Dissertation

Deep Learning in Video Understanding

The Role of Motion in Action Recognition

Filip Ilic

July 26, 2024

Supervisor
Thomas Pock

Co–supervisor
Richard P. Wildes

External Assessor
Christoph Feichtenhofer

Comitee Chair
Friedrich Fraundorfer

Awarding Institution
Graz University of Technology
Institute of Computer Graphics and Vision

Doctoral Dissertation of
Filip Ilic to achieve the degree of
Doktor der technischen Wissenschaften

Supervisor
Thomas Pock

Co–supervisor
Richard P. Wildes

External Assessor
Christoph Feichtenhofer

Comitee Chair
Friedrich Fraundorfer

Awarding Institution
Graz University of Technology
Institute of Computer Graphics and Vision

Preface

I want to thank my supervisor, Prof. Thomas Pock, for his guidance and trust in me. You taught me how to pursue research with passion and dedication, and I am grateful for the opportunity to work with you.

The unwavering support of my secondary supervisor, Prof. Richard Wildes, was invaluable. I am thankful for our regular and thorough discussions, which brought order and structure to our research. If I were to put a picture in the dictionary next to the word ’mentor’, it would be yours. Your dedication to your students and fellows, your gentleness, patience, and wisdom, along with your ability to see the bigger picture, are qualities I aspire to emulate. Working with you made me feel appreciated. Thank you!

With a touch of sadness, I acknowledge my friends and colleagues at the ICG for the conducive working atmosphere. As I prepare to start a new chapter, I cherish the time we spent working together and the opportunity I had to get to know each of you. I will miss our daily interactions. In particular, I thank Markus, who helped me find my footing in the first months of my PhD. His work ethic and broad knowledge in many different fields helped me grow as a person and a researcher. To my office mates, Alex, and honorary office mate Lea, who made me look forward to coming to work every morning; and to all my other colleagues who made the institute a great place to be: **Thank you!**

I want to thank my family and loved ones for their support and encouragement. Mom, Dad, and Jan, thank you! You have always been there for me, and I am grateful for your love and support.

To my footnote-sized girlfriend[†]: Thank you for passionately engaging with me in all our little side projects, all our minor and major discoveries, as well as all the rabbit holes that you fall into with me. I am immensely grateful to have your shoulder to lean on. ♥.

[†] Diana

Abstract

The dynamic nature of the world is integral to human perception, underscoring the importance of the temporal dimension. Many phenomena, ranging from simple locomotion to complex interpersonal interactions, only fully reveal their meaning when observed over time. This temporal perspective is crucial in computer vision, particularly in tasks such as action recognition, where the estimation of optical flow—how objects move across a sequence of images—plays a pivotal role. Furthermore, the principles of image understanding, traditionally focused on static imagery, provide a foundational framework that can inform and enhance video analysis. By adapting insights gained from static contexts, we can approach the complexity of video with strategies for capturing motion.

This thesis explores the considerable advancements in motion-guided action recognition and video understanding within the field of computer vision, facilitated by the integration of machine learning and deep learning technologies. It focuses specifically on the temporal aspects that distinguish video from static image analysis, where optical flow is often a key descriptor in capturing motion dynamics.

We discuss the evolution of machine learning in the broader domain of image and video understanding, detailing the design decisions and trade-offs of successful architectures and the various techniques employed to optimize their functionality. Through this exploration, we aim to provide a deeper understanding of how contemporary video analysis techniques can be applied effectively.

A particular case study in our research is dedicated to action recognition based entirely on motion information, deliberately excluding static cues to avoid biases and enhance system accuracy. We show through a psychophysical study that humans are more adapt at recognizing motion with very limited information, and propose an explicit correlation block to model optical flow directly within the network. Moreover, we use our gained insights to introduce an innovative approach to privacy-preserving action recognition. This approach is particularly relevant given the growing concerns over surveillance and privacy in digital environments.

Kurzfassung

Die dynamische Natur der Welt ist ein wesentlicher Bestandteil der menschlichen Wahrnehmung. Viele Aktionen und Gesten, von der einfachen Fortbewegung bis hin zu komplexen zwischenmenschlichen Interaktionen, offenbaren ihre Bedeutung erst dann vollständig, wenn sie über die Zeit beobachtet werden. Diese zeitliche Perspektive ist in der Computer Vision von entscheidender Bedeutung. Dies ist besonders offensichtlich bei Action Recognition - dem Erkennen von menschlichen Aktivitäten. Dabei wird oft unter, zuhilfenahme des optischen Flusses, welcher die Bewegeung von Objekten in Bildfolgen beschreibt, eine Repräsentation der Dynamik in der Szene geschaffen. Darüber hinaus bieten die Bausteine des Bild- und Videoverstehens, die sich historisch bedingt mehr mit statischen Bildern beschäftigen, einen grundlegenden Rahmen, der die Videoanalyse vorantreiben kann. Erkenntnissen die aus der Bildanalyse kommen, weisen uns in gewisser Maßen den Weg und verdeutlichen welche Strategien und Verfahren in der Videoanalyse verwendung finden können.

In dieser Arbeit untersuchen wir die beträchtlichen Fortschritte der Action Recognition aus Videos. Viele der Verbesserungen in der Videoerkennung wurden erst in den letzten Jahren durch Deep Learning und Machine Learning Ansätze ermöglicht. Diese Verbesserungen konzentriert sich insbesondere auf die zeitlichen Aspekte, die Videos von der Analyse von statischen Bildern unterscheiden. Wir erörtern die Entwicklung des maschinellen Lernens im breiteren Bereich des Bild- und Videoverstehens, wobei wir die Designentscheidungen und Kompromisse erfolgreicher Deep Learning Architekturen und die verschiedenen Techniken zur Optimierung ihrer Funktionalität detailliert beschreiben. Mit dieser Analyse wollen wir ein tieferes Verständnis dafür vermitteln, wie moderne Bild- und Videoalgorithmen effektiv eingesetzt werden können.

Eine besondere Fallstudie in unserer Forschung ist der Action Recognition gewidmet, die ausschließlich auf Bewegungsinformationen basiert. Dabei werden statische Videoelemente bewusst mit einem Rauschen versehen, die es uns erlaubt Bewegungen ausschließlich entkoppelt von Textur, Kontur, und Gestalt zu sehen. Wir zeigen anhand einer User-Study, dass Menschen ebendiese Bewegungen besser als herkömmliche neuronale Netzwerke erkennen können, und schlagen einen expliziten Korrelationsblock vor, um den optischen Fluss direkt im Netzwerk zu modellieren. Darüber hinaus nutzen wir diese Erkenntnisse, um einen Ansatz zur Action Recognition unter der Wahrung der Privatsphäre, im besondereren bezogen auf einzelne Attribute die nicht für Action Recognition relevant sind, vorzustellen. Dieser Ansatz ist angesichts der zunehmenden Realität der permanenten Überwachung und der abnehmenden Privatsphäre in digitalen Umgebungen besonders wichtig.

Affidavit *I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. I agree to the electronic publication of this thesis by the University Library.*

Date	*Filip Ilic*

Contents

List of Figures

List of Tables

Background

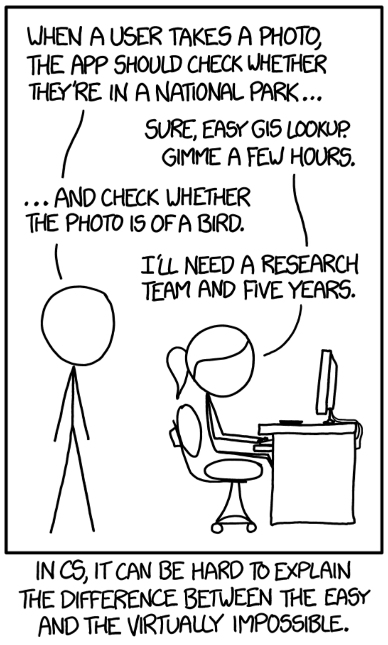
Introduction 1

Contents

1.1 Image and Video Understanding

Motivation

Given the vast array of problems that can be solved with the human visual system, it is no surprise that automation in this regard is and has been a widely studied field for many years. As such, Computer Vision aims to develop algorithms that can automatically extract information from images which enable tasks such as object recognition, image segmentation, and many more. The increase of computational power and the availability of large datasets on the internet have led to significant advancements in the field. A rather recent paradigm shift that can be pinpointed to the early 2010's is the rise of neural networks, and in particular deep learning techniques that leverage the amount of available data very successfully. Many tasks that were once considered to require general intelligence and problem-solving capabilities, or deemed *virtually impossible*, through the lens of the early 2010s (Fig. 1.1), have now been made possible with deep learning [1–3]. These changes affected more than just the computer vision community and shaped all manners of research including natural language processing, robotics, bio-chemistry, and material sciences.



WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.
IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Figure 1.1: Published in September 2014. The comic depicts how it is sometimes difficult to understand from a layperson's perspective what easy and hard problems in computer science are. In this particular case it describes the task of detecting a bird in the image as *Virtually Impossible*. The real punchline of the comic however - winding up over last decade - is that detecting birds in images, *i.e.* image classification is reasonably advanced; solved with learning algorithms. Image classification is in a state that is mature enough to be present in our daily lives, working with great accuracy and efficacy showing the rapid progress in computer vision. Image taken from [4].

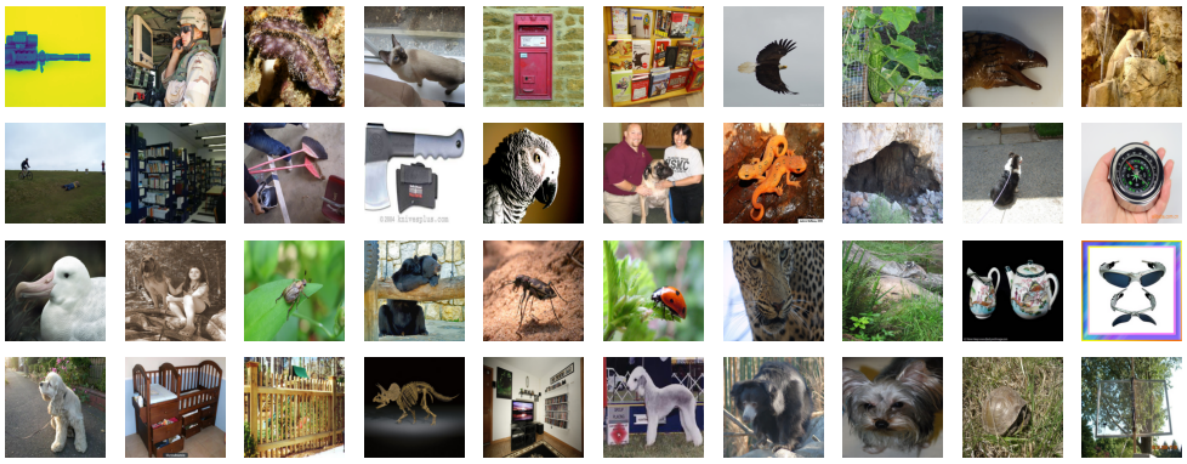


Figure 1.2: Image Classification: Samples from ImageNet [8]. This figure presents a selection of images from the first truly large scale Image dataset containing more than three Million samples with more than 5000 classes. Each image shown represents a different classes such as animals, objects, and scenes. These examples demonstrate the diversity and complexity of images used in training machine learning models for accurate image classification.

Taxonomy of Related Computer Vision Tasks

The subsequent paragraphs explore various aspects of computer vision that are pertinent to broader image and video analysis. This overview serves as a concise taxonomy of different tasks within the field and their applications.

Image Classification Image classification - Fig. 1.2 - is one of the foundational tasks in computer vision, where the goal is to categorize what is depicted in images into predefined classes. Machine learning models, particularly convolutional neural networks (CNNs), have shown remarkable success in this area by learning to identify patterns and features that distinguish different categories [5–7]. Image classification is the basis for more complex tasks like object detection and segmentation. Applications of image classification are vast, ranging from identifying spam images in email filters to diagnosing diseases from medical scans. For example, in social media platforms, image classification algorithms help filter and categorize content, while in medical diagnostics, they assist in classifying images of skin lesions as benign or malignant. The continuous development of more sophisticated models and training techniques has significantly increased the accuracy and reliability of such systems.

Semantic Segmentation Semantic segmentation - Fig. 1.3 - classifies each pixel in an image into predefined categories, providing a detailed and comprehensive understanding of the scene [9–11]. This pixel-level precision is vital for applications requiring a granular interpretation of images. In medical imaging, for example, semantic segmentation is used to delineate anatomical structures, tumors, or other pathological regions, aiding in accurate diagnosis and treatment planning. Autonomous vehicles rely on semantic segmentation to distinguish between different parts of the environment, such as roads, sidewalks, vehicles, and pedestrians, ensuring safe navigation and decision-making.

Object Detection Object detection in visual computing applications involves the identification and, potentially, the localization of objects within an image. This task goes beyond merely recognizing what objects are present; it determines their precise positions within the frame and serves as a basis for many vision tasks that require interaction with the world [13–18]. Modern machine learning algorithms have excelled in this area due to their ability to learn hierarchical feature representations from large datasets. In autonomous driving, for instance, object detection is crucial for identifying pedestrians, other vehicles, traffic signs, and obstacles, allowing the vehicle to navigate safely. Surveillance systems utilize object detection to monitor and detect

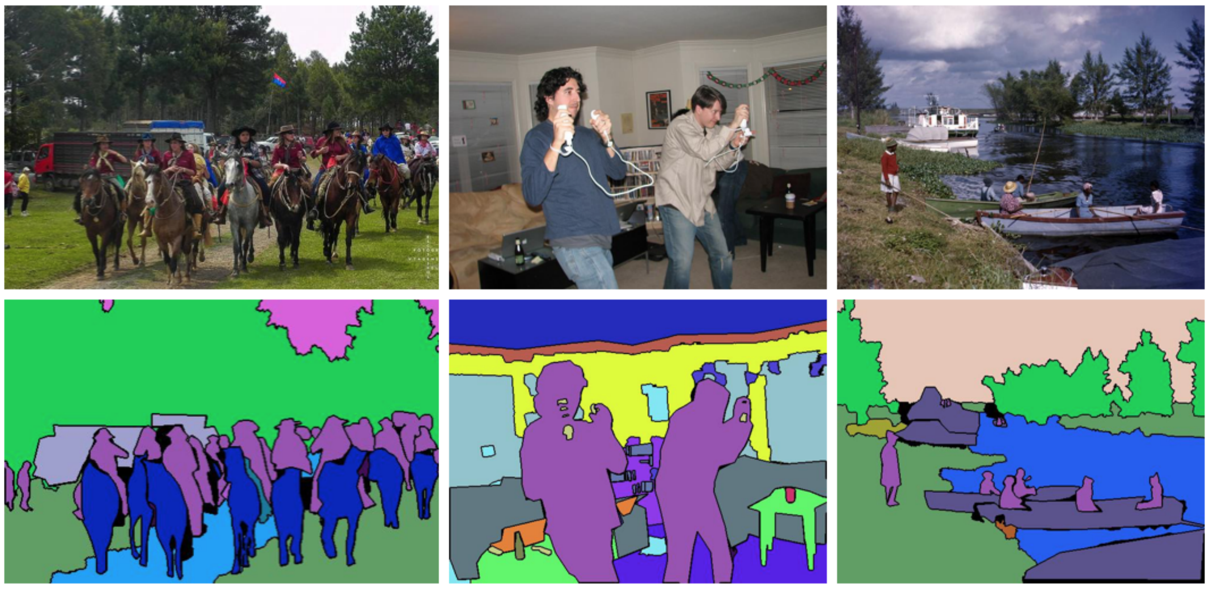


Figure 1.3: Semantic Segmentation: Samples from MS-COCO [12] with corresponding Segmentation Maps. This figure illustrates the concept of image segmentation by displaying a selection of images alongside their segmentation maps. Each image from diverse scenes—such as a festive gathering, a casual indoor activity, and a serene outdoor setting—is paired with its color-coded map that assigns a class label to every pixel in the image. These examples highlight how image segmentation is used to identify and separate various components of an image, useful for tasks such as autonomous driving.

suspicious activities by identifying potential threats. The precision and speed of modern object detection models are continually improving, making them indispensable in these and many other applications that are already in use.

Object Tracking Building upon object detection, object tracking involves the continuous monitoring of identified objects across a sequence of frames in a video. This task is essential for applications requiring an understanding of the movement and interaction of objects over time [20–22]. Advanced techniques have been developed to maintain high accuracy and robustness even in complex and dynamic environments. For example, in video surveillance, tracking a suspect through a crowded area requires algorithms that can handle occlusions, changes in appearance, and varying lighting conditions. In sports analytics, object tracking can analyze player movements, strategies, and performance over the course of a game. Furthermore, in wildlife monitoring, tracking animals helps in studying their behavior and migration patterns. Object tracking, thus, plays a critical role in scenarios where continuous and precise monitoring of moving objects is required.

Action Localization Action localization aims to identify when and where specific actions occur within a video. This task is particularly challenging due to the need to analyze both spatial and temporal information to understand the context and sequence of actions [23–25]. Machine learning models trained for action localization are crucial in sports analytics, where identifying and analyzing player movements and key actions are critical. For instance, action localization can automatically generate highlight reels by pinpointing significant moments in sports footage. In security applications, action localization can detect and alert to suspicious behaviors, such as a person leaving a bag unattended in a public place. Additionally, in human-computer interaction, action localization enables more natural and intuitive interfaces by recognizing user actions and gestures.

Action Recognition While action localization focuses on identifying the temporal and spatial aspects of actions, action recognition, Fig. 1.5, aims to classify the type of action being performed [26–29]. This task is critical in various applications, from video content analysis and human-computer interaction to security and

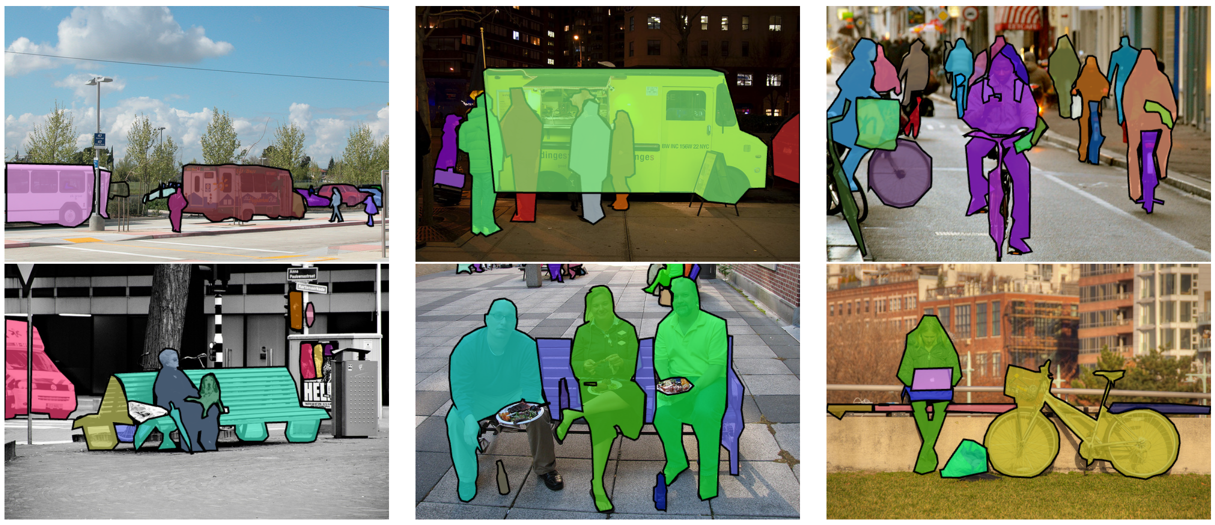


Figure 1.4: Object Detection: Samples from MS-COCO [12] showing the ground truth object segmentation masks for all objects in the scene. While early datasets contained bounding boxes around objects of interest, dataset annotations continued to improve as to minimize the noise that is present in the ground truth. The output of current State-of-the-Art segmentation approaches such as Segment-Anything [19] can come very close to the ground truth annotations and is used for many downstream tasks.

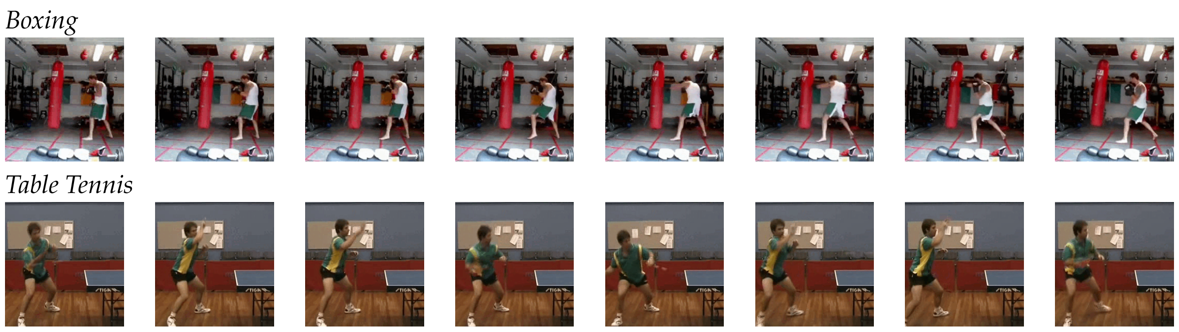


Figure 1.5: Action Recognition: Samples from the UCF101 Dataset [30]. Action Recognition is analogous to Image Classification where a single label is assigned to the input video. By analyzing the spatial and temporal information an action can be deduced. This is crucial for enhancing automated surveillance, video retrieval and search, as well as improving other tasks that depend on the temporal component.

behavior monitoring. For example, in elder care facilities, action recognition systems can monitor residents to detect falls or other emergencies, enabling prompt intervention. In video content analysis, action recognition helps in categorizing and searching large video archives based on specific actions. The field continues to evolve with the integration of multi-modal data, such as combining visual and audio inputs, to enhance recognition performance.

1.2 Outline

The structure of this thesis begins with an introduction to machine learning and its fundamental concepts in Chapter 2. This foundation supports the subsequent exploration of deep learning architectures. Chapter 2 also addresses strategies for visualizing learned representations and discusses enduring concepts in the rapidly evolving field of deep learning. Chapter 3 discusses some of architectures, emphasizing the importance of understanding broader concepts that go beyond simple architectural choices within these networks. This discussion is then extended to focus to include the temporal dimensions that lead to the principal contributions of this work. Chapters 4, 5, and 6 explore specific challenges within the video understanding field, and their solutions. These chapters represent the core technical contributions of the thesis, with the final Chapter 7 summarizing the work and providing an outlook.

Contributions

The primary contribution of this thesis is exploring the role of motion information in video understanding and its impact on action recognition, through three different application. To allow for a better understanding of the various contributions, the preliminary chapter of this thesis - Chapter 2 - introduces core concepts related to video understanding that are used throughout the thesis. A synopsis of each primary contribution is, as follows:

- An in-depth analysis of the challenges associated with deep learning, and concepts that pertain to action recognition and video understanding in particular.
- A method to obfuscate arbitrary videos, probing the capabilities of neural networks to model and discern motion information in videos.
- An application of the aforementioned obfuscation method to create videos which can hide arbitrary privacy relevant attributes in videos, while maintaining the action recognition capabilities of most models.
- A novel method for spatio-temporal clustering based on low-level image features and optical flow, shown to be effective in identifying individual object parts and their movements in videos.

Preliminaries 2

Contents

2.1 Mathematical Preliminaries

This chapter will serve to define required mathematical notation, and introduce core concepts that are important in the context of Machine Learning. We will start with a brief overview of Linear Models, their shortcomings, their strengths and conclude with the introduction of a small Neural Network to solve a simple classification task. The goal of this chapter is to lay the foundation and vocabulary required to understand the chapters that follow, that deal in depth with Deep Learning, different types of Neural Network Architectures, and in particular their applications in Computer Vision and Video Understanding.

Definitions & Core Concepts

Vectors are defined as a one-dimensional array of numbers, with the (scalar) elements of the vector denoted as x_i. We will denote vectors as bold lowercase letters $\mathbf{x}$.

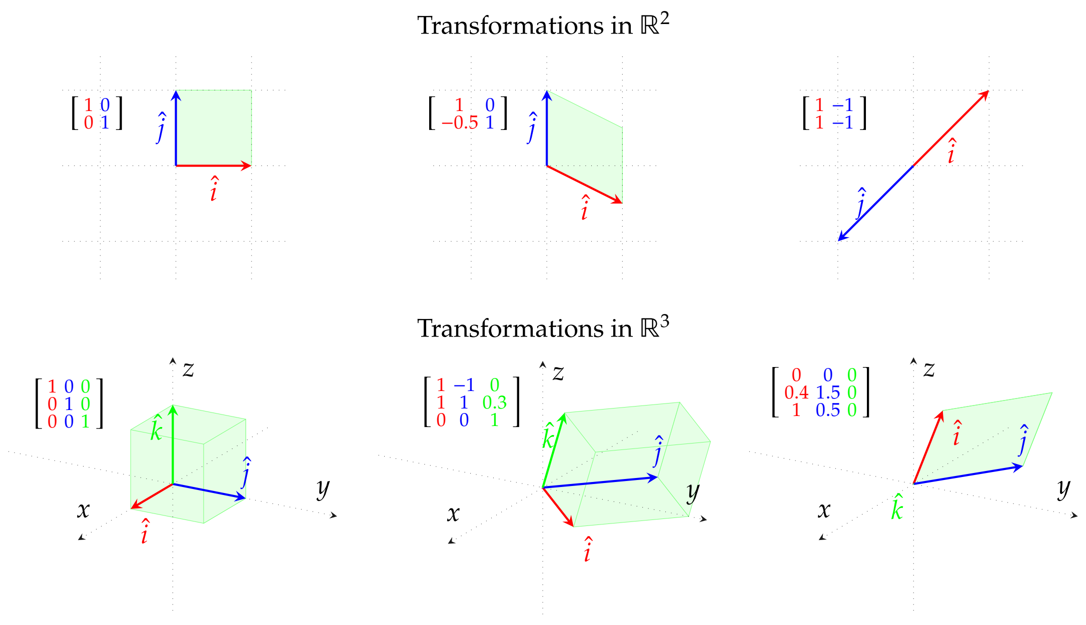
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \mathbb{R} \tag{2.1}$$

It is often required to compute the similarity between two vectors or to project one vector onto another. All of these operations can be expressed in terms of the inner product (often also referred to as dot product) of two vectors; this operation is defined as the sum of the element-wise product of the two vectors.

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i \tag{2.2}$$

It is useful to consider the meaning of the inner product in terms of the angle between two vectors, which leads to another often used similarity measure between two vectors: the cosine similarity.

$$\cos(\theta) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{||\mathbf{x}|| \, ||\mathbf{y}||} \tag{2.3}$$


Transformations in $\mathbb{R}^2$
$\begin{bmatrix}1&0\\0&1\end{bmatrix}$
$\hat{j}$
$\hat{i}$
$\begin{bmatrix}1&0\\-0.5&1\end{bmatrix}$
$\hat{j}$
$\hat{i}$
$\begin{bmatrix}1&-1\\1&-1\end{bmatrix}$
$\hat{i}$
$\hat{j}$
Transformations in $\mathbb{R}^3$
$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$
z
$\hat{k}$
$\hat{j}$
y
x
$\hat{i}$
$\begin{bmatrix}1&-1&0\\1&1&0.3\\0&0&1\end{bmatrix}$
z
$\hat{k}$
$\hat{j}$
y
x
$\hat{i}$
$\begin{bmatrix}0&0&0\\0.4&1.5&0\\1&0.5&0\end{bmatrix}$
z
$\hat{i}$
$\hat{j}$
y
x
$\hat{k}$

(a) Identity **(b)** Full-Rank Matrix **(c)** Singular Matrix

Figure 2.2: Visualization of transformation matrices in $\mathbb{R}^2$ and $\mathbb{R}^3$. The colored vectors represent the basis vectors of the coordinate system, whereas the green parallelogram / parallelepiped represents the transformation of the unit cube, from which the rank and determinant can be inferred. In low dimensions, such as shown here, the rank and determinant can be easily visualized. However, in higher dimensions this becomes increasingly difficult.

Practically, the cosine similarity is often used to compare the similarity of two vectors, as it is invariant to the magnitude of the vectors and only depends on the angle between them. Common norms to measure the length of a vector are the L_1 and L_2 norm. They, in turn, are part of the family of L_p norms (Fig. 2.3) which are defined as

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}, p \geq 1. \tag{2.4}$$

Oftentimes we will encounter the L_1 and L_2 norm particularly in the context of machine learning, either as regularization terms, or as a measure of the error of a model. In other fields of mathematics, such as functional analysis and optimization other norms play a more important role [31].

Another essential entity are tensors of any dimensionality. Tensors are multi-dimensional arrays of numbers, with the elements of the tensor denoted by subscripts $x_{i,\ldots,n}$. Sometimes it is useful to think of a tensor in terms of its components in its column space, i.e. its basis vectors, which are the vectors that span it. Sometimes, however, the notion of a tensor as a container to store numbers is more useful, as in the case with tensors containing image and video data, where there is no meaningful interpretation of the column- or row-space. We will denote tensors as bold uppercase letters $\mathbf{T}$.

$$\mathbf{T} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,m} \end{bmatrix}, \quad \mathbf{T} \in \mathbb{R}^{n\times m} \tag{2.5}$$

A property that all tensors share is that they are linear operators, mapping between two vector spaces that satisfy two key properties of Additivity and Homogeneity:

$$f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) \qquad \text{(Additivity)}$$
$$f(\alpha\mathbf{x}) = \alpha f(\mathbf{x}). \qquad \text{(Homogeneity)}$$

Common operations such as $+, -, \times$, do not require additional explanation with the exception of the

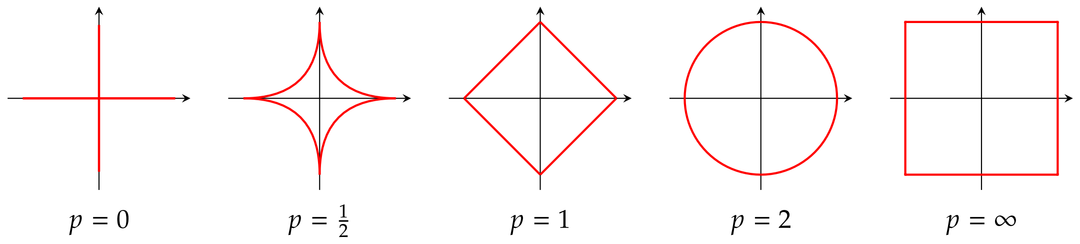


Figure 2.3: The p-norms of a vector $x \in \mathbb{R}^2$ for different values of p. The red curve represents the unit circle in each norm. Note that only $p \geq 1$ are actual norms, whereas $p < 1$ are commonly refered to as *pseudo-norms*.

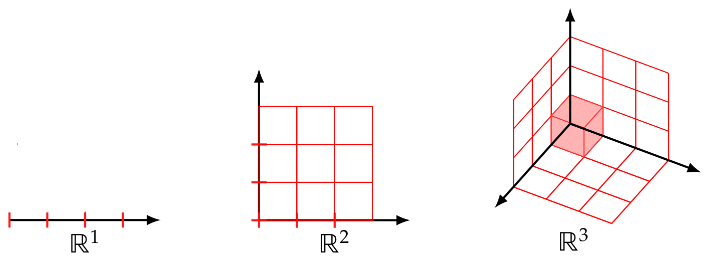


Figure 2.4: The curse of dimensionality. The addition of each dimension increases the volume of the space exponentially. This figure shows the volume of a unit hypercube in $\mathbb{R}^1$, $\mathbb{R}^2$, and $\mathbb{R}^3$. Consider the implications of the curse of dimensionality for image data: Each an image of medium resolution (e.g. $256 \times 256 \times 3$) can be thought of living in a $\approx 200,000$ dimensional space. This means that unless the dataset is very large the space will not be densely populated. Furthermore, an argument can be made that natural images do not indeed need to populate the entire space, but can rather live in a subspace or a manifold embedded in this higher dimensional space.

Hadamard product, denoted by the symbol $\odot$. It is the element-wise product of two tensors of the same size as seen in Eq. 2.6.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \odot \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} ax & by \\ cz & dw \end{bmatrix}. \tag{2.6}$$

Our particular use-case of the Hadamard product we will encounter throughout the thesis is applying element-wise operations such as masking operations in the context of images or video. In particular, we are often dealing with tensors that are of the form $\mathbf{X} \in \mathbb{R}^{C \times H \times W}$, where C is the number of channels, H is the height and W is the width of the tensor. It is assumed that the Hadamard product is applied element-wise to each leading dimension - in this case C if the dimensions of the tensors in all but the leading dimensions are identical.

Dimensionality Reduction We can visualize low-dimensional tensors directly in terms of their basis vectors, as shown in Fig. 2.2. From these visualizations some properties of the tensor can be directly inferred, such as the rank, determinant, and in some cases even the eigenvectors of the tensor. However, for other applications in machine learning, where the tensors might represent images or weights of a network and are high-dimensional, plotting them is not feasible. Dimensionality reduction techniques such as PCA [32] T-SNE [33] and related approaches such as the Singular Value Decomposition [34] can be used to visualize high-dimensional data, see Fig. 2.4, for a visualization in a lower-dimensional space.

Linear Models

The core concept of linear models is that they apply fixed nonlinear functions, known as basis functions, to the input data. These models then express the output as a linear combination of these nonlinear functions. In the case of polynomial regression, the input data is transformed into polynomial terms, but the model remains linear in its coefficients.

This is in general done by applying a set of basis functions ψ to the input data and linearly combining them with a weight w, which is formulated as

$$\langle w, \psi \rangle = w^\top \psi \quad \text{where} \quad \psi(x) = \begin{bmatrix} \psi_1(x) \\ \psi_2(x) \\ \vdots \end{bmatrix}.$$

To illustrate an example we choose basis functions ψ to be the n-th order polynomial, and the input data x to be a 1-D vector. The model is then given by

$$y = \sum_{i=0}^{n} w_i \psi_i(x) \quad \text{where} \quad \psi(x) = \begin{bmatrix} x^0 \\ x^1 \\ \vdots \\ x^n \end{bmatrix},$$

where w_i corresponds to the coefficient for the respective polynomial term in $\psi(x)$. The linear model aims to find the best set of w_i values that allow the model to accurately represent the relationship between the input x and the output y. The individual outputs y_s for each sample s are often joined together in the single vector

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix},$$

as well as the weights w being combined in a single matrix $\mathbf{W}$. The linear model can then be written in matrix form with the basis functions already applied to the input data, i.e. ,

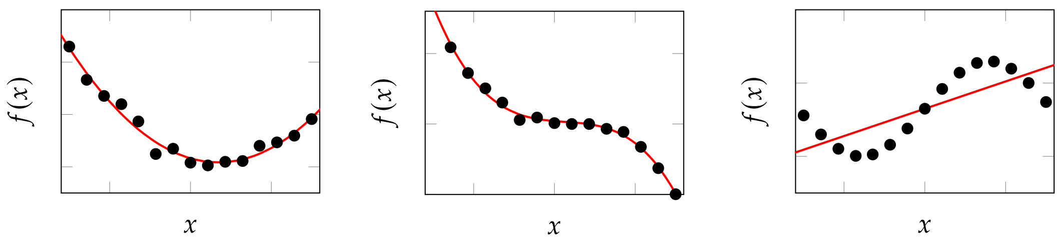
$$\mathbf{Y} = \mathbf{\Phi} \mathbf{W}, \tag{2.7}$$

where $\mathbf{\Phi}$ is the design matrix. It contains the transformations of individual samples s_i, in this case the polynomial basis functions of the n-th order polynomial, i.e.

$$\mathbf{\Phi} = \begin{bmatrix} | & | & & | \\ s_1 & s_2 & \dots & s_m \\ | & | & & | \end{bmatrix}.$$

To be more explicit, assume that we are given a sample $x_1 = 2.1$ and $x_2 = 3.4$ with their respective ground truths as $y_1 = 3.2$ and $y_2 = 1.2$. We can then transform the input data into the polynomial basis functions, and stack them into a design matrix $\mathbf{\Phi}$, i.e.

$$s_1 = \begin{bmatrix} 1 \\ 2.1 \\ 2.1^2 \\ \vdots \\ 2.1^n \end{bmatrix}, \quad s_2 = \begin{bmatrix} 1 \\ 3.4 \\ 3.4^2 \\ \vdots \\ 3.4^n \end{bmatrix}, \quad \text{and} \quad \mathbf{\Phi} = \begin{bmatrix} 1 & 1 \\ 2.1 & 3.4 \\ 2.1^2 & 3.4^2 \\ \vdots & \vdots \\ 2.1^n & 3.4^n \end{bmatrix}.$$



(a) Good Quadratic Fit **(b)** Good Cubic Fit **(c)** Bad Quadratic Fit

Figure 2.5: Linear regression on three different samples, with different polynomial features. We can see that the quadratic model in 2.5a and the cubic model in 2.5b fit the data well as they match the underlying generating function $f(x)$. However, note how in 2.5c a quadratic model is used to fit a sinusoidal function, which results in a poor fit.

Similarly we can transform the ground truth into a vector $\mathbf{Y}$, i.e.

$$\mathbf{Y} = \begin{bmatrix} 3.2 \\ 1.2 \end{bmatrix}.$$

The design matrix then contains the entire dataset with each column representing a sample and each column representing a feature of all data points. We can now try to find the best weights $\mathbf{W}$ such that our prediction, $\mathbf{\Phi W}$, matches the ground truth $\mathbf{Y}$. One intuitive error function to minimize is the sum of squared errors, which is given by

$$\min_{\mathbf{W}} \frac{1}{2} \|\mathbf{\Phi W} - \mathbf{Y}\|_2^2 . \tag{2.8}$$

For models of this form, gradient descent algorithms are not necessary, as an analytic solution exists, resulting in what is known as the Moore-Penrose Pseudoinverse [35, 36], $\mathbf{\Phi}^+$.

$$\mathbf{W} = \underbrace{(\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T}_{\mathbf{\Phi}^+} \mathbf{Y} \tag{2.9}$$

Examples of a quadratic and cubic linear polynomial model that fit given data very well are shown in Figure 2.5a and 2.5b. The problem with this method of function fitting is apparent in 2.5c. The basis functions are fixed in advance and cannot be adapted based on evidence provided by the data. If the data, in this particular case a simple $\sin(x)$, is not well represented by the chosen basis functions, the model simply cannot accurately represent the relationship between input and output.

2.2 Neural Networks

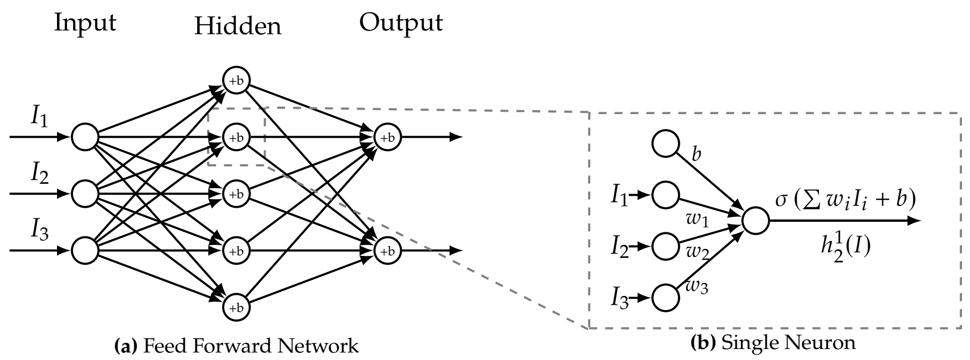
Feed Forward Networks

Instead of using a priori fixed basis functions, neural networks learn the basis functions from the data. The computation in a neural network is accomplished by applying a linear transformation followed by a non-linear activation function at each neuron. This approach allows neural networks to automatically adapt and optimize the basis functions based on the given data, making them highly flexible and powerful for a wide range of tasks. In a typical neuron of a neural network, the computation can effectively be written as a combination of weighted inputs, summed with a bias term, and then passed through an activation function, i.e.,

$$h(x) = f\left(\sum_{i=0}^{n} w_i x_i + b\right), \tag{2.10}$$

where $h(x)$ represents the output of the neuron, x_i the input, and w_i associated with the input at position i. The bias b is added to the sum before it is passed through a non-linear activation function σ, or in literature often confusingly referred to as f. To avoid confusion between the whole networks that are usually called f_θ (with θ being the collection of all weights and biases in a network) we will use σ for the activation function. Neural networks are often arranged in layers of neurons, where all layers that are not directly the output or input nodes are referred to as a hidden layer h, as seen in Fig. 2.6. When referring to hidden neurons we will use the convention of using h_k^l where l refers to the layer in the network, and k identifies the k-th neuron in that layer.

The distinguishing feature of linear models is that the weights w_i that act as composite functions in a neuron are learned from the data, and the non-linear activation function σ is applied element-wise to the output of the transformation. A few common activation functions are shown in Fig. 2.7. The Rectified Linear Unit (ReLU) and Gaussian Linear Unit (GeLU) are the de facto standard in many deep learning applications due to their ease of computation and robustness with respect to vanishing gradients. Understanding how different activation functions impact the network's learning process is crucial for optimizing model performance. Moreover, activation functions play a vital role in introducing non-linearity into the model, enabling it to learn complex patterns.



(a) Feed Forward Network **(b)** Single Neuron

Figure 2.6: A simple feed forward neural network with one hidden layer (left) and the anatomy of a single neuron (right). The left side illustrates a network architecture consisting of an input layer, a hidden layer, and an output layer. Each layer is fully connected to the next, allowing information to be passed forward through the network. The right side focuses on a single neuron within the network. Note that in the single neuron diagram, $\sigma(\cdot)$ represents the fixed activation function, as described in Fig. 2.7. The activation function determines the neuron's output based on its input. The learned parameters are the bias b and the weights w_i, which are adjusted during training to minimize the error of the network's predictions. The bias arrow b for every hidden unit is omitted in (a) to help with clarity, but is attached to every hidden neuron, as shown in (b). The used shorthand notation places each corresponding bias term in the hidden unit, denoted with $+b$. All learnable parameters of a neural network are often denoted collectively as θ, and are referred to as the weights of the network. These parameters are crucial as they define the network's ability to model complex functions.

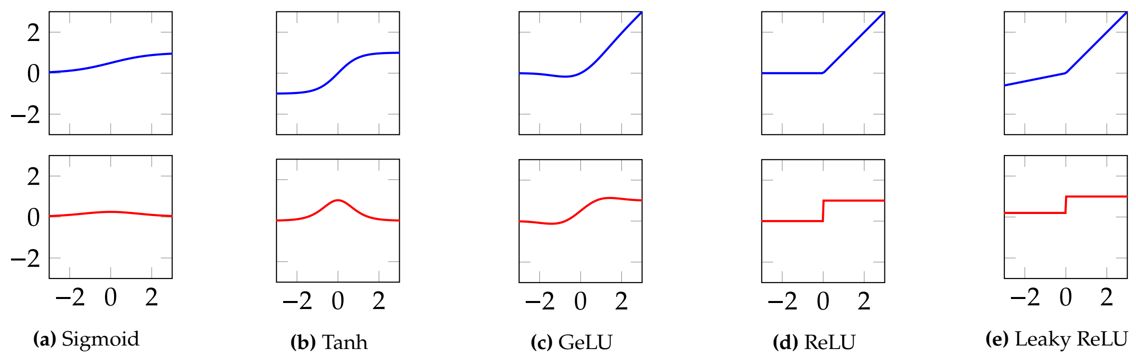


Figure 2.7: Common Activation functions (Top) and their derivatives (Bottom). Some functions like the Sigmoid and Tanh may suffer from saturation problems, meaning that the gradient becomes very small for large or small inputs if the inputs are not normalized appropriately. This can lead to problems in deep networks as very little gradient information might be propagated, leading to no discernible improvement in the loss. The *Rectified Linear Unit* (ReLU) and its adaptation, the *Leaky Rectified Linear Unit* (Leaky ReLU) are popular choices for deep networks as they do not suffer from saturation problems. Recently, the *Gaussian Error Linear Unit* (GeLU), a smooth approximation of the ReLU, has shown real world benefits in the domain of large language models, even considering an increase in computational complexity compared to the ReLU.

Example: Approximating a 1-D function We will briefly look at a simple example of a feed forward neural network that consists of a single hidden layer with three neurons, and a single output neuron. The network is trained to approximate a simple 1-D function, see Fig. 2.8. It is useful to think of each neuron separating its input into two regions. Each neuron effectively divides its input into active and inactive regions, enabling piece-wise linear approximations of the target function, with each neuron in the hidden layer contributing to this approximation. The output is then a linear combination of these activations, weighted by the output neuron's weights. This simple example illustrates the power of neural networks to learn functions from data, and how they can be used to approximate functions that are not well represented by fixed basis functions.

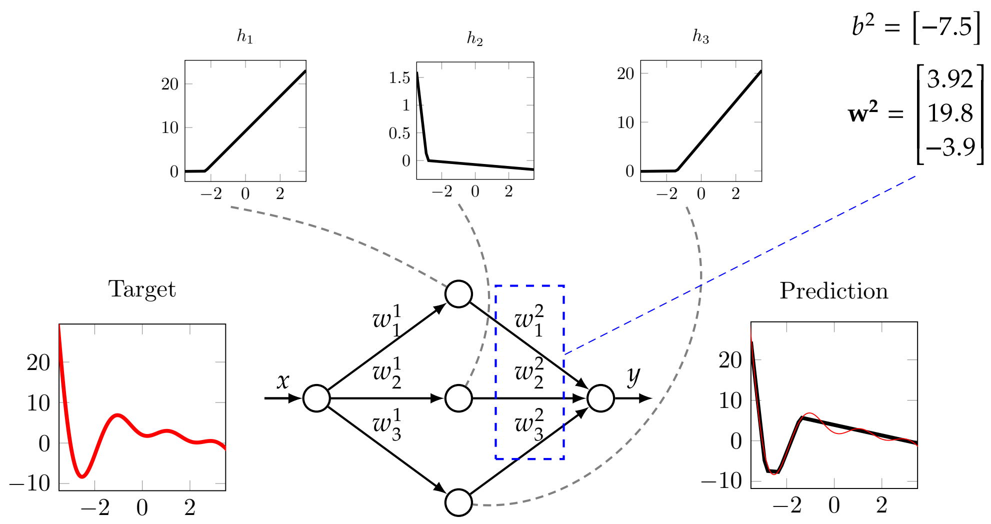


Figure 2.8: We can visualize the outputs of a hidden layer in a neural network by plotting the output of each neuron in the hidden layer. The hidden layer activations are shown in the plots above the hidden layer nodes. The target function, $f(x) = -4 + e^{-x}\sin(2x) + e^{\frac{x}{2}} + \cos(3x)$, is shown on the left and the output of the network is shown on the right. We can see the piece-wise linear approximation of the target function by the network. As the activation function we use the *LeakyReLU*, see Fig. 2.7. We can also see how the final output is a linear combination of the hidden layer activations. For example we can see that h_2 is responsible for the sharp drop from -2.5 to -2 and is therefore scaled accordingly with the weight $w_2^2 = 19.8$ and is positive. The same argumentation and reasoning can be applied to the other hidden layer activations to produce the final output. Observe that the piecewise linear approximation, is depending on the application already a good approximation of the target function, even with a shallow network only consisting of one hidden layer, with three neurons.

An example of how different activation functions effect the approximation of a target function is shown in Fig. 2.7. The experimental setup is identical to Fig. 2.8 and only the activation functions are changed.

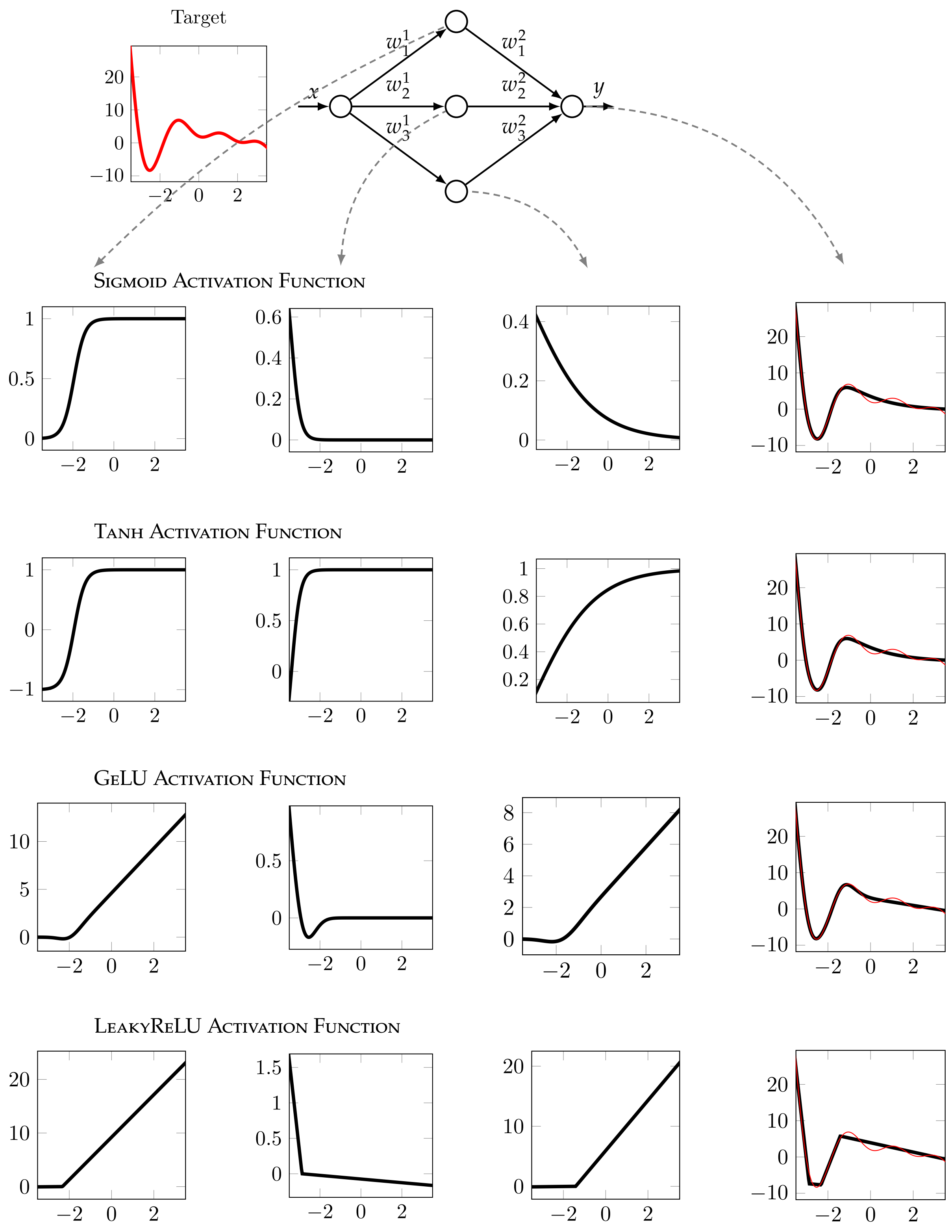


Figure 2.9: This figure builds on the scenario shown in Fig. 2.8 by illustrating the impact of various activation functions on the individual neuron activations in a feed-forward neural network. We visualize the outputs of a hidden layer with different activation functions (see Fig. 2.7) - Sigmoid, Tanh, GeLU, and, LeakyReLU. The same target function, $f(x) = -4 + e^{-x}\sin(2x) + e^{\frac{x}{2}} + \cos(3x)$, is used. Each activation function influences the neurons and the resulting approximation of the target function differently. For instance, LeakyReLU provides sharp transitions, while Tanh and Sigmoid produce smoother curves. The outputs of each hidden unit are displayed in a stacked format below, with the final output being a linear combination of these hidden layer activations, shown in the rightmost column. We have organized the hidden units so that those with similar contributions to the shape of the output function are grouped together in columns. It is important to note that some hidden neurons have negative weights, meaning that an activation function producing an output that appears dissimilar might actually contribute in a similar way to its counterpart with a different activation function, due to the sign flip from the negative weight.

The *Universal Approximation Theorem* [37] underpins this example, stating that even a single hidden layer can approximate any continuous function to any degree of accuracy, provided it has enough neurons, as shown in Fig.2.10. We can see that the same target function is approximated with a network consisting of a single hidden layer, with 3, 10, and 50 hidden units. It is important to keep in mind that the approximations of the function in Fig. 2.10 are only accurate in the region where the network has seen sample data. In the regions where the network has not seen any data, the approximation may be extremely poor. This is shown in Fig. 2.11, where a target function is approximated fairly well by a network with 50 hidden neurons. However if we pass x values outside of the range of the training data - indicated by the gray regions - the prediction is wildly inaccurate.

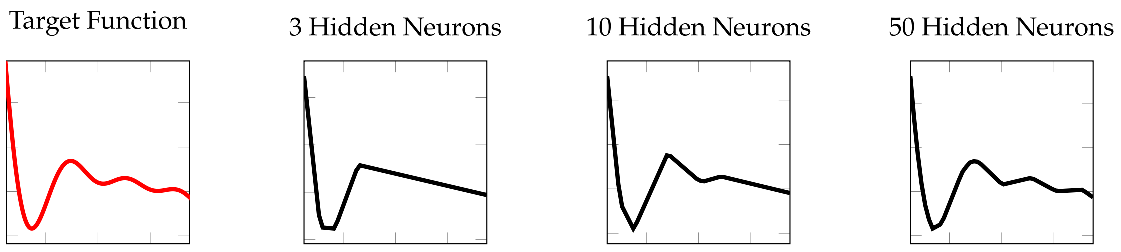


Figure 2.10: A single hidden layer neural network can approximate any continuous function, to arbitrary precision, given enough hidden neurons. In literature [37] referred to as the *Universal Approximation Theorem*. In this particular example, we see the approximation of a function using a neural network with 3, 10, and 50 hidden neurons, following the example in Fig. 2.8.

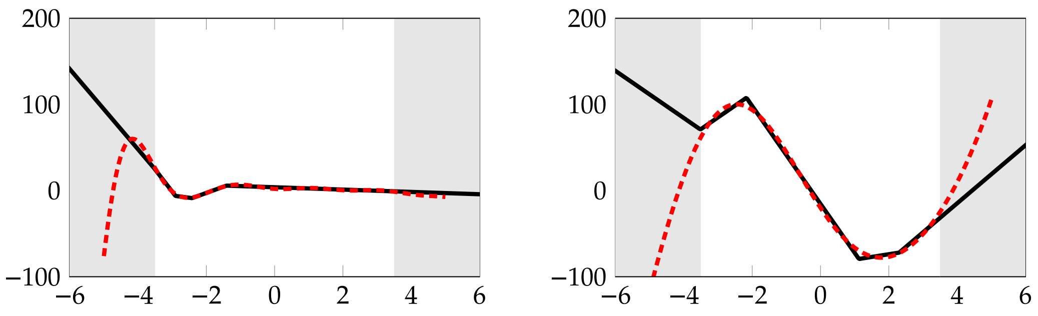


Figure 2.11: While the target function (red) is well approximated by the neural network (black) in the white region, the approximation is poor in the gray regions, as no data in the gray regions was used to train the network.

Loss Functions In the previous example of regressing a 1D function we intuitively knew that a reasonable error is the error between the prediction and the true value of the target function. This difference is known as the residual, and is a measure of the error of the model. This error function is referred to as a Loss function, $\mathscr{L}$. There are many feasible loss functions, depending on the problem at hand. In the case of regression problems, the Mean Squared Error (MSE), the Mean Absolute Error (MAE), and Huber-Loss with the parameter δ, of the residual error (x) between prediction $(\hat{y})$and ground truth (y) are commonly used

$$\mathscr{L}(x) = |x| \tag{MAE}$$

$$\mathscr{L}(x) = \frac{1}{2}x^2 \tag{MSE}$$

$$\mathscr{L}_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq \delta \\ \delta\left(|x| - \frac{\delta}{2}\right) & \text{if } |x| > \delta \end{cases} \tag{Huber}$$

For a visual intuition of the increase in error as the prediction deviates from the true value, see Fig. 2.12.

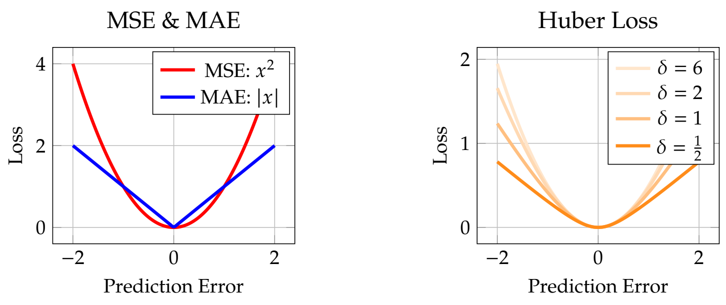


Figure 2.12: Loss functions for different prediction errors. (a) shows the Mean Squared Error (MSE) and Mean Absolute Error (MAE), showing how the MSE increases rapidly as prediction errors increase, and the MAE is shown by a blue V-shaped curve that linearly increases with prediction errors. The right plot (b) shows variations of the Huber loss function for different values of δ. These curves demonstrate how the Huber loss interpolates between MAE and MSE, becoming more quadratic as δ increases. This provides a smooth and differentiable approximation to MAE, which is less sensitive to outliers than MSE. This visualization helps to understand how the choice of δ affects the sensitivity of the Huber loss to outliers and errors of individual samples during training.

Assume a network f, parametrized by learnable parameters[†] θ, that maps input data $\mathbf{x}$ to a prediction $\hat{\mathbf{y}}$ through possibly multiple hidden layers, and a ground truth $\mathbf{y}$. As discussed previously we can use the L_2 norm (= MSE) to measure the dissimilarity between the prediction and the ground truth, which written explicitly is given as

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{y}}) = \frac{1}{2} || \underbrace{f_\theta(\mathbf{x})}_{\mathbf{y}} - \hat{\mathbf{y}} ||_2^2. \tag{2.11}$$

However, we will mostly be concerned with classification problems, where the output of the network is not a single scalar value, but rather a probability distribution over classes. In the context of a two class classification problem, the output can still be a single scalar value, which is then passed through a sigmoid function to normalize the output into the range $[0, 1]$, implicitly assigning all values below ≤ 0.5 to one class, and values above > 0.5 to the other class. The sigmoid function is given by

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{2.12}$$

where z is the output of the network, and is commonly called Logit. The extension of the sigmoid function to multi-class classification is the SoftMax function, which normalizes the output of the network into a probability distribution over K classes. The SoftMax is given by

$$\text{SoftMax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}. \tag{2.13}$$

The loss function used for a multi-class is the cross-entropy loss, which is given by

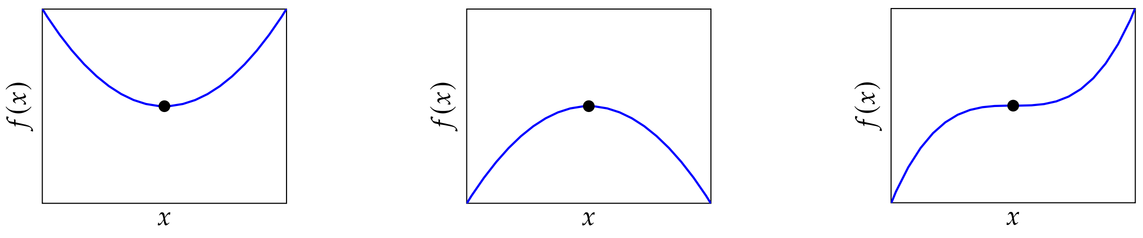
$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i), \tag{2.14}$$

where y_i is the true label of the i^{th} class, and $\hat{y}_i$ is the predicted probability of the i^{th} class.

The error surface, E, with respect to the parameters, θ, of the network is given by the sum of the loss function over all samples N in the dataset containing, i.e.,

$$E(\theta) = \sum_{n=1}^{N} \mathcal{L}(f(x_n), \hat{y}_n). \tag{2.15}$$

[†] In literature w is often used to refer of the weights of linear models or small feed forward networks, whereas θ is used for larger collections of weights. It is therefore common to see f_θ to refer to a network and its parameters.



(a) Minimum **(b)** Maximum **(c)** Saddle Point

Figure 2.13: Stationary Points of $f(x)$ are characterized by $f'(x) = 0$. For any meaningfully complex function, the loss surface will have multiple local minima, maxima and saddle points. If no analytic solution exists, gradient methods are used to find minima / saddle points of the loss function.

Trying to find a closed form solution for arbitrary neural networks is, unlike in the case of the linear models, not possible as no closed form solution to of the form $\nabla_\theta E(\theta) = 0$ exists. We therefore need to resort to iterative optimization algorithms to find the optimal weights of the network that minimize the error function $E(\theta)$. Gradient Descent is such an algorithm that can be used to iteratively update θ in the direction of the negative gradient of the error function with respect to the weights of the network.

Gradient Descent The gradient of the loss function is computed with respect to the weights of the network, and is used to update the weights in the direction that minimizes the loss function. The update rule for *Steepest Descent* for the weights is given by

$$\theta_{t+1} = \theta_t - \eta \nabla E(\theta_t), \tag{2.16}$$

where θ_t is the value of the weights at iteration t, η is the learning rate, and $\nabla E(\theta_t)$ the gradient of the error function with respect to the network weights. This gradient contains the partial derivatives of the error function with respect to each weight of the network,

$$\nabla E(\theta) = \left[\frac{\partial E}{\partial \theta_1}, \dots, \frac{\partial E}{\partial \theta_n}\right]. \tag{2.17}$$

The learning rate η is a hyper-parameter that controls the size of the step taken in the direction of the gradient. If the learning rate is too large, the optimization algorithm may overshoot the minimum and diverge. If the learning rate is too small, the optimization algorithm may take unreasonably long to converge to a minimum. Different heuristics exist to deal with the fact that the first order approximation of the function is only valid in the vicinity of the current point, such as strategies that employ an adaptive learning rate [38], momentum based optimizers [39, 40], or even optimizers that use second order information to make informed decisions about the descent direction. In the case of the latter, the Hessian matrix is used to compute the curvature of the error function, and can be used to improve the descent direction based on the curvature of the loss function at the current point. However, in practice, the full Hessian matrix,

$$\mathbf{H} = \nabla^2 E(\theta) = \begin{bmatrix} \frac{\partial^2 E}{\partial \theta_1^2} & \cdots & \frac{\partial^2 E}{\partial \theta_1 \partial \theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial \theta_n \partial \theta_1} & \cdots & \frac{\partial^2 E}{\partial \theta_n^2} \end{bmatrix}, \tag{2.18}$$

is rarely used due to its computational complexity. Instead, approximations of $\mathbf{H}$, such as the diagonal of $\mathbf{H}$, are sometimes used [38].

Optimizers We discussed in the previous section that the weights of a neural network are adjusted (*'learned'*) by minimizing a loss function. This is done by iteratively updating the weights of the network in the direction of the negative gradient of the loss function that is computed over the entire dataset; this is *Gradient Descent*

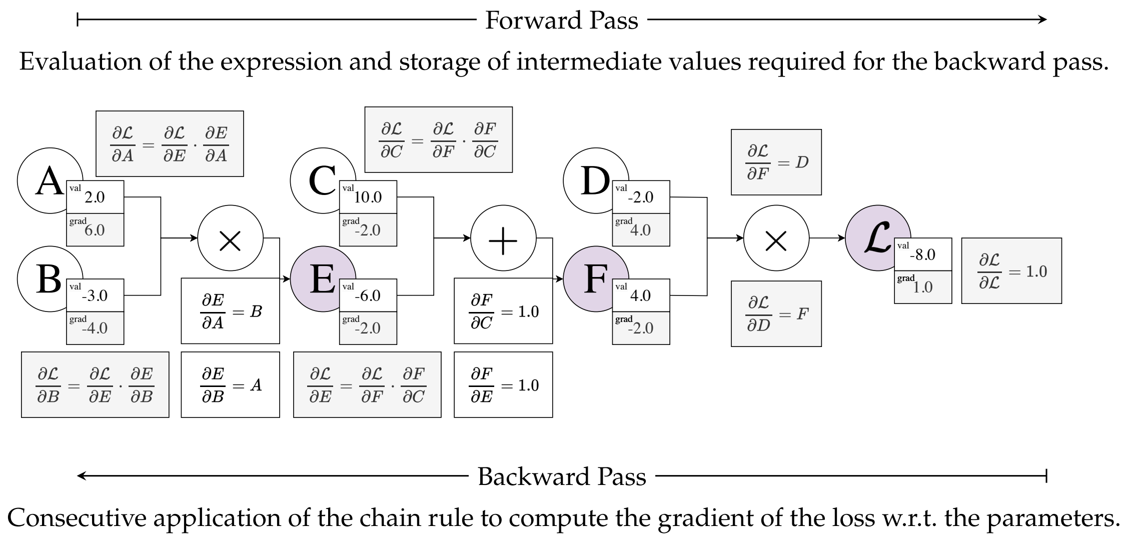

Forward Pass
Evaluation of the expression and storage of intermediate values required for the backward pass.
$\frac{\partial\mathcal{L}}{\partial A} = \frac{\partial\mathcal{L}}{\partial E}\cdot\frac{\partial E}{\partial A}$
$\frac{\partial\mathcal{L}}{\partial C} = \frac{\partial\mathcal{L}}{\partial F}\cdot\frac{\partial F}{\partial C}$
$\frac{\partial\mathcal{L}}{\partial F} = D$
A
val 2.0
grad 6.0
B
val -3.0
grad -4.0
C
val 10.0
grad -2.0
E
val -6.0
grad -2.0
D
val -2.0
grad 4.0
F
val 4.0
grad -2.0
$\mathcal{L}$
val -8.0
grad 1.0
$\frac{\partial\mathcal{L}}{\partial\mathcal{L}} = 1.0$
$\frac{\partial E}{\partial A} = B$
$\frac{\partial F}{\partial C} = 1.0$
$\frac{\partial\mathcal{L}}{\partial D} = F$
$\frac{\partial\mathcal{L}}{\partial B} = \frac{\partial\mathcal{L}}{\partial E}\cdot\frac{\partial E}{\partial B}$
$\frac{\partial E}{\partial B} = A$
$\frac{\partial\mathcal{L}}{\partial E} = \frac{\partial\mathcal{L}}{\partial F}\cdot\frac{\partial F}{\partial C}$
$\frac{\partial F}{\partial E} = 1.0$
Backward Pass
Consecutive application of the chain rule to compute the gradient of the loss w.r.t. the parameters.

Figure 2.14: The computational graph of the function $\mathcal{L}(A, B, C, D) = ((A \times B) + C) \times D$, showing the computation of the intermediate values, E and F, in the forward pass (left-to-right), and the gradient computation with the partial derivatives in the backward pass (right-to-left).

see Eq. 2.16. For large datasets Gradient Descent itself is rarely used because it requires the computation of the gradient over the entire dataset for a single update. Furthermore, it does not take any previous iterations of the gradient step into account to potentially accelerate the convergence of the optimization algorithm. *Stochastic Gradient Descent* which uses a subset of the data to compute the gradient, is often more practical. The subset of the data that is used to compute the gradient is called a mini-batch. The update rule for the weights stays the same, with the learning rate η often adapted based on heuristics throughout training. The learning rate can be adapted in a number of ways, such as using a learning rate schedulers [41, 42], warmup-schedules [43], and good initialization schemes [44, 45], or momentum based optimizers [39, 46]. The de facto standard optimizer for training neural networks however, is the Adam optimizer [47], which is an adaptive learning rate optimizer that computes the learning rate for each parameter individually and also incorporates momentum.

Backpropagation and Automatic Differentiation Finding the gradient of the loss function with respect to the weights of the network is a non-trivial task, as the loss function is a composition of many functions. Fortunately *Backpropagation* [48, 49], a technique that computes the gradient of the loss function with respect to the weights of the network, is an efficient way to compute the gradients needed for the optimization algorithm to work effectively and at scale. It works by applying the chain rule of calculus throughout the network back-to-front to break down complex derivatives into simpler parts, which can be numerically computed.

In Fig. 2.14 we show a sample computational graph consisting of simple operations such as addition and multiplication. Throughout the forward pass intermediate nodes that contain the current result are computed, as well as values stored that are needed to compute the partial derivatives in the backwards pass. For the backwards pass we start at the output node $\mathcal{L}$ and trivially compute the gradient of $\frac{\partial L}{\partial L} = 1.0$. This kickstarts the backward pass and the gradient is passed to the predecessors of L, namely D and F. The local gradients are computed and stored in the respective nodes. Because the $\times$ operator was used to combine L, *i.e.* $L = D \times F$ we can compute the partial with respect to both variables, and store them. It is helpful to think of each node in the computational graph incorporating local gradient information, and the in-flowing gradient information from the parent nodes, which in this case is just $\frac{\partial L}{\partial L} = 1.0$. To compute $\frac{\partial L}{\partial C}$ we need to apply the chain rule for the fist time; The local gradient information $\frac{\partial F}{\partial C}$ is combined with the previously computed in-flowing gradient from $\frac{\partial L}{\partial F}$. The code that implements the computational graph of Fig. 2.14, and the corresponding gradients computed by an autograd engine, Pytorch [50], is shown in Fig. 2.15. In this fashion it is possible to compute arbitrarily complex expressions with automatic differentiation, which is neither a fully numerical gradient method, nor a purely symbolic gradient method. From an implementation perspective, it is a highly extensible and robust way to compute gradients, as each function only needs to implement a forward and backward pass to be integrated into any graph. A through and in depth exploration of the history, development, and optimization of automatic differentiation see [51].

code

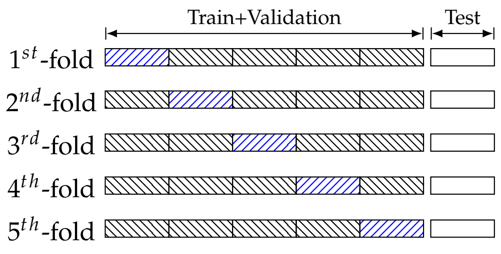
```
import torch

# Define tensors with gradients
A = torch.tensor(2.0, requires_grad=True)
B = torch.tensor(-3.0, requires_grad=True)
C = torch.tensor(10.0, requires_grad=True)
D = torch.tensor(-2.0, requires_grad=True)

# Perform forward pass
E = A * B
F = C + E
L = D * F

# Retain gradients for intermediate variables
for var in [E, F, L]:
    var.retain_grad()

# List of variables and their names
vars = {"A": A,"B": B,"C": C,"D": D,"E": E,"F": F,"L": L}

# Backward pass
L.backward()

# Display values and gradients
for name, node in vars.items():
    print(f"| {name} | val:{node.item()} \t | grad:{node.grad.item()} \t|")
```

output

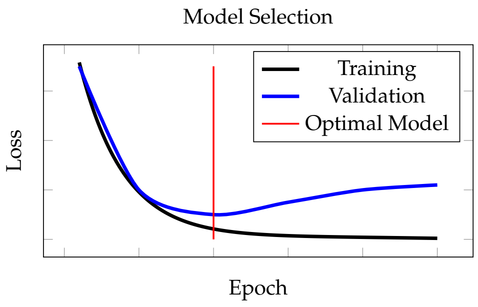
```
| A | val:2.0    | grad:6.0     |
| B | val:-3.0   | grad:-4.0    |
| C | val:10.0   | grad:-2.0    |
| D | val:-2.0   | grad:4.0     |
| E | val:-6.0   | grad:-2.0    |
| F | val:4.0    | grad:-2.0    |
| L | val:-8.0   | grad:1.0     |
```

Figure 2.15: Computational graph of a simple function, and the code that uses automatic differentiation to compute the gradients.

Datasets & Model Selection In this section, we discuss the structure and use of datasets in training machine learning models, as well as strategies for model selection to ensure that the trained networks generalize to unseen data. A dataset typically consists of n individual samples, which is divided into a train- and a test-set. The train-set is used to train the model, while the testing set is reserved exclusively to evaluate the performance of the model *once*. This division helps to assess how well the model can generalize to new, unseen data. Moreover, a technique called k-fold cross-validation is often used, where the dataset is segmented into k equal parts. During the validation process, each segment is used once as a validation-set while the remaining segments collectively form the train-set. This method is illustrated in Figure 2.17a.

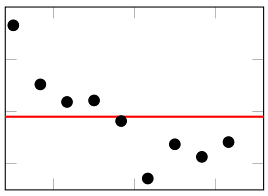


(a) k-fold cross validation

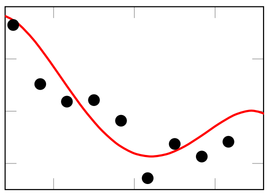


(b) Early Stopping

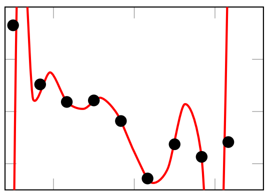
Figure 2.17: Model selection strategies during training. (a) k-fold cross-validation is often used to determine the performance of a model, to tune hyperparameters, and for the model selection to avoid over- and under-fitting. The dataset is divided into k equal parts. $k-1$ parts are used as the train set, whereas the last part is used as validation set. The performance of the model is then averaged over the k folds. (b) Early Stopping; during training the validation loss is monitored to prevent overfitting. This is done by comparint the training loss and the validation loss. It is expected that the training loss will continously decrease, while the validation loss will decrease initially, but at some point will begin to increase again as overfitting on the data happens. The optimal model that should be chosen is the one that minimizes the validation loss.



(a) Underfitting



(b) Good Fit



(c) Overfitting

Figure 2.16: Depiction of model complexity and data fit: (a) Underfitting, where the simple red curve fails to represent the underlying structure of the data points; (b) Good Fit, where the red curve accurately captures the structure of the data points; (c) Overfitting, where the complex red curve not only represents the structure but also the noise within the data points. Strategies such as k-fold cross-validation can help to determine model good parameters.

A significant disparity between train error (near zero) and test error (substantially higher) usually indicates overfitting, see Fig. 2.16a. This occurs when the model fits precisely to every data point in the training set, neglecting the fact that these points may include noise and may not represent the underlying data structure accurately. To select the epoch at which overfitting has not yet occurred the validation loss is monitored, and training is stopped when the validation error starts increasing while the train error is still decreasing, as this indicates overfitting cf. Fig. 2.17b. Additionally it is advisable to begin training with a model of lower complexity (i.e. fewer hidden units) and gradually increase the capacity by adding more layers and units until it sufficiently captures the essential patterns of the data.

In addition to the dataset itself, datasets are often released with predefined splits, i.e., divisions between training, validation and test sets. This standardization allows various models to be fairly compared and evaluated on the same data, allowing competing algorithms to be evaluated fairly against each other.

Example: Classification of 2D data In Section 2.2 we discussed the structure of a simple feed forward neural network, and how it can be used to approximate a 1-D function. We minimized the MSE, and used the gradient of the error function to update the weights of the network. We will now look at a slightly more complex example of a neural network that is used to classify 2-D data points, with the cross-entropy loss as defined in Eq. 2.14. In particular, we consider our dataset to consist of two classes, where the class is determined by the distance to the origin, as shown in Fig. 2.18. The goal now, is to train a neural network to classify the points into the correct class. The network consists of a single hidden layer with three neurons, ReLU activation functions, and two output neurons. The use of two output neurons is simply to demonstrate that this approach scales to an arbitrary number of classes, since for a 2 class classification problem a single sigmoid activation would suffice, see Eq. 2.13 and Eq. 2.12. After training the network according to the

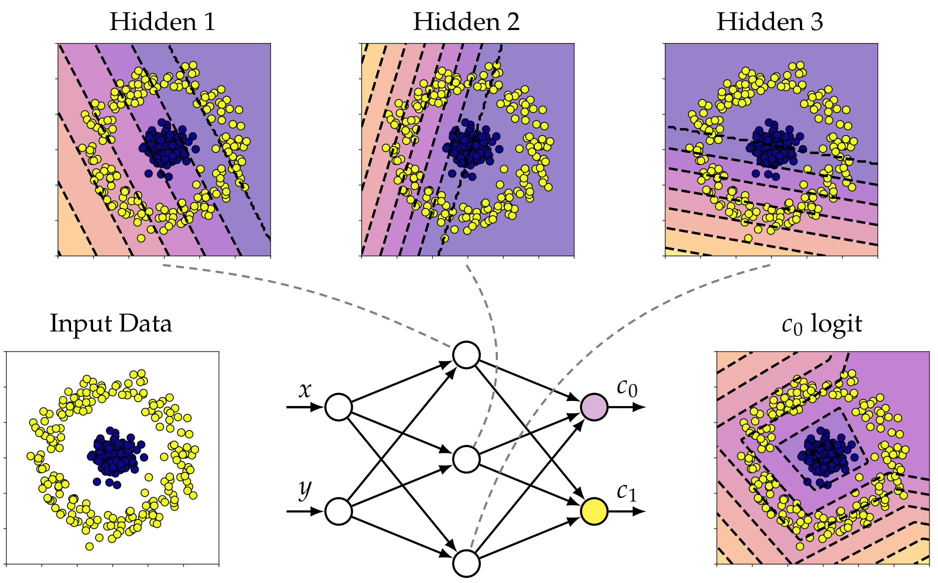


Figure 2.18: The dataset contains samples with x, y coordinates. Depending on the location of the sample, it is classified as c_0 or c_1, which we visualize as violet and yellow respectively classes respectively. The dashed lines show the visualization of the hidden layer neurons as decision boundaries just as was the case in the 1-D example, see Fig. 2.8. Visualization of the decision boundaries of the hidden layer neurons are shown in the top row, and the decision boundary that is a linear composite of the hidden layer neurons is shown on the right. We only show the logit for c_0 for simplicity. We can see how the hidden layer neurons that are each able to separate the input data into a flat region and a region with a slope, can be combined such that a decision boundary that appears to be a square is formed.

principles discussed in Section 2 with the proper train- and test-set splits we can inspect the learned piecewise linear mappings in the hidden neurons.

The decision boundaries of the network for different activation functions can be plotted by passing a dense grid of input points to the network, and monitoring the logits for each class. This is shown in Fig. 2.19 on the right. Again note how the piece-wise constant activations induced by the ReLU activation function allow the network to learn piece-wise linear decision boundaries, which are able to separate the two classes in the input space. However as the input is two dimensional (x, y coordinates of each point) each input neuron splits the input space again into two regions; one where the neuron is active and the slope is positive, and one where the neuron is inactive and the slope is zero. The output neuron then combines these regions to form the final decision boundary. Figure 2.18 shows the visualization of the planes formed by each neuron in the hidden layer, and the final decision boundary that is obtained by a linear combination of the hidden neurons. Note how the Sigmoid / SoftMax function squashes and amplifies the discrepancy (*i.e.* steepness of the plateau) between the two classes, Fig. 2.19.

The progression of the learning processes for a few selected epochs is shown in Fig. 2.20, as well as the influence of the different activations functions that are used in the hidden layer. The activation functions used were shown earlier in Fig. 2.7.

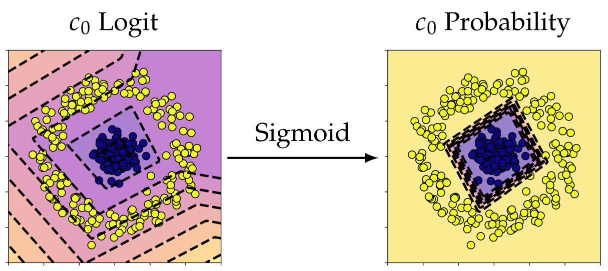


Figure 2.19: Given the logit of class c_0 we can apply the sigmoid function to get the probability of the class. We see how the sigmoid function squashes the logit to a probability, which produces this sharp decision boundary along which the probability changes from 0 to 1. Note that the steepness of the decision boundy can inform us about the confidence of the model.

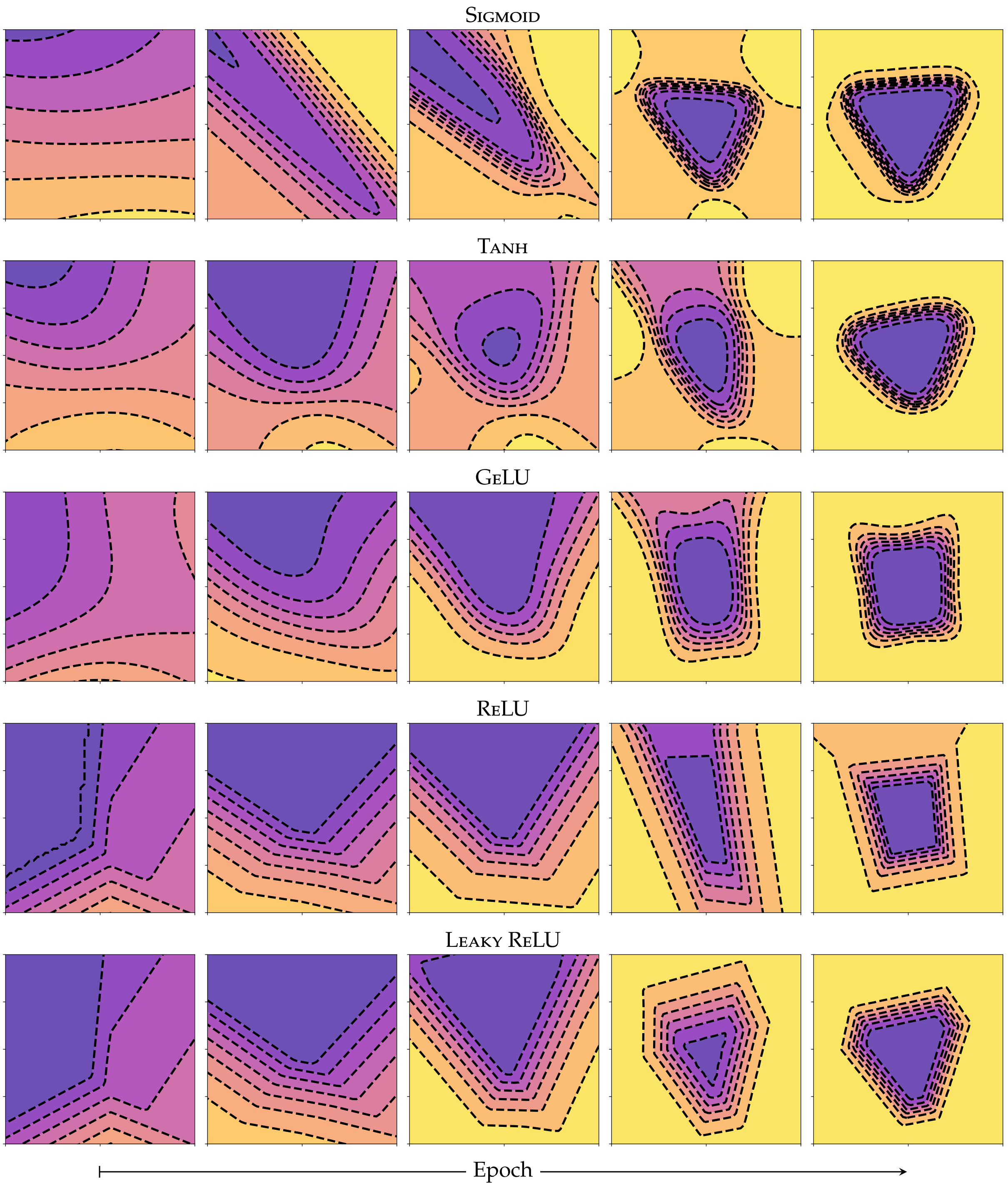


Figure 2.20: Adaptation of the decision boundary over time for different activation functions. The decision boundary is shown in the x, y plane, where purple indicates a high confidence for class 0 (c_0) and yellow indicates a high confidence for class 1 (c_1). Note how the decision boundary changes over time and how the activation function affects the shape of the decision boundary. Also note that it is only possible for such a simple dataset to display the decision boundary directly in the input space. Note that the plots above do not represent a fully converged network and are for illustration purposes only. A fully converged network and its decision boundary is shown in Fig. 2.19.

Convolutional Networks

Until now we limited ourselves to discussing feed forward neural networks, where the input data is transformed through a series of linear transformations and non-linear activation functions. However, in many applications, such as image and video processing, the data is high-dimensional and has a grid-like structure. In such cases, convolutional neural networks (CNNs) which are specifically designed to handle grid-like data, are great alternatives. CNNs are composed of convolutional layers, pooling layers, and fully connected layers. The convolutional layers apply a set of filters to the input data; an example of the convolution operation with a fixed kernel **K** applied on an image **I** is shown in Fig. 2.21. which allows the network to learn spatial hierarchies of features. The pooling layers downsample the input data, reducing its dimensionality; this is shown in Fig. 2.22.

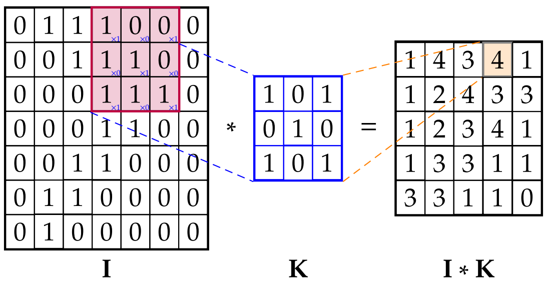


Figure 2.21: Functional diagramm of a 2D convolution. In convolutional neural networks the weights of **K** are learned to detect certain patterns in the input **I**. The output of the convolution is obtained by sliding the kernel **K** over the input **I** and computing the element-wise product and summing the results. In convolutional neural networks the output **I** ∗ **K** is refered to as *Activation Map* or *Feature Map*. In the case of a neural network the activation map is also passed through a non-linear activation function, as previously discussed.

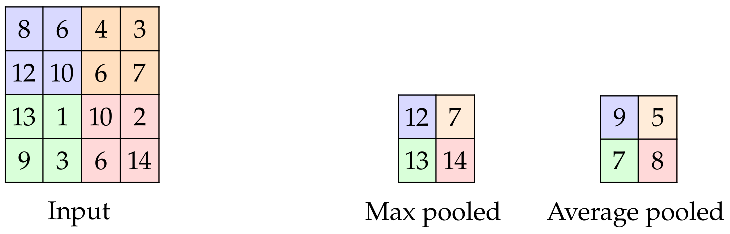


Figure 2.22: The most common pooling operations are max pooling and average pooling. In this example we show the result of applying a 2×2 max, and average pooling operation with stride 2 on a matrix. Their purpose is to reduce the spatial dimensions of the input tensor, and thereby reduce the number of parameters in the model, as well as shift invariance. Pooling operations are common throughout many different neural network architectures and are commonly applied after the activation function in a convolutional layer.

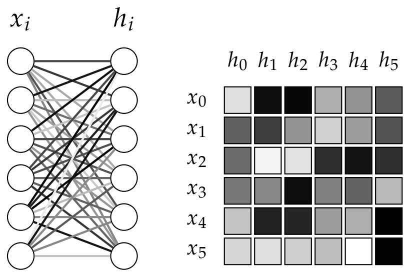
Concretely the convolution operation[†] in 1D is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \qquad \text{(Continuous)}$$

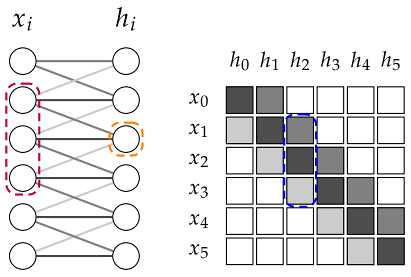
$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m), \qquad \text{(Discrete)}$$

with f being the input signal, g being the kernel, and $*$ being the convolution operator. The convolution operation is commutative, associative, and distributive. Note that while convolutions are defined analytically as integrations from negative to positive infinity, in practice the kernels have relatively small support, e.g., on the order of 3 to 7 pixels in modern convolutional neural networks.

[†] In practice cross-correlation is used, which is just convolution with a flipped kernel. However as the machine learning literature refers to this as convolution we will follow the convention. Practically, it also makes no difference whatsoever as the kernels are learned from data.

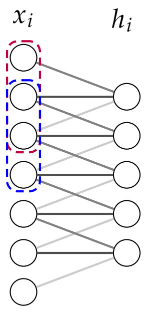


(a) Fully connected layer with 25 weights.

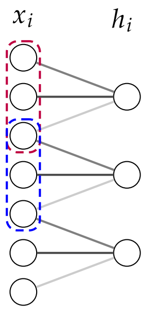


(b) Convolutional layer with 3 weights.

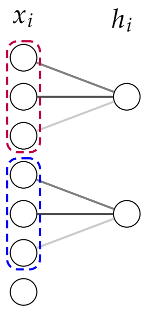
Figure 2.23: A visual representation of the different scaling of weights in a fully connected layer and a convolutional layer. The fully connected layer has a total of 25 multiplicative weights and 5 bias terms totalling 30 learnable parameters, whereas the convolutional layer consists of 3 multiplicative weights and 3 bias terms totalling 6 learnable parameters. Note that we chose the receptive field to be 3 for the convolutional layer, and a stride of 1. The stride is the number of steps the receptive field is moved across the input. In (b) we highlight the actual multiplicative kernel weights and how they are overlayed on top of the inputs x_i, indicated with the black dashed box to produce h_2. It also graphically shows how the idea of *weight tying*, is used to create the inductive bias in convolutional networks. Figure inspired by illustrations from [52].



(a) Stride 1



(b) Stride 2



(c) Stride 3

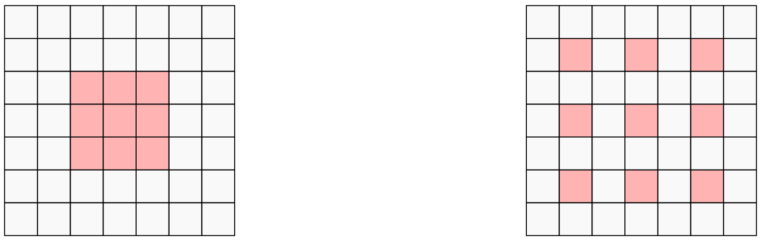
Figure 2.24: The stride parameter in convolutional layers determines the step size at which the convolutional kernel is slid across the input. We can observe that larger strides result in smaller output sizes. All figures show a convolutional layer where no padding on the input is applied. The purple and blue boxes indicate the first and second position of the convolutional kernel as it is slid across the input.

We also want to draw the distinction between convolutional layers and fully connected layers. In a fully connected layer, each neuron is connected to every neuron in the previous layer, as shown in Fig. 2.23. This results in a weight matrix that is unique between every connections between neurons in the previous and current layer. This is in contrast to convolutional layers, where each neuron is connected to a local region of the input data, which can also be seen in Fig. 2.23 as the banded structure of the weight matrix, that is the same in the support of the convolutional kernel.

The convolution operation itself is also often parametrized with stride and dilation parameters that greatly influence the receptive field of of deep convolutional neural networks. The stride of the convolution determines the step size at which the kernel is slid across with the input signal, see Fig. 2.24. Often the stride parameter is used as a way to downsample the input signal, instead of directly applying pooling layers. This has the benefit of saving computation, but trades of spatial sampling of the input signal.

Another important concept is that of kernel dilation, see Fig. 2.25. It is a technique used to expand the receptive field of filters without increasing the number of parameters or the computational complexity. Dilation involves spacing out the elements of the convolution kernel, inserting gaps between each element within the kernel window. This allows the kernel to cover a larger area of the input signal, enabling the network to incorporate broader contextual information from the input data into the feature maps it produces. By adjusting the dilation rate, a network can control the scale of context it extracts, with higher dilation rates capturing wider spatial relationships. Dilation effectively enlarges the field of view of filters, allowing deeper layers to see further into the input without the need for pooling or larger convolutional filters.

As shown in Fig. 2.21, the spatial resolution decreases if the convolution is directly applied to the input. In practice it is often desirable to keep the spatial resolution constant. To achieve this the input is often padded



(a) Dilation factor 0 **(b)** Dilation factor 1

Figure 2.25: A 3×3 convolutional filter with dilation factors 0 and 1. The dilation factor determines the spacing between the elements in the convolutional kernel, and is used to increase the receptive field of the network without increasing the number of parameters. In the example above we can see how the receptive field of a filter increases, at the cost of spatial fidelity.

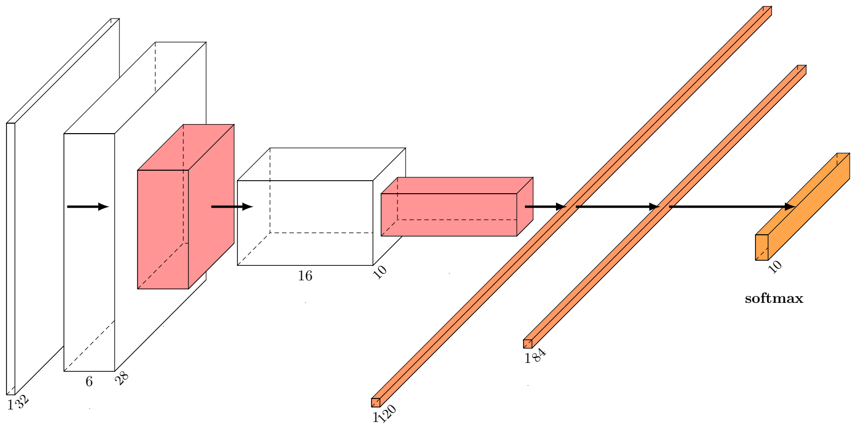


Figure 2.26: The LeNet-5 architecture consists of 3 convolutional layers, 2 pooling layers, and 2 fully connected layers. It introduced several key components that have become standard components in modern CNNs, such as downsampling through convolution and pooling, followed by fully connected layers. The network was designed to classify handwritten digits from the MNIST dataset.

around the edges of the input image with half the size of the convolution kernel. The values with which the input is padded is often set to zero, but can vary depending on the application and can include reflection or padding with the value at the border itself.

Examples of early Convolutional Networks Fukushima et al. [53] introduced in their seminal work the idea of convolutional networks, motived by the structure of the visual cortex. Other early work such as LeNet-5 [54], developed by Yann LeCun et al. in 1989 was one of the first successful applications of convolutional networks to handwritten digit recognition. A schematic of the LeNet-5 architecture is shown in Fig. 2.26. The architecture is relatively simple by todays standards, yet it introduced several key components that have become standard components in modern CNNs. The network features a series of layers that include convolutional layers, subsampling layers (pooling layers), and fully connected layers. What makes LeNet-5 notable is its systematic use of convolutional layers to learn hierarchical feature representations of the input images - a concept that is at the heart of all modern convolutional networks.

VGG16 [6], developed and published in 2014 (25 years after LeNet-5), marked a significant advancement in convolutional neural networks. This architecture distinguished itself through its depth and the use of very small (3×3) convolution filters, a novel approach at that time. The model consists of 16 layers, including 13 convolutional layers and 3 fully connected layers, demonstrating the effectiveness of deeper networks for complex image recognition tasks. The schematic of the network is shown in Fig. 2.27. The appeal lies in its straightforward, uniform structure that allows for the stacking of small filters to achieve a receptive field similar to those of larger filters but with deeper non-linearities and fewer parameters. However, despite its successes, VGG16 contrasts with modern convolutional networks in several significant ways. Today's architectures are more parameter-efficient, using techniques such as batch normalization, skip connections, or depth-wise separable convolutions to reduce the parameter count while enhancing performance. The

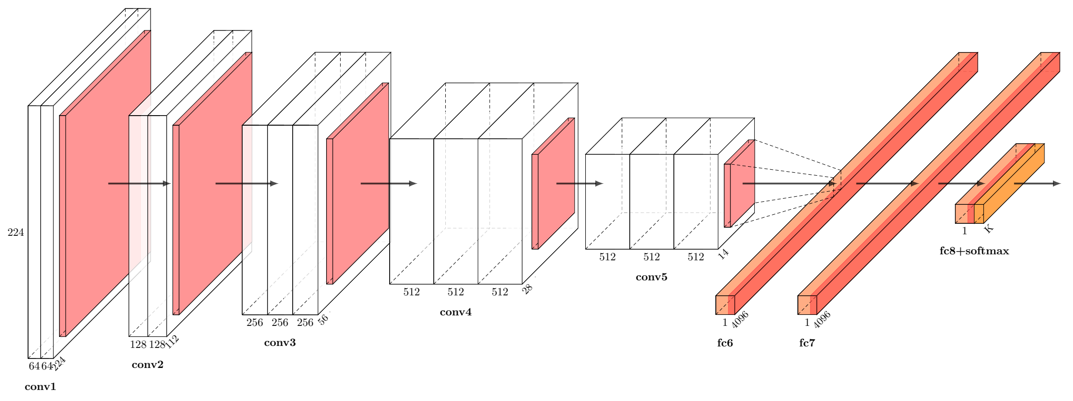


Figure 2.27: The VGG16 architecture consists of 5 distinct convolutional stages, where in each stage the spatial resolution stays the same. In each stage convolutions are followed by a ReLU activations, and at the end of each stage the activations are pooled to reduce the spatial resolution. After the **conv5** block three fully connected layers further reduce the number of activations until the final fully connected layer has the same number of outputs as there are classes in a classification task. Lastly a SoftMax function is applied to the output of the network to normalize the output into a probability distribution over the classes. In terms of terminology, all convolutional layers are often referred to as the *Backbone* of the network, and the fully connected layers are referred to as the *Head* of the network.

introduction of residual connections, which will be discussed in the next Chapter, in newer models helps to train very deep networks more effectively by alleviating issues like vanishing gradients, which can affect deep networks like VGG16.

Inductive Bias Convolution is invariant with respect to translation of the input signal, which means that the network can learn to recognize patterns in the input data regardless of their position in the input signal. This inductive bias is particularly useful in image processing, to detect objets in images, independent of their position in the image. This is in contrast to fully connected networks, which do not have this inductive bias and must learn to recognize patterns in the input data regardless of their position.

It is important, however, to distinguish between invariance and equivariance, as they are often used interchangeably but have different meanings. Given a function $f(\cdot)$ and a transformation $T(\cdot)$ the following properties hold:

$$
\begin{aligned}
f(T(x)) &= f(x) && f \text{ is Invariant w.r.t. } T\\
f(T(x)) &= T(f(x)). && f \text{ is Equivariant w.r.t. } T
\end{aligned}
$$

It is desirable that networks for image classification are invariant to some transformations in the image space as they do not change the underlying class that is depicted in the image. A cat is still a cat regardless of its position in the image. We therefore want the prediction of the network to be invariant to translations of the input image. CNNs often incorporate mechanisms to handle invariance implicitly through pooling layers, which help to abstract away positional details while retaining crucial information. The final fully connected layers that operate on the features extracted from the convolutional layers are then responsible for combining the features to form a classification decision that is invariant to the position of the object in the image. However, CNNs inherently lack the ability to be equivariant to transformations such as rotation or scaling. Other methods of convolution, such as harmonic networks [55], have been developed to address this limitation. They are not as widely used as traditional CNNs for many common tasks such as object detection, as many problems relating to rotation and scale can be solved through data augmentation, indirectly. Methods to introduce positional awareness into convolutional networks exists and generally follow the idea of adding spatial coordinates to the input data, see Fig. 2.28. Depending on the task and especially if the dataset enforces some sort of spatial structure, it can be beneficial supplement the input images with image coordinates [56].

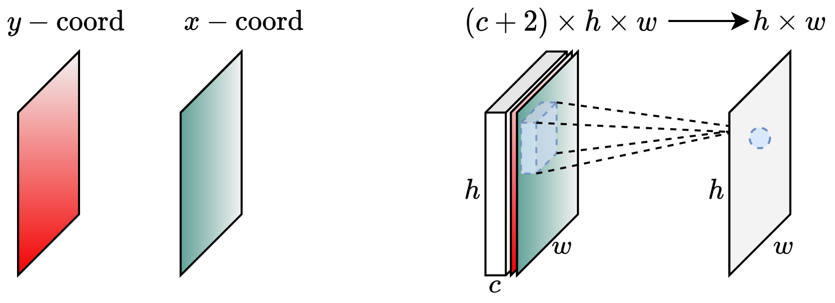


Figure 2.28: The figure showcases the Coordinate Convolution (CoordConv) approach, which integrates the Y-coordinate and X-coordinate matrices, each matching the input's height and width $h \times w$, into the input image's channel dimension, resulting in an augmented input of $(c + 2) \times h \times w$, with c being the original channel count. This strategy incorporates spatial coordinates into the convolution, enabling the network to leverage pixel position information. Many types of coordinate convolution have been tried such as polar or sinusoidal coordinate representations, where spatial context is key. All these strategies have in common that they try to enhancing the model's spatial awareness and understanding.

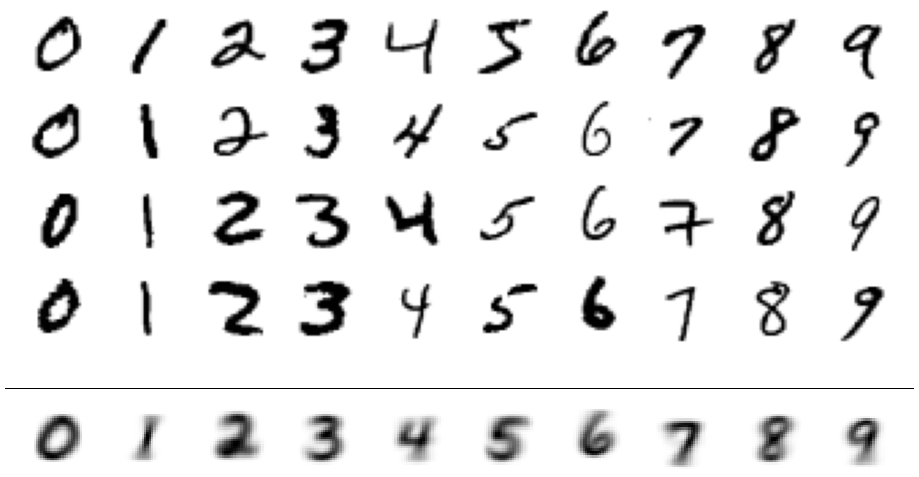


Figure 2.29: The MNIST dataset comprises 70k images of handwritten digits, each 28×28 pixels in size. The last row shows the average digit computed over the entire dataset.

Interpretation and Visualization

Linear Models and simple MLPs The interpretation of the weights of a model is a powerful tool to understand the inner workings of the model, and to gain insights into the structure of the data that the model has learned. We will compare the weights of a linear model to the weights of a neural network (without non-linear activation functions) that has been trained on the same dataset. We will see that the weights of a linear model can be interpreted as templates, with a high response if the input image resembles the template. Similarly we will see that the weights of a neural network in the last layer can just as easily be interpreted as templates. This insight is important because for most classification tasks a network can semantically be split between its *Backbone*, and its *Head*. The *Backbone* is responsible for extracting rich feature representations through different mechanisms such as convolution, pooling, non-linear activation, normalization or attention. The *Head* however is often just one (or more) fully connected layers that act on the extracted features, and can therefore be interpreted as a template matcher, *i.e.* a linear classifier.

We will use the MNIST dataset to illustrate this point, and compare the weights of a linear classifier to the weights of a neural network that has been trained on it. The MNIST dataset consists of 70,000 images of handwritten digits, each 28×28 pixels in size. It is popular due to its simplicity and the notable interclass diversity it presents, such as variations in stroke thickness and slant, as can be seen from plotting a few samples of the dataset in Fig. 2.29. We will consider a linear classifier that directly operates on the pixel values of the input data. More concretely this means that we will reshape each 28×28 image into a 784-dimensional

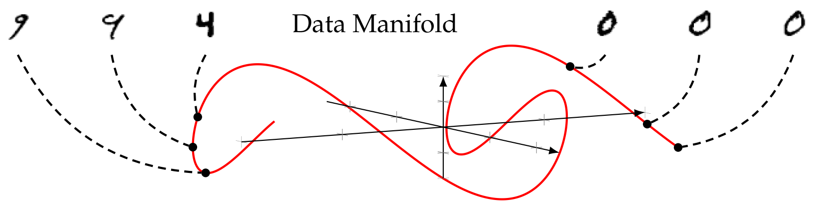


Figure 2.30: The Manifold Hypothesis: The data is embedded in a low-dimensional manifold within the high-dimensional image space. It effectively means that images (or other data) do not populate the entire space densely, but rather are concentrated around certain structures within it. It explains the success of deep learning models in learning representations, as the networks both transform the data into a lower-dimensional feature representation while learning at the same time a partitioning of that subspace according to the task. A prominent example of this is the success of auto-encoders [57–59] in learning low-dimensional representations of data.

vector, i.e. $x_i \in \mathbb{R}^d, d = 784$ and assemble our design matrix Φ from the reshaped images, where each image is treated as a column in the matrix

$$\Phi = \begin{bmatrix} | & | & & | \\ \mathrm{x}_1 & \mathrm{x}_2 & \dots & \mathrm{x}_n \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{d \times n}.$$

We further encode the labels of the dataset as one-hot vectors, and denote the labels as $\mathbf{Y} \in \mathbb{R}^{10 \times n}$, where n is the number of samples in the dataset, and 10 is the number of classes in the dataset. This allows us to learn a linear classifier that maps the input data to the labels of the dataset, i.e. $\mathbf{W}\Phi = \mathbf{Y}$. Graphically expanded it looks as follows:

$$\underbrace{\begin{bmatrix} — & w_1 & — \\ & \vdots & \\ — & w_{10} & — \end{bmatrix}}_{10 \times d} \underbrace{\begin{bmatrix} | & | & & | \\ \mathrm{x}_1 & \mathrm{x}_2 & \dots & \mathrm{x}_n \\ | & | & & | \end{bmatrix}}_{d \times n} = \underbrace{\begin{bmatrix} | & | & & | \\ y_1 & y_2 & \dots & y_{10} \\ | & | & & | \end{bmatrix}}_{n \times 10}. \tag{2.19}$$

Visualization of the learned templates Each weight of the weight matrix **W** of the neural network can be reshape to the dimensions of the input data. The resulting images that can be interpreted as the templates for a particular class are shown in Fig. 2.31. The reason they can be interpreted as a template, is that the output **Y** is computed as the inner product of the input data with all the weights of the network, ⟨sample, templates⟩. Seeing the structure in Eq. 2.19 we can observe that one sample $\mathbf{x_i}$ is multiplied with all weights **W** where each row in $\mathbf{w_i}$ is responsible for the logit in the resulting vector $\mathbf{y_i}$. The learned and reshaped templates of the linear classifier and the neural network are shown in Fig. 2.31. Similarly we can also visualize the weights that are obtained by solving the linear classification problem with the Moore-Penrose Pseudoinverse, as described in Eq. 2.9. Note how both the linear classifier and the neural network have learned templates that resemble the average digit of the MNIST dataset, as shown in Fig. 2.29 in the bottom row.

Connected to the idea of feature extraction and partitioning is the manifold hypothesis that suggests that the data manifold is embedded in a much lower dimension than the input data. This is an important concept as it suggests that the essential features or variations in the data can be captured by a lower-dimensional manifold. Networks at the same time transform the input data in a way that the essential features can be captured by a lower-dimensional representation, and then can perform partitioning in this lower-dimensional space. An analogy of how such a manifold might look is shown in Fig. 2.30. This way of thinking about the manifold also gives a nice visual explanation of data augmentation, as it suggests that the data manifold is much larger than the data that is available, and that the network can learn to generalize better by learning the structure of the manifold which can be achieved by incorporating transformations of the input data that should not change the underlying class of the data

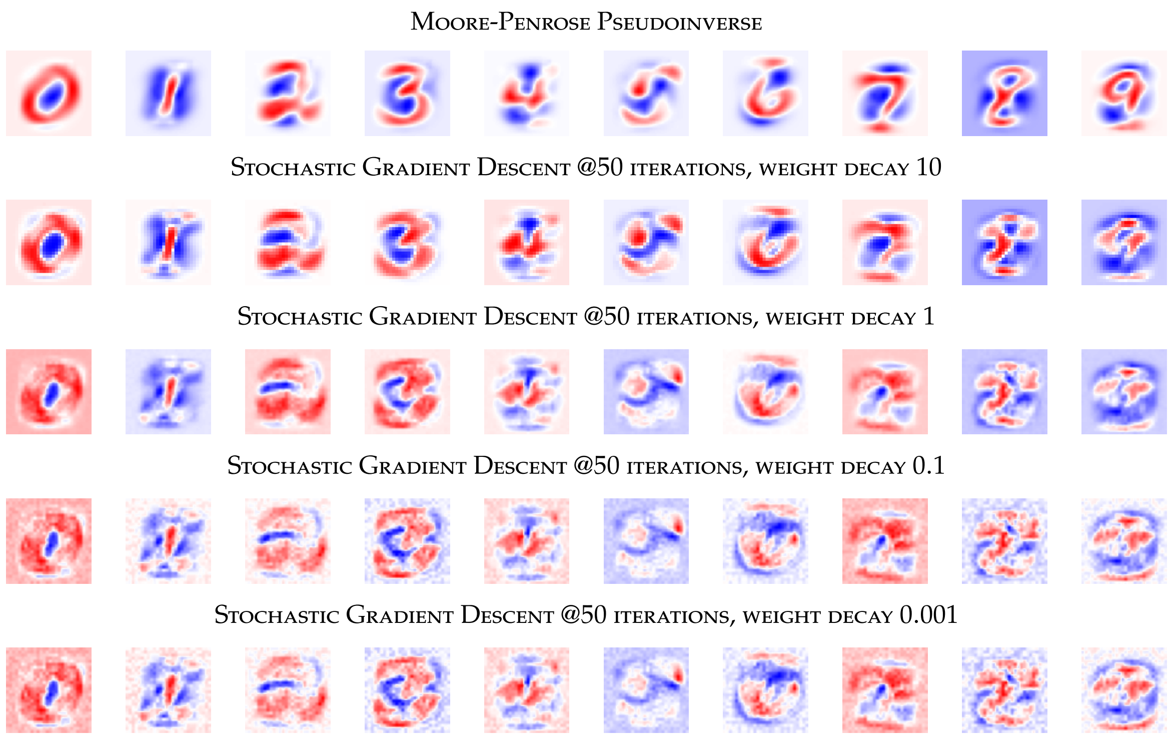


Figure 2.31: Visualization of the weights of a linear classifier on the MNIST dataset. The linear classification problem reduces to $\mathbf{W}\Phi = \mathbf{Y}$, where $\mathbf{W} \in \mathbb{R}^{10\times784}$, $\Phi \in \mathbb{R}^{784\times n}$, $\mathbf{Y} \in \mathbb{R}^{n\times10}$. This means that we can reshape the weights of the linear classifier into the shape of the input data. As the dot product of each row of the input data with the weights is computed ⟨sample, templates⟩, the weights can be interpreted as a template, with a high response if the input image resembles the template.

Activation Maximization

Throughout this introduction we have been able to visualize the weights of the networks directly. In the case of the feed forward network in 1-D and 2-D we were able to visualize the weights as a function of the input space. In the case of the linear model on the MNIST dataset we were able to interpret the learned weights as templates for the individual classes. This direct way to visualize the weights of a network, however, reaches its limits when considering more complex networks with high dimensional inputs such as images. In these cases, the weights of the network are not directly interpretable as a function of the input space, but rather as a function of the activation maps of the previous layer. This makes it difficult to draw direct conclusions from the weights of the network. One way to visualize arbitrary neurons of a learned network is Activation Maximization [60]. It is a technique that uses the gradient information of a neuron in the network to adapt the input image to the network in such a way that this particular neurons activation is maximized. This method is particularly valuable for probing deeper layers of a network, as it results in data that lives in the space of the input. For example, this means that an image is produced by probing individual neurons in a network that is trained for object detection.

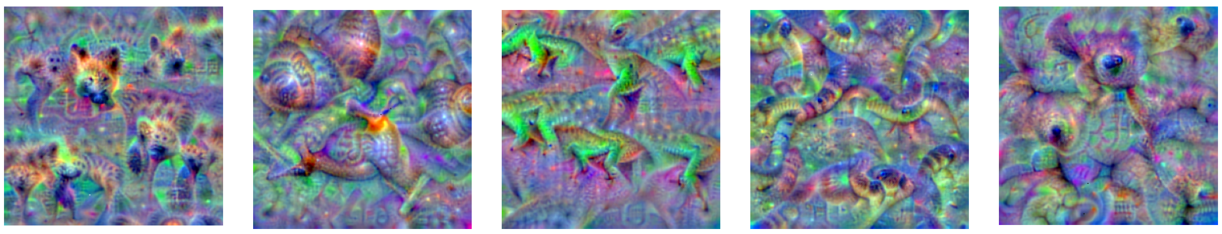

(a) Hyena **(b)** Snail **(c)** Chameleon **(d)** Snake **(e)** Teddy

Figure 2.32: Activation Maximization with data term regularization can be used to generate images that maximize that maximize the activation of a specific class neuron in a pretrained network. The images above are generated with a pretrained VGG16 network on ImageNet. One can clearly see typical patterns such as canine facial features in the hyena image, or the shell of the snail. Note that due to the gradient ascend algorithm and different initial images the generated images can vary for different runs.

Activation Maximization aims to find the input that maximizes the activation of a target neuron or sum of neurons (i.e. a whole layer) in a neural network. This is achieved by iteratively adjusting the input image to increase the activation of the target neuron or layer by using the gradient of the activation with respect to the input image. The optimization process can be regularized to ensure that the generated image is natural and not just a noise pattern that maximizes the activation of the target neuron. In the case for a image classification network this means that we are able to generate images that maximize the activation of a specific class, i.e. look like a *template* of that class, see Fig. 2.32.

A neural network, f, with weights θ, and trained to have c class outputs, maps an input x to an output y in $\mathbb{R}^c$:

$$f(x;\theta) : x \mapsto y \in \mathbb{R}^c$$

where c represents the number of classes. Activation Maximization aims to find an input x that maximizes the activation of a particular neuron. For a neuron corresponding to class e (where e is one of the c classes), this can be formulated as:

$$x^* = \arg\max_x f(x;\theta)_e,$$

and is shown in Fig. 2.32. If the goal is not to maximize the output neuron but rather a collection of neurons in a specific layer l, computing the collective activation involves summing the activations of all neurons in that layer. Let h_i^l represent the activation of the i-th neuron in layer l. The input to these neurons is $f^{l-1}(x;\theta)$†, which is the input x processed by all layers up-to, and excluding l. The objective is to maximize the sum of these activations, denoted by A_l:

$$A_l(f(x;\theta)) = \sum_i h_i^l(f^{l-1}(x;\theta)) \tag{2.20}$$

† Another way to think of this processing is the concatination of all hidden layers up to l, resulting in the concatenated formulation of $A_l(f(x;\theta)) = \sum_i h_i^l((h^{l-1} \circ \cdots \circ h^2 \circ h^1)(x;\theta))$.

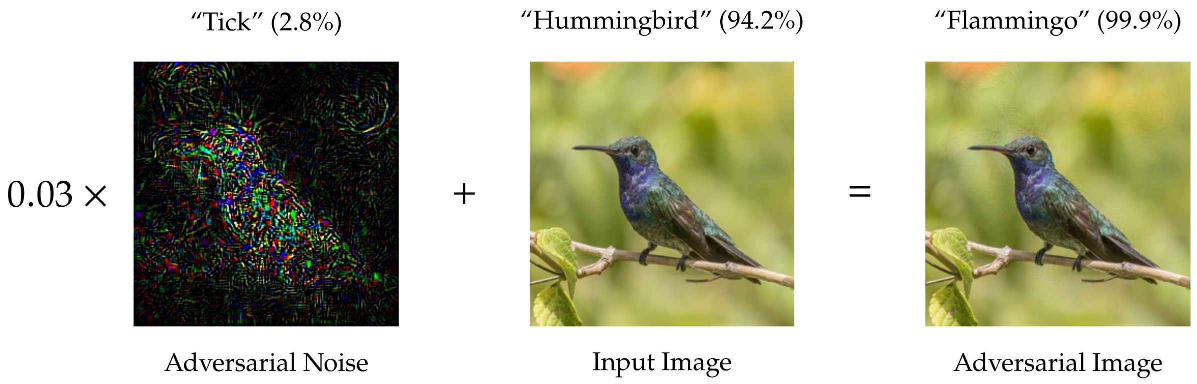


Figure 2.33: Activation Maximization can be applied by providing a real image as the starting point, and maximizing the activation of a logit corresponding to the target class that does not match the actual class of the image. The resulting image is an adversarial example that is misclassified by the model, with an extremely high confidence. The difference between the adversarial image and the input image is the adversarial noise, and is shown with a scaling of ×33 because the noise would otherwise be too small to see. Again note that this is only possible because no regularization on the image is applied during the optimization process, cf. Eq. 2.21

The input that maximizes the activation of layer l can be expressed as:

$$x^* = \arg\max_x A_l(f(x;\theta))$$

To achieve this, x is iteratively adjusted using gradient ascent, where η represents the learning rate:

$$x \leftarrow x + \eta \cdot \nabla_x A_l(f(x;\theta))$$

Here, $\nabla_x A_l$ is the gradient of A_l with respect to x.

To ensure the generated image resembles natural inputs, regularization techniques are applied during the optimization process. These may include TV regularization, jitter, blurring, and value clipping. This can be interpreted as projected gradient descent with a regularization term. This is given by:

$$x \leftarrow x + \eta \cdot \nabla_x A_l(f(x;\theta)) - \lambda \cdot R(x), \tag{2.21}$$

where λ is the regularization strength, and R a particular Regularization strategy such as $||\cdot||_2$. Activation Maximization provides a powerful tool for visualizing and understanding the internal representations of deep neural networks, *because* it allows us to generate patterns in the image domain that show us what type of pattern most activate a neuron. The down side of this approach is that the resulting images can be hard to interpret and in any case they place the ultimate burden of interpretation on the viewer of the images.

Adversarial Examples Adversarial attacks are a class of techniques can be used to generate input images that are misclassified by a neural network with high confidence. Activation Maximization can be used to generate adversarial examples by optimizing the input to maximize the activation of a target class *without* explicitly regularizing the image during the optimization process. This effectively creates images that are misclassified by the network, but are visually indistinguishable from the original input. An example of such an adversarial example is shown in Figure 2.33, where the input image is a picture of a hummingbird, but the network classifies it as a flamingo with a confidence of 99.9%. The adversarial noise is the difference between the adversarial image and the input image.

2.3 Optical Flow

Considering that our world is dynamic and that objects in the world move around, it stands to reason to model that motion in order to understand the world better. Modeling the true 3D motion of objects in the world requires knowledge about the depth of objects, their relative velocities, and the motion of the camera. The task of understanding and calculating the *real* 3D motion from images is called *Scene Flow*. Estimating the *apparent* 2D motion of objects as they appear in images, is a easier task in many ways. This apparent motion of pixels in the image - the *Optical Flow* - is for many tasks a sufficient representation. An example of the optical flow between two consecutive frames is shown in Fig. 2.34. The optical flow is a as a 2D vector field as it describes the displacement of a pixel in one image to its corresponding position in the second image. A more intuitive and better visualization of that flow field is the Middleburry [61] color coding that assigns a color hue to the direction and the saturation to its magnitude. An example of this encoding in either vector form and Middleburry coding is shown in Fig. 2.35.

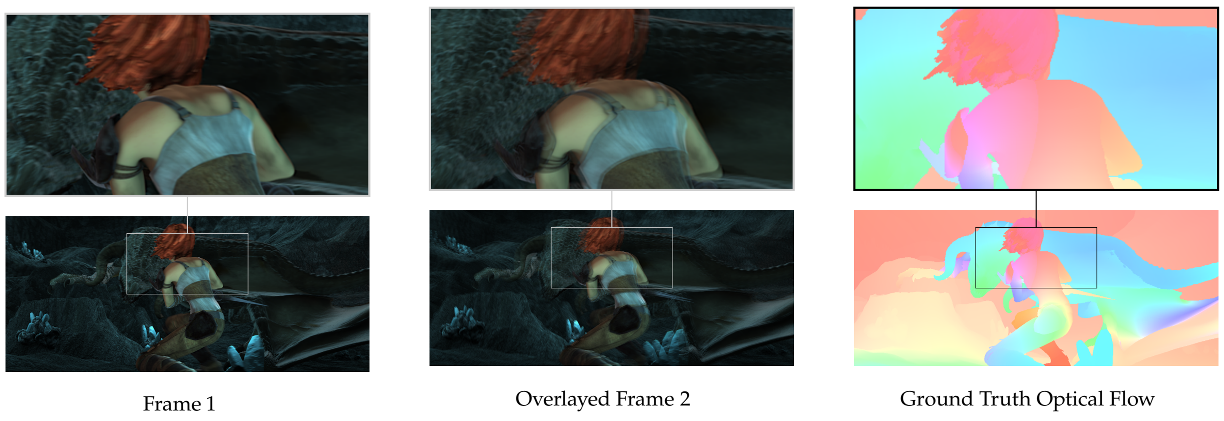
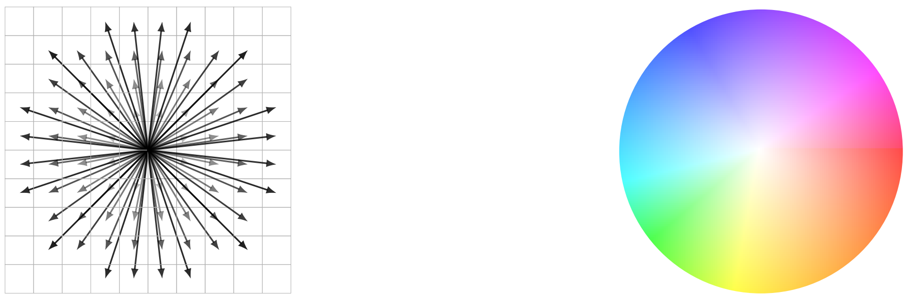


Figure 2.34: Two Frames from a Synthetic Video, and the optical flow between the two frames. Frame 1 is shown on the left, with Frame 2 overlayed on top of Frame 1 in the center. On the right the ground truth optical flow field is shown. The ground truth is available available directly because the used dataset, Sintel [62], is synthetic.

Note that due to the *Aperture Problem* the optical flow for a given image pair is not unique[†], and the same motion can be described many underlying motions. This is illustrated in Fig. 2.36. For action recognition, where relative motion between neighboring pixels is often sufficient, optical flow is a reasonable compromise between computational complexity and the accuracy of motion estimation.



(a) Optical Flow Field as a Grid of Vectors. **(b)** Middlebury Color Coding

Figure 2.35: Optical flow is the apparent motion of objects in the scene. It can be represented as a grid of vectors, where each vector represents the motion of a pixel in the image. The length and direction of the vector indicate the magnitude and direction of the motion which can also be color coded such that the speed corresponds to the saturation of the color and the direction to the hue of the color. Such color coding is known as the Middlebury[61] color coding.

[†] For some image pairs, e.g. noise patterns, it is unique.

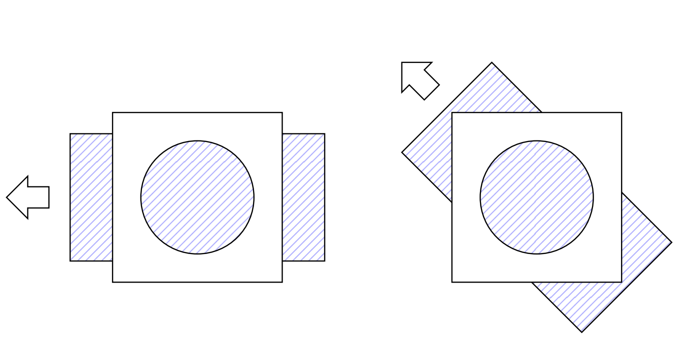


Figure 2.36: Illustration of the Aperture Problem in Optical Flow Estimation. The aperture problem arises when motion detection is attempted through a "aperture" limiting the observer's view to a small portion of a larger scene that might contain context for disimbiguation. In this illustration, only the local motion of a single edge of a moving object is visible through the aperture. Due to this limited view, the direction of motion appears ambiguous: the object could be moving vertically, horizontally, or along any diagonal that projects the same edge motion within the aperture. This problem highlights the difficulty in determining the full, two-dimensional vector of motion from limited one-dimensional local visual information.

Classical Methods

We will briefly discuss methods of computing the optical flow, and more broadly discuss techniques and concepts that are useful even beyond the scope of optical flow.

Variational Methods Classical approaches such as Horn-Schunck [63] and Lucas-Kanade [64] estimate the motion from two consecutive frames of a video sequence using a variational framework. At its core, the problem can be written as an energy cost function with a data fidelity term D, and a regularization term R:

$$E(u) = D(u, I) + \alpha R(u), \tag{2.22}$$

where I refers to the input images, u to the optical flow field, and $\alpha \in \mathbb{R}$ is a weighting parameter. Often the image I is also thought of as an *image function* of spatial coordinates. We already encountered a type of regularizer R in Sec. 3 when we discussed the Total Variation Regularization in the context of adversarial examples produced by activation maximization. The purpose of both terms is the same; to penalize solutions that are quantifiably bad, with respect to the regularization term. In the context of optical flow estimation, the regularization term R can be used to ensure that the estimated optical flow is smooth, and the data fidelity term D ensures that the estimated optical flow is consistent with the input images. The energy function is minimized to estimate the optical flow field. To weight the individual terms, an additional parameter α is introduced which can trade off the accuracy with respect to the data fidelity versus the smoothness of the optical flow estimation.

Horn-Schunk [63] model the data term D with the brightness constancy such that the brightness of a matched pixel in the two frames is the same, i.e.

$$I_1(x) \approx I_2(x + \Delta x). \tag{2.23}$$

Additionally, it is assumed that a single imaging function I is responsible for the brightness of the pixels in both images, I_1 and I_2, and this brightness only depends on the spatial coordinates x and the time t. Using a (first-order) Taylor expansion around the point (x, t) and assuming that the motion is small, the brightness constancy assumption can be expressed as:

$$I(x + \Delta_x, t + \Delta_t) \approx I(x, t) + \frac{\partial I}{\partial x}\Delta_x + \frac{\partial I}{\partial t}\Delta_t. \tag{2.24}$$

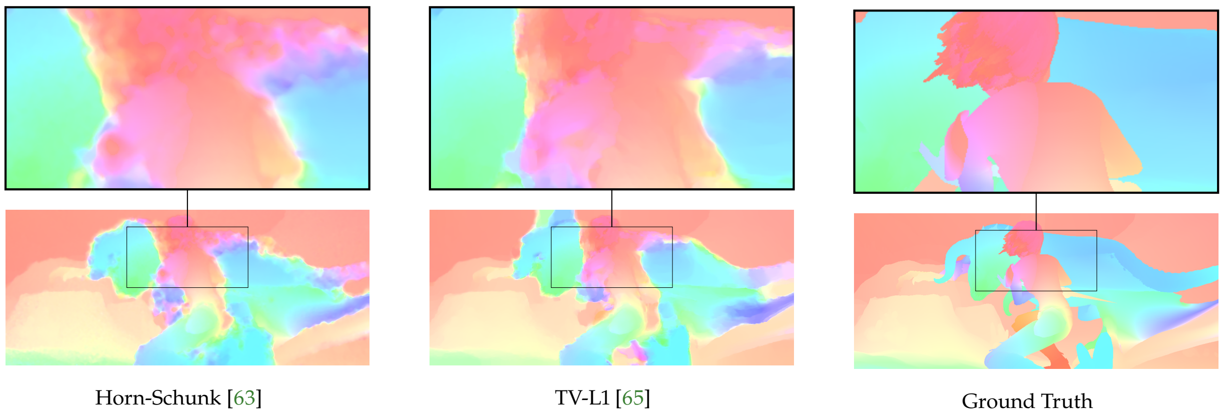


Figure 2.37: Optical Flow estimates of two different variational methods. The difference to the ground truth is most apparent at the object boundaries, due to the penalization of discontinuities.

Defining the optical flow velocity v as a displacement in space Δ_x over a displacement in time Δ_t,

$$v = \frac{\Delta_x}{\Delta_t}, \tag{2.25}$$

and rearranging the terms under the assumption that $I(x + \Delta_x, t + \Delta_t) = I(x, t)$ leads to

$$\begin{aligned} \frac{\partial I}{\partial x}\Delta_x + \frac{\partial I}{\partial t}\Delta_t &= 0 && | \div \Delta_t \\ \frac{\partial I}{\partial x}\frac{\Delta_x}{\Delta_t} + \frac{\partial I}{\partial t} &= 0 && |\text{insert Eq.2.25 for } v \\ \frac{\partial I}{\partial x}v + \frac{\partial I}{\partial t} &= 0. \end{aligned}$$

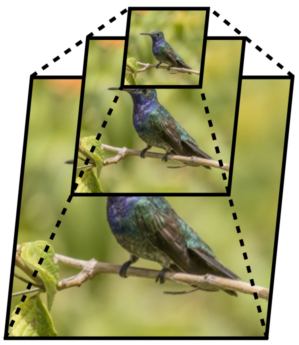
Such approaches however suffer from the problem that the edges along objects in the scene are not well preserved, and the optical flow is often not accurate at object boundaries due to L2 regularization. Methods such as the TV-L1 [65] method have been proposed to address this issue, by using a L1 regularization term instead. In Fig. 2.37 we show the differences of the optical flow estimation between the Horn-Schunck, TV-L1 method.

Feature Pyramids The idea to process the same image at multiple resolutions and combine the features from the different resolutions is a common technique in computer vision. Features from the lower resolution provide a good initial guess on top of which fine-grained details can be added. This idea of using a hierarchical representation to estimate optical flow at various levels is shown in the schematic in Fig. 2.38b. Also note that the ability of coarse-to-fine pyramids can be useful to extend the capture range of optical flow estimation.

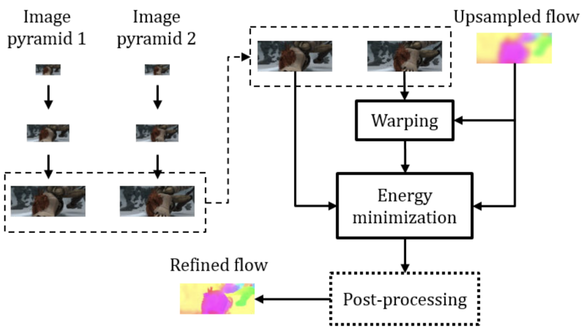
Quality and Warping of Optical Flow The quality of an optical flow estimate can be computed if the ground truth optical flow exists. The error between the ground truth optical flow and the estimated optical flow can be computed using the MSE or the End Point Error (EPE). The commonly used EPE is the average Euclidean distance between the ground truth optical flow and the estimated optical flow.

The direction of computation of the optical flow is often referred to as the forward flow, and the backward flow. The forward flow $F_{1\to 2}$ and the backward flow $F_{2\to 1}$ warping process is visualized in Fig. 2.39.

This can of course only be done if the ground truth optical flow is available, which is often not the case. One method to evaluate the quality of the optical flow field without ground truth is warp image 1 with the computed optical flow to produce image 2. This then allows the computation of an error in pixel-space between the warped image 1 and image 2. This error is often called the photometric error, and is used



(a) Image Pyramid with 3 Levels



(b) Optical Flow Estimation: Coarse-to-Fine

Figure 2.38: Incorporating ideas such as a Image Pyramid (a) into optical flow estimation is useful as it allows the gradual incorporation of fine grained details at higher resolutions without degrading the overall flow by inducing noise. Figure (b) taken from [66].

to evaluate the quality of the optical flow field. Note that warping is also a useful technique for iterative refinements: Calculating the optical flow between two images, warping one image to the other using the optical flow, and repeating the procedure with the just generated warped image to accumulate the flow across iterations for a final solution.

Another interesting application is the use of cycle consistency to evaluate the quality of the optical flow field. This is done by warping image 1 with the optical flow to produce image 2, and then warping image 2 with the inverse optical flow to produce image 1, consider Fig. 2.35 on how to invert the direction from $F_{1\rightarrow 2}$ to $F_{2\rightarrow 1}$. The error between the original image 1 and the warped image 1 is then computed. This error is often called the cycle consistency error.

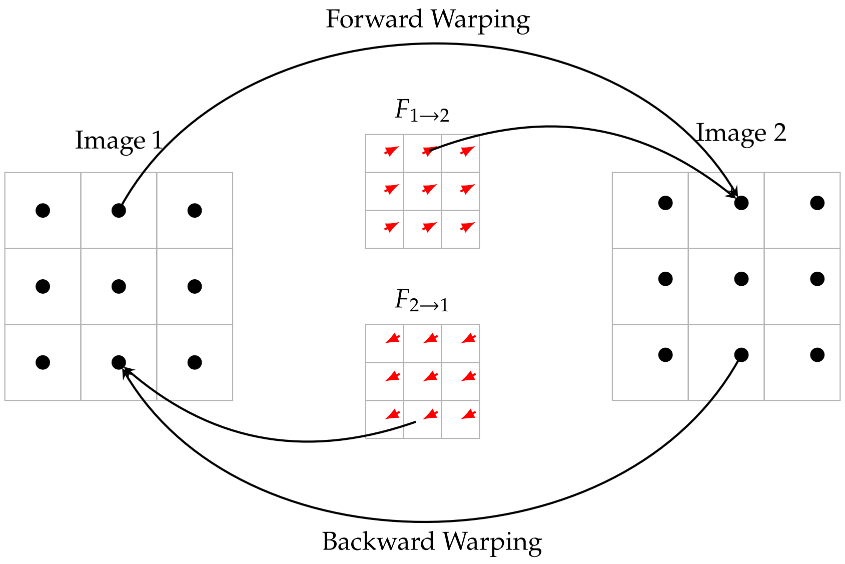


Figure 2.39: Illustration of backward and forward warping. Given the two frames I_1 and I_2, the forward warping process maps the pixels from I_1 to I_2 using the optical flow field. The backward warping process maps the pixels from I_2 to I_1 using the inverse optical flow field. With dense optical flow the resulting forward and backward flow are often not exactly indentical when inverting the backward flow for example. This is due to occlusions and border effects present in the images.

Learning Based Methods for Optical Flow Estimation

The aforementioned variational methods for optical flow estimation relied on explicit assumptions that were modeled through the data term D and the Regularizer R. Other approaches that do not rely on these assumptions have been proposed, and are often referred to as learning-based methods. The learning-based methods learn the optical flow estimation directly from the data, and do not rely on explicit assumptions about the data. In general all methods for optical flow will compute the flow field based on matching features between two frames. The main difference between the methods lies in how the features that are matched are computed, and how the resulting matches are refined. In classical methods the features themselves were often the image values or derivatives of the intensities. In learning based methods features are extracted, often with convolutional networks. These features are then matched in the second image, producing a better initial displacement which is then refined. It is noteworthy that just as in classical methods the optical flow estimation is often computed in a coarse-to-fine manner.

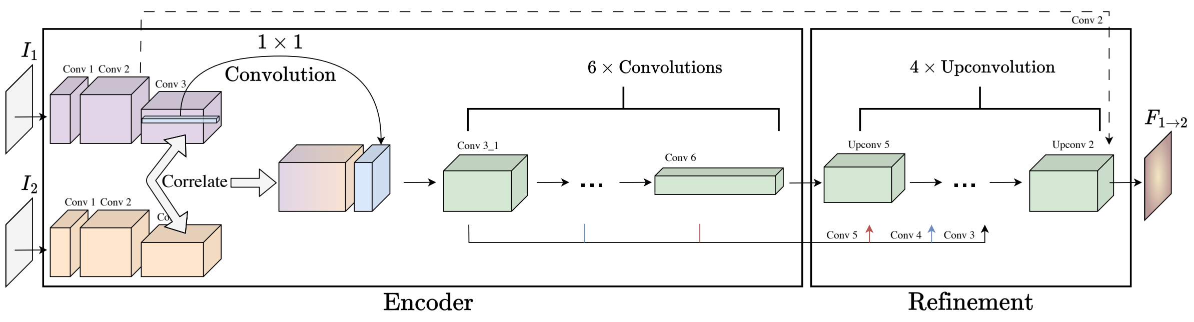


Figure 2.40: The FlowNet architecture for optical flow estimation. It consists of a series of convolutional layers, which are used to extract features from the input frames, and a correlational block that matches them. A series of deconvolution layers in the refinement section are used to upsample the features to the resolution of the optical flow. Additional features from one frame are used in the refinement to give potential edge aware context to the predicted optical flow.

FlowNet The earliest End-to-End architecture for optical flow estimation is FlowNet [67]. FlowNet is based on a convolutional encoder-decoder architecture, and its schematic is illustrated in Fig. 2.40. Convolutional layers are used to extract a feature representation from two consecutive frames which is then explicitly correlated, resulting in a correlation volume. That correlation volume is then continuously refined with more convolutional layers until a bottleneck is reached. A decoder then upsamples the features in the bottleneck and additionally incorporates features of the image before the correlation volume into the upsampling stages. These upsampling stages are implemented with transpose convolutions and increase the resolution of the feature blocks until they match the original input image. In the last layer the channels are compressed to 2 channels such that it can be interpreted as a displacement vector. The EPE between the predicted optical flow and the ground truth is used to provide a signal for training. The FlowNet architecture was the first to show that it is possible to learn the optical flow estimation directly from the data, and it has since been improved upon by many other architectures.

PWC-Net PWC-Net [66] adapted ideas from earlier work such as SpyNet [68] and incorporates a learnable feature pyramid into the optical flow estimation. Furthermore a cost volume is computed from the features extracted from the feature pyramid, and the optical flow is then computed from the cost volume. The PWC-Net architecture is shown in Fig. 2.41. PWC-Net also introduced dedicated warping layers, which were spiritual predecessors to the recurrent refinement architectures that followed.

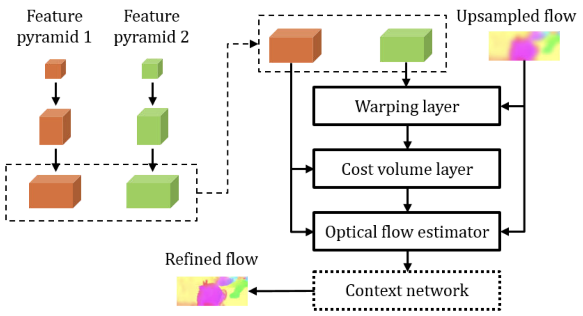


Figure 2.41: PCW-Net adopts the idea of a multi scale feature pyramid that is used to estimate the optical flow. Figure taken from [66], Fig. 3.

Recurrent All-Pairs Field Transforms RAFT [69] drastically improved the performance across many optical flow benchmarks when it appeared. In RAFT, the feature extraction process involves first downsampling the input images and then extracting features with a convolutional neural network. This serves to condense the image data into a more computationally manageable form while emphasizing essential, robust features. The core of RAFT processing power lies in its construction of a 4D correlation volume from these downsampled features. This volume, see Fig. 2.42, which represents the similarity measures across every possible pairing of pixels between the two frames, allows for finding good matches across the feature pyramid from which initial motion vectors can be derived. These initial motion vectors are then iteratively refined with a type of recurrent neural network.

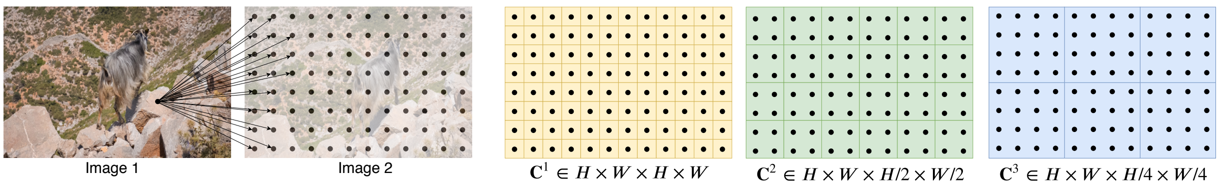


Image 1
Image 2
$\mathbf{C}^1 \in H \times W \times H \times W$
$\mathbf{C}^2 \in H \times W \times H/2 \times W/2$
$\mathbf{C}^3 \in H \times W \times H/4 \times W/4$

Figure 2.42: At the heart of RAFT is the 4D correlation volume at different scales. Computing this volume allows matching a pixel in one image to the other images such that the matches are consistent across the different feature scales. These matched correlation volumes are then passed to the GRU refinement units, as shown in Fig. 2.43. Figure taken from [70], Fig. 2.

Despite the downsampling, RAFT generally captures small displacements through its unique iterative refinement approach, which works on the basis of Gated Recurrent Units (GRU) [71] that also are fed features from a context encoder that uses the first frame as guidance, see Fig. 2.43. The context encoder plays a pivotal role in enhancing the optical flow because it operates on the full resolution of the input images and not the downsampled $\times\frac{1}{4}$ version that is used in the matching volume. The iterative refinement strategy of RAFT uses these Gated Recurrent Units to incrementally refine the flow predictions. Each iteration has access to the context encoder features which allows the model to improve the flow vectors, aligning them along precisely to object boundaries.

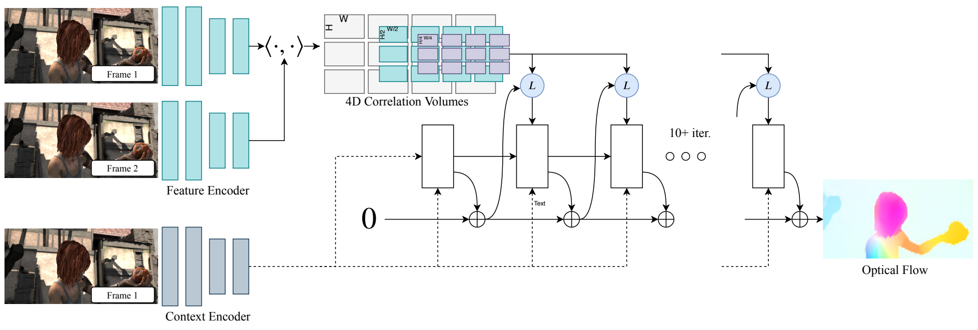


Figure 2.43: The RAFT architecture for optical flow estimation. Figure taken from [70], Fig. 1. Note that the Gated Recurrent Units (GRUs) [71] are a type of neural network architecture designed for sequence prediction tasks. They simplify the traditional LSTM [72] design by combining the forget and input gates into a single "update gate," and by merging the cell state and hidden state. This makes GRUs computationally more efficient and easier to train, without sacrificing much of the capability to handle long-term dependencies in data, which makes them a popular choice to model temporal dynamics.

This all-pairs correlation approach, coupled with the recurrent architectures ability to utilize global information via the context encoder, sets RAFT apart in its capacity to handle dynamic scenes and complex motion patterns accurately. The introduced RAFT architecture has been successfully applied to other tasks where dense correspondence estimation is vital such as stereo-matching, and depth estimation [73, 74].

Summary Concluding with the optical flow section, we want to highlight the immense improvements that have been made in the field of optical flow estimation with the advent of deep learning. To highlight this advance, we compute the optical flow between two frames, showing both classical variational methods, and deep learning based methods side by side, as well as the ground truth flow, see Fig. 2.44. The ability to learn the optical flow estimation directly from the data has allowed for the development of more accurate and robust optical flow estimation methods with RAFT style architectures in particular.

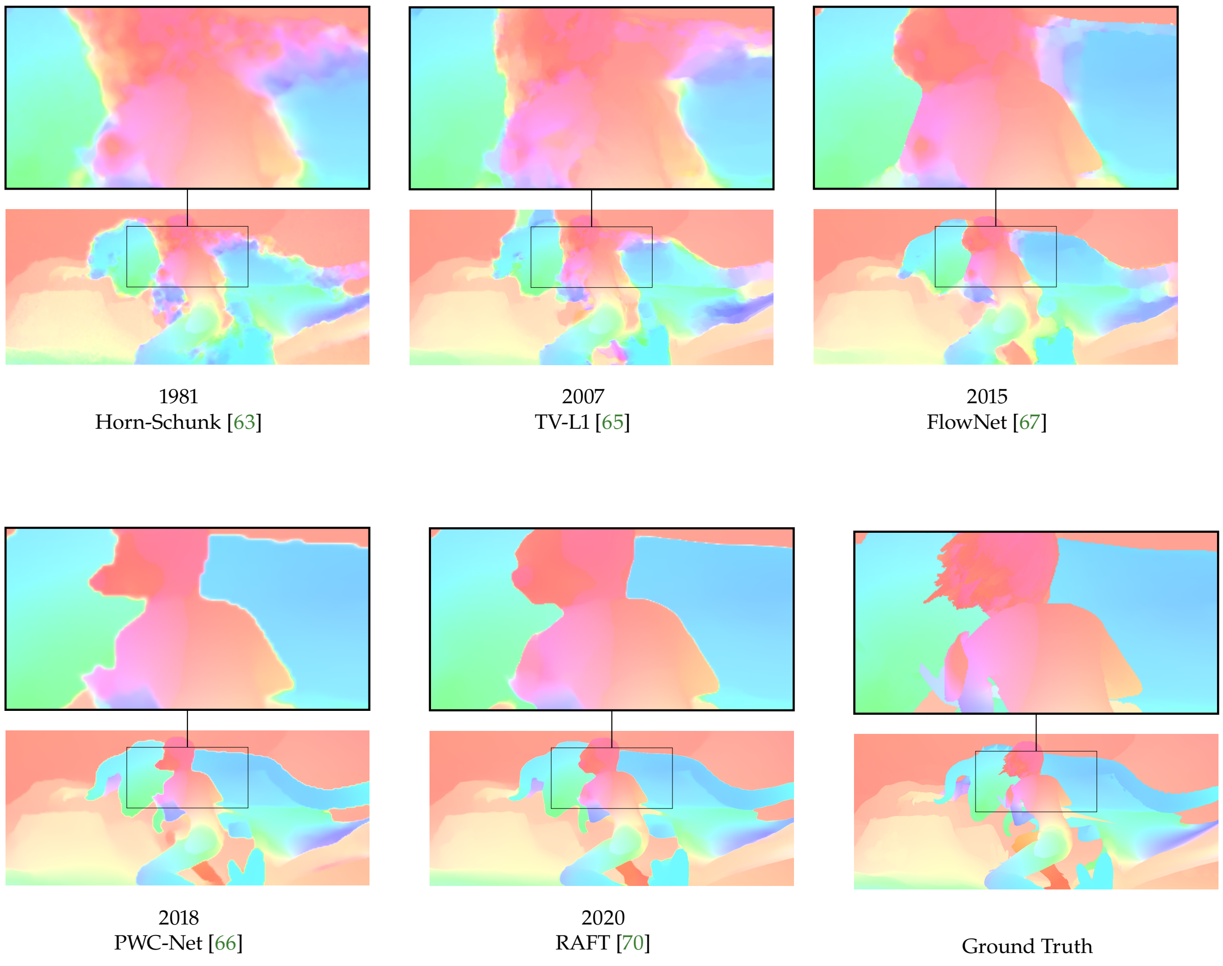


Figure 2.44: The computation of Flow from images is an important topic in visual computing. Shown here are the improvements made from 40 years of research, dating back to 1981. While contemporary methods are generally able to capture rough object boundaries with high accuracy, they still struggle with very fine structures during motions, such as the hair of the person. The ground truth is shown for reference.

Video Understanding

Deep Learning 3

Contents

In the previous chapter we discussed the general principles and methods that are used to solve tasks such as classification from data. We discussed simple cases such as classifying digits, or classifying points based on their x, y coordinates. However for applications that require a deeper understanding such as object detection, semantic segmentation, and video understanding, better models, and a lot more data are required.

The goal of this chapter is to provide a meaningful introduction into the regime of large models and large datasets, and to use the basic principles discussed in the previous chapter to build models that can deal with, and understand image data.

3.1 Deep Learning for Video Understanding

We briefly touched on *shallow* networks which consisted of a few hidden layers at most, and introduced VGG16 in Sec. 3 as a deep network. The term *deep* refers to the depth of the network, which in that specific instance was the number of layers in the network. The term itself is quite arbitrary and different subfields have different interpretations of what constitutes a deep network. In any case, the depth of the network is a crucial factor in the performance of the network, as it allows the network to learn hierarchical representations of the data, which in turn can lead to better performance. Increasing the depth of the network, however, does introduce some problems regarding the training procedures and data handling that need to be addressed. Additionally, the increase in depth also generally increases the number of parameters in the network which leads to a higher computational cost, and a higher memory requirement. We will discuss different architectures of neural networks that have been proven instrumental in image and video tasks, the principles behind them, and the training procedures that are used to train these networks.

As Image Understanding Tasks often predate their video counterparts, many lessons learned from networks that operate on images are transferred to video networks. This chapter will introduce current and historic developments in the state-of-the-art image understanding networks and discuss how they were adapted to video understanding tasks.

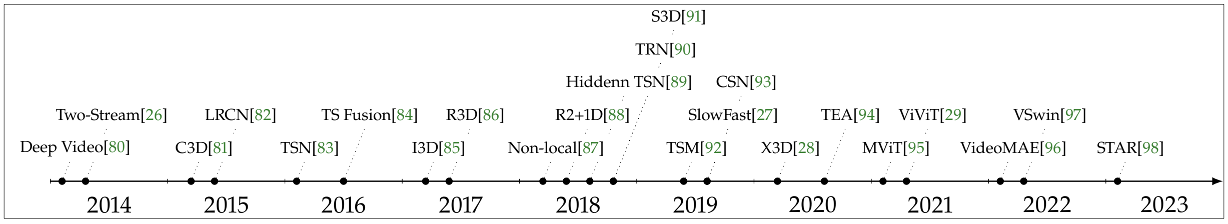


Figure 3.1: Chronological overview of models that introduced novel ideas used in deep learning applications and were successful, enjoying a widespread adaptation and study in the community. Another interesting aspect is the slow transition and evolution Convolutional Neural Networks to Transformer based architectures in the video domain.

3.2 Timeline of Video Action Recognition

The early stages of action recognition followed a similar path to that of Image Understanding where the features that were used for classification were handcrafted and underpinned by sound mathematical formulations and heuristics. An example of such features for Action Recognition are the Improved Dense Trajectories(IDT) [75] which were used in conjunction with Support Vector Machines (SVMs) [76] for classification. Other descriptors such as Space-Time-Interest points (STIP) [77], a natural extension of Harris Corners [78] in 3D, were also used in describing the occurrence of motion in videos, which could then be used for other downstream tasks. These methods, however, were not without their limitations which posed significant hurdles for large-scale deployment. Furthermore, these methods do not necessarily scale for downstream applications as they rely on heuristically or mathematically sound formulations, which had to be tuned according to the task. For a review of pre-deep learning approaches to action recognition see, e.g., [79]. Following the limitations of traditional feature-based techniques, the field of Action Recognition began to pivot towards machine learning models that could learn representations directly from data, particularly with the advent of Deep Learning. The introduction of Convolutional Neural Networks (CNNs) marked a significant leap in this area.

Models

Convolutional Networks Convolutional Networks [99] (CNNs) had their breakthrough moment in the field of computer vision with the introduction of AlexNet [5] in 2012, which won the ImageNet [8] Large Scale Visual Recognition Challenge (ILSVRC) - a object recognition challenge, by a large margin. This model was the first to demonstrate the power of deep learning in image classification. Following this success, CNNs for video action recognition gained traction with the seminal introduction of models like DeepVideo[100] which applied 2D CNNs to individual video frames, exploring temporal connectivity through fusion techniques. Other works such as Two-Stream networks [26, 84, 89], processed appearance and motion information separately. The motion was modeled with optical flow algorithms that were applied on consecutive frames of the whole video. Most of that research was focused on the exact architecture and fusion methodology between the *Motion* and *Appearance* stream. This line of work relied heavily on determining the correct way of fusing the two distinct streams, with connections to biologically inspired models of the human visual system, and ways to visualize the learned features in the network [101].

The research of action recognition relied on an explicit optical flow estimation in these models and used optical flow estimators as a preprocessing stage. Effective action recognition necessitates a nuanced understanding of motion, a dimension that static images or frame-by-frame analysis could not fully capture. By leveraging optical flow, two-stream networks achieved widespread success. The simultaneous introduction of 3D CNNs offered a conceptually parallel track of research, proposing a more integrated approach to model spatio-temporal information within a unified framework with no explicit distinction between the spatial and temporal dimensions. While these models showed promise, their practical deployment was hampered by significant computational demands, leading to architectures that kept the computational cost manageable. This resulted in novel approaches, such as Temporal Shift Modules (TSM) [92], which introduce temporal dynamics into 2D CNNs with minimal computational overhead, and the exploration of internal flow-like representations [27, 88] aiming to bypass the need for pre-computed optical flow, reducing the computational burden associated with two-stream optical flow networks.

Transformers The success of Transformers in Natural Language Processing (NLP) and image understanding tasks has also influenced the architectures and techniques that are researched in video understanding tasks. Transformers leverage their inherent capability to model complex dependencies across time and space, with the added benefit of rudimentary in-built interpretability in the form of attention maps. The central mechanism of self-attention applied on video data offers a flexible and powerful tool to capture the dynamics of video sequences. This approach allows for direct modeling of interactions between any two positions across a video's spatial and temporal dimensions, irrespective of their distance. This would be impossible for traditional convolutional networks that depend on network depth to increase their receptive field. The introduction of the Vision Transformer (ViT) [102] model in 2020 marked a significant milestone in the application of Transformers to image understanding tasks. It has since been adapted to video understanding tasks with models such as the Multi-scale Vision Transformer (MViT) [95], the Video Vision Transformer (ViViT) [29], the TimeSformer [103], and many more [104]. These models have shown promising results in video action recognition tasks, with the potential to outperform traditional CNN-based models in certain scenarios.

Recurrent Architectures Recurrent architectures, particularly Long Short-Term Memory (LSTM) [72] networks, have also been used to some success in video understanding models [105]. LSTMs, with their ability to capture long-term dependencies across time, also offer a promising avenue for modeling the temporal sequences inherent in video data. Early implementations of LSTMs in the context of two-stream networks attempted to aggregate frame-level features over time, aiming to produce a more coherent and comprehensive understanding of video sequences. The primary factor why LSTMs are not widespread is that their training is very brittle and they do not perform as well on transfer learning tasks. Some variants of Recurrent Networks such as GRU [71] have, however, found their niche application in iterative refinement networks, as they are common in optical flow [70].

We will limit our detailed discussion of Deep Learning models for Video Action Recognition mostly to Convolutional Networks and Transformers, as they are the most widely used, and the best performing models in the field. A timeline of significant models in Video Understanding and more specifically Action Recognition is shown in Fig. 3.1.

Datasets

Datasets for Video Understanding have also evolved over time, with the introduction of large-scale video datasets such as Kinetics [106], Sports-1M [100], and UCF-101 [107], which provide a rich source of labeled video data for training and evaluation. We plot a timeline of datasets that have been important in advancing the state-of-the-art in video action recognition in Fig. 3.3. Note that the y-axis, showing the number of videos in the dataset, is displayed in log-scale. The scale of such datasets has only been possible by the widespread adaption of video sharing platforms (mostly YouTube) with scraping of user provided tags and descriptions to create the labels for the videos.

Similarly, the number of action classes has also steadily increased with most action recognition datasets having more than 100 classes, see Fig. 3.2. This is important because backbones trained on such datasets are often used for a wide variety of downstream tasks, as the features learned by the model are more general if the dataset is also multi-faceted. This means that released models that are trained on such large datasets can be used and fine-tuned for other downstream tasks; only the last classification layer needs to be retrained as the features learned by the model are general enough. This trend has been a significant accelerator for the development of deep learning models in the field of video action recognition, and has also been observed in other fields such as image classification and object detection. Samples from some of the mentioned datasets are shown in Fig. 3.4.

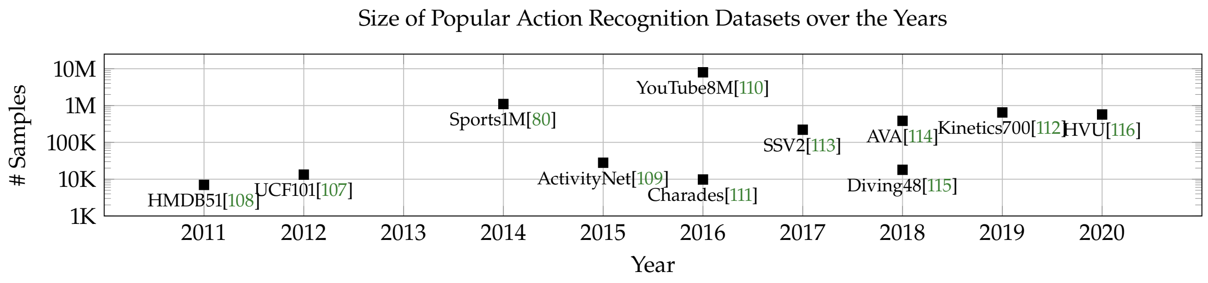


Figure 3.2: Action Recognition datasets needed for deep learning approaches became widespread around 2011. The number of samples in these datasets has been increasing over the years, as is needed for training deep learning models.

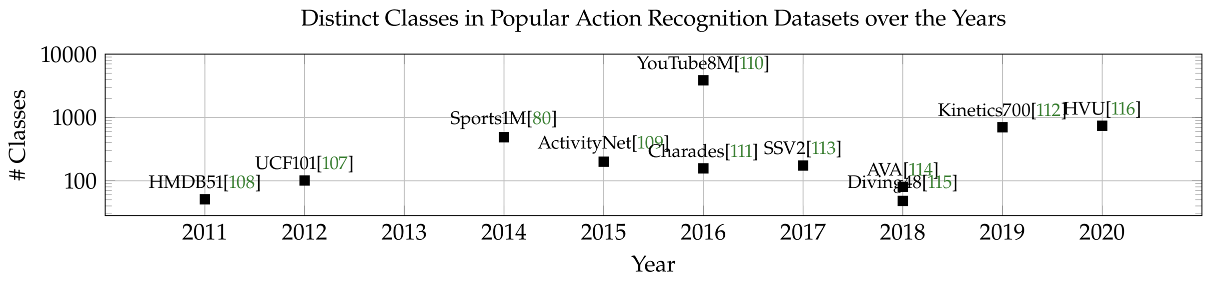


Figure 3.3: At the same time as the number of samples in action recognition datasets has been growing, the number of distinct classes in these datasets has also been increasing. This abundance of data and classes resulted from the abundance of videos freely available on the internet representing a great accelerator for the development of deep learning models.

Video Length The number of frames varies significantly between videos in action recognition datasets, with some videos containing only 10 to 20 frames while others spanning several seconds worth of video material. This variability poses a challenge for models that have to process videos of different lengths, as they must be able to effectively capture temporal dynamics regardless of the video's duration. Common methods to deal with differing video lengths often depend on the network architecture, with some models requiring fixed-length inputs and others being able to process variable-length sequences, and then perform adaptive temporal pooling in the bottleneck layers before the classification head. For models that require fixed-length inputs, videos are often trimmed or padded to a specific number of frames, which can lead to information loss or inefficiencies in processing. In particular, for video sequences where the action itself depends on just a small number of frames, this can lead to a loss of information. An example of this can be seen in Fig. 3.4 for the Diving dataset. If the frames are sampled in a way that do not show the direction of the rotation, which is a crucial part in determining the action label, the model might not be able to correctly classify the action.

Evaluation Metrics For Action Recognition the most commonly used evaluation metric is the Top-1 Accuracy. It measures the percentage of correctly classified videos, where the model's prediction matches the ground truth label. This metric is generally used in classification tasks, and sometimes is augmented with Top-3 and Top-5 Accuracy to provide a more comprehensive evaluation of the model's performance; this type of metric is called the Top-k Accuracy and it measures the percentage of videos where the ground truth label is within the top k predictions of the model.

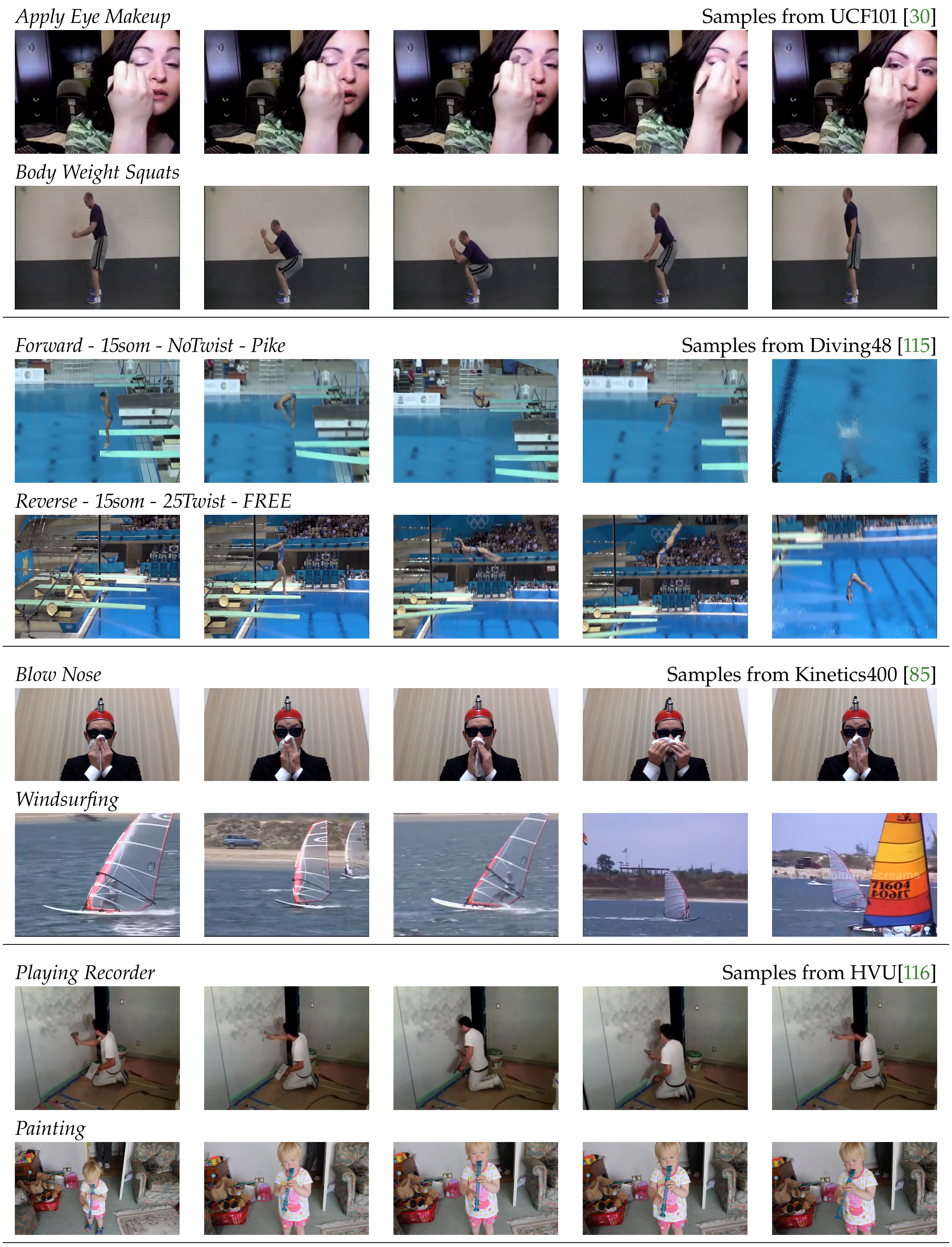


Figure 3.4: Samples from popular Action Recognition Datasets used for Video Understanding. Note that for the sake of brevity only 5 images are shown – this already highlights a hurdle as the length of individual videos is generally not constant and modeling long range dependencies across time is a challenging task. Many sampling strategies for networks that have to deal with arbitrary amounts of frames have been proposed, but the most common strategy is to limit the number of frames, and sample uniformly from the video until the maximum number of frames that the architecture can handle is reached.

3.3 Architectures

We offered a brief overview of the evolution of datasets and high-level ideas that were instrumental in the development of image and video understanding models. This section aims to provide more insight into the development of computational mechanisms and architectural designs, driven by the need to scale up neural networks to handle large-scale image and video data.

Regularization Strategies for Deep Networks

The VGG16 [6] and AlexNet [5] models demonstrated substantial success across various vision tasks, fostering further exploration into deep convolutional network architectures. The main idea behind increasing network depth in both of these networks compared to shallow networks is to reduce the parameter count in individual layers while enabling the model to learn more complex, hierarchical representations.

However, merely adding additional layers to a network does not inherently enhance performance. In fact, deeper networks maintaining the traditional architecture pattern of convolution, pooling, and activation layers often prove challenging to train. The reason for this is that the gradient in deep networks needs to be propagated through many layers on operations which are multiplicative in nature. This leads to the gradient being distributed (according to the chain rule, see Sec. 2.2, Fig. 2.14) across the layers which results in very small values. As a result, meticulous hyper-parameter tuning and learning rate adjustments are critical to optimizing these deeper models.

An elegant way to solve this is to introduce residual connections [7, 117] (or sometimes referred to as *Skip-connection*) between individual stages of convolutional blocks; see Fig. 3.5. In this way, the gradient can be distributed along the residual connections all the way to the early layers of the network, which allows for the training of very deep networks. This is the main reason why the ResNet architecture has been so successful in the field of image recognition, and has been the basis for many other architectures that have been developed since.

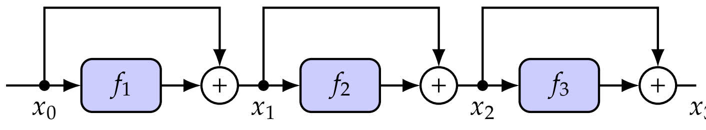
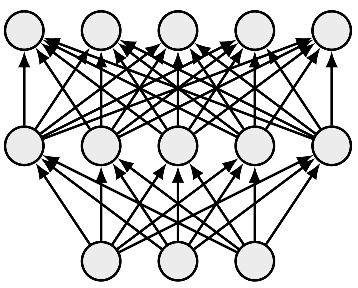


Figure 3.5: Schematic of residual blocks. Each block consists of a function f_i and a skip connection. The skip connection adds the input to the output of the function f_i. This allows f_i to easily distrubute the gradient during backpropagation along the residual path, which is an integral part of training deep neural networks as it effectively combats the vanishing gradient problem, as discussed in Sec. 3.

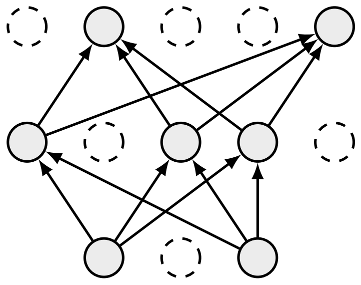
Mathematically, a residual connection in a neural network can be written as

$$\mathbf{x_t} = f_{t-1}(\mathbf{x_{t-1}}) + \mathbf{x_{t-1}}, \tag{3.1}$$

where f_{t-1} is the convolutional block, $\mathbf{x_t}$ is the output of the block, and $\mathbf{x_{t-1}}$ is the input to the block. The residual connection is then added to the output of the block, through which the gradient is backpropagated through the block and the residual connection. The idea here is to learn additive contributions to the transformations, which also allows the network to learn the identity function when needed. As stated, the upside is that the gradient is distributed additively around the block, and therefore more f's can be stacked without leading to vanishing gradients during backpropagation. This type of residual connections are used in all types of architectures as they allow for training of very deep networks without the need for extensive hyper-parameter tuning or different learning rates depending on the layer of the network, by allowing the gradient to flow reliably through the network.



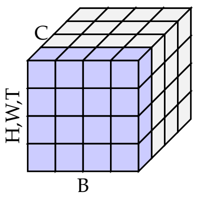
(a) All Neurons Active



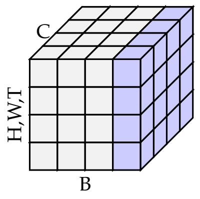
(b) Dropout Regularization

Figure 3.6: During training, a fraction of neurons are randomly set to zero. Which particular neurons are set to zero varies every iteration. This forces the network to learn redundant representations, which have proven to be beneficial for generalization.

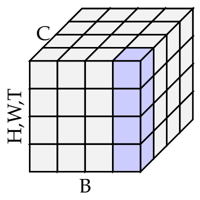
Dropout Dropout, as introduced by Srivastava et. al [118], is a powerful regularization technique used in training deep neural networks. Additionally, work of Gal and Ghahramani [119] established a connection of dropout to a probabilistic view of model uncertainty. In practice however, dropout is used mostly to mitigate the problem of overfitting. The idea is simple yet effective: During the training process, a randomly selected subset of neurons in the network is ignored or *dropped out* at each iteration. This means that their contribution to the activation of downstream neurons is temporarily removed, and their weights are not updated during backpropagation for that particular mini-batch, see Fig. 3.6. The random elimination of a fraction of the neurons forces the network not to rely on any single set of neurons, thus preventing them from co-adapting too closely. Instead, the network develops more robust features that are useful in conjunction with many different random subsets of the other neurons. This technique leads to a network that is less sensitive to the specific weights of neurons, thereby enhancing its ability to generalize well to new data.



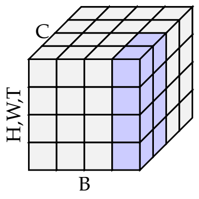
(a) Batch Norm



(b) Layer Norm



(c) Instance Norm



(d) Group Norm

Figure 3.7: Common Normalization Techniques for Deep Neural Networks. The labels on the axis represent the Batch (B), Channel (C), Height (H), Width (W) and Time (T) dimensions of the input tensor. These techniques have proven to be necessary in training deep neural networks.

Normalization Layers Normalization layers also play a important role in stabilizing and accelerating the training process of neural networks. These layers adjust the input data within each batch to have a mean of zero and a standard deviation of one, which helps in reducing internal covariate shift [120] — the phenomenon where the distribution of network activations varies significantly during training, impeding efficient learning.

Each normalization technique is designed to address specific challenges encountered during network training and to improve the efficiency and effectiveness of learning.

- **Batch Normalization** Normalizes the input of a layer for each batch, stabilizing the learning process by reducing the internal covariate shift.
- **Layer Normalization** Instead of normalizing across the batch, it normalizes across each feature map in the data. This is particularly useful in situations where the batch size is small.
- **Instance Normalization** Applies normalization for each training example separately by computing the mean and variance used for normalization from each individual sample. This is often used in style transfer applications.

- **Group Normalization** Divides the channels of the input into groups and normalizes each group separately. This technique is particularly useful when the batch size is small or when the input data is not well suited for batch normalization.

Figure 3.7 provides a visual representation of various normalization strategies applied across different dimensions of input tensors. All normalization techniques share common benefits that contribute to the overall performance of neural networks: By ensuring that the scale of inputs remains similar along a selected dimension, it prevents the model from becoming overly sensitive to the scale of features.

Data Augmentation Data augmentation involves artificially expanding the size and diversity of training datasets by generating transformed versions of existing data. This method significantly enhances the robustness and generalization ability of machine learning models. The primary reason for data augmentation is to prevent overfitting. Deep learning models, particularly those used in image and video classification, possess a vast number of parameters, making them highly susceptible to memorizing the noise and details in the training data, thereby failing to generalize to new data. Data augmentation artificially introduces variability during training, enabling models to learn more generalized features rather than memorizing specific data. This is illustrated in Fig. 3.8.

For image classification tasks, several augmentation techniques are commonly used to simulate the variability that a model might encounter in a real-world scenario:

- **Geometric Transformations** include flipping (horizontally and vertically), rotating, and scaling images. These transformations help a model learn to recognize objects regardless of their orientation and size in different images.
- **Random cropping** of images helps the model focus on different parts of an image, enhancing its ability to recognize objects no matter where they appear in the field of view.
- **Color Alteration** techniques such as adjusting brightness, contrast, saturation, and hue, or adding color jitter, help in training models that are robust to varying lighting conditions and color variations.
- **Adding random noise** to images can make models more robust against different types of sensor noise or compression artifacts in real-world digital images.

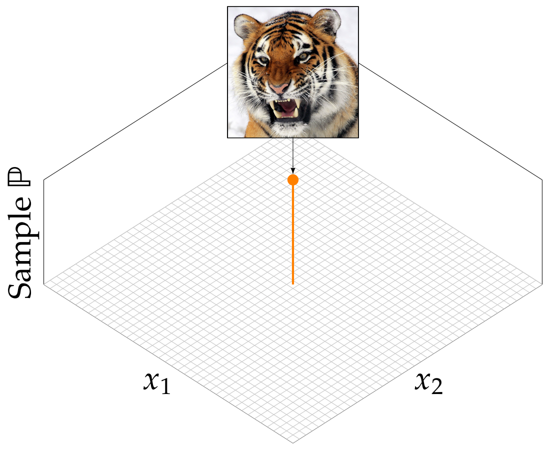
Concerning video action classification, data augmentation extends beyond image-specific techniques due to the temporal nature of video data. In addition to the spatial augmentations applied on a per-frame basis, temporal augmentations are crucial for improving model performance:

- **Temporal Cropping**, similar to image cropping but applied across frames, helps the model learn to recognize actions from different segments of the video.
- **Temporal Scaling** speeds up or slows down the video playback, training the model to recognize actions that occur at different speeds.
- **Temporal Jittering** involves shuffling frames slightly within a video sequence.

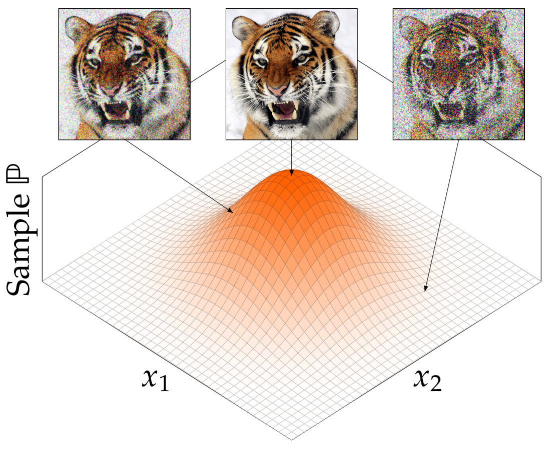
Incorporating these spatial and temporal techniques effectively can significantly enhance the ability of a model to understand and classify complex activities in videos, making it adept at handling the intricacies of real-world video data. While data augmentation is beneficial, it must be applied judiciously. The types of chosen augmentations should reflect realistic variations that are likely to be encountered in the specific application for which the model is being developed. Over-augmenting can lead to a situation where the transformed images no longer represent realistic samples, which could deteriorate model performance on actual data.

Deep Convolution Neural Networks (CNN)

Residual Connections in Convolutional Neural Networks In the context of CNNs, a complication arises when the channel dimensions of the input and output of a convolutional block differ, thereby preventing the direct application of residual connections. To solve this problem, the input is passed through a convolutional layer with a kernel size of 1×1 (without bias) to match the dimensions of the output of the convolutional



(a) Sample in Dataset



(b) Sample with Data Augmentation

Figure 3.8: Data augmentation modifies the data distribution to increase model robustness against noise by allowing the model to fit the data-manifold with specified invariances. The illustration above adds Gaussian noise, denoted as $\mathcal{N}(0,\cdot)$, to each pixel of the input image independently. Each axis in the figure (x_1 and x_2) represents a pixel value from the flattened image, which for a $224 \times 224 \times 3$ image, results in a 150528-dimensional vector. Each point in this high-dimensional space corresponds to a potential image in the dataset. Other augmentation techniques like flipping, rotating, scaling, cropping, and color jittering adjust the data in this space creating distinct "islands" that correspond to the same object class. This figure also illustrates again the Curse of Dimensionality (see Sec. 2.1) and the issues that can arise with large quantities of data.

block. This is called the *identity mapping*; see Fig. 3.9. In scenarios where the dimensions of the input x and the function output $f(x)$ naturally align, the network employs the formulation $out = x + f(x)$. Here, $f(x)$ acts as a residual correction to x, allowing the network to possibly change $f(x)$ toward zero when x already represents the desired output. Conversely, when the dimensions of x and $f(x)$ do not align directly, the network introduces a transformation Wx, where W denotes the weights of the 1×1 convolution used to match the channel dimensions of x to those of $f(x)$. This transformation ensures that Wx remains similar to x, but adjusted for optimal addition to $f(x)$. Throughout training, the model learns to treat $f(x)$ as the appropriate residual addition to Wx instead of directly to x. The use of W is essential when there is a mismatch in channel dimensions. Additionally, convolution operations without padding typically reduce the spatial dimensions of x. To counteract this, padding can be applied to ensure the spatial dimensions of x align with those of $f(x)$. Details on padding were discussed in Sec. 2.

These are the main ideas behind the successful ResNet architectures [7]; we list the most common ResNet architectures in Table 3.1, which differ in the number of convolutional layers. Each variant scales in complexity, adjusting the depth and the number of blocks at each convolutional stage. Each stage typically doubles the number of filters while reducing the spatial dimension by half, maintaining a constant computational cost per layer. The use of 1×1 convolutions, often referred to as bottleneck layers in deeper ResNets (ResNet$_{50}$ and above), serves to reduce the dimensionality before applying more computationally expensive 3x3 convolutions. The incremental complexity is balanced by the use of residual connections, ensuring that even as the network depth increases, training of the network remains stable. This facilitates deeper network architectures without vanishing gradients or degraded training accuracy.

Visualizing Learned Convolutional Filters Visualizing the first layer in a CNN is straightforward, as the kernels are applied directly to the input data. In the case of images, the input data is of a $3 \times h \times w$ shape, where h and w are the height and width of the image, and the other dimension containing the red, green, and blue channels of the image. In the case of ResNet-101 the first layer has 64 kernels, each of shape $3 \times 7 \times 7$ which can be displayed as a 7×7 rgb image, see Fig. 3.10.

Visualizing deeper layers of a convolutional network in terms of the filters that are learned is difficult and does not yield meaningful insights into the learned features. This is due to the fact that the filters themselves do not operate on image data, but on activation maps that are a product of previous convolutional filters. As the depth in a neural network increases, the features extracted become more abstract, integrating simple

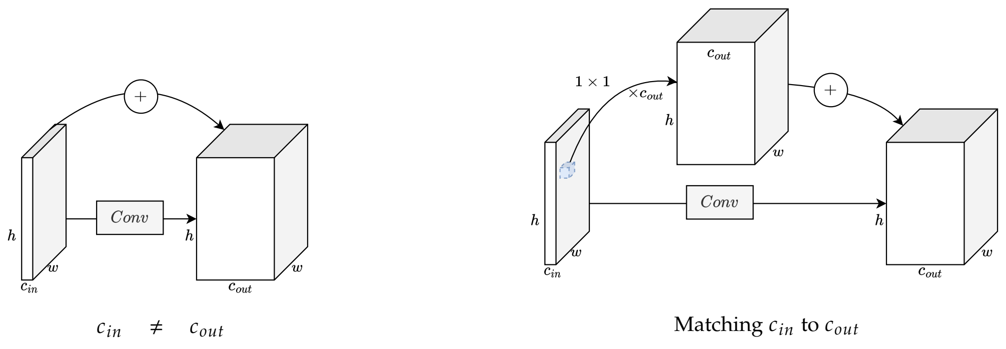


Figure 3.9: The implementation of residual connections in convolutional neural networks requires adaptation as the input and output dimensions before and after the convolutional block may differ. In the case where the number of input channels c_{in} does not match the number of output channels c_{out}, a 1×1 convolutional layer with c_{out} number of filters is applied to the input to match the dimensions of the output. This transformation ensures that the residual connection can be added to the output of the convolutional block. In the 1×1 convolutional blocks the bias is also often omitted and the transformation can therefore be written simply as Wx with W being the weights of the 1×1 convolution. This type of adaptation of x is often referred to in the literature as *identity mapping* [7]. Also note that the spatial dimensions of the input and output are often kept the same by applying zero padding to the input.

Table 3.1: Popular ResNet Architectures used for Image Recognition tasks. Each Layer is seperated into stages. For each stage, the number of blocks and the number of times each block is repeated is shown. The output size of each stage is also shown.

Layer	**Output size**	**ResNet$_{18}$**	**ResNet$_{50}$**	**ResNet$_{101}$**
conv$_1$	112×112	7×7, 64, stride 2 3×3 Max. Pool, stride 2		
conv$_2$	56×56	$\begin{bmatrix} 3\times3, & 64 \\ 3\times3, & 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, & 64 \\ 3\times3, & 64 \\ 1\times1, & 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, & 64 \\ 3\times3, & 64 \\ 1\times1, & 256 \end{bmatrix} \times 3$
conv$_3$	28×28	$\begin{bmatrix} 3\times3, & 128 \\ 3\times3, & 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, & 128 \\ 3\times3, & 128 \\ 1\times1, & 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, & 128 \\ 3\times3, & 128 \\ 1\times1, & 512 \end{bmatrix} \times 4$
conv$_4$	14×14	$\begin{bmatrix} 3\times3, & 256 \\ 3\times3, & 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, & 256 \\ 3\times3, & 256 \\ 1\times1, & 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, & 256 \\ 3\times3, & 256 \\ 1\times1, & 1024 \end{bmatrix} \times 23$
conv$_5$	7×7	$\begin{bmatrix} 3\times3, & 512 \\ 3\times3, & 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1\times1, & 512 \\ 3\times3, & 512 \\ 1\times1, & 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, & 512 \\ 3\times3, & 512 \\ 1\times1, & 2048 \end{bmatrix} \times 3$
	1×1	Avg. Pool ⟶ Number Of Classes: FC ⟶ SoftMax		

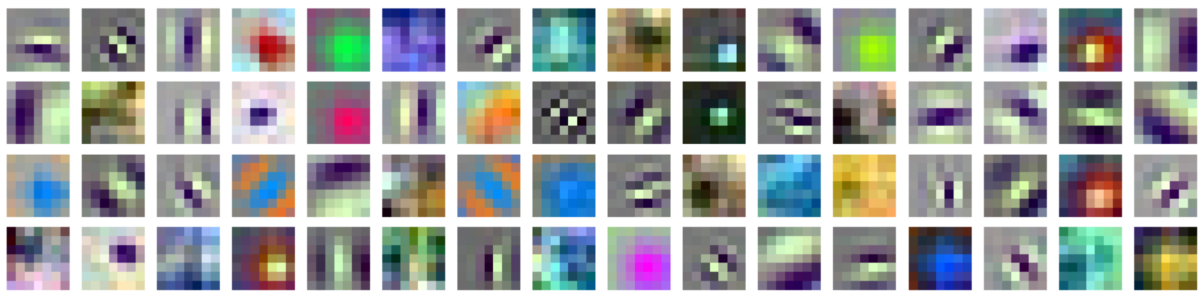


Figure 3.10: Visualization of the First Convolutional Layer Kernels in ResNet-101: This figure showcases a selection of kernels from the first convolutional layer of a pretrained ResNet-101 model, highlighting their diversity and specificity in capturing fundamental visual features such as edges and blobs. These initial filters are directly applied to the raw input images, allowing them to learn basic patterns that are universally present in natural images, and in the following represent the input image in terms of these filters. The clarity and interpretable nature of these filters—often resembling traditional oriented or isotropic bandpass operators are a consequence of their position in the network's hierarchy, operating on the pixel level of the input data. In contrast, kernels in deeper layers of the network process features that are increasingly abstract and distanced from the raw pixel data. As a result, filters beyond the first layer cannot be directly displayed to gain meaningful insights. This reflects the hierarchical feature extraction process inherent in deep convolutional neural networks, where initial layers capture simple, low-level features and deeper layers integrate these into more complex, high-level representations needed for visual tasks.

patterns from earlier layers into more sophisticated representations that might correspond to parts of objects or complex arrangements of features. Activation Maximization, as discussed in Sec. 3 is one notable method that allows us to inspect the response of particular layers in the network indirectly, by generating input images that maximize the activation of a particular layer or neuron within the hierarchy. In Fig. 2.32, we see how Activation Maximization is applied on the output layer (the logits) of a trained neural network to generate images that maximally activate a specific class. This resulted in images that look like the class itself. However it can also be applied on intermediate layers of the network to generate images that maximally activate that specific layer. This can be implemented by summing over the activations of all neurons in a specific layer:

$$A_l(f(x;\theta)) = \sum_i h_i^l(f^{l-1}(x;\theta)) \tag{3.2}$$

where A_l is the activation of layer l, computed by the hidden units i in the full layer h^l. The input to h^l are the previous layers of the network, abbreviated by $f^{l-1}(x;\theta)$, compare with Eq.2.20 in Section 3. The weights, θ, are the parameters of the network which are not updated. Then, gradient ascend can be applied to obtain the updated x, often referred to as x^*. Note that it is necessary to regularize the updated x^* with methods as described in Sec. 3 to prevent degenerate solutions such as adversarial examples. Samples of this process for different layer depths are shown in Fig. 3.11 for the same ResNet-101 model where the first convolutional filters were visualized directly (Fig. 3.10). The generated images show varying degrees of complexity, and show that early layers in the network capture patterns such as orientation of edges and blobs, whereas deeper layers capture more complex and abstract features that are plausibly useful for classification. The output of the last convolutional layer before the fully connected layers looks, indeed, like templates of the objects that the network is trained to recognize. There exist other slightly more advanced methods such as DeepDream [121] and DeepVis [122], which do not operate directly on the image space of x^* but rather a feature representation of it.

Computational Cost The VGG16 model, a significant advancement in deep learning, demonstrates that increasing network depth is an effective strategy to improve performance, particularly in tasks requiring low-level feature extraction like image matching, stitching, and rectification. Although deepening networks can lead to better outcomes, the rise in computational costs does not always correlate with the increase in benefits. This discrepancy has led to the development of various strategies aimed at reducing the computational demands of deep neural networks.

The inception block, tackles the challenge of efficiently capturing image features at varying scales within a single convolutional layer. By integrating multiple filters of different sizes (1×1, 3×3, and 5×5) and a max-pooling operation, this module allows for the extraction and merging of a diverse range of features in parallel, see Fig. 3.12. The parallel structure minimizes additional computational costs while processing spatial hierarchies between features at different scales. The outputs from these paths are concatenated; commonly multiple inception blocks are stacked to form the Inception architecture, which has been used in various models such as GoogLeNet [123] and InceptionV3[124].

Spatio-temporal Convolutional Networks for Video Understanding Many of the ideas that have been developed for image classification have been adapted to video understanding with great success, and have historically been the basis for many of the models in the field. The defining feature of video data is that it has a temporal dimension, which means that the network must be able to capture the temporal information in the video input. In the case of convolutional neural networks, aggregating this temporal information can be done in a number of ways. In Fig. 3.13 different ways of aggregating spatio-temporal information with convolutions is shown; 3D convolutions operate at the same time on the spatial and temporal dimension. Many popular architectures [81, 85, 91] that serve as reliable backbones for feature extraction and action recognition have been built on top of 3D convolutions. However, 3D convolutions are computationally expensive, and do not scale well with the number of frames in the video. One way to reduce the number of parameters required is to split the 3D convolution into a 2D convolution that operates on the individual frames, followed by a 1D convolution that operates on the temporal dimension. This is the idea behind the R(2+1)D network [88] and other approaches that improve upon it.

Two-Stream Networks An alternate approach to capturing the temporal information in videos is to use two separate streams of convolutional networks, one that captures the spatial information and one that captures the temporal information. The two streams are then combined either with continuous fusion throughout the layers, or late in the network, termed *late fusion*. This idea was first explored in depth by Simonyan and Zisserman [26] in the context of action recognition. Many of the two-stream approaches relied on explicit pre-computation of optical flow that was used as input in the motion stream. Some approaches used only a single frame in the appearance stream, and 2D convolutions in the motion stream along the channel dimensions, while other approaches combined the two-stream approach with 3D convolutions as well. Later approaches such as SlowFast networks [27], see Fig. 3.15, also used two streams but did not compute optical flow explicitly before passing it into the motion stream. Instead they use two 3D convolutional streams that capture the input video at different temporal sampling rates and with different capacities and spatio-temporal extents of the convolution kernels to facilitate different representations in the two streams. The schematic of this SlowFast architecture is shown in Fig. 3.15. It is important to note that the temporal stream is sampled at a higher rate than the spatial stream. This allows the network to capture the temporal information at a higher resolution, which is crucial for capturing fast motion in videos. The two streams are then combined in a fusion layer to produce the final output.

Learned Spatio-temporal Patterns Similarly as in the case of the ResNet-101 model, we can apply activation maximization to the I3D network, which is a 3D convolutional network that has been trained on the Sports1M dataset. The results of this are shown in Fig. 3.16. In the early layers of the network, we see simple moving patterns such as scintillating edges, whereas in the deeper layers we see more complex motion patterns such as revolutions, rotations, and diverging patterns. Since the I3D network uses 3D convolutional kernels, the spatial and temporal information is captured at the same time, which makes it difficult to observe only spatial or only temporal patterns by themselves. The I3D network uses a fixed input length of 16 frames, which means that the resulting video that can be displayed also consists of 16 frames. Additionally to the spatial TV regularization that was used in the ResNet-101 model, it is also necessary to regularize the temporal dimension to prevent degenerate solutions. This also highlights one of the downsides of Activation Maximization as the space of solutions is being limited by the choice of regularizer.

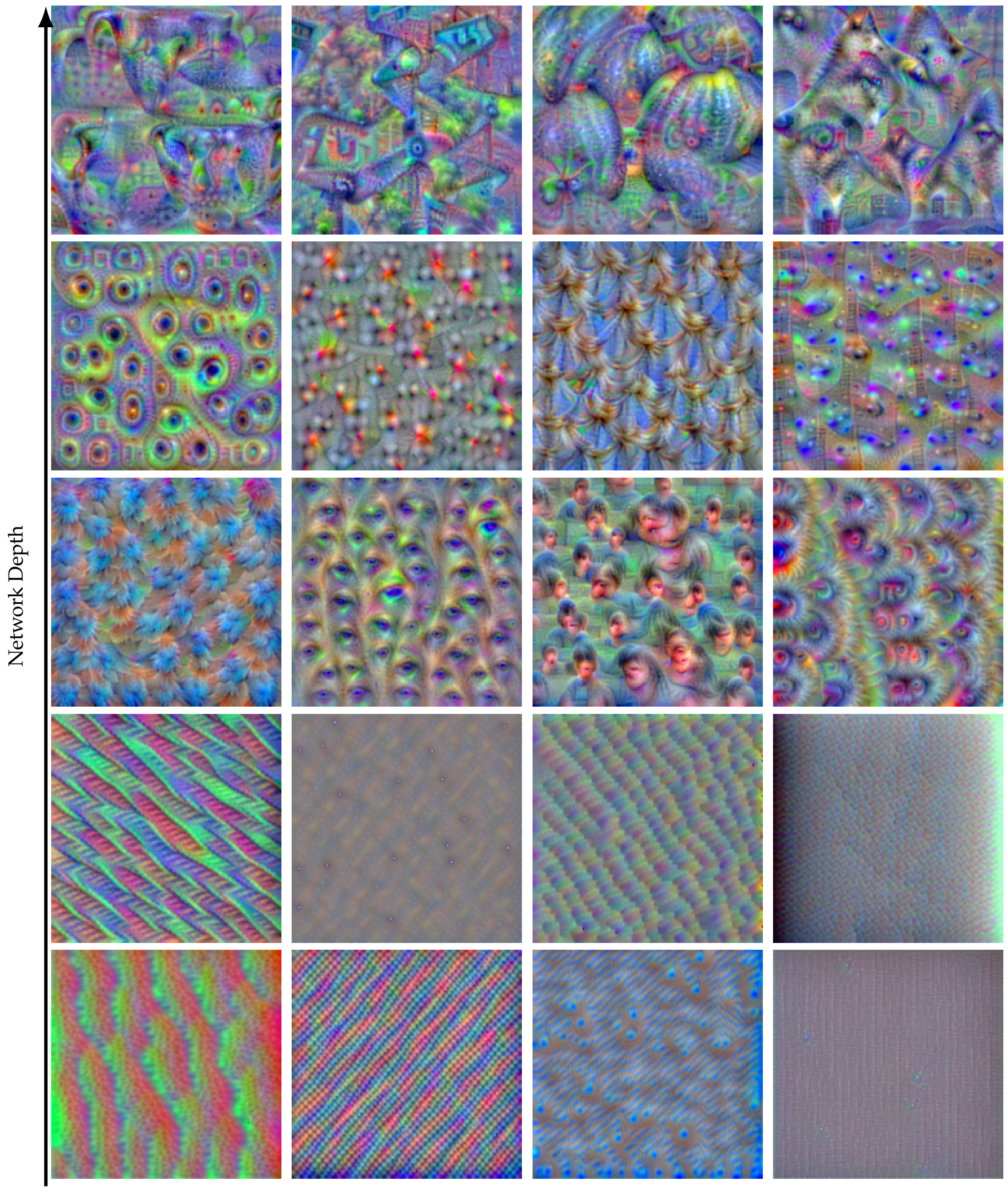


Figure 3.11: Activation Maximization on select filters of a ResNet-101. We visualize 4 filters at 5 different depths, ranging from early to late layers in the network. Note the increasing complexity of the patterns as we move deeper into the network, where the output of the last convolutional layer before the fully connected layers does, indeed, look like templates of the objects the network is trained to recognize. This again harkens back to the idea of the feature extraction serving the purpose of extracting templates which are then used in a *linear* classifier.

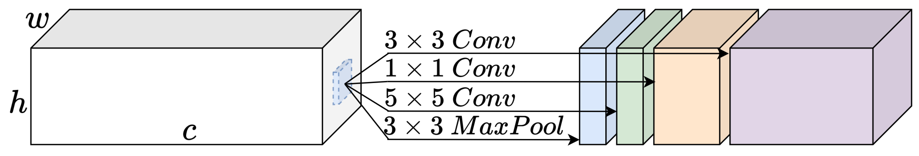
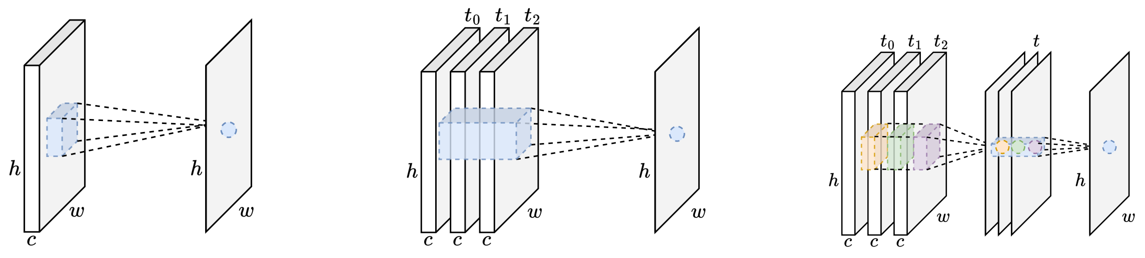


Figure 3.12: Schematic of the Inception architecture [124] that uses Inception blocks. This diagram illustrates the parallel convolutional and pooling paths which are concatenated to form a single output, enhancing the network's ability to handle multi-scale information within an image.



(a) 2D Convolution **(b)** 3D Convolution **(c)** 2D + 1D Convolution

Figure 3.13: Comparative Visualization of Convolutional Techniques: 2D, 3D, and (2D+1D) Convolution. This figure illustrates the core differences between three convolutional strategies used in neural network architectures for processing spatial (2D) and spatio-temporal (3D) data. The 2D convolution operates on two-dimensional input, effectively handling spatial features within images by applying filters across the height and width dimensions. The 3D convolution extends this concept to incorporate the temporal dimension, applying filters across the height, width, and time (or depth), making it suitable for video or volumetric data analysis. However, the 3D convolution comes with significantly higher computational costs due to the increased dimensionality. The (2D+1D) convolution, as a computational optimization, first applies a 2D convolution across the spatial dimensions to capture spatial features, followed by a 1D convolution across the temporal dimension. This sequential approach dramatically reduces computational requirements compared to direct 3D convolution, offering a more efficient way to capture both spatial and temporal information without the full computational burden of traditional 3D convolutions.

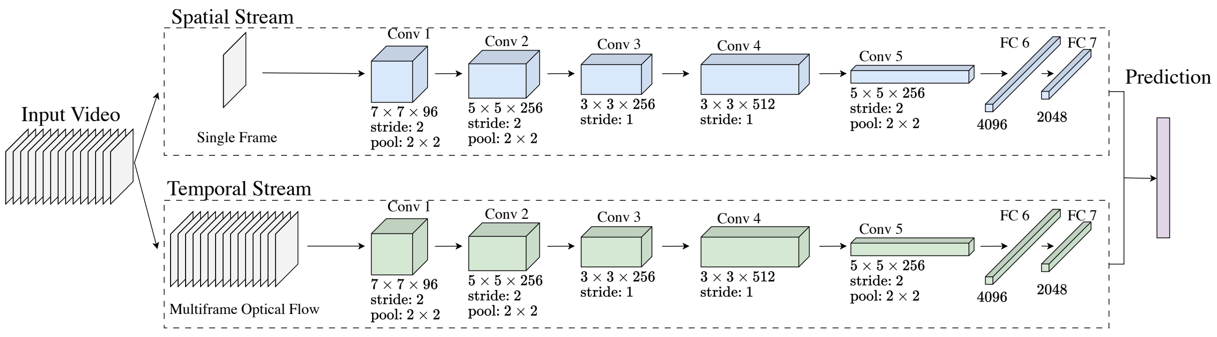


Figure 3.14: The original two-stream network introduced by Simonyan and Zisserman [26] for action recognition. The network consists of two separate streams, one that captures the spatial information and one that captures the temporal information. The spatial stream operates on a single "Anchoring" frame of the video sequence and is a standard 2D convolutional network. The motion stream takes as input the precomputed L optical flows between neighboring frames in the video. The d_x and d_y components of of the optical flow are stacked together to form a $2 \times L$-channel input that is passed through another 2D convolutional network to capture the temporal information. The two streams are processed separately and their SoftMax-ed outputs are combined in a fusion layer to produce the final output. The depicted figure is taken from the original paper Fig. 1 [26].

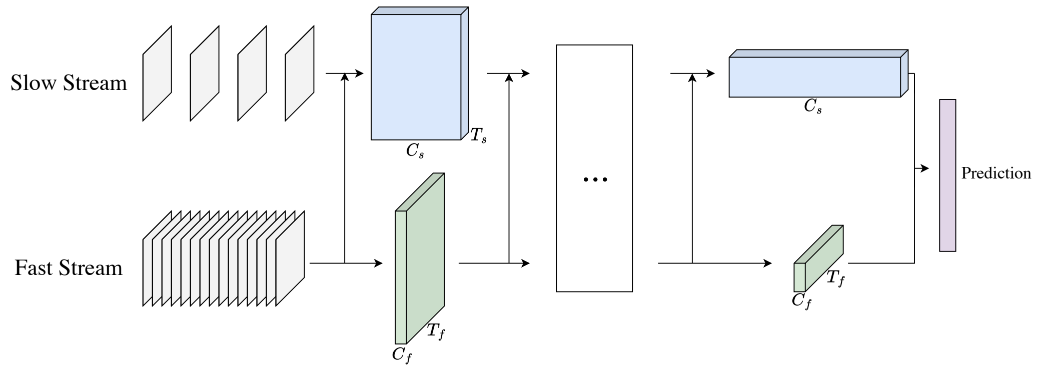


Figure 3.15: This architecture utilizes convolutional neural networks across two distinct streams, each using a different temporal sampling rate. The slow stream is designed to capture spatial information by processing fewer frames, while the fast stream, which has fewer parameters to ensure efficiency, captures temporal dynamics by sampling at a higher rate. This dual-stream approach allows for the efficient processing of both spatial and temporal aspects without the need for explicit optical flow estimation. Additionally, the capacity in each stream is strategically chosen to optimize the representational capabilities and computational efficiency. Figure adapted from [27].

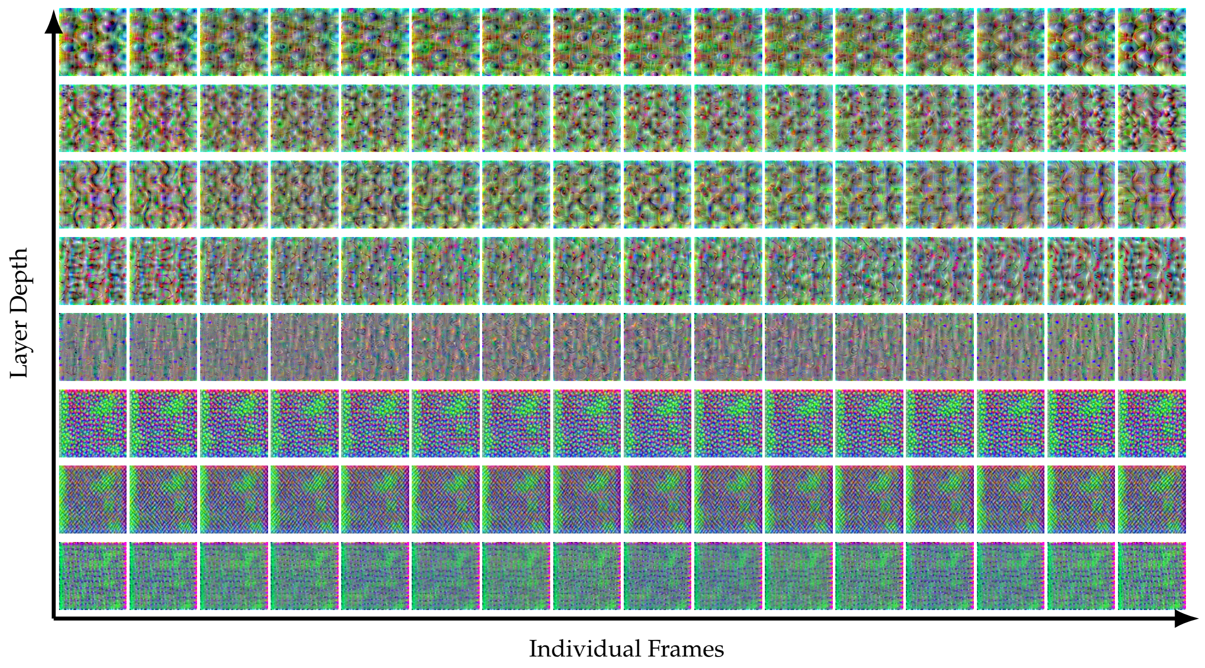


Figure 3.16: Activation maximization applied to I3D network trained on Sports1M. Depending on the target layer in the network - the layers' depth increases along the y-axis, we obtain different video, which we display here as image sequences along the x-axis. In early layers simple moving patterns such as scintillating edges can be seen, whereas deeper layers show more complicated motion patterns such as revolutions, rotations and diverging patterns. Since in the I3D uses 3D convolutional kernels, the spatial and temporal information is captured at the same time, which makes it difficult to observe only spatial or only temporal patterns by themselves.

Transformers

So far, we discussed convolutional neural networks, which are the backbone of many successful models in computer vision. However, another architecture called the transformer has emerged as a powerful alternative to CNNs for a wide range of tasks, including image recognition, natural language processing, and video understanding. First introduced in the context of natural language processing by Vaswani et al. [125], the transformer architecture is based on the idea of intrinsically allowing the model to capture long-range dependencies in the input data within one layer, without having to rely on network depth to increase its receptive field sufficiently, see Fig. 3.17. In this section, we will provide an overview of the transformer architecture, its key components, and discuss how it can be applied to video understanding tasks.

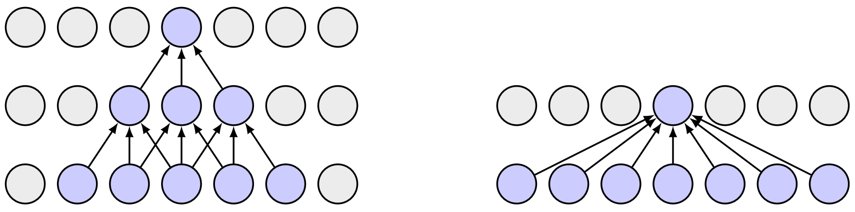

(a) Convolutional Neural Network **(b)** Transfomer

Figure 3.17: The receptive field of a CNN increases with network depth, and while techniques like dilated or strided convolutions can enhance this expansion, a certain level of depth is still essential for the feature representation to encompass the entire input. In contrast, transformers inherently possess a global receptive field, allowing them to access all input data simultaneously from the outset. However, these differences are worth discussing as both approaches have upsides and downsides; CNNs, with their local receptive fields, naturally introduce useful inductive biases, which can be beneficial depending on the task - for details refer back to Sec. 2. This difference in receptive fields also explains why transformers, despite their global view, tend to be more data-intensive, requiring substantial datasets to train effectively as the inductive biases of a comparable CNN need to be learned from data.

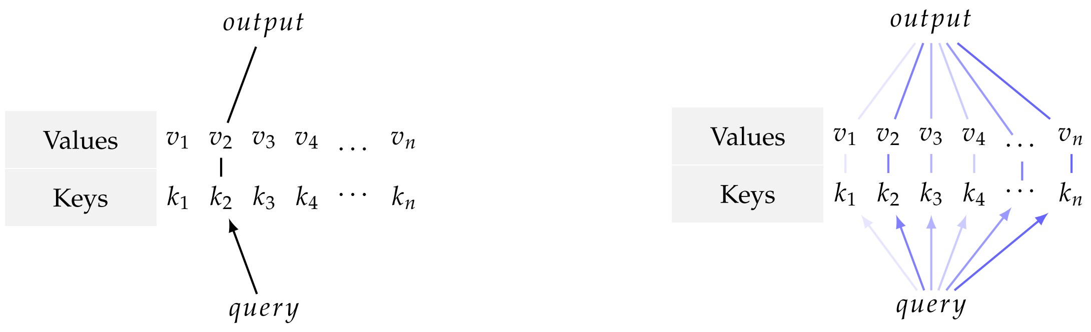
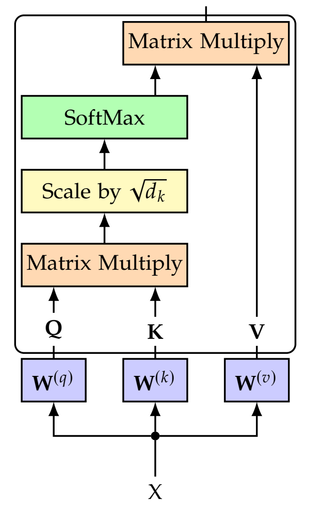

output
Values
Keys
v_1 v_2 v_3 v_4 ... v_n
k_1 k_2 k_3 k_4 ... k_n
query
output
Values
Keys
v_1 v_2 v_3 v_4 ... v_n
k_1 k_2 k_3 k_4 ... k_n
query
(a) Relational Retrieval **(b)** Proportional Retrieval

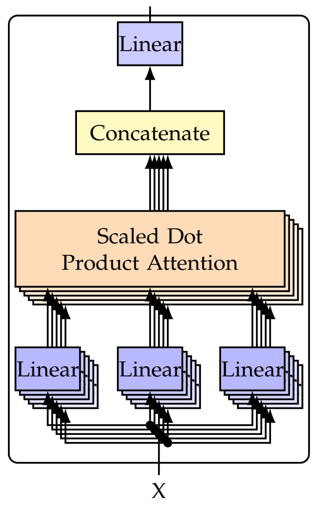
Figure 3.18: A common challenge across disciplines, particularly in computer science, is the retrieval of information from extensive databases. Traditional relational databases, for example, might utilize bag-of-word descriptors to construct query-key pairs, but such queries merely return the value associated with a directly matched key, as illustrated in figure (a). In contrast, a more sophisticated approach might involve *proportional retrieval*, where a mechanism computes the similarity between the query and all keys. The results are then aggregated into a weighted sum of values based on this similarity, as depicted in figure (b). This concept of aggregating and weighting response values forms the foundational principle of the transformer module.

The Attention Module At its core, the attention mechanism in machine learning architectures is a method to dynamically relate different parts of input data to each other.

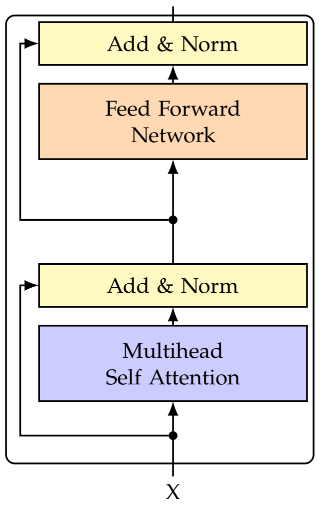
Relational systems rely on exact matches where a query directly retrieves a corresponding value through a key. This is illustrated in Fig. 3.18a and typically uses descriptors like bag-of-words for feature representation, ensuring a direct retrieval of the value connected to the precise key match. Conversely, attention mechanisms allow for proportional retrieval - as shown in Fig. 3.18b. This method calculates a similarity score between each key and the query using a predefined similarity function. These scores are then applied to adjust the weighting of values associated with each key. A normalization step, often through a SoftMax function, ensures these weights sum to one, reflecting the relative importance of each key in relation to the query.



(a) Scaled Dot Product Attention



(b) Multihead Self Attention



(c) Encoder Block

Figure 3.19: Building blocks of the transformer architecture. The attention mechanism (a) is the smallest unit of computation that shows how a single key and query are matched together to allow for proportional retrieval of the $value$ that corresponds to that key. After the "SoftMax" block, it is possible to inspect the attention of the query to all other keys. The multihead attention block (b) is a parallelized version of the attention block, where the input is split into multiple heads and the attention is computed in parallel. The different projections learned allow the different heads to focus on different aspects of the input sequence. The encoder block (c) is finally the combination of the multihead attention block and different mechanisms such as the feed forward network and the residual connections. There is also the decoder block that can be used especially for tasks that require cross attention (i.e., the query is generated from a different input than the key and value). For the tasks that we are interested in, the encoder block is sufficient, as we do not employ multimodal input. Figure inspired by [126], Fig. 12.6.

More concretely, the attention in a transformer is computed in the transformer architecture is as follows: The input is first tokenized and then transformed through linear projections into three vectors: the query, key, and value. In the context of natural language, the tokenization happens through the use of word embeddings which are then passed through (three separate) linear transformation to get the query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$. In the case of images, the image is divided into patches, which are then flattened into a sequence of vectors, that are then passed similarly through linear transformations, again resulting in $\mathbf{Q}, \mathbf{K}, \mathbf{V}$. This highlights one key property of transformers, and their adaptation to various different domains; the type of input data that the model can handle mostly comes down to the tokenization function. The transformer architecture and attention mechanism are general purpose compute structures, agnostic to input data as long as a tokenization mechanism can be used to split the input data into smaller chunks. This is in contrast to convolutional neural networks, which are designed to operate on grid-like data such as images, and recurrent neural networks, which are designed to operate on sequential data such as text.

The general formulation of the attention mechanism for $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ is given by:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{SoftMax}\left(\mathbf{Q}\mathbf{K}^\top\right)\mathbf{V}. \tag{3.3}$$

However in practice the attention mechanism is often scaled by the square root of the dimensionality of the key, which is denoted as d_k:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{SoftMax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \tag{3.4}$$

Scaling is done to prevent the dot product from individual $\mathbf{Q}, \mathbf{K}$ pairs from becoming too large, which would result in one of the terms dominating the SoftMax function, and assigning a weight of 1.0 or close to it to the corresponding value. This would effectively reduce the attention mechanism to a simple lookup mechanism, removing the contextualization that can happen through proportional retrieval, cf. Fig.3.18b. The scaling ensures that the SoftMax function is more evenly distributed across the values, allowing the model to learn more complex relationships between the query and the keys.

Before we continue with additional architectural details of the transformer, we will discuss the attention mechanism more abstractly. By splitting the input data into queries, keys, and values, the attention mechanism allows the model to compare a learned representation of a token (the query) with other learned representations of the entire input data (the keys). The result is a similarity score between the query and each key, which is used to weight the corresponding values. This similarity allows the model to aggregate information that, in turn, was also built on the context of the entire input. An example of such contextualization can be seen in the canonical example from Natural Language processing; "Bank of the river" vs. "Bank account". The word "Bank" has different meanings depending on the context of the sentence, and the attention mechanism allows the model to learn these relationships by comparing the query "Bank" with the keys "of the river" and "account". The attention mechanism then weights the value "of the river" higher when the query is "Bank" in the context of the sentence "Bank of the river", and weights the value "account" higher when the query is "Bank" in the context of the sentence "Bank account". A diagram of the attention mechanism, more specifically the scaled dot product attention, as discussed in Eq. 3.4, is shown in Fig. 3.19a. This dot product self attention block serves as a building block for parts of the transformer architecture, and can be written as

$$\begin{aligned}\mathbf{Q} &= \mathbf{X}\mathbf{W}^{(q)}\\ \mathbf{K} &= \mathbf{X}\mathbf{W}^{(k)}\\ \mathbf{V} &= \mathbf{X}\mathbf{W}^{(v)},\end{aligned} \tag{3.5}$$

where $\mathbf{X}$ is the input token, and $\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)}$ are the linear transformation weights for the query, key, and value respectively.

Multi-headed attention is a straightforward extension of the scaled dot product attention. Attention is computed in parallel blocks where the input tokens are transformed by different weight matrices $\mathbf{W}_i^{(q)}, \mathbf{W}_i^{(k)}, \mathbf{W}_i^{(v)}$ to form the $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ in the i^{th} attention head. The resulting $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are concatenated to form the output of the multi-headed attention block, see Fig. 3.19b. In convolutional networks this is conceptually similar to increasing the number of filters in a convolutional layer, which allows the network to learn different features in parallel.

These multi-headed attention blocks are combined with skip-connections (Section 3.3) and layer normalization blocks to form *Encoder Blocks*, Fig. 3.19c. As the input and output dimensions of an Encoder Block are identical, they can be stacked directly on top of each other to form the encoder part of the transformer architecture. This modular design is quite flexible and allows for the easy addition of more encoder blocks to increase the depth of the network.

The Vision Transformer The Vision Transformer (ViT) is a Transformer that has been adapted for image recognition tasks. The key idea behind the ViT is to treat an image as a sequence of patches, which are then flattened and passed through the transformer, see Fig. 3.20. This is in contrast to convolutional neural networks, which operate directly on the pixel data of the image. The ViT architecture consists of an encoder part, which is made up of a stack of encoder blocks, and a classification head that is attached to the output of the encoder. The encoder blocks are made up of multi-headed attention blocks, normalization blocks, and feedforward blocks.

This means that the queries, keys and values are computed from each of the tokens in the sequence, and the attention mechanism is applied to each token in parallel. The attention to one Query can then be visualized as it is the dot product of the Query with all the Keys. When the Query and the Key are created from the same token, we speak of self-attention; however if the Query and the Key are created from different tokens, the term of cross-attention is used. A schematic of the computation for both self- and cross-attention is shown in Fig. 3.21. We will focus on self-attention mechanisms as we will use transformer architectures to extract features and to classify images and videos. Cross-attention is typically used in tasks where the input data is of different modalities, such as in the case of image captioning, where the image and the text are passed through the transformer, and the model must learn to attend to the relevant parts of the image to generate the caption. The blocks of the transformer that use cross-attention are generally referred to as *Decoder Blocks*.

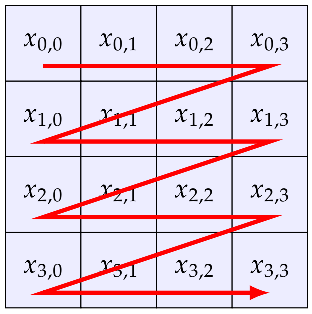


Figure 3.20: The ordering of the tokens before they are passed to the transformer. In the case of the common $\mathrm{ViT}_{16\times16}$, the image is divided in patches where each patch has the dimensions of 16×16 pixels. Commonly, the input images are rescaled to a size of 224×224 which gives a total of $14 \times 14 = 196$ tokens. The tokens are then arranged, row by row, and reshaped into from a 16×16 patch to a 256×1 vector, before being passed into the linear projection layers that feed into the transformer. Figure adapted from [126], Fig. 12.23.

Positional Embeddings Another interesting aspect that can be seen from the schematic, Fig. 3.21, is that every token is treated the same, independently of its position in the sequence. However, we do know that information about the relative or absolute positions of patches in the image are vital for understanding the spatial relationships and objects within an image. To allow the transformer to distinguish between the position of each token in the image, positional encodings are added to each token. The positional encodings are either fixed, or learned during training, and are added to the input tokens before they are passed through the transformer. An example of how the positional encoding is constructed is shown in Fig. 3.22. In the case of fixed positional encodings, sine and cosine functions with fixed frequencies as basis are used. In particular, the positional encoding for a position t and dimension i is defined as:

$$PE_{(t,i)} = \begin{cases} \sin(\omega_k \cdot t) & \text{if } i \text{ even} \\ \cos(\omega_k \cdot t) & \text{if } i \text{ odd} \end{cases}. \tag{3.6}$$

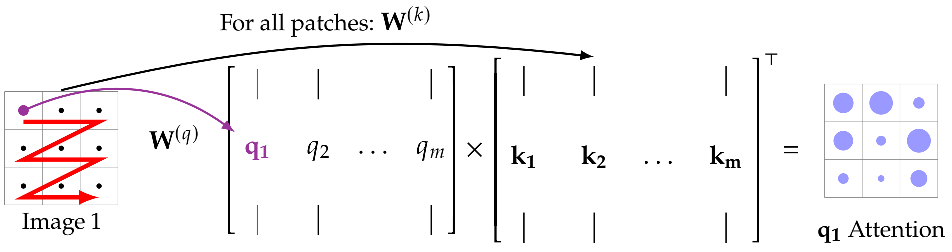
The purpose of using this specific formulation is to ensure that the encoding varies smoothly and uniquely for each position t, providing the model with a way to discern the relative or absolute positioning of tokens in a sequence.

In the case of the vanilla Vision Transformer (ViT) [127] the images are rescaled to a size of 224×224 pixels, and split into 16×16 sized patches, which results in $14 \times 14 = 196$ tokens. This means that the positional encoding must be able to distinguish between $14 \times 14 = 196$ positions. Furthermore, an embedding dimensionality of 768 per token is used. As the flattened image patches have a dimensionality of $16 \times 16 = 256$ and the embeddings have a dimensionality of 768 the resulting total token size is $256 + 768 = 1024$ which is then passed through the transformer.

It is also possible to have non-fixed, learnable positional embeddings instead of the fixed positional embeddings. In the following paragraph we will discuss some methods of inspecting the inner workings of the vision transformer which will also include differences between fixed and learnable positional embeddings.

Classification with the Vision Transformer We have now gathered all the required ingredients to build a vision transformer, and can now discuss how a vision transformer can be used for image classification. The input image is first split into patches, which are then flattened and passed through the transformer. In addition to the patch embeddings, an additional token called `[CLS]` (short for `classification`), which contains the zero vector, is added to the input tokens. After the preparation of the tokens, the input tokens are passed through the transformer, which consists of a stack of encoder blocks.

The output of the transformer at the position of the `[CLS]` token is passed through a classification head, which is a linear layer that maps the output of the transformer to the number of classes that the model should classify the image into, identical to earliest classifiers discussed in Sec. 2.2. As the tokens aggregate more information by contextualization through self-attention, the representation of each token and the similarity to other tokens in the image changes. Because the `[CLS]` token does not belong to any particular patch in the


For all patches: $\mathbf{W}^{(k)}$
Image 1
$\mathbf{W}^{(q)}$
$\begin{bmatrix} \mathbf{q_1} & q_2 & \dots & q_m \end{bmatrix} \times \begin{bmatrix} \mathbf{k_1} & \mathbf{k_2} & \dots & \mathbf{k_m} \end{bmatrix}^\top =$
$\mathbf{q_1}$ Attention

(a) Self Attention

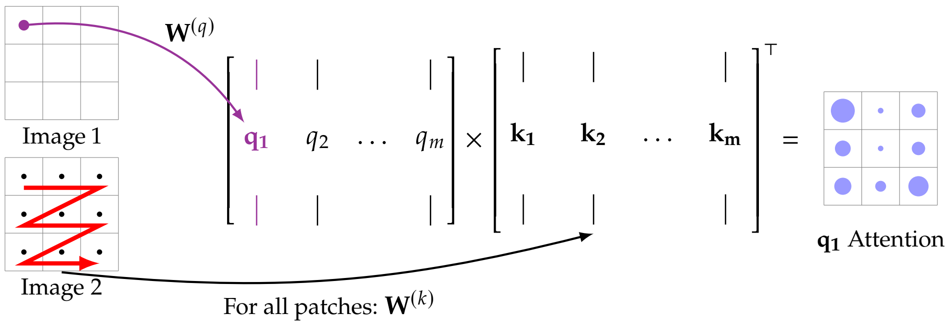
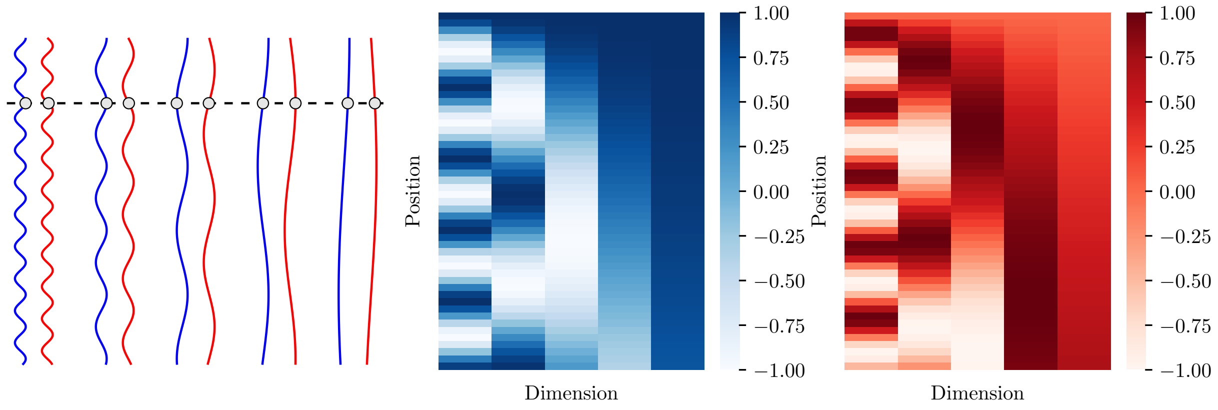


Image 1
Image 2
$\mathbf{W}^{(q)}$
$\begin{bmatrix} \mathbf{q_1} & q_2 & \dots & q_m \end{bmatrix} \times \begin{bmatrix} \mathbf{k_1} & \mathbf{k_2} & \dots & \mathbf{k_m} \end{bmatrix}^\top =$
$\mathbf{q_1}$ Attention
For all patches: $\mathbf{W}^{(k)}$

(b) Cross Attention

Figure 3.21: The attention computed of the first query q_1 with respect to all keys $k_1, k_2, \dots, k_m$. As each $q_1 \times k_i$ dot product gives a scalar, the result can be reshaped back again to the (tokenized) image dimensions. The size of the dots in the right image is proportional to the dot product value, and shows how much the query q_1 *attends* to each key k_i. In the context of Vision Transformers the visualized query is often the [CLS] token, and the keys are the image patches. Some applications for cross attention include semantic segmentation, depth estimation. It is noteworthy that it is not necessary to have the same number of queries and keys, as the attention mechanism can be applied to any pair of sequences, nor that they have to come from images.



(a) Schematic **(b)** Cos part **(c)** Sin part

Figure 3.22: Positional embedding is an important aspect of the transformer architecture. The overall schematic of how to create the sin and cos embeddings is shown in (a). A fixed number of sin and cos with increasing frequency are used to determine the position of the token in the sequence. Figures (b) and (c) show the sin and cos embeddings for a sequence of length 128 respectively. Often times however the positional embedding is learned as well.

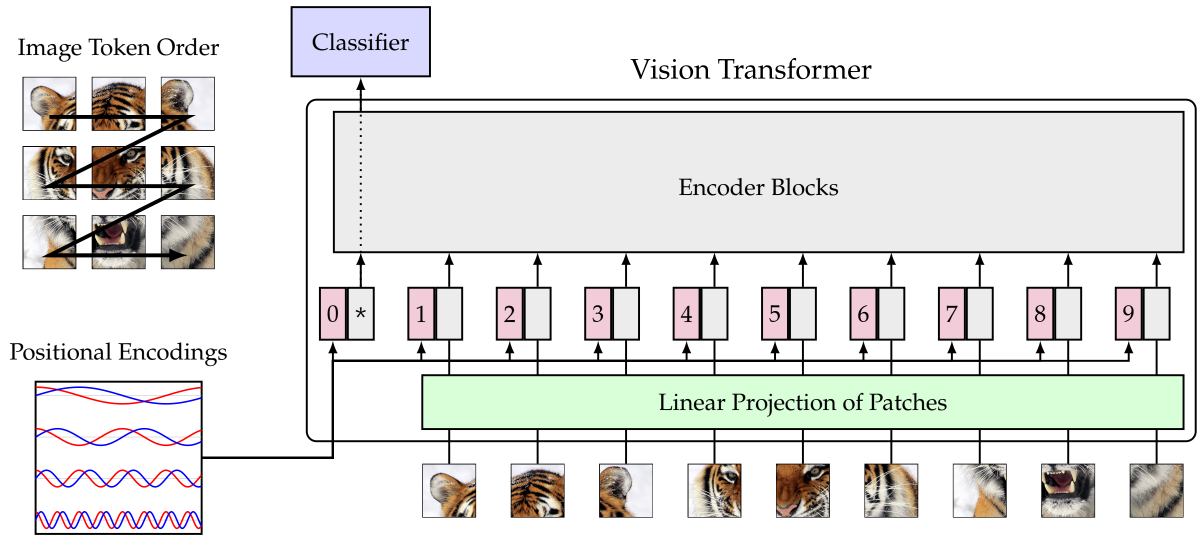


Figure 3.23: Vision Transformer (ViT) architecture. The vision transformer is a natural extension of the transformer to the domain of images. The input image is split into patches (16x16) which and then flattened into a sequence of vectors; an additional positional encoding is added to each token as the transformer does not have any inductive bias for the spatial structure of the input. An important aspect for image classification is the addition of another token, often called `[CLS]` or $*$ token. Initialized with a learnable embedding, and initialized with a fixed value, this token undergoes the same transformations in each encoder block as the other tokens generated from the image itself. The attention mechanism is therefore also applied on the `[CLS]` token. For tasks of image classification it is the output of the `[CLS]` token that is used for the final classification. This leads to global contextual information important for the classification of the image to be stored in the `[CLS]` token, and by extension, the attention maps that lead to the `[CLS]` token.

input image, and is used for classification, information regarding the class of the image is aggregated in it. This leads to the effect that the attention of the `[CLS]` token coincides with image regions that are important for the classification task. The full architecture of the vision transformer is shown in Fig. 3.23.

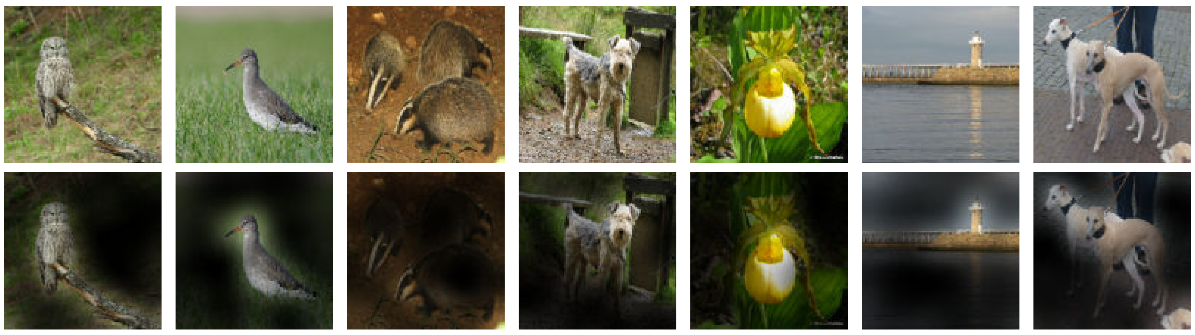


Figure 3.24: Attention of the Vision Transformer with respect to the `[CLS]` token. The attention of the `[CLS]` in the last encoder block is visualized as a heatmap, and overlayed on top of the input image. The attention of the `[CLS]` token can be used to understand which parts of the input image are important for the classification task. Figure adapted from [127], Appendix D, Fig. 14.

Visualizing Aspects of a Vision Transformers In the case of CNNs we visualized the first convolutional kernel weights directly, and those in deeper layers of the network indirectly through Activation Maximization. In the case of the vision transformer such workarounds are not always necessary to gain insight into what is useful for the final classification. The `[CLS]` token, which is appended to the input sequence of image tokens, is of particular interest in that case. As previously discussed, it is this `[CLS]` that accumulates important information about the image, because it is used in the final classification step. The attention of the `[CLS]` token can be used to understand which parts of the input image are important for the classification task. Figure 3.24 shows the attention of the `[CLS]` token in the last encoder block visualized as a heatmap, which is overlayed on top of the input image. Note that the attention map has a lower spatial resolution than the input image, as the attention is computed over the tokens (patches of 16×16 pixels), and not the pixels of the image. However it is commonplace to upsample the attention map to match the size of the input image so that they can be overlayed.

Average Attention Distance It is also of interest to investigate what the average attention distance for tokens inside of a transformer is. Results about the average attention distance can be used to understand how the attention is spread across the tokens in the input sequence. The average attention distance is computed by passing a large number of images through the transformer, and then computing the attention distance for each token to all other tokens in a particular head, for a particular layer. We define the attention distance as the weighted average of the attention weights of all token to all other tokens. We also choose to display the average attention distance with respect to a particular encoder block (depth) and average over all heads in the transformer, as is shown in Fig. 3.25. The x-axis of our visualization shows the depth of the encoder block, and the y-axis the mean attention distance. The heatmap shows that the attention distance is not uniform across the depth of the transformer, and that the attention is spread across the tokens in the early layers, and becomes more focused in the later layers. This is in contrast to convolutional neural networks, where the receptive field of a neuron is fixed and does not change across the network depth. In other terms, deeper layers in the network become more selective in the tokens they attend to. In early layers the mean attention distance is quite large, and is spread across the tokens, meaning that local and global information is being aggregated. In later layers the attention distance is not spread, and becomes larger, showing that later encoder blocks are quite picky with respect to which tokens they focus on. This is in stark contrast to convolutional neural networks, where the receptive field of a neuron is fixed and does not change across the network depth.

We can also visualize the embedding layer through which the image patches are linearly projected, see Fig. 3.26. To better understand the linear projection layer in terms of its influence on image patches we can use the Singular Value Decomposition (SVD) to decompose the weight matrix of the projection layer. The SVD decomposes the weight matrix into three matrices, such that

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$$

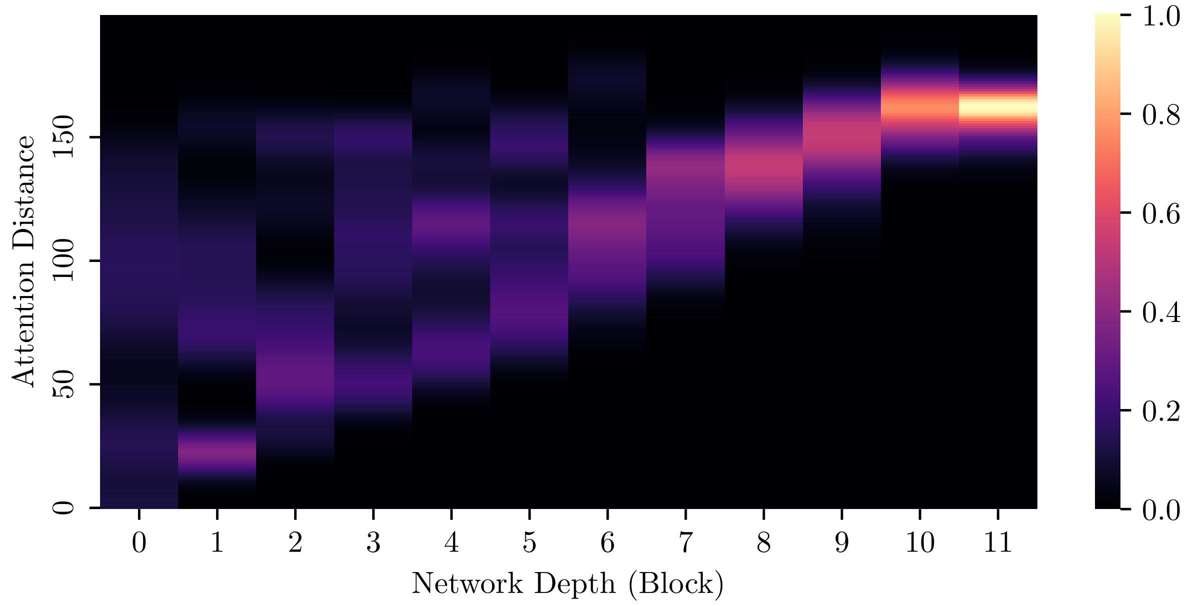


Figure 3.25: The attention distance across the block-depth of a vision transformer [127] is not uniform. We evaluate it by passing 1000 test images from ImageNet [128] through the vision transformer and then calculate the attention distance for all tokens to all other tokens in a particular head, for a particular layer. We do not calculate the attention distance for the `[CLS]` token, as it is a special token that is used for classification and not for the image itself. We can see that the attention distance is not uniform across the block-depth; in early layers the attention is spread to cover local and global patches of the image, whereas later layers are cherry picking the most important tokens to focus on.

where **X** is the weight matrix of the projection, **U** represents the left singular vectors, Σ contains the singular values, and **V** the right singular vectors. Arranging the weights of the projection matrix **X** into a matrix, we can then decompose the weight matrix into three matrices, such that

$$\mathbf{X} = \underbrace{\begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix}}_{m\times n} = \underbrace{\begin{bmatrix} | & | & & | \\ u_1 & u_2 & \dots & u_m \\ | & | & & | \end{bmatrix}}_{m\times m} \times \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n \\ 0 & 0 & 0 \\ & \vdots & \end{bmatrix}}_{m\times n} \times \underbrace{\begin{bmatrix} — & v_1^\top & — \\ & \vdots & \\ — & v_n^\top & — \end{bmatrix}}_{n\times n} = \mathbf{U\Sigma V}^\top. \tag{3.7}$$

In the case of the vision transformer which receives input patches of dimension $16 \times 16 \times 3$ pixels (768×1 when flattened), the linear projection layer transforms this into a 384×1 vector. This means that the weights of the linear projection can be written as $\mathbf{X} \in \mathbb{R}^{m\times n}$ with $m = 768, n = 384$. The SVD decomposition of the weight matrix **X** allows us to look at the singular values Σ, which give us an idea of the importance of each singular vector in the decomposition. The singular values are sorted in descending order, and the first few singular values are typically much larger than the rest capturing most of the variation in the data. This, in combination with the left singular vectors **U**, allows us to look at the filters that are applied to the image patches. Each projection is therefore a linear combination according to the weights in **X** that is applied to the image patch. This has the added benefit that we can visualize the filters that are applied to the image patches in image domain, as a single basis vector in **U** has the dimensionality 768×1 and can be reshaped into the image patch dimensions of $16 \times 16 \times 3$. This gives us insight into what the model has learned and what it deems to be appropriate basis functions to transform the individual patches into. The corresponding example of the first 64 filters in **U** that are applied to the image patches are shown in Fig. 3.26. The filters that are applied to the image patches capture different aspects of the input image, such as edges, textures, and colors. This is similar to the first convolutional layer of a CNN, which also captures low-level features in the input data. But very interestingly, comparing Fig. 3.26 to Fig. 3.10 we can see that the filters in the vision

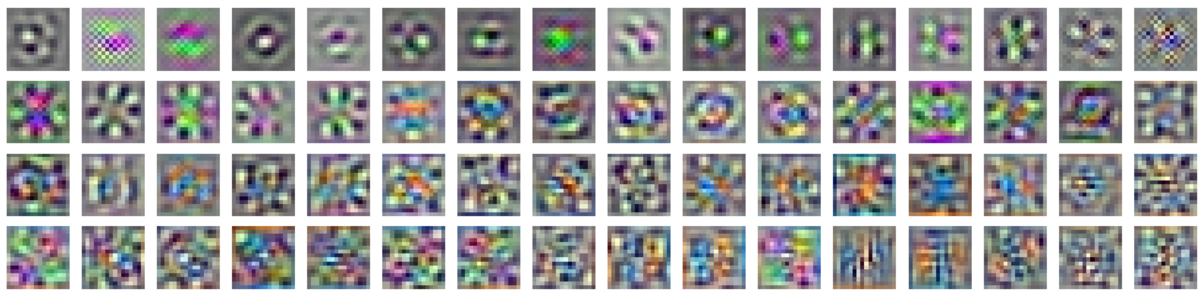
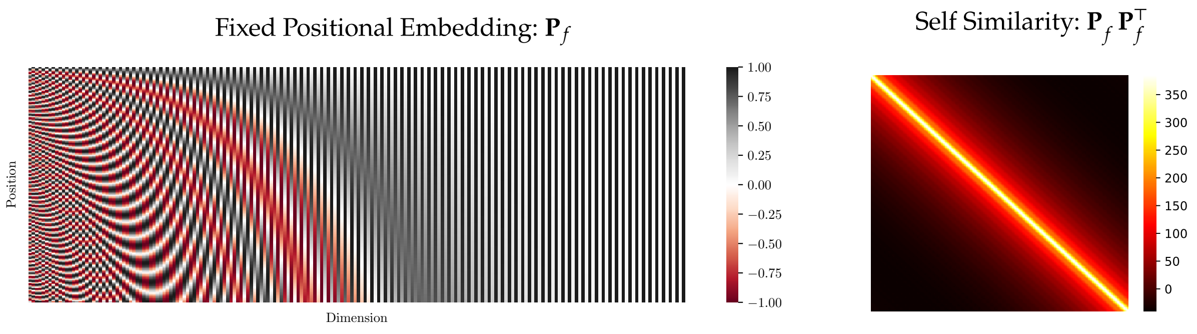


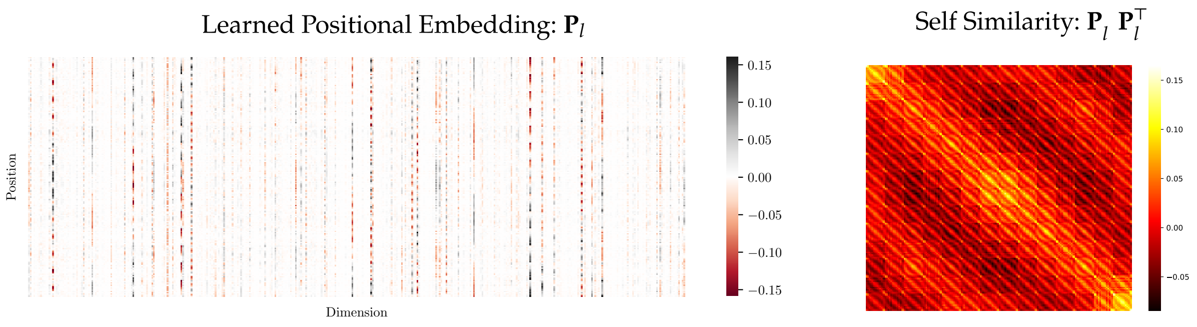
Figure 3.26: Linear projection layer of a vision transformer. We visualize the first 64 columns of the U matrix (see Eq. 3.7), which are the filters that are applied to the image patches. The filters are visualized as $16 \times 16 \times 3$ images, corresponding to the size of the image patches. Each row of U can be interpreted as the weights of a fully connected layer that is applied to the image patches. The filters capture different aspects of the input image, such as edges, textures, and colors. To compare the filters to the first convolutional layer of a CNN, which also captures low-level features in the input data, see Figure 3.10.

transformer are are radial and not oriented in a specific direction. Some more details on the relation between self-attention and convolutional layers see [129, 130].

Lastly, we will discuss the differences and similarities between fixed and learned positional embeddings that are added to the input tokens before they are passed through the transformer. We already discussed before that the positional embeddings are added to the input tokens to allow the transformer to distinguish between the position of each token in the image, and that the positional embeddings are either fixed or learned during training. When fixed positional embeddings are used, they are generated using sine and cosine functions, with each position in the sequence uniquely encoded. The formulation of the positional encoding in Eq. 3.6, and an interlaced fixed sine and cosine positional embedding $\mathbf{P}_f$ as it is used in a ViT is shown in Figure 3.27 (a). A learned embedding $\mathbf{P}_l$ that used the same ViT is shown in 3.27 (b). It is apparent that the learned embedding is more sparse and has a more nuanced structure than the fixed embedding. This becomes even clearer when we compute the pairwise dot product of the individual token positions $\mathbf{P}\mathbf{P}^\top$, as they are shown on the right side of Fig. 3.27. Each row of $\mathbf{P}_f\,\mathbf{P}_f^\top$ is most similar to itself (prominent diagonal), and the similarity between neighboring embeddings gradually decreases with distance. In contrast $\mathbf{P}_l\,\mathbf{P}_l^\top$ shows less regularity; One thing that immediately stands out is that it seems as if the similarity matrix is itself composed of smaller submatrices. These submatrices are of size 14×14 and reflect the number of horizontal and vertical patches into which the input image gets divided; cf. Fig.3.20. This effectively means that the sudden increase in similarity every 14 pixels in $\mathbf{P}_l\,\mathbf{P}_l^\top$ (or after each sub matrix) is in fact a neighboring token in the input image. To further understand the structure of the similarity matrix we can decompose it into its eigenvectors in the same fashion as for the linear projection matrix of the kernels (Fig. 3.26, Eq.3.7). The eigenvectors of the self-similarity of the positional embeddings can then be visualized and are shown both for the fixed and learned positional embeddings $\mathbf{P}$, see Fig. 3.28. The prominent feature in the learned positional embedding is that they correlate highly with embeddings that are in the same row or column as the embedding itself. This is in contrast to the fixed positional embeddings, which show a more regular pattern of correlation across the embeddings.

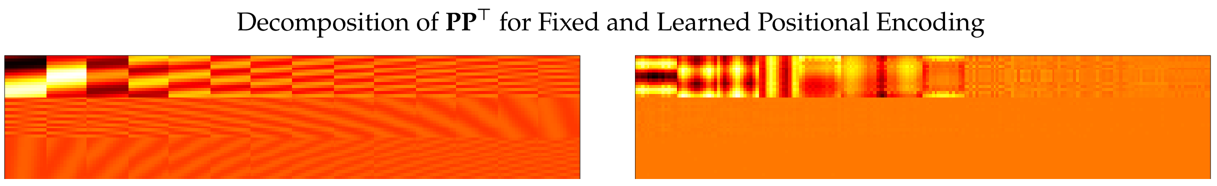


(a) Fixed positional embedding, based on Eq. 3.6.



(b) Learned Positional Embedding, optimized during training.

Figure 3.27: Left: Positional embedding. Right: Similarity of positional embeddings. The fixed positional embedding (a) is generated using sine and cosine functions, with each position in the sequence having a unique encoding. The learned positional embedding (b) is a trainable parameter that is optimized during training, allowing the model to adapt the embeddings to the specific task at hand. The similarity of the fixed and learned embeddings is visualized using a heatmap, showing the pairwise cosine similarity between the embeddings. The learned embeddings exhibit a more nuanced and task-specific structure, reflecting the model's ability to tailor the embeddings to the data distribution and learning objectives.



(a) Fixed Positional Encoding **(b)** Learned Positional Encoding

Figure 3.28: Decomposition of the Self-similarity matrix $\mathbf{PP}^\top$ for Fixed and Learned Positional Encoding, compare with Fig. 3.27. We display the first 42 principal components of the positional encoding matrix, scaled by the corresponding singular values. These components show how the learned positional encoding is mostly senstive to tokens that are located either in the same column or row as the token itself.

Extensions to the Vision Transformer

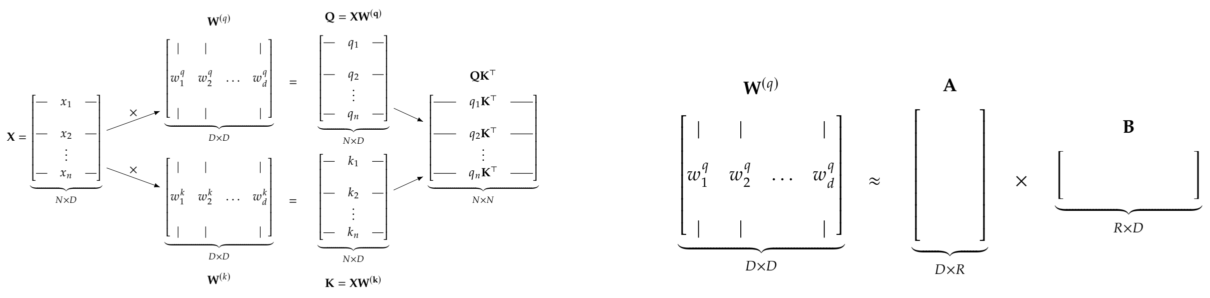
One of the reasons why Transformers spread across many different domains rapidly is the fact that they are highly modular and function as general purpose units of computation, given that the tokenization of the input data can be adapted to the specific task. This does not mean, however, that specific adaptation to individual domains have not occurred and are not necessary. In the domain of image understanding one of the main issues with the vision transformer is that it does not scale well with the input image size. The ViT has a fixed patch size of 16×16 pixels, which means that the number of tokens in the input sequence scales quadratically with the image size. This can lead to memory issues when processing high-resolution images, and can also limit the model's ability to capture fine-grained details in the input data. To address this issue, several extensions to the vision transformer have been proposed, which aim to improve the model's scalability and performance on high-resolution images.

Low Rank Approximations of Attention One downside of the attention mechanism is that it scales quadratically in the number of tokens in the input sequence. The Linformer [131], a linear complexity variant of the transformer model, approximates the self-attention mechanism so that it scales linearly with the sequence length, rather than quadratically. This is achieved by projecting the sequence dimension of the keys and values to a lower dimension before computing the attention scores.

The keys and values $\mathbf{K}, \mathbf{V}$ are projected into a lower dimension k (where $k \ll n$, n being the original sequence length), using a projection matrix $\mathbf{P} \in \mathbb{R}^{n \times k}$, resulting in

$$\mathbf{K} = \mathbf{KP}, \quad \mathbf{V} = \mathbf{VP}.$$

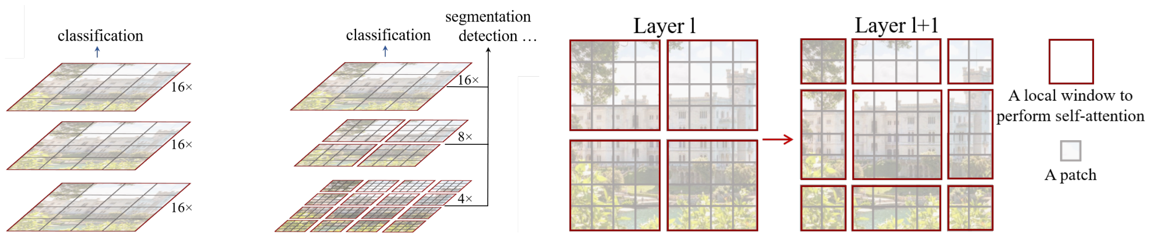
This transformation leads to a significant reduction when computing the self-attention - from $\mathcal{O}(n^2 \cdot d)$ to $\mathcal{O}(nk)$. Other ways of dealing with this quadratic complexity are a highly researched topic with the emergence of Large Language Models (LLMs) such as GPT-3 [132], and the desire to increase the context size that the model can effectively capture. However many of these methods can be described by some sort of low-rank approximation of the self-attention matrix, see Fig. 3.29.



(a) Full Self Attention **(b)** Low Rank Approximation of $\mathbf{W}^{(q)}$

Figure 3.29: Self Attention and a low rank approximation of the projection layers. This low rank approximation decreases the number of parameters and the computational complexity of the self attention mechanism. It allows for the use of larger models and longer sequences.

Windowed Attention Another way to reduce the computational complexity of the attention mechanism is to limit the attention computation to a local neighborhood around the current token. This approach also happens to solve other issues that the ViT has; The fixed patch size of 16×16 pixel patches is constant throughout the encoder blocks. This also means that the number of tokens that are created per image depend on the size of the image. While for many image and video summarization tasks (such as classification) the resolution of 16×16 is not a limiting factor, for other tasks such as for example segmentation it definitely decreases accuracy as every single pixel needs to be classified, and the patch of 16×16 might by too coarse. The *Shifted Window Transformer* (SWin Transformer) [133] proposes to solve the issues listed above. The key idea behind the SWin Transformer is to hierarchically process the image and its features across the transformer blocks - Fig. 3.30. This allows the model to capture multi-scale information in the input data. In particular, in early blocks the image is not subdivided into 16×16 pixel patches, but rather into 4×4 pixel patches. The resulting linear transformations take these patches and limit the attention to a window, Fig.3.30c, restricting the attention mechanism to a local neighborhood, reducing the computational cost. This local attention window mechanism is used in transformer architectures that need a way to limit the attention complexity from $\mathcal{O}(n^2 \cdot d)$ as in Vision Transformers to $\mathcal{O}(n \cdot m \cdot d)$, where n is the total number of patches, d is the dimensionality of the embeddings, and m is the number of patches within each window. Since $m \ll n$, this significantly reduces the computational complexity, making it more efficient for larger images or larger token sequences in general. The way in which the SWin transformer processes the data is reminiscent of the hierarchical processing in convolutional neural networks as discussed in the previous section.

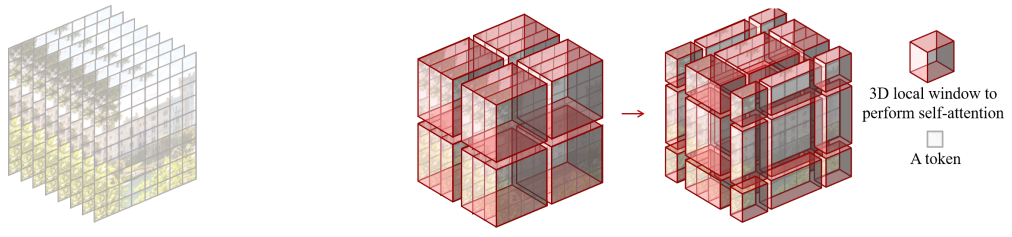


(a) ViT **(b)** Swin Transformer **(c)** Local Attention Window

Figure 3.30: The Swin transformer extracts patches of different spatial resolutions in each encoder block. It allows for scaling the Vision Transformer architecture for larger input images, as it reduces the computational cost of the attention mechanism, since attention is only computed in the local neighborhood of each token. The depth of the network is again responsible for increasing the receptive field, as it is in Convolutional Neural Networks. The hierarchical processing of the image features across the transformer blocks allows the model to capture multi-scale information in the input data and can be particularly useful if small spatial details are important, such as for segmentation tasks where individual pixels need to be assigned to a class. Figures taken from [133], Fig. 1 and Fig. 2.

Extensions to Handle Video Data

Video Swin With little modification, it is possible to adapt the originally proposed Swin Transformer to Video (Video Swin [97]), or any other multi-dimensional data, see Fig. 3.31. The tokenization is done in the same way as for images, *i.e.* extracting patches from all images individually, but computing the attention over a locally increasing spatio-temporal grid. The local window is therefore a small 3D volume, which means that a certain depth of the network is needed to represent long-range dependencies in the data.



(a) Tokenization **(b)** 3D-Local Attention Window

Figure 3.31: The SWin architecture can be extended to the spatio-temporal dimension in a quite straightforward manner. Tokens are still extracted from individual frames, and the attention is computed in a hierarchically increasing spatio-temporal grid. This means, like in the Swin Transformer, that a certain depth of the network is needed to aggregate the full information of the video into the latent representation. Figure taken from [97], Fig. 3.

Tokenization and Attention Factorization To extend the vision transformer to video input, there are two main parts to be considered:

1. The tokenization function that extracts tokens from the video data.
2. The attention computation that can be factorized in different ways.

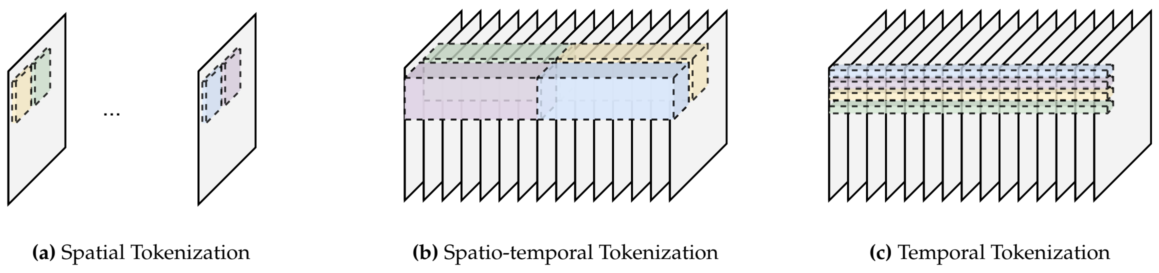


Figure 3.32: Illustration of different tokenization functions for extending the Vision Transformer to video data. **(a)** Each frame of the video is treated as an individual image, and tokens are extracted from each frame independently. This method does not capture temporal dependencies in the tokens themselves, but relies on the attention mechanism between frames to build temporal dependencies. **(b)** A sequence of frames are used and stacked to a volume from which tokens of both spatial and temporal extent are extracted. **(c)** Tokens are extracted from the whole video volume with a small spatial extent, treating temporal tubes as single tokens. This method captures temporal dependencies in the tokens themselves, and uses the attention mechanism to build spatial dependencies.

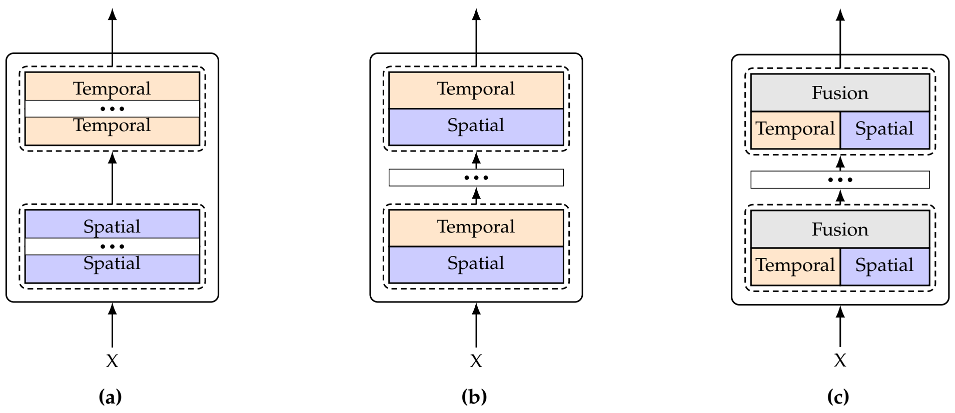


Figure 3.33: The possible factorization methods when extending the vision transformer into the domain of a Video Vision Transformer. **(a)** An extension of the ViT to video can be imagined as first processing the spatial information separately through multiple encoder blocks and limiting the attention to each spatial block. The temporal information can then be incorporated by dedicated temporal encoder blocks which perform the self-attention computation on the temporal dimension. **(b)** An alternative approach is to factorize the self-attention computation itself. The spatial and temporal information can be processed separately through spatial and temporal encoder blocks, but at the same network depth, and the self-attention computation can be factorized into spatial and temporal self-attention blocks. **(c)** A third approach is to fuse the spatial and temporal information at each layer through fusion blocks. Figure inspired by [29], Fig. 1.

For images, the choice of the tokenization function is quite straightforward. Non-overlapping patches of the image are extracted, and the tokens are reshaped into a vector, before being passed through the linear projection layers that feed into the transformer. The addition of the temporal dimension for video data allows for more flexibility in the tokenization function, see Fig. 3.32. Each frame of the video can be treated as an individual image, and the tokens can be extracted from each frame in the same way as for images, Fig. 3.32a. It is also possible to extract tokens along the temporal dimension, by considering a sequence of frames as a single token, Fig. 3.32b. Taken to the extreme it is also possible to extract tokens from a video volume with a small spatial extent such that the temporal tubes are treated as a single token, Fig. 3.32c.

The tokenization is intertwined with the attention computation, as the attention mechanism is applied to the tokens in the input sequence. However, the attention computation can be factorized in a similar way to the tokenization function, see Fig. 3.33c Prominent architectures that employ combinations of the aforementioned tokenization and attention strategies are the Video Vision Transformer (ViVit) [29], the Multiscale Vision Transformer (MViT) [95], and the TimeSformer [103]. Each of these architecture additionally introduces other modifications such as attention pooling to improve the performance.

Figure 3.34 shows the difference in the quality of the representation of the feature space of Kinetics-400 videos by including the temporal dimension in the tokenization and attention computation. In particular a TSNE-clustering [33] into 2 dimensions is performed, and each class is assigned a different color. The features

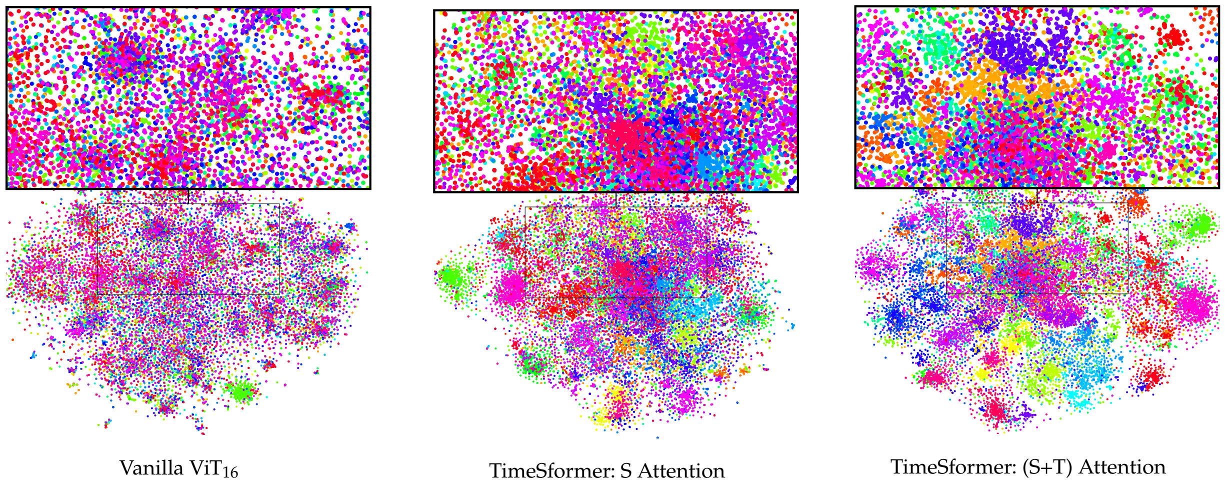


Figure 3.34: TSNE clustering of the features of Kinetics400 [106] with different transformer architectures. It shows how the features are better separated when using spatio-temporal attention. The vanilla ViT$_{16}$ has a lot of overlap between the classes, whereas the TimeSformer with spatio-temporal attention has better separation. Figure taken from [103].

of the vanilla ViT architecture show that the classes themselves are not well separated, and that the addition of spatio-temporal attention (as implemented with the TimeSformer [103] increases class separability.

3.4 Representation Learning and Training Schemes

Self-Supervised Learning

So far we have mainly discussed modes of learning where the target label is provided to the network, and can be used to calculate the loss based on the classification error. This type of training that requires the target label is called *Supervised Learning*. There are, of course, other modes of learning. We will discuss *Self Supervised Learning*. There are two main ways to motivate self-supervised learning. In the context of image and video understanding, the obvious reason to choose self-supervised learning is that it is often difficult to obtain labeled data; Large datasets need to be carefully annotated, and this is a time-consuming and expensive process. However the more interesting reason to choose self-supervised learning even in the presence of ground truth labels is that it can be used to learn representations of the data that are shown to have better generalization properties.

Contrastive Learning *Contrastive Learning* stands out as a pivotal technique, and traces its origins to Maximum Likelihood Learning [134–136], particularly within the context of image and video analysis [137–139]. This method leverages unlabeled data by learning to differentiate between similar (positive) and dissimilar (negative) pairs of data samples, see Fig. 3.35. Essentially, the model is trained to understand which samples are alike and which are not, without direct reference to explicit labels.

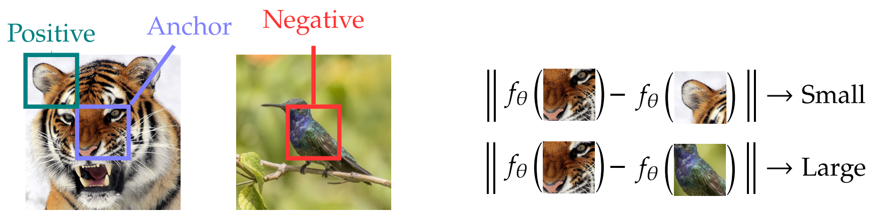


Figure 3.35: Example of a contrastive loss function for learning robust image representations. The function f_θ is a neural network that maps images to a feature space. The loss function measures the similarity between two images. An anchor image is compared to a positive image and a negative image. The loss function is designed to minimize the distance between the anchor and positive image while maximizing the distance between the anchor and negative image.

A core component of contrastive learning is the use of a feature space where similar samples cluster together, while dissimilar ones are pushed apart. This is achieved by optimizing a contrastive loss function, which minimizes the distance between positive pairs and maximizes it between negative pairs. Such an approach allows the model to develop a robust understanding of the inherent characteristics of the data. One significant advantage of contrastive learning is its ability to learn useful representations that can be transferred across different tasks or domains. These learned representations often exhibit enhanced generalization capabilities. By focusing on the relationships between data samples, rather than on predefined labels, contrastive learning can effectively harness the vast amounts of unlabeled data available, circumventing the limitations of labeled datasets. Thus, it emerges as a powerful tool in machine learning techniques, enabling more efficient and scalable learning solutions.

The contrastive loss for a pair of samples (x_i, x_j), where y_{ij} is 1 if the samples are similar (positive pair) and 0 if they are dissimilar (negative pair), can be formulated as:

$$\mathcal{L}(x_i, x_j) = y_{ij} \cdot \mathrm{d}(x_i, x_j)^2 + (1 - y_{ij}) \cdot \max\left(0, m - \mathrm{d}(x_i, x_j)\right)^2 \tag{3.8}$$

where $\mathrm{d}(x_i, x_j)$ is a distance metric between the features of x_i and x_j, and m is a margin that quantifies how far the dissimilar (negative) pairs should be pushed apart. Usually the positive and anchor samples are augmented patches of the same image, and the negative samples are patches from different images. The distance metric can be the Euclidean distance, the cosine similarity, or other similarity measures.

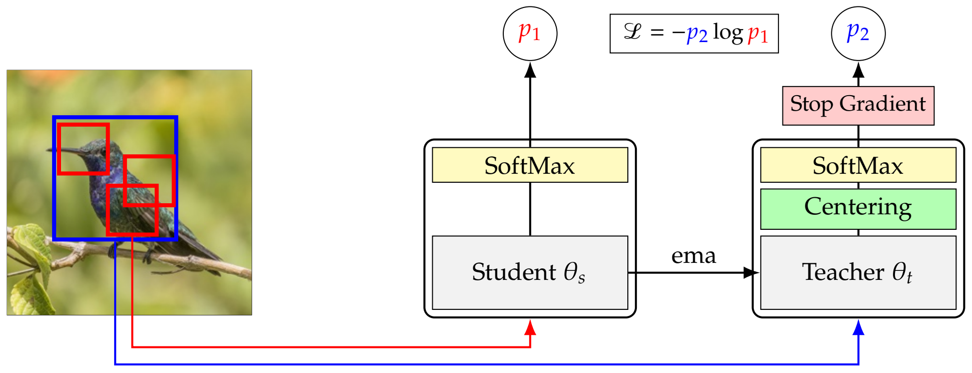


Figure 3.36: DINO: A self supervised learning method. DINO is a self-distillation process, where the teacher network is updated using an Exponential Moving Average (EMA) of the student network. The student network is trained to predict the teacher network's output. The important aspect is that the teacher network is given more context of the input image, i.e. larger crops of the full image compared to the student network. This allows the teacher network to have a better understanding of the image and thus provide better supervision to the student network. However in turn this also means that the student networks learns to predict output classes based on smaller regions of the image which leads to robust feature representations.

In practical implementations, the contrastive loss is computed over batches of samples rather than pairs. Each sample in a batch is treated as an anchor, and pairs are formed with every other sample in the batch. This results in a quadratic increase in the number of comparisons per batch, specifically $N(N-1)$ comparisons for a batch size of N. Efficiency in sampling is crucial because not all pairs contribute equally to learning; easy negatives (very dissimilar pairs) might not provide useful gradients after early training stages. Hence, strategies like hard negative mining [140] are employed where more emphasis is placed on negative pairs that are closer to the anchor in the feature space, as these are more informative and contribute to more effective training.

DINO *Distillation with No Labels (DINO)* has been developed as a novel framework that leverages self-distillation techniques to facilitate representation learning without labeled data. This approach particularly benefits image and video processing tasks by employing multiple instantiations of the same network, such as Vision Transformers, to achieve consensus on the representations of augmented versions of the same input image.

DINO's methodology is rooted in the principle of knowledge distillation, where a student model is trained to emulate the output of a teacher model. The difference between the two models is not architectural, but rather that the teacher model is given a larger input image, whereas the student only receives a crop of that image. The teacher's weights are updated as an Exponential Moving Average (EMA) of the student's weights. Mathematically, this relationship can be expressed as:

$$\theta_t = \tau\theta_t + (1-\tau)\theta_s, \tag{3.9}$$

where θ_t and θ_s represent the parameters of the teacher and student models, respectively, and τ is a decay coefficient controlling the update rate. For a schematic of the DINO training process refer to Fig. 3.36.

This configuration promotes an internal consistency in the feature space, encouraging the model to focus on invariant features across different augmentations of the same image.

The distillation process is governed by a cross-entropy loss between the softened outputs of the teacher and the student, where 'softened' refers to the use of a temperature setting in the SoftMax function to produce smoother probability distributions. This technique enhances the stabilization of the learning process by providing more informative gradients [141]. This cross-entropy loss can be expressed as:

$$\mathcal{L} = -\sum p_2 \log p1,$$

where p_1 and p_2 are the SoftMax outputs of the teacher and student models respectively.

Robust and Transferable Features One of the significant advantages of DINO is its ability to produce semantically meaningful embeddings where similar instances are closely aligned in the feature space, without relying on explicit label information. This property enables the learned features to be robust and transferable across different tasks and domains. By utilizing the inherent variability within unlabeled datasets and circumventing the need for extensive manual labeling, DINO provides a scalable and efficient solution for extracting valuable information from large volumes of unlabeled data. For an example of the robustness of the feature representations learned by DINO versus a supervised training scheme we will compare features of a Vision Transformer (ViT_{16}) where the only difference is the supervised versus self-supervised training scheme, see Fig. 3.37.

We extract keys ($\mathbf{K}$) of all tokens in the input image, at various depths in the network. As the (ViT_{16}) uses 11 encoder blocks, we investigate the difference of the representations from the 2^{nd} to 11^{th} block. To visualize the major contributing factors we use PCA on the extracted keys and plot the first three principal components after each block. The representations learned by DINO are less noisy and semantically meaningful as they roughly correspond to object parts. This is interesting as the model was trained without any explicit labels, and the representations are learned by the model itself.

During the clustering we omitted the `[CLS]` token that is used in the final classification layer of the model. This token is used to summarize the information of the image and is not used for the clustering. However we can also visualize the attention of the `[CLS]` token of the DINO model and compare it to that of a supervised ViT. We randomly choose 3 attention heads for both the DINO and the supervised model and visualize the attention of the `[CLS]` token in the last encoder block for a given image. The results show spurious activation, which roughly indicates that the representations are very specific, as the network finds some small part in the input image and attends to that token, whereas more semantically meaningful and interpretable attention patterns emerge when looking at the attention of the DINO model, see Fig. 3.38.

Notes on Pre-training Pre-training involves training a model on a large, general dataset before fine-tuning it on a more specific dataset tailored to a particular task. This method leverages the idea that models can learn useful, transferable features from a comprehensive dataset which can then be applied to enhance performance on narrower tasks that may have less data available for training. In the context of video action recognition, this approach is particularly beneficial due to the complex and data-intensive nature of video processing. Pre-training allows models to learn robust spatial and temporal features that are crucial for understanding video content, which can significantly reduce overfitting when the model is later fine-tuned on specific action recognition tasks.

One common technique in video action recognition is the inflation of 2D CNNs into 3D CNNs. This process, often referred to as *network inflation* [85], involves adapting architectures originally designed for image analysis to handle video input. The 2D convolutional filters, which typically operate on image height and width, are expanded into the temporal dimension. This expansion turns them into 3D filters that can also process time, allowing the network to learn from the motion information encoded in multiple consecutive frames. When inflating a 2D model into 3D, the pretrained weights of the 2D model are used as a starting point. These weights are replicated across the newly added temporal dimension, maintaining the model's ability to detect spatial features while beginning to learn temporal features.

Pre-training and sufficient data augmentation have proven to be important factors in enhancing the performance of models. Experiments in the TimeSformer demonstrate the significance of pre-training on large-scale datasets like ImageNet, where just Pre-training on ImageNet increases the Top-1 Accuracy on the Kinetics-400 dataset from 64.8% to 80.5% [103].

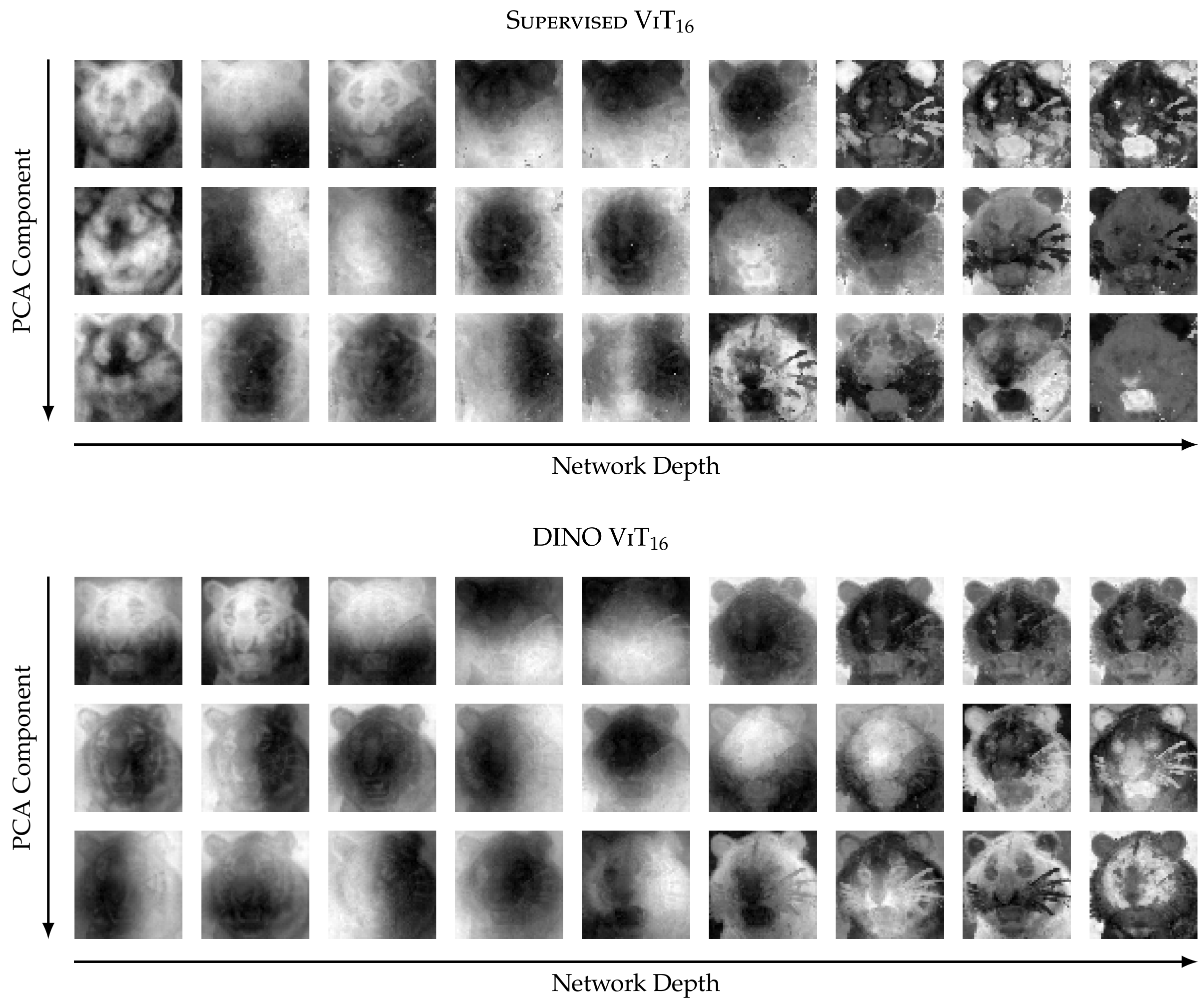


Figure 3.37: We visualized the learned representations of the same vision transformer trained in a supervised manner (top) versus a self-supervised manner (bottom). We cluster the keys of all patches in a encoder block using *PCA* and show the first 3 components. The supervised network shows more spurious activations and also very strong activations in regions that do not directly correspond to semantically meaningful regions. This is contrasted with the DINO trained ViT that is grouping things semantically in the later layers. Both networks show a strong bias towards grouping things based on locality in the early layers, which is due to the fixed positional encoding that is used, cf. Fig.3.22. This also shows that not just the attention maps, but also the learned features themselves can be used for feature description, especially if the semantics of the features are of interest.

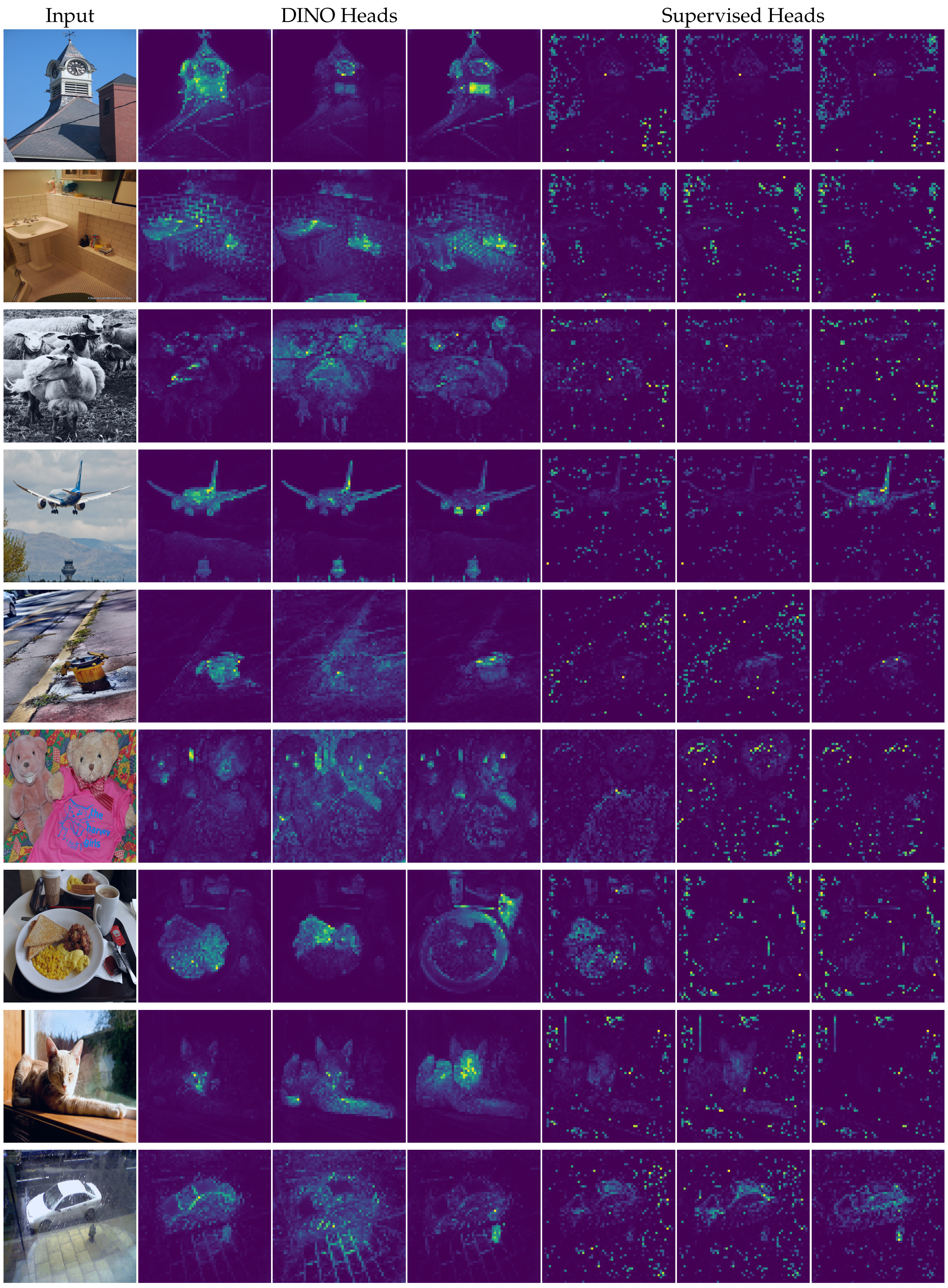


Figure 3.38: Visualizing the attention heads from the last layer of the `[CLS]` token as a query for the different attention heads. We can see that the learned attention is less spurious, and that the attention heads capture correct semantic meaning and similarity between semantically similar parts in the input image. Figure adapted from [142], Appendix A, Fig. 10.

The Importance of Motion for Action Recognition 4

Contents

Appeared as
"Is Appearance Free Action Recognition Possible?"
in ECCV 2022.

Intuition might suggest that motion and dynamic information are key to video-based action recognition. In contrast, there is evidence that state-of-the-art deep-learning video understanding architectures are biased toward static information available in single frames. Presently, a methodology and corresponding dataset to isolate the effects of dynamic information in video are missing. Their absence makes it difficult to understand how well contemporary architectures capitalize on dynamic vs. static information. We respond with a novel Appearance Free Dataset (AFD) for action recognition. AFD is devoid of static information relevant to action recognition in a single frame. Modeling of the dynamics is necessary for solving the task, as the action is only apparent through consideration of the temporal dimension. We evaluated 11 contemporary action recognition architectures on AFD as well as its related RGB video. Our results show a notable decrease in performance for all architectures on AFD compared to RGB. We also conducted a complimentary study with humans that shows their recognition accuracy on AFD and RGB is very similar and much better than the evaluated architectures on AFD. Our results motivate a novel architecture that revives explicit recovery of optical flow, within a contemporary design for best performance on AFD and RGB.

Keywords: *Action Recognition, Static and Dynamic Video Representation, Human Motion Perception*

https://f-ilic.github.io/AppearanceFreeActionRecognition

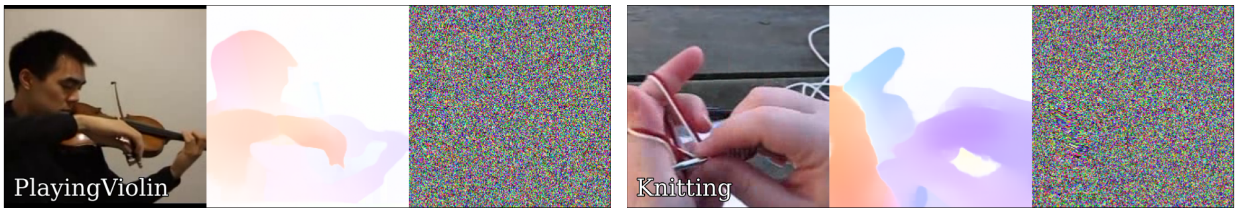


Figure 4.1: Appearance Free Dataset (AFD) for Action Recognition. Within each set of three images we show: a single RGB frame (left); corresponding optical flow in Middlebury colour coding [61] (middle); corresponding appearance free frame (right). When viewed as video the AFD reveals the motion in the original video even as any single frame provides no relevant discriminative information.

4.1 Introduction

Motivation

Action recognition from video has been subject of significant effort in developing and improving both algorithms and datasets [79, 143]. This interplay between algorithm and dataset advances has led to paradigm shifts in both. Algorithms have evolved from primarily hand-crafted mathematically, physically, and heuristically driven approaches to methods based on deep-learning architectures. Datasets have evolved from relatively small, carefully selected videos to massive, web-crawled collections. As a result, current state-of-the-art algorithms for action recognition achieve impressive levels of performance on challenging datasets. In contrast, the internal representations learned by the architectures remain under explored [144]. This lack of understanding is unsatisfying both scientifically as well as pragmatically. From a scientific perspective, such detailed analysis is integral to understanding how a system operates and can guide further improvements. From a pragmatic perspective, multiple jurisdictions are beginning to require explainability as a precondition for deployment of artificial intelligence technologies [145, 146]; technologies lacking such documentation may not see real-world application.

Some evidence suggests that contemporary deep-learning architectures for video understanding can perform well on the task of action recognition with little to no regard for the actual actors [147–149]. These studies suggest that static visual information available in a single frame (*e.g*. colour, spatial texture, contours, shape) drives performance, rather than dynamic information (*e.g.*, motion, temporal texture). While some methodologies have been aimed at understanding the representations learned by these architectures [115, 150, 151], none provide an approach to completely disentangle static vs. dynamic information. In response, we have developed an Appearance Free Dataset (AFD) for evaluating action recognition architectures when presented with purely dynamic information; see Fig. 4.1. AFD has been rendered by animating spatial noise, historically known as Random Dot Cinematographs RDCs [152], using image motion extracted from the UCF101 dataset [107] by a state-of-the-art optical flow estimator [69]. The resulting videos show no pattern relevant to action recognition in any single frame; however, when viewed as video reveal the underlying motion. We have produced AFD for the entire UCF101 dataset, evaluated a representative sample of contemporary action recognition architectures and used our results to drive development of a novel architecture that enhances performance when confronted with purely dynamic information, as present in AFD.

The ability of action recognition (and other) video architectures to work in absence of static appearance information is not only an academic exercise. Real-world deployment scenarios may require it, such as in the presence of camouflage. Two examples: Video surveillance for security should be able to cope with nefarious actors who artificially camouflage their activities; video-based wildlife monitoring must be robust to the natural camouflage that many animals possess to hide their presence.

Related work

Datasets A wide variety of datasets for development and evaluation of automated approaches of video understanding are available [153]. For action recognition, in particular, there is a large body ranging from a few dozen classes [100, 107, 108, 115] to massively crawled datasets with classes in the hundreds [85, 100, 113, 114]. Some work has specifically focused on curating videos where temporal modeling is particularly important for action recognition [113, 115, 154, 155]; however, they still contain strong cues from single frame static appearance (*e.g.* colour, shape, texture). Overall, no action recognition dataset completely disentangles single frame static information from multiframe dynamic information.

Camouflaged actors would provide a way to evaluate recognition systems based primarily on dynamic information. While there are camouflage video understanding dataset available, they are of animals in the wild as selected for object segmentation (*e.g.* [156, 157]). In contrast, it is unlikely that a non-trivial number of real-world videos of camouflaged actors can be found. In response, our Appearance Free Dataset (AFD) provides synthetic camouflaged action recognition videos as coherently moving patterns of spatial noise, historically known as Random Dot Cinematograms (RDCs) [152]. These patterns are defined via animation of an initial random spatial pattern, where the pixelwise intensity values of the pattern are randomly selected. By design, they reveal no information relevant to the motion in any single frame; however, when viewed as video the motion is apparent; examples are shown in the right columns of Fig. 4.1. Such videos have a long history in the study of motion processing in both biological [158, 159] and machine [160, 161] vision systems. Interestingly, humans can understand complex human body motion solely from sparse dots marking certain body points, even while merely a random dot pattern is perceived in any single frame [162].

Synthetic video, both striving for photorealism [62, 67, 163–165] and more abstracted [105, 166], is a common tool in contemporary computer vision research that allows for careful control of variables of interest. Our effort adds to this body with its unique contribution of a synthetic camouflage action recognition dataset for probing a system's ability to recognize actions purely based on dynamic information. Notably, while our texture patterns are synthetic, they are animated by the motion of real-world actions [107].

Models and action recognition architectures A wide range of action recognition architectures have been developed [153]. Most contemporary architectures can be categorized as single stream 3D (x, y, t) convolutional, two-stream and attention based methods. Single stream approaches are motivated by the great success of 2D convolutional architectures on single image tasks [5, 6, 167, 168]. In essence, the 3D architectures extend the same style of processing by adding temporal support to their operations [28, 81, 85, 88, 169]. Two-stream architectures have roots in biological vision systems [170, 171]. The idea of having separate pathways for **static** video content and **dynamic** information has been variously adopted. A key distinction in these designs is whether the pathway for dynamic information explicitly relies on optical flow estimation [26, 172] or internally computed features that are thought to emulate flow like properties [27]. Attention-based approaches rely on various forms of non-local spatiotemporal data association as manifested in transformer architectures [87, 95]. For evaluation of architectures on the new AFD, we select a representative sampling from each of these categories.

Interpretability Various efforts have addressed the representational abilities of video understanding architectures ranging from dynamic texture recognition [173], future frame selection [174] and comparing 3D convolutional vs. LSTM architectures [175]. Other work centering on action recognition focused on visualization of learned filters in convolutional architectures [150, 151], or trying to remove scene biases for action recognition [115]. Evidence also suggests that optical flow is useful exactly because it contains invariances related to single frame appearance information [176]. More closely related to our study is work that categorized various action classes as especially dependent on temporal information; however, single frame static information still remains [155]. Somewhat similarly, an approach tried to tease apart the bias of various architectures and datasets towards static vs. dynamic information by manipulating videos through stylization; however, single frame static information remained a confounding factor [177]. While insights have been gained, no previous research has been able to completely disentangle the ability of these models to capitalize on single frame static information vs. dynamic information present across multiple frames. We concentrate on the ability of these systems to capture dynamic information, with an emphasis on action recognition.

Contributions

In light of previous work, we make the following three major contributions.

1. We introduce the Appearance Free Dataset (AFD) for video-based action recognition. AFD is a synthetic derivative of UCF101 [107] having *no static* appearance cues, however, revealing real-world motion as video.
2. We evaluate 11 contemporary architectures for action recognition on AFD; *i.e.* dynamic information alone. We conduct a psychophysical study with humans on AFD, and show significanly better performance of humans than networks. These results question the ability of the tested networks to use dynamic information effectively.
3. We provide a novel improved action recognition architecture with enhanced performance on AFD, while maintaining performance on standard input.

We make AFD, associated code and our novel architecture publicly available.

4.2 Appearance free dataset

Our proposed appearance free dataset, AFD101, is built from the original UCF101 [107] by employing a state-of-the-art optical flow estimator [69] to generate the corresponding framewise flow that is used to animate spatial noise patterns. The resulting AFD101, consisting of 13,320 videos, depicts realistic motion even while any individual frame is devoid of static appearance information that is relevant for action recognition. Figure 4.1 and the project webpage provide illustrative examples. We use UCF101 from many possible choices [153] as the basis for AFD because it was widely used in action recognition evaluation and is large enough to support training, yet small enough to facilitate numerous experiments.

Dataset generation

Generation of AFD from RGB video follows three key steps; see Fig. 4.2.

1. Initialization: Generate a single frame of noise, with same spatial dimensions as the frames in the input video. The pixel values are sampled uniformly i.i.d. as three-channel RGB values. Let this frame be denoted as R_0.
2. Flow calculation: Generate interframe optical flow for each temporally adjacent pair of input RGB frames, $I_{t-1}, I_t, t \in [1, T]$, yielding the flow $F_{t-1,t}$, with T being the number of frames in the input video. We use RAFT [69], for the extraction of optical flow.
3. Warping: We warp the initial noise frame R_0 using $F_{t-1,t}$, $t \in [1, T]$, to generate the next noise frame, R_t. The output frames, R_t, are an appearance free version of the input RGB video, where at each frame all that is seen is i.i.d. noise, but whose interframe motion corresponds to that of the original video; this output video has the same spatial and temporal dimension as the input video.

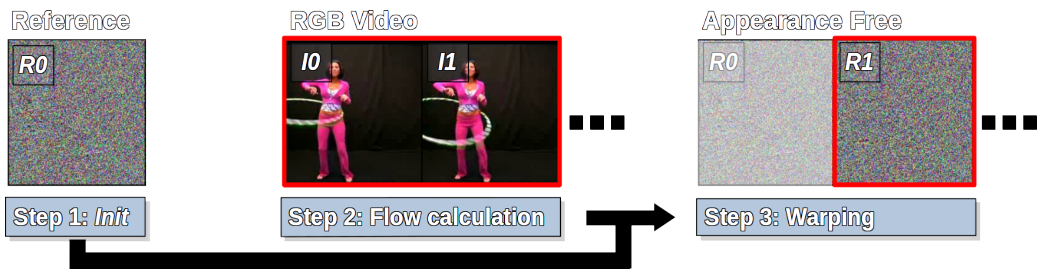


Figure 4.2: Appearance free videos are created by 1) initializing noise *R0*, 2) calculating optical flow between input frame pairs, and 3) using the optical flow to warp *R0*. Steps 2 and 3 are repeated for each frame-pair in the input video.

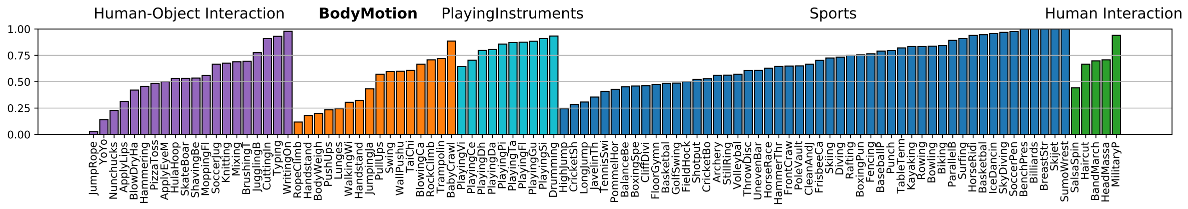


Figure 4.3: ResNet50 Top-1 Accuracy on single frame UCF101 by *Action Groups*.

Implementation details

The details of our proposed methodology are particularly important, as all *static* discriminatory single frame information must be removed; to this end nearest neighbor interpolation during warping is performed. For each warp, newly occluded pixels are overwritten by the new value moved to that location; de-occluded pixels are filled with R_0 values to ensure that the spatial extent of the noise is constant. Along the border where the warping leads to undefined values (*e.g.* pixel at that location moves elsewhere and no new value is warped into that location) are treated as de-occlusions as well. In preliminary experiments, we found that other choices of implementation, *e.g.* bilinear interpolation as opposed to nearest neighbor interpolation, or always estimating flow with respect to I_0 always yielded inferior results and unwanted artifacts. In the worst cases these artifacts led to revealing motion boundaries of actors in single frames.

Appearance in optical flow

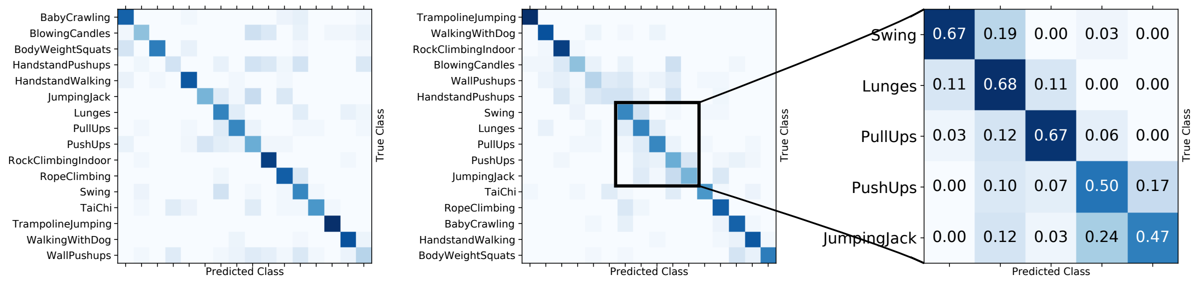
The concept of appearance consists of multiple intertwined qualities, including texture, contour and shape. While optical flow is indeed free of texture it is not free of contour and shape induced by motion of the camera or the actor itself; this fact can be observed qualitatively in Fig. 4.1: The flow visualizations alone reveal violin playing and knitting. We quantify this observation as follows. We train a strong single frame recognition architecture, ResNet50 [168], on single frames from three datasets: UCF101 (the original), UCF101Flow, comprised of optical flow frames generated from UCF101 by RAFT [69], and AFD101. Top-1 recognition results for UCF101, UCF101Flow and AFD101 are 65.6%, 29.4% and 1.1%, respectively. These results show that only AFD101 contains no discriminatory information in a single frame, as only 1.1% top-1 accuracy is in line with random guessing.

A dataset subset for exploration

To evaluate the performance of state-of-the-art architectures in the context of human performance, a reduced subset of the classes in UCF101 is needed, simply because it is not feasible to conduct an experiment with humans with the original 101 categories. We chose to have five classes as it is possible for humans to hold five action classes in working memory for a prolonged duration [178]. In the following, we refer to the reduced size version of AFD101 as AFD5 and the corresponding reduction of UCF101, *i.e.* the RGB video, as UCF5.

A defining property of the subset is that it discourages recognition from static information in single frames even within the original RGB input. To this end we train a ResNet50 [168], used by many state-of-the-art architectures as inspiration for their backbone [153], on the entire UCF101 dataset in a single frame classification manner, *i.e.*, we randomly sample a frame from UCF101. By design, this procedure isolates the ability of any given class to support recognition on the basis of a single frame. UCF101 is subdivided into five action groups [107]: *Human Object Interaction*, *BodyMotion*, *PlayingInstruments*, *Sports* and *Human Human Interaction*; Fig. 4.3 has single frame recognition accuracy by group.

Our selection criterion is to choose videos with similar appearance; therefore, we select a subset of actions within a group. To select the action group we consider the following points. The *Body Motion* group is particularly attractive because it is the only category that has five or more classes with below 25% single frame



(a) Alphabetically Sorted **(b)** Reordered by diagonal **(c)** Selected Subgroup

Figure 4.4: Singleframe ResNet50: After finetuning on the *BodyMotion* action group of UCF101, we a) inspect the resulting confusion matrix b) reorder it by its diagonal weight so that classes that are often confused among each other get grouped together, and c) select the most prominent subgroup.

recognition accuracy. The *Human Object Interaction* group is the next closest in terms of having several (albeit fewer that five) categories with such low single frame accuracy. However, its classes tend to be distinguished by featuring a prominent object and we do not want to promote categorization based on single frame object recognition. The *PlayingInstruments* group also features prominent objects and likewise is a poor choice. The remaining groups (*Sports* and *Human Human Interaction*) have generally higher single frame recognition accuracies; so, also are excluded. These considerations lead us to *Body Motion* as the group of interest for present concerns.

With the selection of *Body Motion*, we finetune the previously trained ResNet on that particular group. Given that finetuning, we perform confusion matrix reordering [179] and select the five classes that are most confused among each other in the group. More precisely we find the permutation matrix that has the highest interclass entropy, excluding classes with less than 50% accuracy; see Fig 4.4. This results in the choice of the following five classes for AFD5: *Swing*, *Lunges*, *PushUps*, *PullUps*, *JumpingJack*. AFD5 consists in total of 583 videos with approximately 116 videos per class.

4.3 Psychophysical study: Human performance

To set a baseline for Appearance Free Data we perform a psychophysical study. The ability of humans to recognize actions from dynamic information alone, *i.e.* without any single frame static information, has been documented previously [162, 180, 181]. These studies are conducted with sparse dots, typically at major body joints, on otherwise blank displays. Our study appears to be unique in its use of dense noise patterns *i.e.* dense random dots.

Experiment design

A session in the experiment consisted of the following seven phases. (i) The participant is presented with a brief slide show explaining the task. The participant is told that they will see a series of videos as well as a menu on the same display from which to indicate their perception of a human action that is depicted. (ii) Training is provided on UCF5, *i.e.*, RGB video. During this phase, four training videos from each class, totaling 20 videos, are presented with correct responses indicated in the menu. The videos are shown in randomized order. (iii) Testing is performed on 90 UCF5 test videos. Each video repeats until the participant makes their selection on the menu. (iv) A rest period of five minutes is provided. (v) Training is provided on AFD5, *i.e.* appearance free video. During this phase, four training videos from each class are presented with correct responses indicated in the menu. The videos also are shown in randomized order. (vi) Testing is performed on 90 AFD5 test videos. Each video repeats until the participant makes their selection on the menu. (vii) Participants are given a questionnaire to complete, with questions including visual impairments (all had normal or corrected vision), familiarity with computer vision, action recognition, as well as their impression

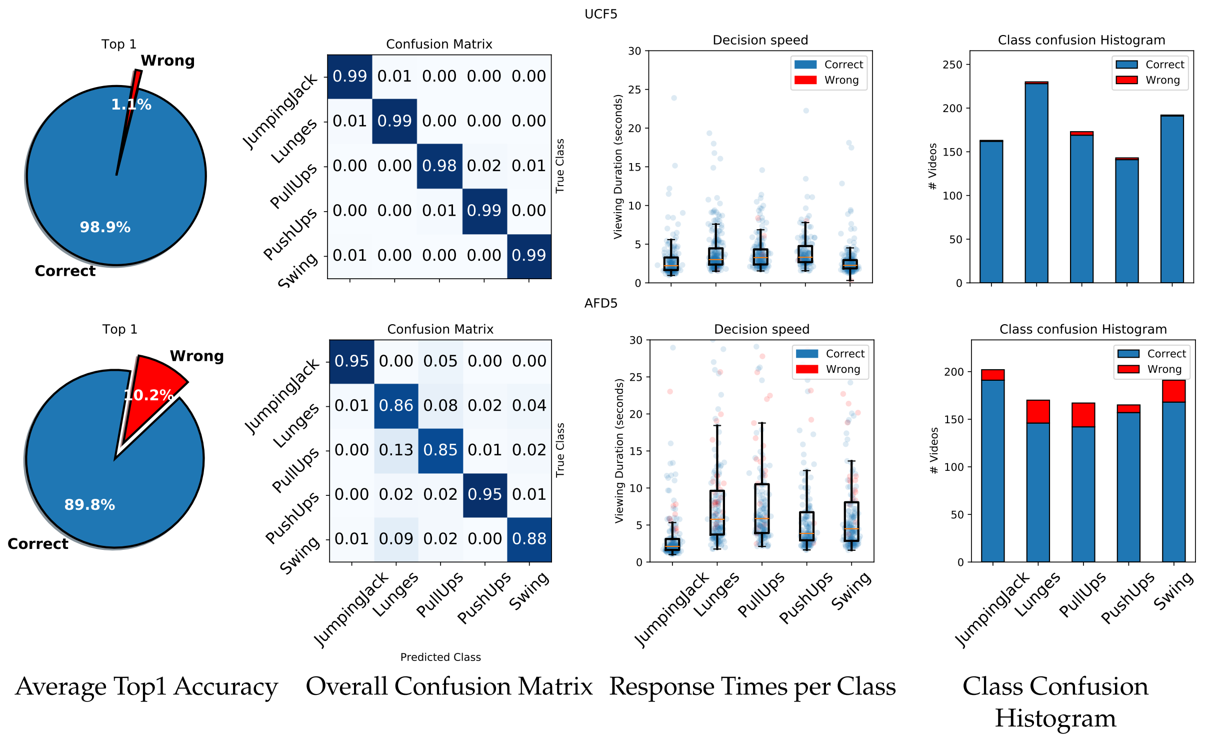


Figure 4.5: Results of psychopysical study. Top row: Performance on standard RGB video, UCF5. Bottom row: Performance on appearance free data, AFD5. Additional results, including a breakdown per participant are presented in the Appendix.

of task difficulty. During both test phases, action category choices and response times were recorded. Ten human participants were recruited as volunteers to participate in the experiment. All experiments were conducted in the same environment, *i.e.* computer/display, lighting and seating distance from the display (0.5 meters). Participants were allowed to view freely from their seat and take as much time in making their responses as they like. All participants completed their session within approximately 30 minutes, including the questionnaire. Participants were informed and consented to the data gathered in written and signed manner, and were provided with sweets and thanks at task completion. The sample size of ten participants is in our view sufficient to show that various people with different backgrounds are *able* to solve AFD data without much trouble and to establish a baseline for comparison to action recognition architectures.

Human results

Results of the psychophysical study are shown in Fig. 4.5. On average, our participants perform with an accuracy of 89.8% on AFD5 and 98.9% on UCF5. Participants report that mistakes on UCF5, the RGB videos, resulted from accidental clicks when making selections. Our analysis of AFD5 samples that were most frequently misclassified by the participants reveals that those videos typically have sudden and high speed camera movement, *e.g.* as induced by quick panning and zooming, which in turn creates very large optical flow displacement. The plots of response time show that participants take longer to make their choice on AFD5, with the exception of *JumpingJack* which remains largely unaffected. The extended time taken for AFD5 suggests that appearance free recognition requires more effort and repeated playback of the video is necessary, even though ultimate performance is high.

Overall, the psychophysical results document the strong ability of humans to recognize actions on the basis of AFD, as well as standard RGB videos.

4.4 Computational study: Model performance

We evaluate 11 state-of-the-art action recognition architectures on UCF101 and our new AFD101. We select representative examples from the currently most widely used categories of action recognition architectures, *cf*. Sec. 4.1: single stream 3D convolutional [28, 85, 88], two-stream convolutional [26, 27] and attention-based [87, 95]. We also evaluate on a standard 2D convolutional architecture (ResNet50 [168]) to verify that our AFD does not support classification based on single frame static information. Notably, our results from the psychophysical study allow us to compare the performance of the network architectures to that of humans on both UCF5 and AFD5.

Training procedure

To guarantee a fair comparison all reported results are obtained by training and testing with our procedure, described below. Furthermore, since we want to investigate architectural benefits on Appearance Free Data, all results (except for one architecture, see below) are obtained without pretraining - concurrently often done on Kinetics400 [85] - as there is no AFD for Kinetics. Our goal is not to show how to archieve best performance on UCF101; it is to show the difference in performance of each network when presented with *only* dynamic information. We also ran preliminary experiments with pretraining on Kinetics and finetuning on AFD, but saw no significant improvement during AFD testing compared to no pretraining. Nevertheless, since all the evaluated architectures, save one, employ a ResNet-like backbone we use ImageNet pretrained versions of those to initialize our training. The exception is MViT, which needed to be initialized with Kintecs400 weights, as its performance was not increasing from baseline with the aforementioned scheme. However the Kinetics weights were only used as an intializer and we allowed for retraining of the entire backbone to make the results comparing UCF and AFD as comparable as possible. We acknowledge that our training strategy does not allow for the architectures to show state-of-the-art performance on RGB input in our experiments; however, our approach is necessary to allow for fair comparison of performance on RGB vs. AFD and measure the respective performance between the modalities.

Training details

We train with Adam [47] using an early stopping scheme, and a base learning rate of $3e^{-4}$. The batch size is chosen as large as possible on two GPUs with 24GB memory each. Data augmentation, a staple of modern deep learning, is kept the same for all evaluated models and datasets; uniform temporal sampling with jitter, random start frame selection, and contrast, hue, and brightness jitter are applied. Random horizontal flipping of entire videos is used, as actions in UCF101 do not depend on handedness. A quite aggressive data augmentation scheme that changes over the course of every 20 epochs to a less aggressive one, and capping at 80 epochs is used. We find that this aggressive data augmentation technique is needed to achieve reasonable results when no pretraining is used. Spatial center crops of 224×224 are used as the final input to all networks with the exception of X3D XS and S, which use a spatial resolution of 160×160. The reported results are the averages of the three standard splits used on UCF101, now applied to both the original RGB as well as AFD. The results on UCF5 and AFD5 were obtained by finetuning the networks previously trained on their 101 class counterparts and using the same splits.

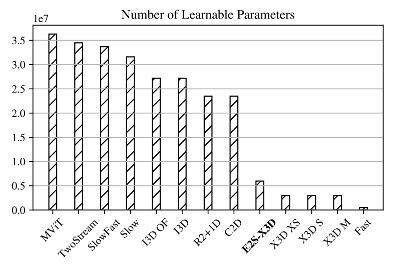
Model results

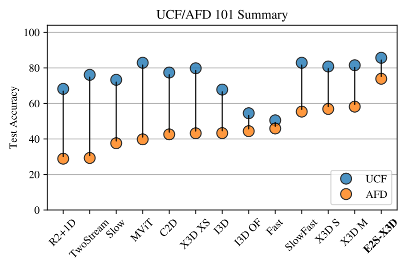
Table 4.1 shows top-1 accuracy results for all evaluated architectures on UCF101, AFD101, UCF5 and AFD5. Note that the results for the single frame ResNet on UCF101 essentially are the same as those shown in Fig. 4.3. The results in Table 4.1 also validate our claim of having removed all *static*, single frame information relevant to recognition, as the performance is equal to random choice.

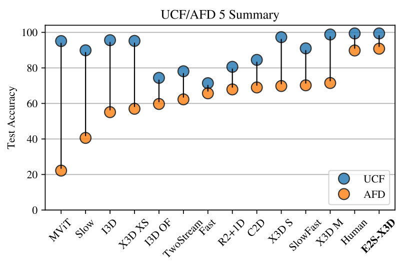
For most of the action recognition architectures, it is seen that obtaining ≈ 70-80% top-1 accuracy is possible. This fact makes the sizable drop of performance on AFD data, the Δ columns in Table 4.1, especially striking. The average performance drops by approximately 30% almost evenly across all architectures - with two notable exceptions: The Fast stream of SlowFast and I3D OF a common I3D architecture with an optical flow estimator before the network input. These two architectures are noteworthy because their ability to maintain performance across regular and appearance free video apparently stems from their design to prioritize temporal information over spatial; although, this focus on temporal information leads them to be less competitive on the standard RGB input in comparison to the other evaluated architectures. The comparison to human performance shows X3D variations to be the best architectures, competitive with human performance on UCF5; *however*, the best architectures on AFD5 (X3D M and SlowFast) are *considerably* below human top-1 accuracy (18% below). These results show that there is room for improvement by enhancing the ability of current architectures to exploit dynamic information for action recognition. It also suggests that optical flow is *not* what is learned by these networks, when no explicit representation of flow is enforced. Importantly, TwoStream [26] and I3D [85] OF were evaluated with RAFT optical to allow for a fair comparison with each other and with our new algorithm, introduced in the next section. The summary plots below Table 4.1 show learnable architecture parameter counts and ordering by AFD accuracy. These plots further emphasize the fact that explicit representation of dynamic information (*e.g.* as in I3D OF and to some extent Fast) is best at closing the gap between performance on RGB and AFD. X3D M is among the top performing architectures on RGB; further, among those top RGB performers, it shows the smallest Δ. Also, it is has a relatively small number of learnable parameters, with only Fast notably smaller. These observations motivate the novel architecture that we present in the next section. It improves performance on AFD, while also slightly improving performance on RGB. This architecture is given as E2S-X3D in Table 4.1. Moreover, it performs on par with humans on UCF5 and AFD5.

Table 4.1: Top-1 accuracy for the evaluated action recognition architectures. The second column indicates the number of frames and the temporal sampling rate, $f \times r$, of each network. For two-stream approaches the notation $\binom{\text{Appearance}}{\text{Motion}}\binom{f\times r}{f\times r}$ is used. A drop in performance across all networks on AFD data is observed. Plots below the table show parameter counts as well as summarize the accuracy findings, sorted by AFD performance. Absolute performance on UCF is of secondary importance; it merely gives a point of comparison with respect to AFD. The Δ columns shows the difference between UCF and AFD performance, which highlights the relative *dynamic* recognition capability of architectures.

	$f \times r$	UCF101	AFD101	Δ	UCF5	AFD5	Δ
Single Image Input							
ResNet50 [168]	1×1	65.6	1.1	64.5	74.2	20.1	54.1
Video Input							
I3D [85]	8×8	67.8	43.3	24.5	95.6	55.1	40.5
TwoStream [26]	$\binom{1\times1}{10\times1}$	76.1	29.3	46.8	78.1	62.3	15.8
C2D [87]	8×8	77.4	42.6	34.8	84.5	69.0	15.5
R2+1D [88]	16×4	68.2	28.9	39.3	80.6	67.9	12.7
Slow [27]	8×8	73.3	37.6	35.7	89.9	40.6	49.3
Fast [27]	32×2	50.5	45.9	4.6	71.4	65.7	5.7
SlowFast [27] $\binom{\text{Slow}}{\text{Fast}}$	$\binom{8\times8}{32\times2}$	82.9	55.4	27.5	91.0	70.2	20.8
I3D [85] OF	8×8	54.5	44.4	10.1	74.4	59.7	14.7
X3D XS [28]	4×12	79.8	43.2	36.6	95.2	57.0	38.2
X3D S [28]	13×6	80.8	56.9	23.9	97.3	69.8	27.5
X3D M [28]	16×5	81.5	58.2	23.3	98.8	71.5	27.3
MViT [95]	16×4	82.9	39.8	43.1	95.1	22.2	72.9
E2S-X3D **Ours** $\binom{\text{M}}{\text{S}}$	$\binom{16\times5}{13\times6}$	85.7	73.9	11.8	99.4	90.8	8.6
Human Average		—	—	—	98.9	89.8	9.1





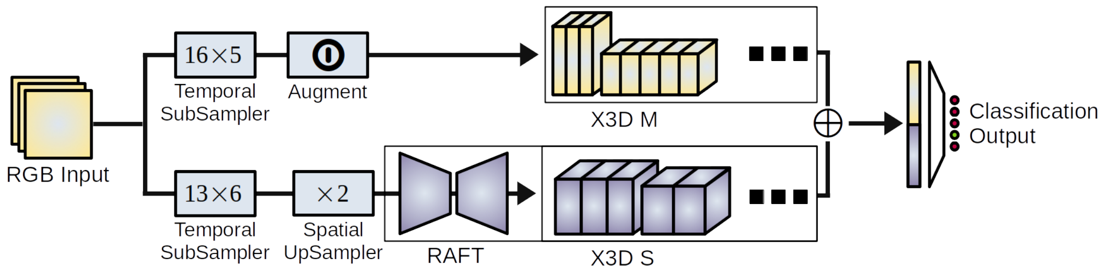


4.5 Two-stream strikes back

The results of our evaluation of architectures in Sec. 4.4 show the importance of explicit representation of dynamic information for strong performance on AFD. This result motivates us to revive the two-stream appearance plus optical flow design [26]. While state of the art when introduced, compared to more recent action recognition architectures, *e.g.* those evaluated in Sec. 4.4, it is no longer competitive [143]; so, we incorporate two-streams into the top performing architecture on RGB input with the smallest drop on AFD, *i.e.* X3D.

Design of a novel network architecture

The design of our novel architecture, ES2-X3D, is shown in Fig. 4.6. Initially, optical flow fields and RGB images are processed in separate 3D convolutional streams. Both streams are versions of the X3D architecture [28], where we use the S variant for optical flow and the M variant for RGB. These two architectures differ in their spatiotemporal resolution of the input and as a consequence the activation maps. X3D M uses larger spatial and temporal extents, whereas S has smaller spatial and temporal supports. Since the input flow field already represents the dynamic information with detail, the additional spatiotemporal extent is not needed; in contrast, the RGB stream without the optical flow benefits from larger spatial and temporal support, see Fig. 4.6 (bottom). We validate these choices below via ablation. The dimensionality of the layers is shown in Fig. 4.6. Following the separate parallel processing, late fusion is performed. We fuse the outputs via



Model	input	conv$_1$	res$_2$	res$_3$	res$_4$	res$_5$	conv$_5$
M	$16 \times 224 \times 224$	$16 \times 112 \times 112$	$16 \times 56 \times 56$	$16 \times 28 \times 28$	$16 \times 14 \times 14$	$16 \times 7 \times 7$	$16 \times 7 \times 7$
S	$13 \times 160 \times 160$	$13 \times 80 \times 80$	$13 \times 40 \times 40$	$13 \times 20 \times 20$	$13 \times 10 \times 10$	$13 \times 5 \times 5$	$13 \times 5 \times 5$

Figure 4.6: Our novel ES2-X3D operates with two parallel streams for processing of RGB (top stream) and optical flow (bottom stream). Late fusion via concatenation is followed by a fully connected and Softmax layer. The table shows the output sizes of each employed X3D architecture in the format T×H×W.

Table 4.2: Performance of X3D architecture configurations, XS, S and M for UCF5 and AFD5. Ablation is performed for both optical flow and RGB inputs.

UCF5

Architecture		Input $\binom{\text{RGB}}{-}$	Input $\binom{-}{\text{RAFT}}$	Input $\binom{\text{RGB}}{\text{RAFT}}$
XS	4×12	95.2	82.1	—
S	13×6	97.3	**89.5**	—
M	16×5	**98.8**	86.8	—
$\binom{\text{M}}{\text{S}}$	$\binom{16\times5}{13\times6}$	—	—	**99.4**

AFD5

Architecture		Input $\binom{\text{RGB}}{-}$	Input $\binom{-}{\text{RAFT}}$	Input $\binom{\text{RGB}}{\text{RAFT}}$
XS	4×12	57.0	77.3	—
S	13×6	69.8	**80.3**	—
M	16×5	**71.5**	78.9	—
$\binom{\text{M}}{\text{S}}$	$\binom{16\times5}{13\times6}$	—	—	**90.8**

simple concatenation of the outputs of the two streams. While more sophisticated fusion schemes might be considered, *e.g.* [172], we leave that for future work. Finally, a fully connected layer followed by a Softmax layer yields the classification output, which is used with a cross-entropy loss to train the network. For optical flow input, we use a state-of-the-art optical flow estimator, RAFT [69] pretrained on Sintel [62]. We do not further train the flow extractor and only use it in its inference mode. To capture fine grained motion and to adjust for the different spatial resolution RAFT was trained with, we add an up-sampling layer to the RGB images prior to RAFT. Data augmentation, as described in Sec. 4.4, is only used in the appearance stream, as indicated in Fig. 4.6. The training, validation and testing of the network follows the same procedure as the other networks; described in Sec. 4.4.

Empirical evaluation

E2S-X3D outperforms all competing architectures on AFD101, with the closest competitor (singlestream X3D) trailing by 15.7%; see Table 4.1. It also improves performance on UCF101, albeit slightly; 2.8% points gained on MViT (MViT has around 6× more parameters). To rule out an ensembling effect, we compared our network in its default configuration (RGB in one stream, optical flow in the other) with an alternative where it inputs RGB to both streams. We find a performance drop of 10.6% on UCF101 and 30.7% on AFD101, validating our hypothesis that explicit optical flow is crucial. Moreover, it is on par with human performance on both AFD5 and UCF5. These results document the advantage of explicit modeling of motion for action recognition. Table 4.2 shows results of ablations across various X3D configurations for both AFD5 and UCF5. It shows that for both input modalities, best performance is achieved when the S configuration is used for optical flow and the M configuration for the RGB stream. This set of experiments empirically validates our final ES2-X3D design.

4.6 Conclusions

We introduce an extension to a widely used action recognition dataset that disentangles static and dynamic video components. In particular, no single frame contains any *static* discriminatory information in our apperance free dataset (AFD); the action is only encoded in the temporal dimension. We show, by means of a psychophysical study with humans, that solving AFD is possible with a high degree of certainty (~ 90% Top1 Accuracy). This result is especially interesting as 11 current state-of-the-art action recognition architectures show much weaker performance as well as a steep drop in performance when comparing standard RGB to AFD input. These results lend to the interpretability of the evaluated architectures, by documenting their ability to exploit dynamic information. In particular, this shows that optical flow, or a similarly descriptive representation, is not an emergent property of *any* of the tested network architectures.

We propose a novel architecture incorporating the explicit computation of optical flow and use insights from recent action recognition research. This explict form of modeling *dynamics* allows our approach to outperform all competing methods, and compete with human level performance. Given the strong performance of our architecture in our evaluation, future work could consider training and investigation of performance on larger datasets [85, 113] as well as application to other tasks where non-synthetic data is available and dynamic information plays a crucial role, *e.g*. camouflaged animal detection [156]. Alternative fusion strategies are also a potential for improvement of our present architecture.

Privacy Attribute Obfuscation for Action Recognition 5

Contents

Appeared as
"Selective, Interpretable and Motion Consistent
Privacy Attribute Obfuscation for Action Recognition"
in CVPR 2024.

Concerns for the privacy of individuals captured in public imagery have led to privacy-preserving action recognition. Existing approaches often suffer from issues arising through obfuscation being applied globally and a lack of interpretability. Global obfuscation hides privacy sensitive regions, but also contextual regions important for action recognition. Lack of interpretability erodes trust in these new technologies. We highlight the limitations of current paradigms and propose a solution: Human selected privacy templates that yield *interpretability by design*, an obfuscation scheme that *selectively hides attributes* and also induces *temporal consistency*, which is important in action recognition. Our approach is architecture agnostic and directly modifies input imagery, while existing approaches generally require architecture training. Our approach offers more flexibility, as no training is required, and outperforms alternatives on three widely used datasets.

Keywords: *Action Recognition, Static and Dynamic Video Representation, Human Motion Perception*

https://f-ilic.github.io/SelectivePrivacyPreservation

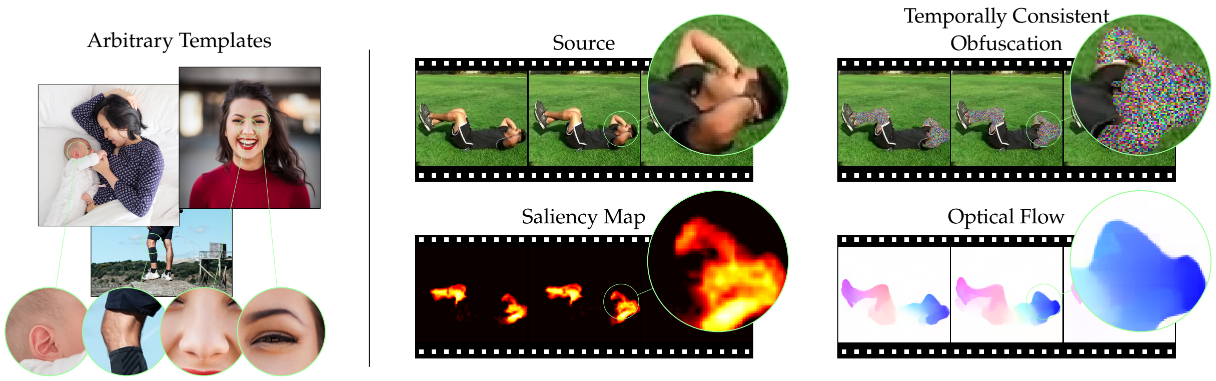


Figure 5.1: Our goal is to hide privacy attributes without action recognition performance dropping. Left: Arbitrary images can be used to specify an interpretable template library defined by privacy attributes. Middle: A salience map is generated from privacy templates; example illustrates use of templates for personal identification. Right: The source video is masked with noise as guided by salience and animated by source video optical flow. Salience makes masking selective to private regions, while preserving *scene context*; optical flow preserves *motion* – both of which are critical for action recognition. The obfuscated video can be input directly to arbitrary privacy and action recognition systems without retraining. Zoomed circles highlight details only for illustration.

5.1 Introduction

Advances in state-of-the-art computer vision and machine learning enable deployment of such systems in the public sphere. Accompanying these initiatives, concerns arise for the privacy of individuals that are captured in acquired imagery [182–185]. In particular, considerations arise regarding attributes that individuals want to keep confidential, yet that are revealed through visual information, even though they are not critical for the functioning of the deployed system, *e.g.* identity, age, gender and race. Video-based action recognition is an area of consideration as it has potential for widespread applications in surveillance and monitoring. These concerns have sparked interest in privacy preserving action recognition [186–189]. These approaches process input imagery to obscure privacy attributes while maintaining action recognition performance. Contemporary approaches typically apply their obfuscation across entire input video frames and improvements have been made within this paradigm. Notably, however, there are downsides to this paradigm, as follows.

Collateral damage. Global masking strategies indiscriminately obscure the entire image, impacting regions within the scene that may exhibit high correlations with actions, albeit lack relevance to privacy. For example, it is known that masking objects and scene context can impair action recognition performance [190], yet these are lost in global masking. Furthermore, global masking strategies do not allow for selective attribute obfuscation and generally hide all attributes at once, even when not all are of concern.

Loss of dynamic information. The large change in input modalities from global masking necessitates the retraining of the action recognition module or the design of custom modules (adversarial training), which adds to the challenge of practical deployment. Indeed, even if applied more locally, the obfuscation can compromise the motion of the actors, which also can be important in action recognition [26].

Lack of interpretability. Finally, given that state-of-the-art approaches are end-to-end trained with limited concern for interpretability, the exact nature of what is being masked and how it is achieved can remain unclear. Lack of interpretability is an important concern in privacy preservation, because its lack can compromise user trust [191, 192].

Contributions

We present an approach to privacy preserving action recognition that responds directly to current limitations in four ways, as illustrated in Fig. 5.1. (i) The approach is based on local detection and selective obfuscation of privacy sensitive regions. This selectivity maintains global context information that is crucial to action recognition, yet unimportant for privacy. (ii) The local processing avoids large modality shifts in the imagery and is independent of the action recognition module itself; therefore, it does not require algorithm retraining, which is sometimes infeasible. (iii) The masking preserves interframe motion; so, that information is available for action recognition. (iv) The privacy sensitive masking is interpretable by design, and allows inspection through the explicitly generated saliency maps.

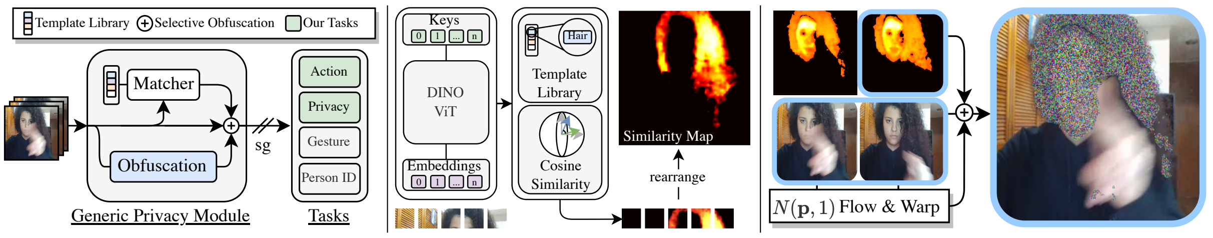
5.2 Related work

Privacy in machine learning. The need to protect privacy has garnered increased attention in the vision research community. Current models commonly consume large amounts of web data to learn generalizable representations [8, 193, 194], which inevitably invade personal information, such as identity and location. Moreover, in model deployment it also may be desirable to preserve privacy information. The concern for privacy is not limited to vision research, but extends across artificial intelligence, including natural language processing [195] and more general machine learning [196]. As these technologies find their way into broader society, privacy concerns must be considered.

Privacy preserving action recognition. We focus on video-based action recognition. Recent developments in this area have yielded systems capable of strong performance on challenging datasets; for review see, *e.g.* [197]. Similarly, applications, including those in privacy-sensitive scenarios (*e.g.* surveillance [198] and monitoring [199]), are being developed. To provide useful spatiotemporal signals for action recognition, video clips are mostly collected to capture actors with a great level of clarity throughout the actions, thus increasing the chance of privacy leakage. Moreover, such datasets are expanding rapidly, *e.g.* [106, 109].

In response to these concerns, research on recognizing actions while preserving private information has emerged in recent years. Early work concentrated on devising models that can work on low-resolution videos [200–202]; these methods often operate at the cost of sacrificing action recognition performance. Other work along these lines developed face-anonymization techniques to prevent models from yielding high accuracy on face recognition [203, 204]. Still, those approaches cannot easily extend to other attributes and are correspondingly limited. More recent work has focused on implicitly learning transformation functions to anonymize videos in a data-driven fashion [186, 188, 205, 206], and also extending such approaches to anomaly detection [207]. To reduce model complexity, a competing approach follows a simple, yet effective procedure [189]: Frame subtraction, followed by broadband-filtering to yield motion descriptors for action recognition, while suppressing privacy attributes. A notable limitation of that approach is the relative weakness of frame differences as motion descriptors compared to common alternatives, *e.g.* optical flow. Another common limitation to all existing work is the lack of flexibility to select arbitrary private information to hide, *i.e.* preserving privacy of only critical attributes, while maintaining the visibility of others, to avoid obfuscation of visual cues that are essential to action classification.

Template matching. Building correspondences between templates and target images via matching is a foundation for modern vision research, *e.g.* SfM [208], image correspondences [209], detection [210] and tracking [211]. Features used in matching have advanced rapidly from primarily hand-crafted (*e.g.* [210, 212]) to learning-based convolutional [7] or transformers [102]. Our solution is inspired by techniques seen in feature-based template matching. We apply their insights on matching templates to localize and selectively obfuscate privacy-sensitive regions. More specifically, we adopt DINO-ViT features [142] and compute similarities using the *keys* of the last attention layer from that architecture. Our use of DINO-ViT keys (rather than queries or values) is based on previous work finding them to perform best when applied to visual correspondence [213, 214].



(a) Overview **(b)** Matcher **(c)** Temporally Consistent Selective Obfuscation

Figure 5.2: Overview of Method. 5.2a We present a privacy module that builds atop three components: (i) a semantic *template library* that contains attributes to be hidden, (ii) a descriptor *matcher* to localize template features in videos to be obscured and, (iii) an *obfuscation* method that is sensitive with respect to motion present in the scene. 5.2b A semantic descriptor matcher based on DINO [142]-ViT [102] keys is used to determine privacy salient regions in a video based on the template library. In our case, regions of interest correspond to those that can identify a person; however, this component can be adapted for other privacy attributes through specification of different templates. The result is a saliency map. 5.2c The saliency map is used as a weight to apply noise to the regions. The noise, however, is not static, but is warped with optical flow with an initial noise pattern image, $N(\mathbf{p}, 1)$, for the purposes of preserving motion information in the source video. The similarity maps of all aggregated relevant privacy attributes are used to weigh the noise and apply it to the input image, obfuscating privacy sensitive information while not destroying the underlying temporal signal.

5.3 Technical approach

Our approach to privacy preserving action recognition is a stand alone module that operates by preprocessing video that subsequently is input to arbitrary action and privacy recognition algorithms, *i.e.* it is independent of the recognition algorithms and does not entail any retraining of those algorithms. Our method consists of three key components; see Fig. 5.2a: (i) A template library that covers privacy attributes to be preserved, (ii) a matcher that produces saliency maps between selected templates and images where privacy is to be preserved and (iii) an obfuscator that uses the saliency maps to hide privacy attributes of concern in a temporally consistent fashion to preserve motion in the video. The remainder of this section details each of these components.

Template library

To obfuscate privacy sensitive regions in images in an interpretable fashion, we need an explicit set of “attributes" to be hidden from the target video. Such a concept template library, T, can be built manually choosing image patches corresponding to features one wishes to obfuscate. In this paper, we concentrate on privacy attributes related to personal identification. Therfore, we use landmark anatomical features, corresponding to detailed facial landmarks and the hand (Fig. 5.3 left: *forehead, hair, eye, cheek, lips, hand*) as well as larger body parts (Fig. 5.3 right: *arm, torso, legs*). The choice to focus on preservation of human identity is motivated by the fact that all three of our evaluation datasets define their privacy attributes on attributes relating to person identity. Notably, however, such a template library easily can be extended depending on the particular task at hand. For example, attributes pertaining to location (e.g. street signs, distinctive scene objects) could guide development of a complimentary set of templates. In any case, given a template library, a user can selectively combine templates to obscure attributes of concern in a particular application.

Formally, we define each template τ_i as an element in the set $\mathsf{T} = \{\tau_1, \tau_2, ..., \tau_n\}$. Further, let $\tilde{\mathsf{T}} \subseteq \mathsf{T}$ be the subset of templates that the user selects from the library, T, for a particular subset of privacy attributes to be preserved for a given dataset or application. Our manual approach to template selection leads to concepts that are *interpretable by design*. In particular, we use two source images, shown in Fig. 5.3, one for small-scale features on the face, and one for large regions, taken from the IPN [215] and SBU [216] datasets. We choose these two datasets because of their complimentarity in template selection: IPN focuses on small scale features

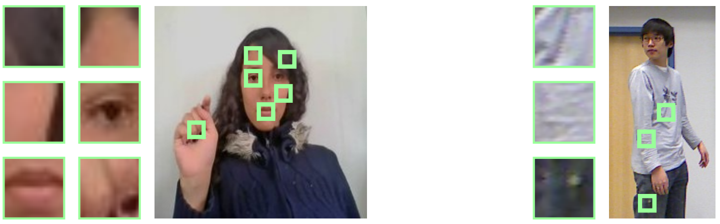


Figure 5.3: Template Library consisting of Patches Chosen from Anatomical Landmark Regions. These specific images are passed through a DINO-ViT feature extractor. The keys, corresponding to spatial locations of the highlighted patches, are chosen as the templates for matching to input images to obtain semantically similar regions for obfuscation.

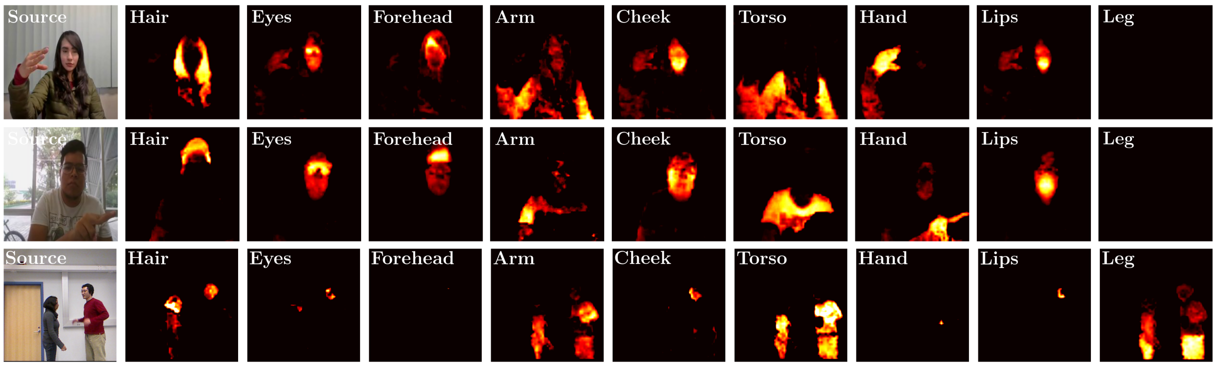


Figure 5.4: Saliency Maps for Descriptors in the Template Library. The manual selection of these templates allows for *interpretability* of the obfuscated parts of the image *by design*. The matched DINO-ViT features capture rich semantic information and allow for detailed spatial localization due to the nature of vision transformers. These saliency maps can then be combined for obfuscating any combination of templates depending on the task at hand.

(*e.g.* facial and hand), while SBU focuses on larger scale features (*e.g.* larger body parts), as detailed later in Sec. 5.4. The choice of the particular template images is not critical to the functioning of our approach, because we extract semantic features from the templates that are known to generalize well across images instances [142], as described next.

Matching: Local patch descriptor templates

Semantic features. Our requirements for good features to match between privacy templates in our library and frames in an action recognition video are straightforward: We require (i) semantically rich features that generalize across (in our case) different individuals and (ii) high spatial resolution to perform privacy obfuscation in a localized manner without destroying regions that are required for action recognition. We find that DINO-ViT features fit our needs very well [125, 142]: (i) They have been trained to yield high similarity for semantically related concepts (*e.g.* objects and their parts), while suppressing the similarity of unrelated matters (*e.g.* background). (ii) They support local patch feature extraction over high resolution images without losing global context. Elsewhere, DINO-ViT keys have proven especially useful in matching between templates and target images [213, 214]. Following these advances, we use last attention layer DINO-ViT keys computed from privacy templates (*e.g.* Fig. 5.3) to match against input images to compute privacy saliency maps.

Privacy saliency matching. To find privacy salient regions in an input video, a similarity map, S, is computed between each frame, I, in the video and the selected set of privacy templates, $\tilde{\mathsf{T}}$. The image is tiled with m patches, $I_j, j \in \{1, \dots, m\}$, and for each patch DINO-ViT keys, $\mathsf{K}(I_j)$, are extracted. These features are matched with DINO-ViT keys, $\mathsf{K}(\tau_i)$, for all selected templates, $\tau_i \in \tilde{\mathsf{T}}$. Clipped cosine similarity is used to establish the saliency of each image patch according to

$$s_j = \frac{1}{|\tilde{\mathsf{T}}|} \sum_{i=1}^{|\tilde{\mathsf{T}}|} \max\left(0, \frac{\langle \mathsf{K}(\tau_i), \mathsf{K}(I_j)\rangle}{\|\mathsf{K}(\tau_i)\|\|\mathsf{K}(I_j)\|}\right), \forall j \in [\![1, m]\!], \tag{5.1}$$

where $\langle \cdot, \cdot \rangle$ is inner product and $|\tilde{\mathsf{T}}|$ is the cardinality of $\tilde{\mathsf{T}}$, the set of selected privacy templates. Clipping is used because only positive values imply salience. This calculation is performed for every patch in every image of the video. Subsequently, the m saliency patches are reassembled in the shape of the original image to produce the final saliency map,

$$S = \mathcal{R}(s_1, ..., s_m; h, w), \tag{5.2}$$

with $\mathcal{R}$ a function that accepts image tiles, s_j, and reshapes them into their original image format of height h and width w. Note that a separate salience map, S, is calculated for each frame in a video of interest, I. The entire process of feature extraction, matching and the resulting saliency maps are summarized in Fig. 5.2b. Example saliency maps in Fig. 5.4 illustrate the ability of our approach to capture all of our templates in a variety of scenarios.

Temporally consistent obfuscation

Our goal is to mask privacy sensitive regions in images to obfuscate them. If we were to apply masks to every frame independently, then temporal information important to action recognition, *e.g.* motion of actors, would be destroyed as well. We empirically document the issue in Sec. 5.4. In response to this challenge, we follow previous work that presented a method for producing temporally consistent spatial noise across videos that supported action recognition, while obscuring appearance information in single frames [217]. While the previous work applied noise patterns uniformly across entire frames, we instead weight them by our privacy salience maps, S, to preserve as much context information as possible. The remainder of this subsection details our approach, with an outline shown in Fig. 5.2c.

Noise pattern initialization. Let $\mathbf{p} = (x, y)$ be image coordinates and t time. We initialize a noise image, $N(\mathbf{p}, 1)$, with the same dimensions as a frame from the input video (*i.e.* $h \times w \times 3$), with h height, w width and 3 the number of colour channels. The dataset mean, μ, and standard deviation, σ, are used to define a uniform distribution from which individual pixel intensities are drawn according to

$$N(\mathbf{p}, t = 1) \sim \mathcal{U}[\mu - \sigma, \mu + \sigma]. \tag{5.3}$$

Motion consistent noise. To create *motion consistent noise* the initial random frame, $N(\mathbf{p}, 1)$, is warped forward with flow fields derived from the original video, $I(\mathbf{p}, t)$, as extracted by an optical flow algorithm. Let $\mathbf{v}(\mathbf{p}, t) = (u(\mathbf{p}, t), v(\mathbf{p}, t))$ be the flow field that maps points, $\mathbf{p}$, in frame t to those in frame $t - 1$, with u and v the horizontal and vertical components of the flow. Then, a motion consistent noise sequence is generated as

$$N(\mathbf{p}, t) = N\big(\mathbf{p} + \mathbf{v}(\mathbf{p}, t), 1\big) \tag{5.4}$$

with $t \in \{2, ..., T\}$ and T the number of frames in the original input video.

Selective privacy obfuscation. The computed video sequence, N, shows no single frame appearance related to the original video as it is random noise; however, when viewed as a video it reveals the motion present in the original, *c.f.* [217]. Direct use of this synthesized video obscures privacy attributes; however, it also obscures other context information that could be of use in action recognition. So, instead we selectively apply N to every frame I in the video by using the privacy salience maps S, according to

$$O(\mathbf{p}, t) = I(\mathbf{p}, t) + \Big(S(\mathbf{p}, t) \times \big(N(\mathbf{p}, t) - I(\mathbf{p}, t)\big)\Big). \tag{5.5}$$

The resulting video, O, contains selectively obfuscated regions, built with interpretable templates by design and contains motion information that (in principle) does not differ from the original input video.

5.4 Empirical evaluation

The privacy obfuscated video, (5.5), serves directly as input to action and privacy recognition. No specialized development, training or other modification of the recognition algorithms nor adaptation of the privacy preservation system is necessary. Indeed, a major difference compared to competing state-of-the-art approaches (*e.g.* [189, 200, 205]) is that *we do not retrain* networks with our obfuscated data, while *they do retrain*. We exploit the fact that privacy attributes are generally *independent* from action recognition cues, which is enabled by our unique *selective obfuscation* approach.

Protocol

Datasets. We use three datasets commonly used for investigating action recognition and privacy, IPN [215], SBU [216] and KTH [218], which pose different challenges concerning both action and privacy.

IPN is a large-scale video-based hand gesture recognition dataset that consists of 50 actors performing 13 static or dynamic gestures, against three different backgrounds [215]. The performed gestures are used as the action labels and actor genders are used as the privacy labels.

SBU is a video dataset depicting eight human interactions. Each video is a video of actor-pairs, in the same laboratory environment. Action labels are derived from the actor interactions and the privacy attributes are the unique pairings of seven different actors resulting in 13 privacy labels.

KTH has 25 actors doing one of six actions [218]. Each action is performed four times in different environments. The six action classes are used for action recognition and 25 actor identities serve as privacy labels. In the originally proposed splits, videos of any one actor are fully contained in one set, since the intent of the KTH dataset was not concerned with actor identity recognition. For privacy identity, each actor must appear in both sets; so, the data is split such that each action is performed twice by each person in the training set, and once in the validation and test sets.

Metrics. Privacy recognition results are obtained following standard practice [189], averaging outputs over multiple frames (32 for IPN and KTH, and 16 for SBU) from the same video. Action recognition results are obtained by reporting the top-1 accuracy, as per convention [186, 187, 205]. It also is interesting to gauge the trade-off between action and privacy recognition. Let $0 \leq a \leq 1$ be normalized action recognition performance derived from dividing accuracy percentages by 100, and let p be defined analogously for privacy recognition. To quantify their trade-off we use a linear combination of a and $1 - p$ and define

$$f_\lambda(a, p) = (1 - \lambda)a + \lambda(1 - p), \tag{5.6}$$

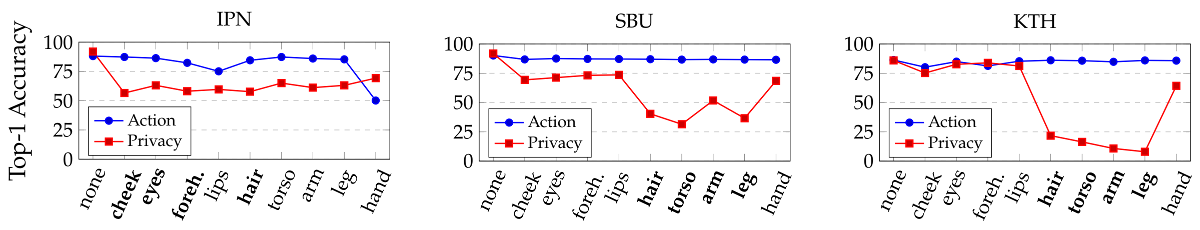


Figure 5.5: Obfuscation with a Single Attribute and the Impact on Performance. Attribute importance is dataset dependent. For example, notice how the 'Hand' template contributes to a large decrease in action recognition performance on IPN, as the action is determined soley by the hand, whereas on SBU it does not. Optimally **blue** is high and **red** is low. Corresponding qualitative examples of saliency maps for each individual template are shown in Fig. 5.4. **Bold** text along the abscissa of each plot indicates the templates used for the final results.

Table 5.1: Top-1 Accuracy for all Privacy and Action models on the original unmodified (source) videos, *i.e.* without privacy obfuscation. **Bold** and underline indicate first and second best, resp. Number of frames and temporal sampling rate indicated as $f \times r$.

Action Recognition on Source Datasets

Network	$f \times r$	IPN	KTH	SBU
C2D [87]	8×8	74.56	83.00	87.50
CSN [93]	32×2	91.73	88.33	90.91
E2S X3D L [217]	16×5	**95.15**	**95.33**	<u>91.11</u>
E2S X3D M [217]	16×5	89.38	92.33	86.36
E2S X3D S [217]	13×6	85.16	92.00	82.98
I3D [85]	8×8	83.15	89.67	82.95
MVIT[95]	16×4	88.00	90.00	**92.55**
R2+1D [88]	16×4	89.80	87.33	81.91
Slow [27]	8×8	85.76	89.00	88.64
SlowFast [27]	32×2	88.06	87.33	90.00
X3D L [28]	16×5	<u>94.41</u>	<u>94.00</u>	90.22
X3D M [28]	16×5	91.67	93.00	81.91
X3D S [28]	13×6	87.81	90.67	75.00
Average		88.05	90.15	86.31

Privacy Preservation for Source Datasets

Network	IPN	KTH	SBU
ResNet_{18} [168]	88.46	87.33	<u>90.43</u>
ResNet_{50} [168]	92.31	90.00	**96.81**
ResNet_{101} [168]	**94.23**	**94.00**	84.04
$\text{ViT}_{b/16}$[102]	**94.23**	**94.00**	79.79
$\text{ViT}_{b/32}$[102]	90.38	**94.00**	78.72
Average	91.92	91.87	85.96

with $0 \leq \lambda \leq 1$ weighing the relative importance of action recognition vs. privacy. This metric is suitable to create a linear ranking as a, and $(1-p)$ are optimally 1 ensuring $f_\lambda \in [0, 1]$. If privacy or action recognition are not equally important, then they can be weighted accordingly; *e.g.* if privacy is critical, then λ can be increased. In our main experiments we use $\lambda = 0.5$, abbreviated as $f_{0.5}$ in Tab. 5.2 and show an ablation over different values of λ in Fig. 5.7.

Competing obfuscation approaches. We briefly describe Naive Baseline approaches that solely rely on full-frame pixelation and bluring as well as actor masking that have been studied across the literature [187, 188]. We also highlight other State-of-The-Art approaches (SoTA) [189, 200, 205] that we compare against. Visual examples of the SoTA approaches are shown in Fig. 5.6; also shown are optical flows of consecutive frames for qualitative comparison.

<u>Mask Obfuscation</u>. All our datasets involve people performing actions and the privacy attributes are related to people. A naive way to preserve privacy in such videos is to completely mask out the actors. To do so, we use the YOLOv8 implementation [17] based on the original YOLO [16]. We report results based on the masked region filled with the mean intensity of the image covered by the mask.

<u>Pixelation</u>. Pixelation applies average pooling over regions of dimensions $x \times x$ on the input image sequence. We choose two scales of $x \in \{4, 16\}$ for the patch-size to pool. In the result tables these methods are abbreviated as $\text{Pix}_{x \times x}$.

<u>Blur</u>. Blur applies a Gaussian blur to the images that have been rescaled to 224×224 pixels. The parameters of the Gaussian are the kernel size, κ, and the standard deviation of the kernel, σ. We choose values for weak and strong blurs, $^{\kappa=13}_{\sigma=10}$ and $^{\kappa=21}_{\sigma=10}$, respectively, as consistent with other work on obfuscation methods [205].

<u>ELR</u> initially reduces frames to Extreme Low Resolution and subsequently applies a set of learned inverse super-resolution transforms to support action recognition [200].

<u>ALF</u> is an Adversarial Learning Framework for action recognition that takes into account a privacy budget [205].

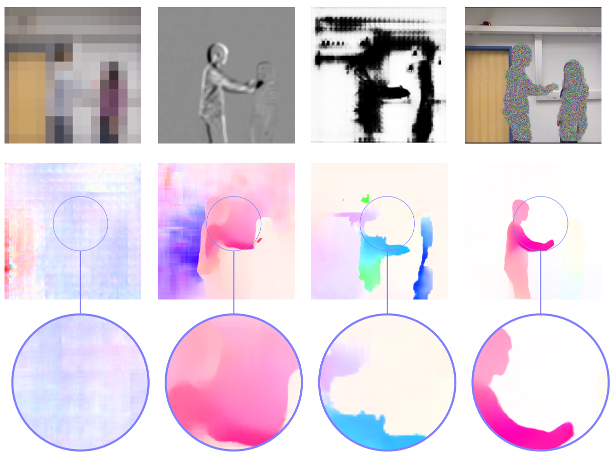


Figure 5.6: Optical flow from two consecutive frames from competing approaches. Only our approach retains dynamic information that supports high-quality optical flow recovery. Top row: Single frames as processed by comparison algorithms. Middle row: Full frame optical flow recovery. Bottom row: Zoomed optical flow details. Optical flow shown in Middlebury colour coding [219].

BDQ is a privacy-preserving encoder that sequentially Blurs, Differences and Quantizes frames. The blur and quantization parameters are learned to maximize action recognition while minimizing privacy recognition [189].

Notably, all the compared SoTA methods operate across entire frames (*i.e.* without selectivity) and have limited interpretability due to their learning-based obfuscation.

Implementation details. Training. No large scale dataset exists that allows for joint training of action and privacy classification on the same videos. Therefore, we pretrain our action and privacy networks on Kinetics400 [85] and Imagenet1k [8], respectively, and then finetune for specific datasets. We use the AdamW [47] optimizer with a learning rate of $3e^{-4}$. The networks are trained with a patience scheme of 100 epochs that monitors the loss on the validation set. For action recognition, videos are temporally uniformly subsampled according to the architecture (Tab. 5.1, '$f \times r$' column).

Note that we never perform any training on videos obfuscated by our approach. To implement warping, (5.4), we use pretrained RAFT to extract optical flow [70].

Privacy template selection. Our approach affords selective combination of a predefined set of privacy templates for a given dataset or application. For the experiments, we choose a subset of templates, $\tilde{\mathsf{T}}$, from our complete identity preserving template library, T, shown in Fig. 5.3, to optimize performance on a given dataset. Figure 5.5 shows template-wise performance based on both action recognition and privacy for each dataset. For SBU and KTH, action recognition is stable with respect to templates; however, the same four templates notably reduce privacy recognition. For consistency on IPN, we also select the four templates that most reduce privacy recognition, even while preserving action recognition; although, the distinction is less striking. This selection process results in templates $\tilde{\mathsf{T}} = \{torso, arm, leg, hair\}$ for SBU and KTH vs. $\tilde{\mathsf{T}} = \{cheek, eyes, forehead, hair\}$ for IPN. Our code yields more results on the overlap of individual saliency maps.

Runtime. The constant overhead to produce our obfuscated videos is ≈100ms/frame on a NVIDIA RTX4080 GPU.

Results

We evaluate our approach to privacy preserving action recognition on 13 different action recognition models and five different privacy attribute recognition models. Competing approaches evaluate on at most five

Table 5.2: Comparison between Ours vs. Naive Baseline (top) and SoTA (bottom) Approaches as Top-1 Accuracy for both Action and Privacy labels, as well as $f_{0.5}$ Introduced in Sec. 5.4. We show Ours and Naive Baseline results averaged across all recognition algorithms presented in Tab. 5.1. Competing SoTA approaches only present results on one specific algorithm for action recognition (I3D [85]) and privacy attribute detection (ResNet50 [168]) as those approaches are network specific. To showcase a fair comparison we also present results with the same single recognition algorithms indicated as "Ours †". First and second best results indicated by **bold** and underline, respectively. Relative performance delta (Δ) is indicated in **green** (improvement) and **orange** (tie) between best and second best method.

Results across Naive Baseline obfuscation methods

	IPN				KTH				SBU			
Method	↑Act	↓Priv	↑$f_{0.5}$	Δ	↑Act	↓Priv	↑$f_{0.5}$	Δ	↑Act	↓Priv	↑$f_{0.5}$	Δ
Original	88.05	91.92	0.48		90.15	91.87	0.49		86.31	85.96	0.50	
Masking	36.45	64.87	0.36		37.54	35.22	0.51		52.62	30.1	0.61	
Pixelate$_{4\times4}$	85.59	73.65	0.56		83.64	59.39	0.62		73.24	46.27	0.63	
Pixelate$_{16\times16}$	49.81	65.76	0.42		38.33	38.43	0.50		25.53	29.84	0.48	
Blur $^{\kappa=13}_{\sigma=10}$ (weak)	76.24	67.40	0.54		44.59	37.73	0.53		72.75	35.69	0.69	
Blur $^{\kappa=21}_{\sigma=10}$ (strong)	58.80	65.92	0.46		30.38	31.84	0.49		57.77	34.21	0.62	
Ours	87.11	51.93	**0.68**	+0.11	88.67	5.46	**0.92**	+0.29	86.74	13.19	**0.87**	+0.18

Results across SoTA obfuscation methods

	IPN				KTH				SBU			
Method	↑Act	↓Priv	↑$f_{0.5}$	Δ	↑Act	↓Priv	↑$f_{0.5}$	Δ	↑Act	↓Priv	↑$f_{0.5}$	Δ
BDQ [189]	81.00	59.00	0.61		91.11	7.15	0.92		84.04	34.18	0.75	
ALF [205]	76.00	65.00	0.56		85.89	19.27	0.83		82.00	48.00	0.67	
ELR[201] s=16	70.82	64.32	0.53		91.22	88.86	0.51		96.27	82.97	0.57	
ELR[201] s=32	52.96	63.29	0.45		85.57	82.56	0.52		92.42	64.89	0.64	
ELR[201] s=64	31.63	62.70	0.34		56.21	58.35	0.49		80.05	43.61	0.68	
Ours †	85.25	51.67	0.67	+0.06	89.44	4.31	**0.93**	+0.01	84.04	11.70	0.86	+0.11
Ours	87.11	51.93	**0.68**	+0.07	88.67	5.46	0.92	±0.00	86.74	13.19	**0.87**	+0.12

action models [189, 200, 205]. Table 5.1 shows results on the original videos from IPN [215], SBU [216] and KTH [218], *i.e.* without any privacy preserving processing. Table 5.2 shows results comparing our approach vs. Naive Baselines and SoTA. Comparison is given as the average across recognition algorithms presented in Table 5.1. Notably, while our approach compromises at most 1.48% action recognition accuracy compared to performance on the original video (88.67% vs. 90.15% on KTH), it *always* greatly improves privacy.

Naive Baselines. Compared to the baselines, our approach scores highest with respect to the $f_{0.5}$ metric across all three datasets: IPN (**+0.11**), KTH (**+0.29**), and SBU (**+0.18**). Naive Baselines show that action recognition correlates highly with privacy performance, as $f_{0.5}$ hovers around 0.5. The notable exception is Pix$_{4\times4}$ that performs well on IPN and KTH, and is the second best Naive Baseline on SBU. A notable benefit of all these approaches is that they do not require retraining of the recognition algorithms. Still, none have a selective ability to obfuscate only certain parts.

State of the art. A noteworthy difference between our vs. the alternative SoTA approaches is that they only work with the recognition algorithms on which they were trained. While this fact disadvantages our approach, as it lacks such retraining, it still outperforms the competing methods on all datasets: IPN (**+0.06**), KTH (**+0.01**), and SBU (**+0.11**).

It is valuable to consider obfuscation approaches in terms of the relative importance of action recognition vs. privacy preservation. Figure 5.7 compares results for the SoTA approaches as that trade-off is varied in terms of f_λ, (5.6). The sweep of λ shows that our obfuscation approach performs better than any competing approach across the entire range. Especially with increasing λ, the gap to other approaches widens as more emphasis is put on privacy preservation.

Importance of selective obfuscation. Our approach is unique in its ability to selectively obfuscate particular regions within a video based on specific privacy attributes. This ability yields two benefits: (i) Selectivity aids in interpretability. Each selected template results in a saliency map, (5.2), which allows for visual inspection

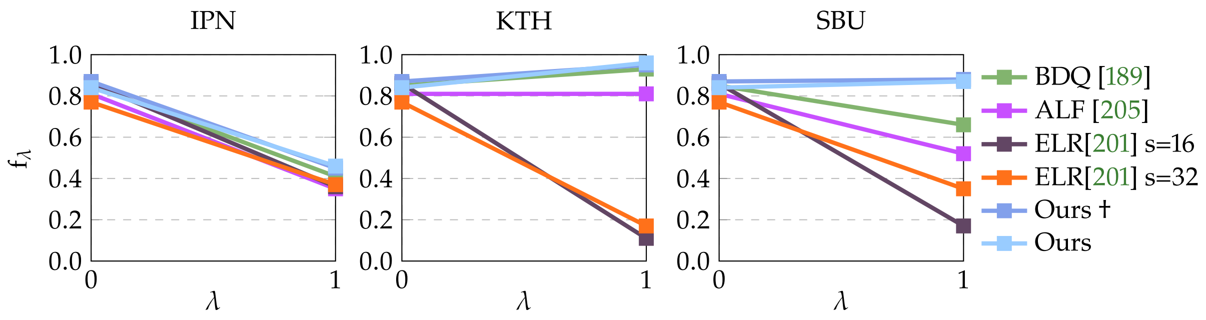


Figure 5.7: Performance across all datasets with varying λ; see (5.6). Values of λ closer to 0 weigh action recognition higher, whereas values closer to 1 increase the importance of privacy preservation.

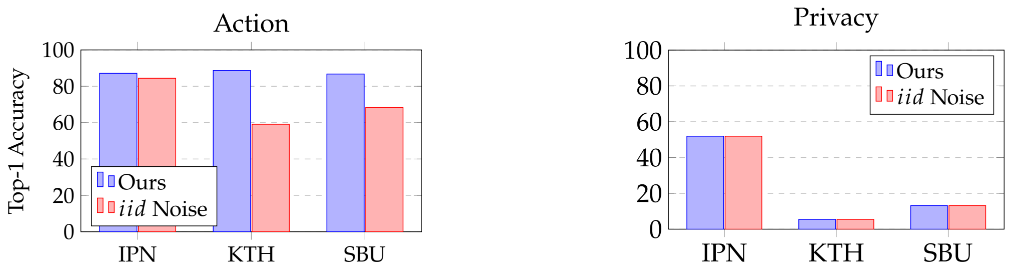


Figure 5.8: Comparing Temporally Consistent vs. *iid* Obfuscation for Action Recognition and Privacy Preservation. Action recognition performance decreases if noise is not temporally consistent.

of what information is being obscured. Figure 5.4 highlights this benefit as the heat maps reveal the degree of obfuscation to be applied on a template-by-template basis. (ii) Individual privacy templates can be chosen and combined for best performance; see Fig. 5.5. Depending on the dataset, different privacy templates differently impact the performance of privacy and action recognition. Action recognition on SBU and KTH is relatively robust to privacy template selection; however, privacy preservation is best when a subset of templates (*torso*, *leg*, *arm*, *hair*) is selected and the rest remain unobfuscated. This fact derives directly from the saliency calculation, (5.1): If multiple individual saliency maps have small values, then other strong responses are scaled down in the combined final saliency map. Subsequently, this effect leads to less noise being applied in our selective obfuscation, (5.5), which can deteriorate privacy preservation. In contrast, action recognition on IPN is compromised if the *hand* template is selected (as expected with a gesture centric dataset), which documents that simply obfuscating the entire person leads to inferior action performance compared to selectively applied obfuscation.

Importance of motion consistent noise. Action recognition is complicated as different datasets, and even individual actions within a dataset might require different recognition capabilities and can rely to different degrees on the modeling of motion [149]. This uncertainty on the role of motion in action recognition is exacerbated by the fact that different deep learning-based architectures are successful in capturing motion to varying degrees [220].

To see the importance of our proposed motion consistent noise for privacy preservation while maintaining good action recognition, we compare to another version of our pipeline that obfuscates video frames using independent, identically distributed (*iid*) noise; see Fig. 5.8 and project page for video results. For all datasets, action recognition performance decreases if the noise is *iid*. This result is sensible, because the *iid* noise does not capture the temporal dynamics of the source video. In contrast, privacy preservation is robust to noise in either case. There is less impact on IPN action recognition, as the critical hand motion is never obfuscated, which underlines the importance of selective masking. Further insight on why motion consistent noise supports better action can be had through consideration of Fig. 5.6, where the ability of such noise to support optical flow estimation is illustrated.

Limitations. Inevitably, there will be a trade-off between action recognition and privacy, as relevant information may be shared. In our approach, that trade-off could happen because our temporally consistent noise maintains dynamic information important for action recognition; however, it also might support motion based identification, *e.g.* gait recognition. Current protocols in privacy preserving action recognition do not consider motion-based identification. However, gait and related motion-based measurements are weak biometrics [221]; so, favouring action recognition may be apt. Moreover, if it becomes a concern, then it may be possible to apply motion perturbations that impede personal identification while maintaining action recognition.

5.5 Conclusion

Our work highlights that it is not necessary to train action recognition and privacy networks in an adversarial fashion for effective obfuscation of privacy attributes while maintaining strong action recognition performance. We show that a system based on local privacy templates, deep features that capture template semantics and selective noise obfuscation that is animated with source video motion can uphold privacy without hindering action recognition. Our approach is unique compared to alternative recent approaches in terms of interpretability and independence from particular action and privacy recognition algorithms. Our nine manually chosen templates, in combination with our proposed obfuscation technique outperforms other state-of-the art approaches across three different datasets.

Spatio-temporal Clustering for Object and Part Segmentation 6

Contents

Appeared as
"Representing Objects in Video as Space-Time Volumes
by Combining Top-Down and Bottom-Up Processes"
in WACV 2020.

As top-down based approaches of object recognition from video are getting more powerful, a structured way to combine them with bottom-up grouping processes becomes feasible. When done right, the resulting representation is able to describe objects and their decomposition into parts at appropriate spatio-temporal scales. We propose a method that uses a modern object detector to focus on salient structures in video, and a dense optical flow estimator to supplement feature extraction. From these structures we extract space-time volumes of interest (STVIs) by smoothing in spatio-temporal Gaussian Scale Space that guides bottom-up grouping. The resulting novel representation enables us to analyze and visualize the decomposition of an object into meaningful parts while preserving temporal object continuity. Our experimental validation is twofold. First, we achieve competitive results on a common video object segmentation benchmark. Second, we extend this benchmark with high quality object part annotations, on which we establish a strong baseline by showing that our method yields spatio-temporally meaningful object parts. Our new representation will support applications that require high-level space-time reasoning at the parts level.

Keywords: *Video Object Part Segmentation, Spatio-temporal Gaussian Scale Space, Action Tubes*

https://f-ilic.github.io/STVI

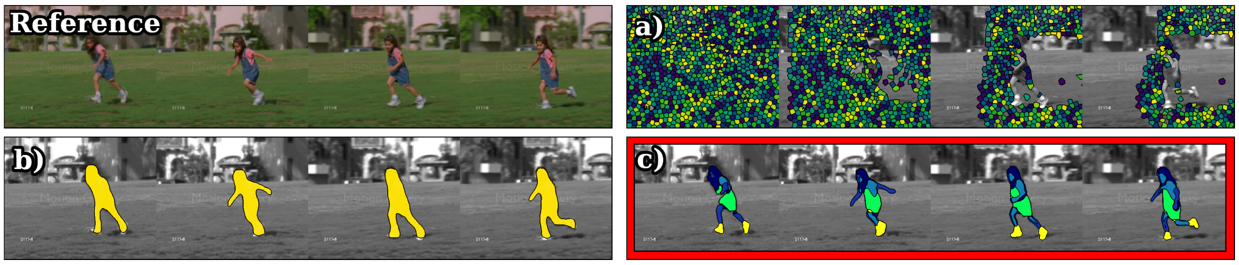


Figure 6.1: Our approach (c) bridges the conceptual gap between low-level video representations such as (a) TSPs [222] which tend to fall apart over time and do not model objects directly, and instance segmentation methods such as (b) Mask R-CNN [15] which lack a decomposition into parts, and temporal consistency. Our unsupervised approach is able to detect salient object parts based on motion and appearance. Sequence taken from SegTrack v2 [223].

6.1 Introduction

Recent research has achieved a lot of progress in Video Object Segmentation, both, with respect to the actual segmentation accuracy in each frame, but also with respect to the temporal consistency of the tracked object. A neglected sub-task is the separation and decomposition of generic objects into their salient components, where object parts can often be inferred by their difference in motion or appearance w.r.t. their surroundings.

The core novelty of our method is the principled *local* use of spatio-temporal Gaussian Scale Space. The enabling top-down methods include a generic instance segmentation framework to detect objects of interest, and an optical flow estimator to track the object and to extract motion information. These top-down anchors are used to guide the bottom-up grouping of features, extracted from appearance and motion in Gaussian Scale Space. This generates consistent regions, which correspond to meaningful object parts. See Fig. 6.1(c) as to how our approach models different parts of a running girl with different spatio-temporal tubes (same color means same tube). Compare this with Fig. 6.1(b), where each object instance is modeled by a singular object tube, and Fig. 6.1(a) where Temporal Superpixels (TSPs) provide no spatial grouping, and fall apart over time. Our approach is able to extract the parts of objects from video in a completely unsupervised manner, including object localization, tracking, and automatic selection of spatio-temporal scales. If required, manual scale selection allows for the explicit extraction of low or high frequency features from motion and appearance structures, see Fig. 6.2.

Such a general representation at the object and parts level could be highly desired in many applications of video understanding, e.g. to analyze motion patterns in sports, to classify actions, or to describe complex dynamics in cluttered scenes with many moving objects. To the best of our knowledge, there is no comparable approach that segments individual objects from video into their parts based on their spatio-temporal saliency.

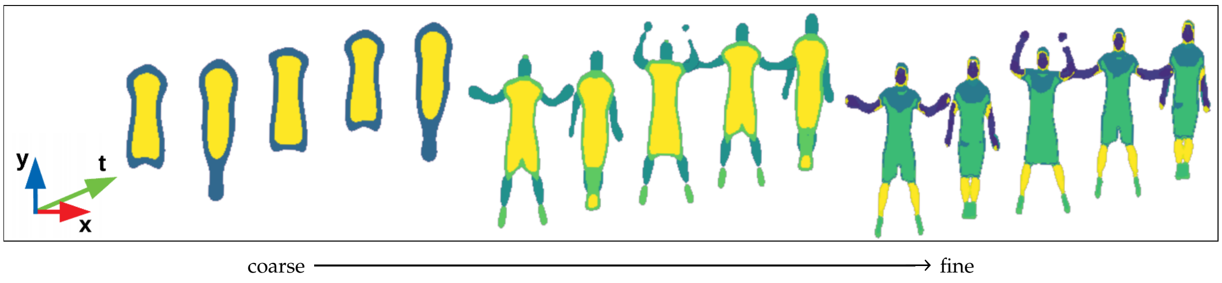


Figure 6.2: STVIs extracted at increasing spatial and temporal scales from a jumping-jack sequence. Note how finer scales capture more detail at the cost of temporal and spatial coherence, whereas coarser scales model more fundamental patterns. Two scales have been selected manually, the representation in the middle is achieved by automatic scale selection.

6.2 Related Work

Frame-based object proposals, e.g. the region proposal network (RPN) [14] provide bounding boxes that inevitably contain background. With the emergence of high quality datasets for instance segmentation [12, 223, 224], methods such as Mask R-CNN [15] improve on those approaches by not only providing tight bounding boxes but also the segmentation mask. However, these masks tend to be imprecise at the object boundaries and temporal correspondence is hard to obtain, especially when partial occlusions occur; therefore, simple temporal stacking of masks from individual frames does not suffice to represent objects in video reliably. Approaches that inherently work on video data, such as SiamMask [225] are able to generate temporally consistent object segmentations through utilizing a Siamese network structure for matching/tracking objects in adjacent frames [226, 227], only requiring the initial object bounding box. Such methods, however, still yield monolithic segmentation masks where the notion of individual salient object parts is not incorporated.

Many excellent solutions exist for object recognition, as well as two-stream architectures for video analysis that use appearance and motion information [20, 84]. These approaches achieve close to human performance in object and action recognition, and object tracking. The stunning performance, however, comes at a price: Besides the huge effort required to train these networks, what they learn is represented implicitly in the millions of parameters that are tuned and thus make it exceedingly difficult to build explicit reasoning atop of them.

One main benefit of our method is that it can recover salient object parts in an unsupervised manner via bottom-up grouping processes. Varquez et al. [228] propose a multiple hypothesis video segmentation algorithm that operates on superpixel flow, in which different superpixelations are created from which overlapping and persistent regions are matched. This indicates that these regions must form some sort of salient area, and are therefore spatio-temporally significant structures. Generally, superpixel/voxel algorithms, such as Simple Linear Iterative Clustering (SLIC) [229, 230], are used to abstract individual pixels into larger groups to reduce the complexity of visual tasks [231]. Temporally Consistent Superpixels (TCS) [232] and Temporal Superpixels (TSP) [222] are for instance such approaches that leverage superpixels created by clustering in the appearance-space. These are considererd low-level video representations as they extend image based superpixels to the video domain, modeling the entire video-volume. Therefore, neither objects nor their motion are modeled explicitly in such representations. While TCS and TSP consider appearance, Levinshtein et al. [233] work with optical flow and extract spatio-temporally closed regions (STC), which is an important step to ensure a temporally consistent representation. The aforementioned approaches are impressive in their own right, but they do not operate at the object level, and instead describe the whole scene with spatio-temporal structures. This gap between object representations and lower-level video representations exemplifies the current void that we aim to fill, and which we show in Fig. 6.1.

We deem the notion of scale, i.e. levels of detail at which an object is represented, an essential aspect of a good representation. We employ the well researched Spatio-temporal Gaussian Scale Space [234, 235] to achieve our goal of variable object scale. In essence, videos at different scales can be obtained by smoothing the video-volume with Gaussians of varying σ in space, and τ in time. Our method extends scale-space to the object level to estimate appropriate spatial and temporal scales at which an object is represented and decomposed.

Early work conducted by Gorelick et al. [236] demonstrated the feasibility of extracting motion-tracks of objects from clean video, and using them to classify human actions such as jumping, running, and walking. There are even object-based spatio-temporal representations [237] that, similar to our approach, use object detectors as "top-down anchors". Other approaches such as [238] use optical flow to anchor an object. However, the distinguishing feature and novelty of our approach is that we are able to represent salient objects in a decomposable, well-behaved, and scalable manner. Seguin et al. even note that *"most of the visual unpleasant artifacts"* [of their approach] *"are due to the use of superpixels"* ([237] caption Fig. 4); this observation holds true for most of the bottom-up driven grouping processes. We aim to avoid such artifacting, and to produce smooth space-time volumes which render them feasible for an incorporation in other applications.

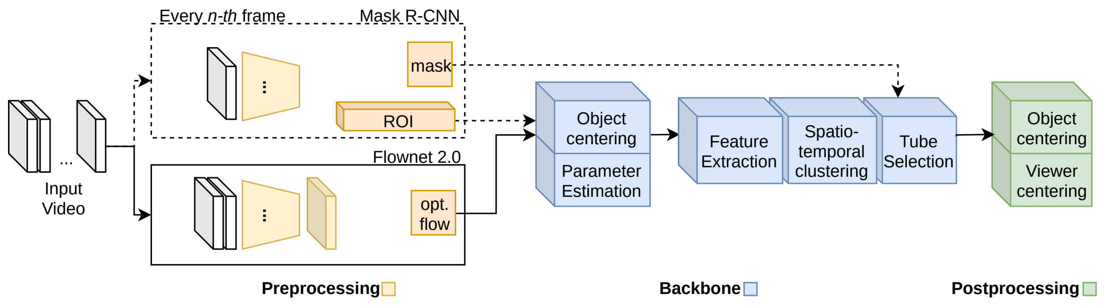


Figure 6.3: The pipeline of our approach is split into three distinct parts. The preprocessing uses an off-the-shelf object detector and a dense optical flow estimator; the per object ROI is used to crop appearance and flow around the current object of interest which results in an object centered subvolume. The backbone then generates spatio-temporally salient structures by smoothing in spatio-temporal scale space with parameters estimated by the object size and motion. Groups of coherent spatio-temporal features are then selected by utilizing mask predictions every *n-th* frame. This naturally tends to yield spatio-temporally significant regions in the context of the object. Hence, our method is able to handle multiple salient objects per video by running multiple times, with almost no overhead, as most of the computational complexity resides in the feature extraction, which can be shared among the objects.

6.3 Space-Time Volumes of Interest

Our approach uses temporally sparse instance segmentation masks and optical flow to build a representation of salient objects and their components, by locating and extracting spatio-temporally salient regions. These regions form structures that resemble *tubes* which model the spatial and temporal extent of objects and their components over time. Instance masks are used to focus on the objects of salience - akin to an attention mechanism, which we refer to as “anchoring”. These masks are provided by Mask R-CNN [15], trained on COCO [12]. Dense optical flow provided by FlowNet2.0 [239] is used to track the object until the next anchor mask is supplied. It is also used to cancel background motion and to support the generation of spatio-temporal tubes, which are extracted by clustering appearance and motion information on the anchored object. The pipeline of our proposed approach is shown in Fig. 6.3.

Preprocessing

Mask R-CNN is used in the first frame to detect objects of interest, and provides us with an initial Region of Interest (ROI). Because we evaluate our approach on datasets that only contain one annotated object, we restrict our approach to that one object per video, though it is capable of handling multiple objects of interest per video by running multiple times. Dense optical flow is computed for every frame pair to track the object, and will later be used to ensure temporal consistency. Every n frames another ROI is supplied to account for drift over time. By stacking the tracked ROIs we create a sub-volume aligned along each of the ROI's center points, and padded to fit to the dimensions of the largest ROI in the video. This yields an object-centered sub-volume around the object.

Because optical flow deals with apparent motion in video, one cannot infer the motion of objects in the scene, but only the relative motion between scene, camera and homogeneously moving regions. We are only interested in representing salient objects, and therefore cancel out the optical flow w.r.t. the object of interest. We do this by subtracting the background's optical flow:

$$\hat{F}_i = \left[F_i - mean(F_i \odot \neg M_i)\right]_{obj}, \tag{6.1}$$

where $\left[\,\cdot\,\right]_{obj}$ denotes the cropping of the volume to the sub-volume around the salient object, F_i the optical flow at frame i, M_i the binary segmentation mask at frame i, and $\odot$ the element-wise multiplication.

The preprocessing step results in two sub-volumes of identical extent (x,y, and t). The first one contains appearance whereas the second one contains the object flow, $\hat{F}_x$ and $\hat{F}_y$. These two sub-volumes are combined, such that each point in the sub-volume holds 8 values: position (x, y, t), color (L, a, b) in the Lab color-space, and flow $(\hat{F}_u, \hat{F}_v)$. With the preprocessing complete, cf. Fig. 6.3, we proceed with the extraction of salient tubes.

Backbone

Spatio-Temporal Smoothing The 8 dimensional feature-volume is processed at the object level, in a bottom-up fashion. To facilitate the creation of connected intra-frame structures of an object, we smooth spatially; to create structures that are temporally consistent and represent the same region even through slight changes in appearance or partial occlusions, we smooth temporally. The combination of these smoothing operations provides a means to weight and emphasize certain object- and motion-patterns in the sub-volume. Gaussian Scale Space with its extension to the spatio-temporal domain and the notation of a space-time scale space family $\mathscr{L}(\cdot)$ as proposed by Laptev and Lindeberg [77] is a cornerstone of our approach:

$$\mathscr{L}(\cdot;\sigma^2,\tau^2) = g(\cdot;\sigma^2,\tau^2) * f(\cdot), \tag{6.2}$$

where the input f is convolved with a spatio temporal Gaussian filter g:

$$g(x,y,t;\sigma^2,\tau^2) = \frac{1}{\sqrt{(2\pi)^3\sigma^4\tau^2}} \cdot e^{\frac{-(x^2+y^2)}{2\sigma^2} - \frac{t^2}{2\tau^2}}. \tag{6.3}$$

Here σ determines the width of the spatial kernel (same for x, y direction), and τ the width along the temporal dimension. With large σ, τ, only lower-frequency spatio-temporal structures remain; convolution with small σ, τ preserves the higher-frequency structures. While it is well-known that particular scales globally emphasize particular space-time structures, we explore the power of scale space *locally* to support the decomposition of objects into meaningful components.

Grouping spatio-temporally salient regions Following smoothing, we cluster the sub-volume containing appearance and optical flow information, encoded as 8-dimensional vectors, see Eq. 6.4. We apply SLIC clustering to the x,y,t sub-volume which contains the feature vector, $\phi(x,y,t)$, at each location:

$$\phi(x,y,t) = \begin{bmatrix} \alpha(x,y,t)^T \\ (L,a,b)^T \\ \beta(\hat{F}_u,\hat{F}_v)^T \end{bmatrix} \text{dimensionality: } 8\times 1 \enspace . \tag{6.4}$$

The scalar values α, β allow us to stretch and compress the sub-volume in which SLICs are generated. High values of α create more compact regions by prioritizing spatial proximity, whereas β is a trade-off between image intensities and flow components, with higher values prioritizing the flow component. We do not enforce spatial connectivity during clustering. Since we work in the 2D projection of 3D data, objects that are connected in 3D might not appear to be in 2D. This more general approach results in tubes that might appear disconnected in individual frames, but are connected somewhere in the temporal sequence. Later additional reasoning could be added for splitting or merging individual tubes.

Tube Selection The smoothed and clustered volume, $\mathbf{A}$, contains clustered spatio-temporal features that we call tubes (v_i), i.e. $\{v_1, ..., v_n\} \in \mathbf{A}$. In most cases background clutter is present, making it necessary to discard some tubes.

We do this by using instance masks from Mask R-CNN every n-th frame, referred to as $\mathbf{M}$. We compute the overlap of each tube with $\mathbf{M}$ and add it to the set of STVIs if more than a certain (volumetric) threshold, ω, is contained. This guarantees that the selected structures are temporally coherent, because they are contained within the instance masks over many frames, see Fig. 6.4.

In our experiments, we verified that tube selection is insensitive to the threshold parameter ω if it is chosen within $0.5 \le \omega \le 0.9$. This can be attributed to the short videos that we work with (usually around 80 frames),

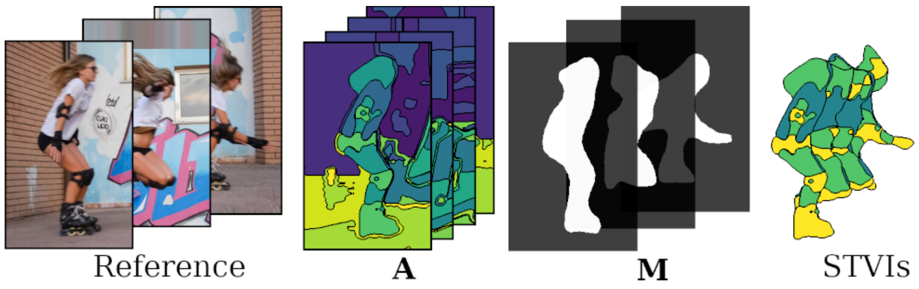


Figure 6.4: We select tubes from the clustered spatio-temporal features **A**, by intersecting individual tubes with the instance masks **M**, which are extracted every n frames. The remaining tubes (STVIs) model the object and its parts. Lower values of n improve the quality (more instance masks), but also incur a larger run-time penalty. In this work we choose n=20.

where single tubes are able to capture and track parts of objects through the whole sequence. In all our experiments reported, we set $\omega = 0.7$. As video length increases, ω needs to be decreased, to allow for tubes that approximate the object well through one section of the video but do not persist over the entirety of the video. Another possible way to deal with longer videos is to split them into shorter chunks. The chunks would then present good short-term volumes, which would need to be merged. We leave this for future work.

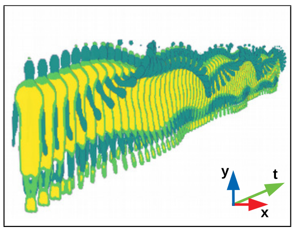
Parametrization The parameters to process a video can be estimated based on the appearance and motion of the object in the video alone. This means that our approach can be run without human interaction (0-Parameter), which is what we use for evaluating our method. However, the parameters can be fine tuned based on the application; e.g. reducing the temporal scale for high-frame-rate footage, or forcing coarser spatial scales. Ablation experiments in which we show the influence of each mentioned parameter are performed in Section 6.4.

- **The spatial scale** σ determines the amount of spatial smoothing that is applied, and is symmetric in the x, y direction. It is set proportial to the object's size, and defaults to the radial approximation, i.e., half of the diagonal of the average bounding box around the object.
- **The temporal scale** τ determines smoothing along the temporal axis and is proportional to the maximum object flow. By default we try to capture all the motion of the object, and therefore set the temporal scale according to the largest object motion. However, to be robust to outliers, the 95^{th} percentile of $||\hat{F}||$ is used.
- **The two clustering coefficients** α, β as introduced in Eq. 6.4 can be used to prioritize either the color- or the flow-components. By default they are set proportional to the spatial and temporal scales ($\alpha \propto \sigma$, $\beta \propto \tau$).
- **The maximum number of components** k determines into how many spatio-temporal tubes the object may be decomposed at most. For our experimental evaluation, k is set to 15, which is suitable for common object categories, as demonstrated in Section 6.4.

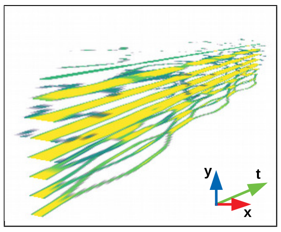
Postprocessing

STVIs are visualized as sliced tubes that, when taken together, model the space-time extent of an object. In contrast, other approaches [236, 240], show object- and motion-tracks as 3D meshes, only showing the hull of the space-time object segmentation. Our visualizations show the different parts of an object, and are especially useful because they allow for a slicing along any of its dimensions, emphasizing motion patterns by looking at different cross-sections, see Fig. 6.5.

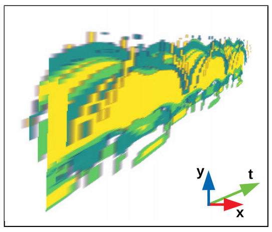
We can also switch between object-centered and viewer-centered perspective, see Fig. 6.6. Since we obtain individual object parts, such an object centered perspective can reveal relative motion w.r.t. the objects' centroid.



(a) t-slices

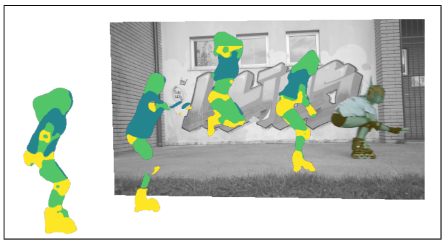


(b) y-slices



(c) x-slices

Figure 6.5: Interesting patterns revealed by slicing along any of the STVI's dimensions; e.g. the oscillating motion of the legs in (b).


(a) Viewer-centered

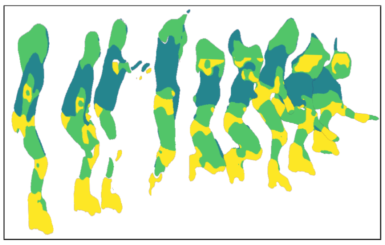

(b) Object-centered

Figure 6.6: The trajectory w.r.t. the observer is shown in (a). The relative motion of object parts w.r.t. the object centroid is shown in (b). This is useful *because* objects are represented as the sum of their parts, and not just as instance masks. For instance, notice the roller blades that move towards the center of the object axis during the jump.

6.4 Experiments & Results

Our experimental validation is split into two major parts:
1) Conventional Video Object Segmentation on the DAVIS [241] dataset, which provides excellent ground truth masks, using well established metrics. In addition, we provide a detailed ablation study.
2) An evaluation of the automatic detection of object parts, for which we extend DAVIS with high-quality, pixel accurate segmentations masks - DAVIS Parts. This new database it is to our knowledge the first Video Object Part Segmentation dataset with pixel accuracy. We evaluate all competing methods on this dataset with metrics that allow us to establish a first baseline for unsupervised object part detection and segmention.

Video Object Segmentation - Evaluation

The metrics we use to evaluate spatio-temporal object representations have to cover a wide range of characteristics. The obvious candidates are segmentation metrics. We also choose to include metrics that quantify interframe contour consistency:

- 3D Segmentation Accuracy **(ACC)**, and Undersegmentation Error **(UE)**, measuring the fraction of correctly identified pixels belonging to the object, and the fraction of pixels extending past the object boundary, respectively, introduced in [242]
- Normalized Temporal Extent **(TEX)** introduced in [222], measuring how long the generated tubes persist in time.
- The Jaccard index $\mathcal{J}$ [243] measuring region similarity and the F-measure $\mathcal{F}$ [241, 244] measuring contour accuracy.

Because lower-level representation approaches such as as TSP [222], TSC [232], SLIC [230] and STC [233] model the whole video volume and not only the objects, a slight adaptation is needed to select appropriate supervoxels which correspond to the objects. We employ the same tube selection process as described in Section 6.3, where we use an intersection criterion w.r.t. the Mask R-CNN segmentation to decide which individual supervoxels are selected for the final object representation. For the evaluation we consider two ways to select the tubes: using masks 1) in the first frame, and 2) every 20 frames, in the following denoted by

Table 6.1: Our method provides competitive results across the board on the DAVIS dataset, and sets a very strong baseline w.r.t. TEX. Our method also achieves the lowest Undersegmentation Error, at the cost of a slightly decreased Accuracy. Other methods that perform high w.r.t. Accuracy tend to generate instance masks that are coarse and blob like.

Method			ACC ↑	UE ↓	TEX ↑	$\mathcal{J}$ ↑	$\mathcal{F}$ ↑
MRCNN [15]			0.82	0.29	—	0.62	0.63
SiamMask [225]			**0.86**	0.28	—	**0.69**	0.67
SLICO [230]			0.30	0.10	0.69	0.27	0.24
		⚓	0.33	0.12	0.71	0.29	0.25
TSP [222]	k_{100}		0.32	0.23	0.62	0.27	0.28
	k_{100},	⚓	0.44	0.12	0.46	0.41	0.37
	k_{800}		0.38	0.05	0.46	0.36	0.37
	k_{800},	⚓	0.63	**0.03**	0.39	0.61	0.60
Ours			0.58	0.15	0.87	0.46	0.56
		⚓	0.78	**0.03**	**0.93**	0.66	**0.68**

the ⚓ symbol. We also include orthogonal - currently more popular - approaches that do not rely on explicit bottom-up grouping processes, to put the obtained results into perspective: Mask-RCNN [15] which yields framewise full object masks and bounding boxes and SiamMask [225] which yields temporally consistent object segmentations. Note that these methods are unaware of part decomposition.

Table 6.1 shows the evaluation of the methods on the DAVIS [241] benchmark dataset. We consider Mask R-CNN trained on COCO [12], a single frame segmentation baseline. TEX is left blank, because the generated frame segmentations are not temporally connected. SiamMask yields temporally connected segmentation masks, however, because it does not provide object components, but rather a single instance segmentation, the temporal extent of the single tube representing the whole object is rather meaningless, and also left blank. SLICO, the 0-Parameter SLIC is also tested as a simple baseline. TSPs, a more sophisticated bottom-up video representation are evaluated in variations k_{100} and k_{800} which refers to the free hyper parameter k which determines the number of components per frame.

Our results show that our approach establishes a very strong TEX score of 0.93 outperforming the next best approach by a considerable margin of 22 percentage points. TSPk_{800} and our method achieve a stunning UE of only 0.03, almost an order of magnitude better than Mask R-CNN and SiamMask. This large UE of Mask R-CNN - indicating that larger than necessary masks for objects are created - also tends to result in a higher ACC, as it is more likely to cover the object, if the mask is larger in the first place.

We group all 50 DAVIS videos into five “meta-classes”, i.e. Human, Animal, Bike-like, Car-like, and Miscellaneous. Class specific metrics are shown in Table 6.2, and provide insights into how the algorithms differ: The UE of the class “Bike-like” is considerably higher for Mask R-CNN and SiamMask than for any other class. This can be attributed to the data that both of these networks were trained on, which causes them to generate blob-like masks that segment whole bike wheels as discs. Our approach, on the other hand, does not have this tendency, but rather generates slightly too small regions, because of the non-edge preserving spatial and temporal smoothing that is applied.

Video Object Segmentation - Ablation Study

During ablation, Fig. 6.7, one parameter is subject to changes within a certain range, while the others are selected automatically, see Section 6.3.

We observe that increasing σ has the effect of decreasing ACC, decreasing TEX, and no change in UE. This is in line with our intuition that spatial low frequency tubes are worse at segmentation and tracking. Nonetheless, they enable us to represent complex objects at coarser scales with fewer and smoother tubes. Coarser temporal scales (larger τ) on the other hand correspond to slightly higher TEX, and considerably lower ACC, and

Table 6.2: The detailed per class performance of the tested approaches shows no large variation in the ranking of the different methods purely based on class. This stability shows that the newly curated DAVIS Parts is a useful dataset for the segmentation of object and their parts and is a suitable dataset for benchmarking approaches that concern themselves with video object segmentation.

Metric	Method		Human	Animal	Bike-like	Car-like	Misc.
ACC↑	MRCNN [15]		**0.85**	0.87	0.86	**0.84**	0.86
	SiamMask [225]		0.82	**0.90**	**0.88**	**0.84**	**0.89**
	TSP [222] k_{800}	⚓	0.61	0.68	0.50	0.66	0.65
	Ours	⚓	0.78	0.77	0.79	0.79	0.85
UE↓	MRCNN [15]		0.22	0.23	0.44	0.19	0.37
	SiamMask [225]		0.19	0.29	0.46	0.23	0.40
	TSP [222] k_{800}	⚓	**0.03**	0.04	0.05	**0.02**	**0.03**
	Ours	⚓	0.04	**0.03**	**0.01**	0.03	**0.03**
$\mathcal{J}\uparrow$	MRCNN [15]		0.65	0.65	0.50	0.59	0.62
	SiamMask [225]		**0.69**	**0.72**	0.64	**0.69**	0.64
	TSP [222] k_{800}	⚓	0.59	0.66	0.48	0.64	0.63
	Ours	⚓	**0.69**	0.68	**0.66**	0.64	**0.77**
$\mathcal{F}\uparrow$	MRCNN [15]		0.64	0.70	0.56	0.54	0.63
	SiamMask [225]		0.70	**0.72**	0.63	**0.62**	0.58
	TSP [222] k_{800}	⚓	0.56	0.64	0.54	0.61	0.66
	Ours	⚓	**0.71**	0.69	**0.68**	0.60	**0.72**
TEX↑	TSP [222] k_{800}	⚓	0.36	0.49	0.20	0.41	0.39
	Ours	⚓	**0.87**	**0.94**	**0.75**	**0.81**	**0.90**

no change in UE. As larger temporal smoothing is applied, fine spatial structures are lost and cannot be recovered.

The compactness of the tube, determined by α, see Eq. 6.4, does not play a significant role in the segmentation performance. It rather influences the qualitative appearance of the generated tubes; in most instances of the videos there is enough contrast in appearance or optical flow of the salient object. Changes w.r.t. β, on the other hand, do have a notable impact; larger values correspond to increasing ACC and decreasing TEX, as the generated tubes incorporate more optical flow information which can cause a rupture in a motion tube; e.g. a back and forth motion is split into two tubes (forward - backward) based on the flow.

k, which is responsible for the number of generated tubes, is more sensitive to change: As more tubes are used to represent the object, they can more closely model spatio-temporal structures, which leads to increasing ACC. This in turn means that smaller, higher frequency structures are modeled which are more likely to disappear over time again, leading to the observed decreasing TEX.

We observe a very constant and low UE. This is, as previously mentioned, inherent to our approach as it tends to select spatio-temporal tubes within object contours, rarely incorporating tubes that exceed object boundaries.

The class variation, Fig. 6.7 bottom row, shows expected results: more compact object classes tend to have a higher ACC because it is easier to approximate compact blobs than classes with deformable parts, such as humans and animals. These results are in line with our expectations towards a well-behaved spatio-temporally scalable representation which clearly shows the blob-like structures at coarse scales, with finer scales allowing for a more detailed, albeit shorter lived object part representation.

Object Parts - Dataset

The major benefit of our method is the automatic decomposition into object parts. To evaluate our claim that the decomposition indeed corresponds to meaningful parts, we extend DAVIS2016 with per-frame, object part annotations. The average number of components per object in our dataset is $\mathcal{M} = 7.15$. We name

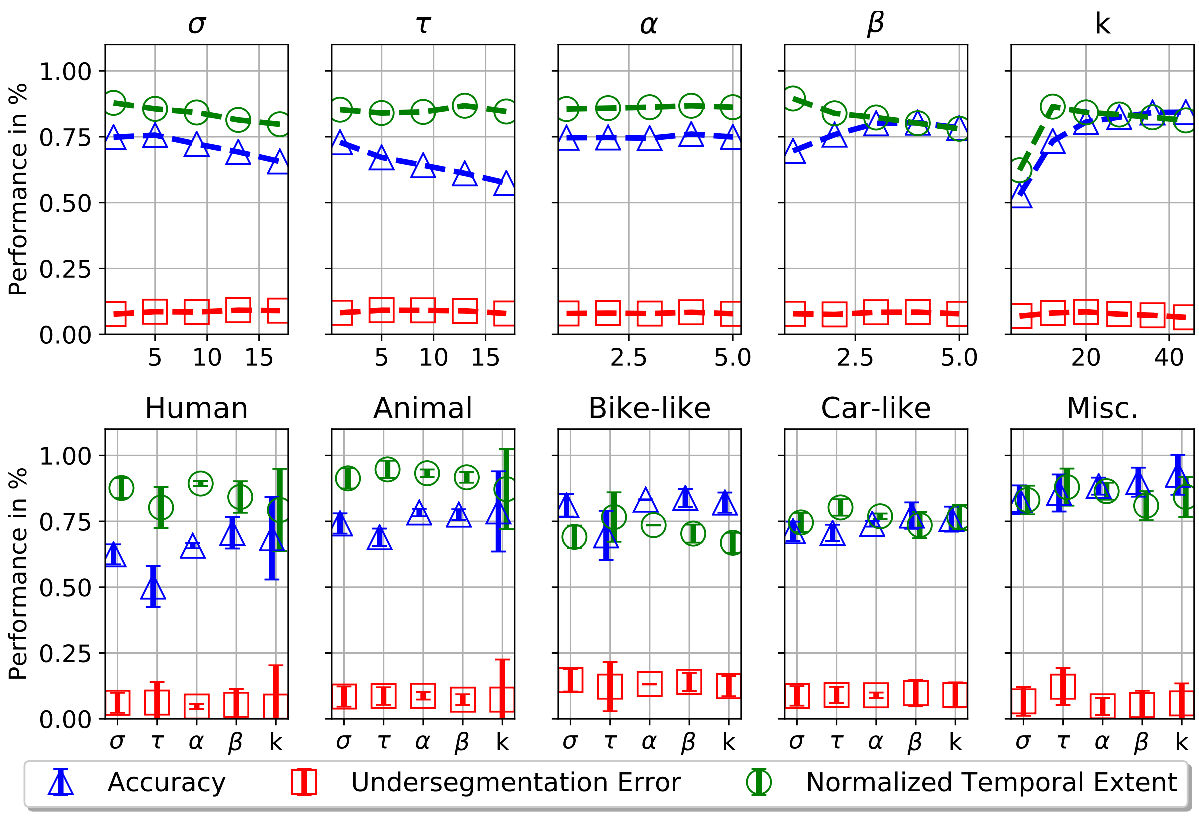


Figure 6.7: Parameter ablation: the top graphs show the ablation w.r.t. the individual parameter, whereas the bottom graphs show the sensitivity of the parameters w.r.t. all classes in DAVIS. σ and τ, parameters responsible for the scale at which features are extracted show a decrease in performance once a certain threshold is crossed at which objects are represented at too coarse scales, which tends to decrease the segmentation accuracy.

this new dataset DAVIS Parts. To our knowledge this is the first dataset providing pixel precision object part segmentation in video. Some example annotations for multiple frames are shown in Fig. 6.8b, and the annotated first frames of each video sequence are shown in Fig. 6.8a.

Object Parts - Evaluation In principle, we want to evaluate the $\mathcal{J}$ and $\mathcal{F}$ measures w.r.t. each individual part in the ground truth labels. To enable a fair comparison between the different methods, slight modifications to these measures are required. For each ground truth label we first need to determine which tubes model it. This is done by selecting those tubes that have a significant intersection with the ground truth tube (we set this significance to 30% overlap in our experiments, with this threshold being insensitive to change). This tube selection allows us to compute $\mathcal{J}$ and $\mathcal{F}$ measures, which we additionally normalize by the number of tubes modeling each particular part. This way we can meaningfully compare approaches which create different numbers of tubes. This score per part is averaged over all parts in the ground truth. We name these two modified measures $\mathcal{J}_P$ and $\mathcal{F}_P$.

Table 6.3: Baseline results on our dataset, no other approach has yet tackled the decomposition of arbitrary objects into their meaningful parts. The optimal results would yield $\mathcal{J}_P$=1, $\mathcal{F}_P$=1, $\mathcal{C}_P$=0.

Method			$\mathcal{J}_P \uparrow$	$\mathcal{F}_P \uparrow$	$\mathcal{C}_P \downarrow$	$\mathcal{M}$
MRCNN [15]			0.05	0.20	—	—
SiamMask [225]			0.02	0.17	—	—
SLICO [230]		⚓	0.04	0.18	0.57	16.2
TSP [222]	k_{100}	⚓	0.05	0.20	0.29	5.3
	k_{800}	⚓	0.06	0.16	0.49	78.8
Ours		⚓	**0.16**	**0.24**	**0.21**	3.7

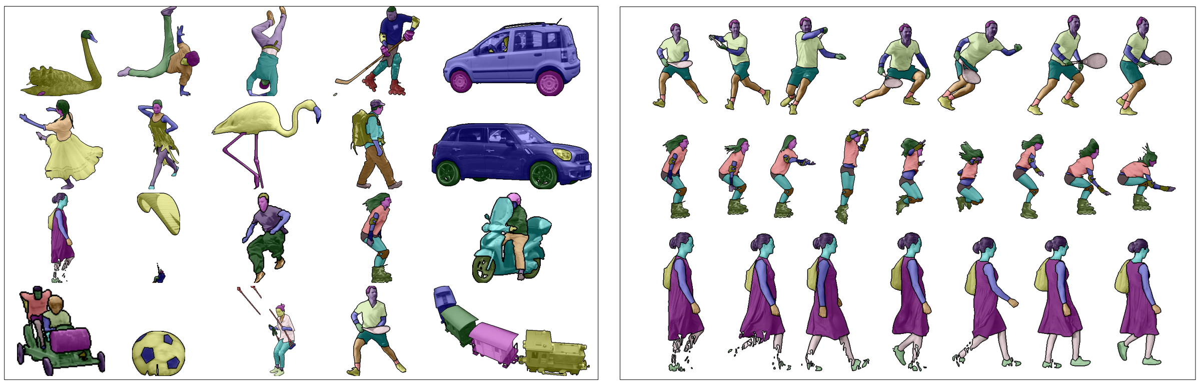

(a) First frame of very sequence **(b)** Samples from sequences

Figure 6.8: All first frame annotations for each sequence from DAVIS, showing the decomposition into its parts.

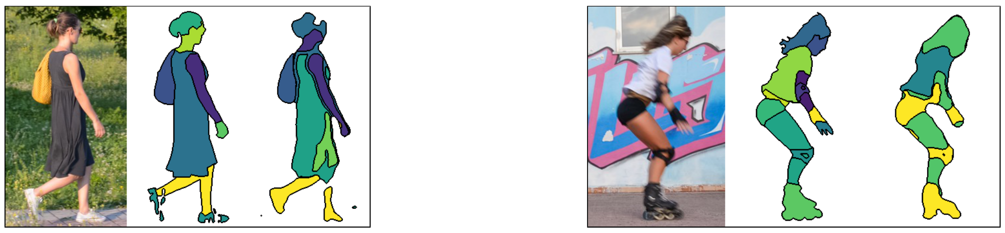


Figure 6.9: Qualitative comparison of part segmentation results, showing three stills from two videos, each with a crop of the reference frame, ground truth, and the result of our method.

Additionally, we introduce a measure for label inconsistency, $\mathcal{C}_P$. Note how in Fig. 6.9 (left), the dress is modeled by two tubes (light- and dark green). While this results in lower $\mathcal{J}_P$ and $\mathcal{F}_P$ values, we want to quantify the fact that the light- and dark green tubes model a single ground truth object part, throughout the whole video sequence, i.e. the dress always consists of the same tubes. $\mathcal{C}_P$ is calculated by accumulating the absolute deviation from the average distribution of labels at each frame. This score is furthermore normalized by the number of frames. To account for changes in the area of each ground truth label over time, the changes of distribution are measured w.r.t. the percentage of the area of a given label in each frame.

The results of our part based evaluation are shown in Table 6.3. STVIs outperform all competing approaches, most notably w.r.t. the Jaccard index $\mathcal{J}$. This performance is especially noteworthy considering that the object segmentation experiments, performed in Section 6.4 showed a much narrower performance gap between the methods. Normalizing the scores by the number of generated tubes is vital to compare the methods in a meaningful way. We therefore also list $\mathcal{M}$, the average number of tubes in Table 6.3. We can see that our method produces the lowest $\mathcal{M}$ of 3.7 tubes per video, whereas TSP k_{800} produces the most tubes - 78.8. This leads to the observation that large $\mathcal{M}$ tend to incur a high label inconsistency, $\mathcal{C}_P$. This is also in line with the observed results in the following object part ablation study.

Object Parts - Ablation Ablation experiments that measure performance of object part segmentation, Fig. 6.11, are performed in the in the same manner, and in the same parameter ranges as those for segmentation performance.

We observe very gradual changes to $\mathcal{J}_P$, and $\mathcal{F}_P$, across all varied parameters, similar to the ablation results w.r.t. segmentation performance. Changes to $\mathcal{C}_P$ are more sensitive, especially those w.r.t. k. This stems from the fact that increasing the number of generated tubes results in tubes that are more likely to be less stable over time, in a similar fashion to TSPs. We conclude that our proposed method is able to achieve a balance between whole object, and object part segmentation, as it shows to be robust in both areas.

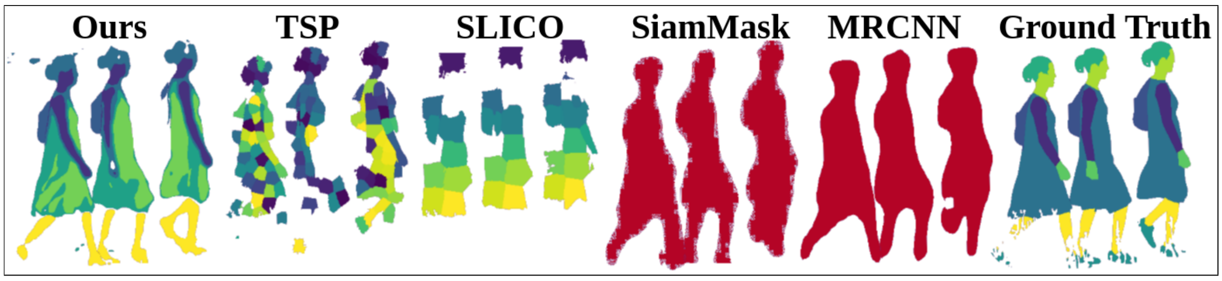


Figure 6.10: Qualitative comparison of part segmentation results. The top row shows three stills from different videos, each with the reference frame, ground truth, and the result of our method. The bottom row shows the results of different approaches, where the improved temporal consistency and object part detection of our method is evident.

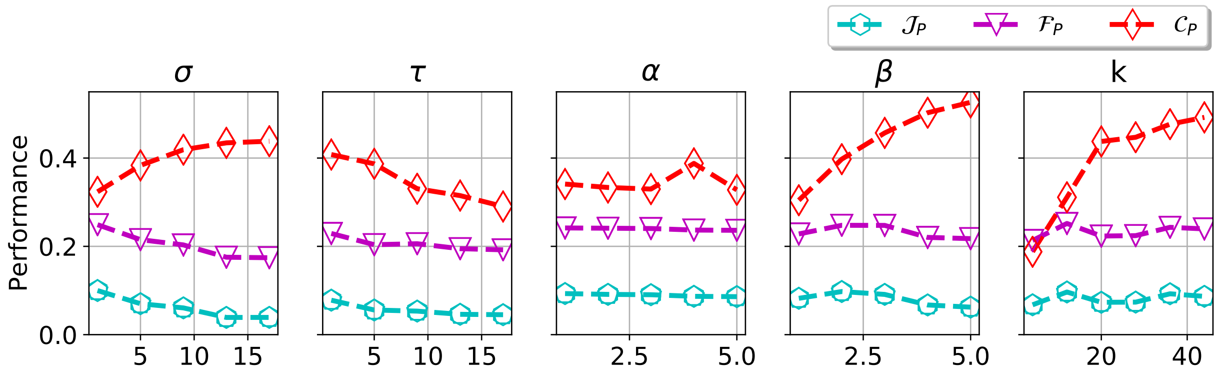


Figure 6.11: Ablation study results for object parts. We observe similar behaviour as in the object segmentation ablation study, where large σ, τ decrease the performance, whereas α, β are more stable, with large β only incurring larger label inconsistency, $\mathscr{C}_P$. Interestingly, k does not effect $\mathscr{J}_P, \mathscr{F}_P$, as drastically as shown in the object segmentation ablation study.

Table 6.4: Our approach offers a spatially and temporally scalable method for extracting a decomposable object representation unlike monolithic approaches that yield segmentation masks, or more traditional video representations that do not focus on salient objects. Furthermore, no existing object representation offers a temporal scale which supports the creation of spatio-temporal tubes at certain frequencies.

Method	Object Representation	Temporally Consistent	Decomposable	Variable Scale Spatially	Variable Scale Temporally
MRCNN [15]	✓	✗	✗	✗	✗
SiamMask [225]	✓	✓	✗	✗	✗
SLICO [230]	✗	✓	✓	✗	✗
TSP [222]	✗	✓	✓	✓	✗
Ours	✓	✓	✓	✓	✓

6.5 Conclusion

A qualitative comparison of our results in both segmentation performance and part based segmentation is shown in Fig. 6.10. We can see that both instance segmentation networks tend to yield masks that are slightly too large, blob-like, do not have crisp object borders, and do not provide object parts. In contrast, our approach yields natural looking object boundaries, and a separation into meaningful parts, where "meaningfulness" is context dependent and established through delineation in appearance and/or motion. Table 6.4 summarizes key properties of all tested approaches.

We have presented a general method to decompose objects into meaningful spatio-temporal parts using relatively simple features. Our method bridges the gap between bottom-up processes used to build spatio-temporally consistent tubes, and video object segmentation networks that yield instance segmentation masks. This novel object representation at different spatio-temporal scales, unlike other approaches, yields temporally coherent object components delineated by motion and/or appearance. We have evaluated our approach on a common Video Object Segmentation dataset where we achieve competitive results. We furthermore have extended this benchmark with high-quality, pixel level annotated individual object parts, where we set a strong baseline with our approach. We hope that our work of unsupervised detection of object parts in combination with the dataset will inspire development of new methods that are able to automatically detect coherent object parts. In general, our part-based decomposition can enable interesting applications in the realm of reasoning about objects, their interactions, or representations tailored to specific tasks, such as feature extraction from spatio-temporal tubes for action recognition and action localization tasks.

Outlook

Outlook 7

Contents

7.1 Considerations

With the widespread adoption of AI and Machine Learning models, ethical considerations become critical. These models are integrated into a diverse array of applications, ranging from self-driving cars to medical diagnostics. More significantly, they are already becoming embedded into everyday tools such as smartphones, laptops and other computing devices, affecting people in various situations and gradually influencing human behavior as new technologies often do. Therefore, it is crucial to address the ethical implications thoroughly to ensure responsible development and deployment of such tools.

Biases in Data One of the most prominent ethical concerns is the potential for biases in the data used to train AI models. Bias can be introduced at multiple stages, including data collection, labeling, and preprocessing. For instance, if the data collection process favors a particular demographic, the resulting model may exhibit biased behavior. Similarly, biases can emerge during data labeling if the labelers have unconscious prejudices or if the labeling guidelines are flawed. Preprocessing techniques can also inadvertently introduce bias by emphasizing certain patterns over others. Since AI models learn from the data provided to them, these biases can become ingrained, leading to unfair or discriminatory outcomes. Addressing these biases is essential to develop fair and trustworthy AI systems. This is particularly well documented for the current state of Generative models such as Midjourney [245]: *"In the case of Midjourney, [decisions are made] by a handful of esoteric nerds, and by a 65-year old mechanical engineer living in Southeastern Wisconsin."* verbatim from [246].

Explainability & Regulatory Concerns The explainability of AI models is another critical concern, particularly in high-stakes applications. For example, in the context of self-driving cars, it is crucial to understand the reasoning behind the car's decisions to ensure safety and accountability. Similarly, in medical diagnostics, healthcare professionals and patients need to comprehend the rationale behind decisions to trust and validate the diagnosis. Transparency in AI decision-making processes is not only important for regulatory compliance but also for user trust and acceptance. Users should be able to understand why a model made a particular decision to feel confident in its reliability and fairness. A lot of work has been done in this area, and the field of Explainable AI (XAI) is rapidly evolving to address these concerns.

Privacy Concerns The data used to train AI models can often include sensitive information, raising significant privacy concerns. This data, if not handled correctly, can potentially identify individuals, leading to privacy violations. Ensuring that data is anonymized before being used for training is a critical step to protect individual privacy. One of the contributions of this thesis is the development of privacy-preserving techniques for videos. These techniques allow for the creation of videos that are still useful for action recognition but obfuscated to remove sensitive attributes. This approach demonstrates how privacy can be safeguarded while maintaining the utility of the data for AI applications. It remains to be seen if such a method is scalable or if other solutions will prove superior.

7.2 Directions for Future Research

The future of action recognition and video understanding looks promising. Many tasks previously considered difficult and computationally expensive are now within reach due to the rapid advancements in AI and deep learning. The development of more efficient models, training techniques and other related knowledge, has enabled the processing of large-scale video data with improved accuracy and speed. These models can capture complex spatio-temporal patterns in videos, leading to better action recognition performance. The integration of these models with privacy-preserving techniques can further enhance their utility in real-world applications. As these models become more sophisticated, they will be able to handle a wider range of tasks. This necessitates understanding the decision making process of these models. Explainability and interpretability are crucial aspects of AI systems that need to be further explored. Even if the models are accurate, if their decisions remain opaque they may not be trusted or accepted by users. Even more so, debugging and understanding the limitations of such opaque models can then only be done through individual failure cases, and is difficult to associate entire failure classes with the model's decision making process.

I therefore believe that endeavors which focus on modularization are crucial because they enable debugging of individual components of the model, inspecting the decision making process of the model as a whole. This is particularly important in the context of safety-critical applications, where the consequences of model failure can be severe.

In more concrete terms, I believe that our introduced Appearance Free Data represents a minimal working example of a dataset that can be used to inspect and debug the temporal reasoning of a model. The dataset is simple enough to be understood by humans, and requires reasoning over varying temporal lengths, as well as the ability to create dense point estimates from sparse spatial data.

Appendix

Appendix A

Contents

A.1 Appendix to Chapter 2

Additional Plots of Decision Boundaries in 3D

Chapter 2, Section 2 describes the decision boundaries of a toy classification problem in 3D. The decision boundary is shown in Fig. A.1 and Fig. A.2. In the case of this small networks the weights can also be directly interpreted as a transformation matrix that maps $\mathbb{R}^2 \rightarrow \mathbb{R}^3$. As such we can visualize the activation of each neuron as a plane in 3D (with an applied ReLU non-linearity).

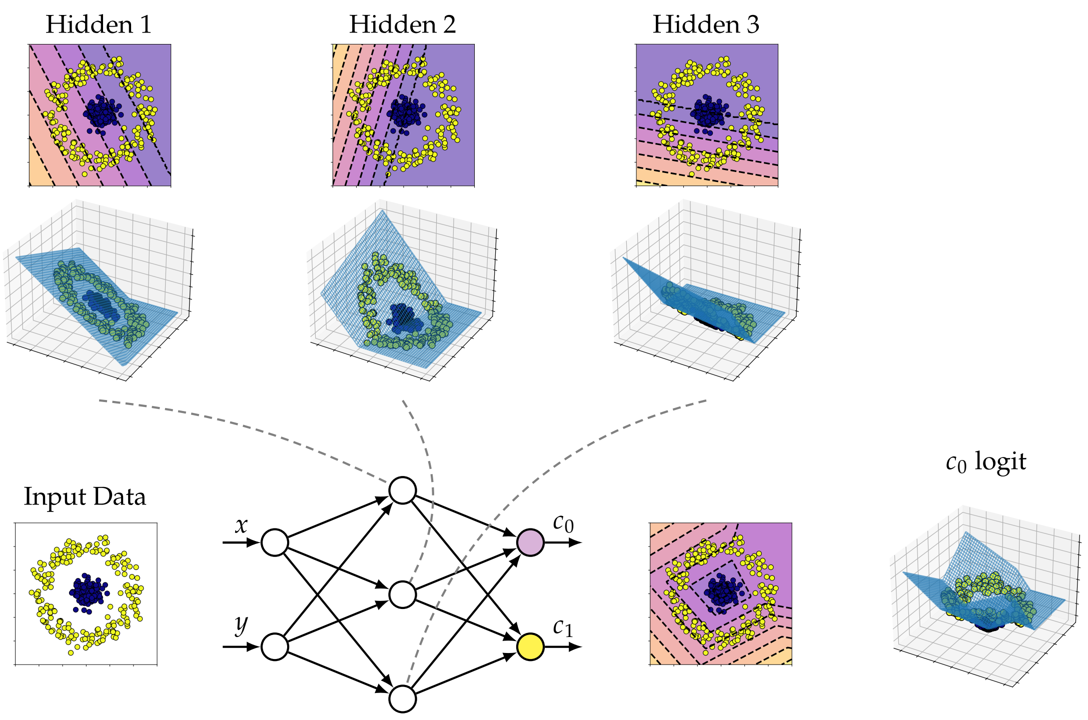


Figure A.1: Appendix to Fig. 2.18. The 2D and 3D visualization of the decision boundaries is shown. We can see how the hidden layer neurons that are each able to separate the input data into a flat region and a region with a slope, can be combined such that a decision boundary that appears to be a square is formed.

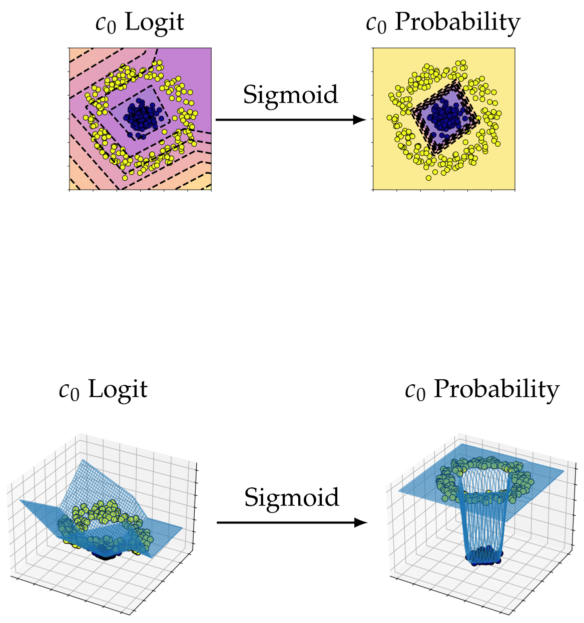


Figure A.2: The decision boundary of the toy classification problem in 3D. The decision boundary is a hyperplane that separates the two classes.

A.2 Appendix to Chapter 4

Additional Behavioural Results of Appearance Free Action Videos

Chapter 4, Section 4.3 describes the performance and the results gathered on AFD5 data with human participants; there, we presented the aggregation and summary of all datapoints. Individual anonymized participant data shows that performance across all participants is similar, see Fig. A.3 (bottom right). From the timing plots (top left), we can see that participants' time-to-choose (*ttc*) decreases from the train to the test set, indicating that the 'training' is beneficial even for humans. The *ttc* per user, Fig. A.3 (bottom left), shows that 3 users in particular show more hesitation / higher *ttc* for samples that are finally classified incorrectly. An increase in *ttc* for wrongly classified samples is seen across all classes and all participants (top right). The questionnaire every participant of the evaluation was required to fill out is shown in Fig. A.4.

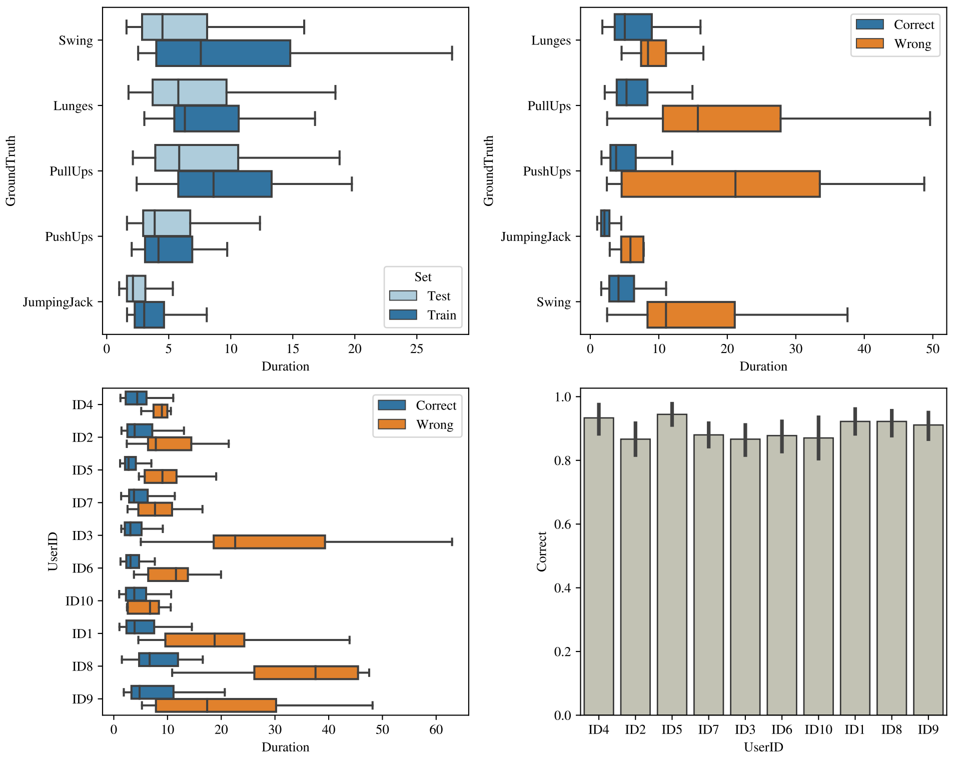
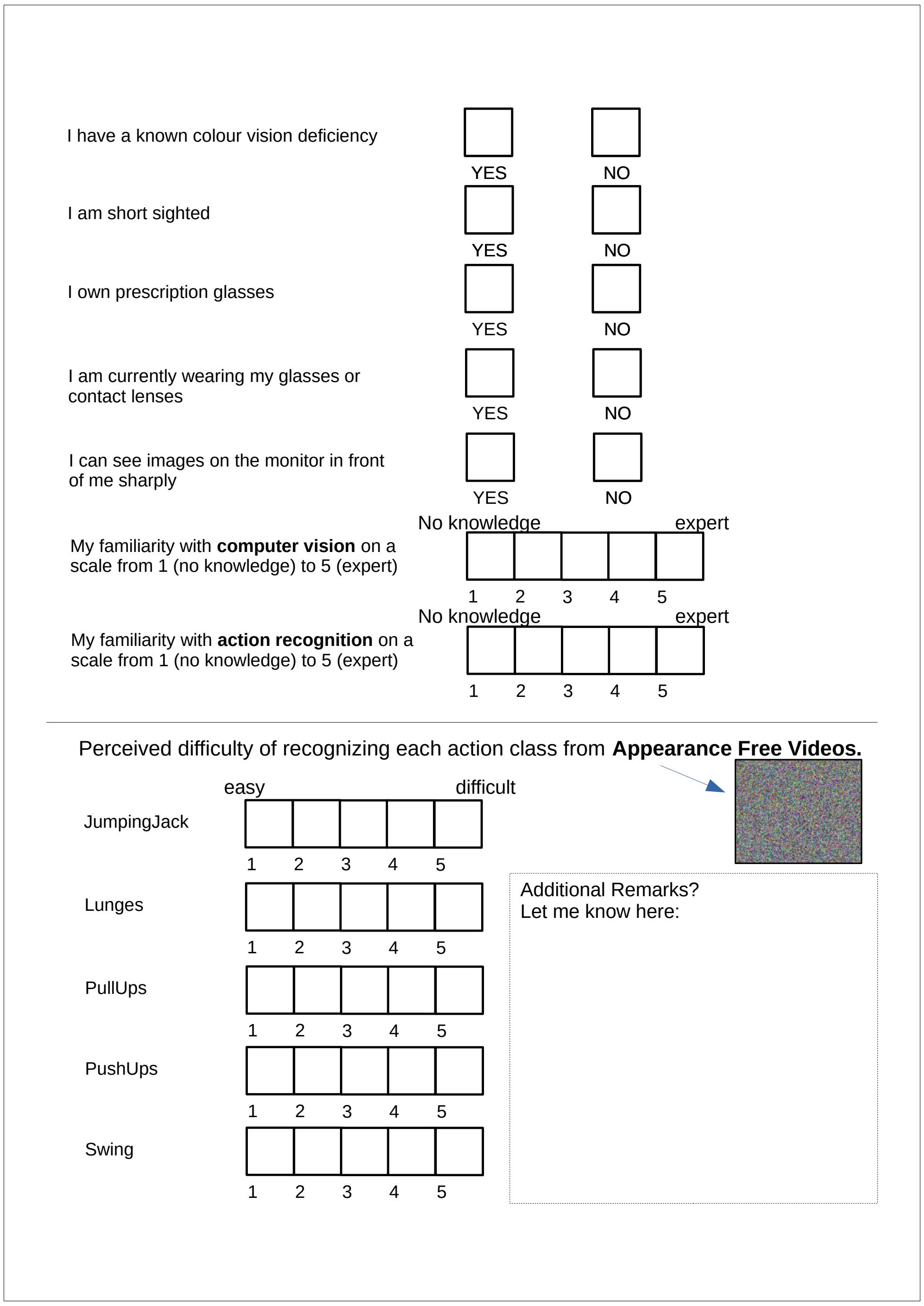


Figure A.3: Additional results of the user study on AFD videos. The figure shows a detailed breakdown of the results obtained from the 10 participants.



I have a known colour vision deficiency	YES	NO
I am short sighted	YES	NO
I own prescription glasses	YES	NO
I am currently wearing my glasses or contact lenses	YES	NO
I can see images on the monitor in front of me sharply	YES	NO

	No knowledge				expert
My familiarity with **computer vision** on a scale from 1 (no knowledge) to 5 (expert)	1	2	3	4	5
My familiarity with **action recognition** on a scale from 1 (no knowledge) to 5 (expert)	1	2	3	4	5

Perceived difficulty of recognizing each action class from **Appearance Free Videos.**

	easy				difficult
JumpingJack	1	2	3	4	5
Lunges	1	2	3	4	5
PullUps	1	2	3	4	5
PushUps	1	2	3	4	5
Swing	1	2	3	4	5

Additional Remarks?
Let me know here:

Figure A.4: Template questionnaire that was filled out by every participant.

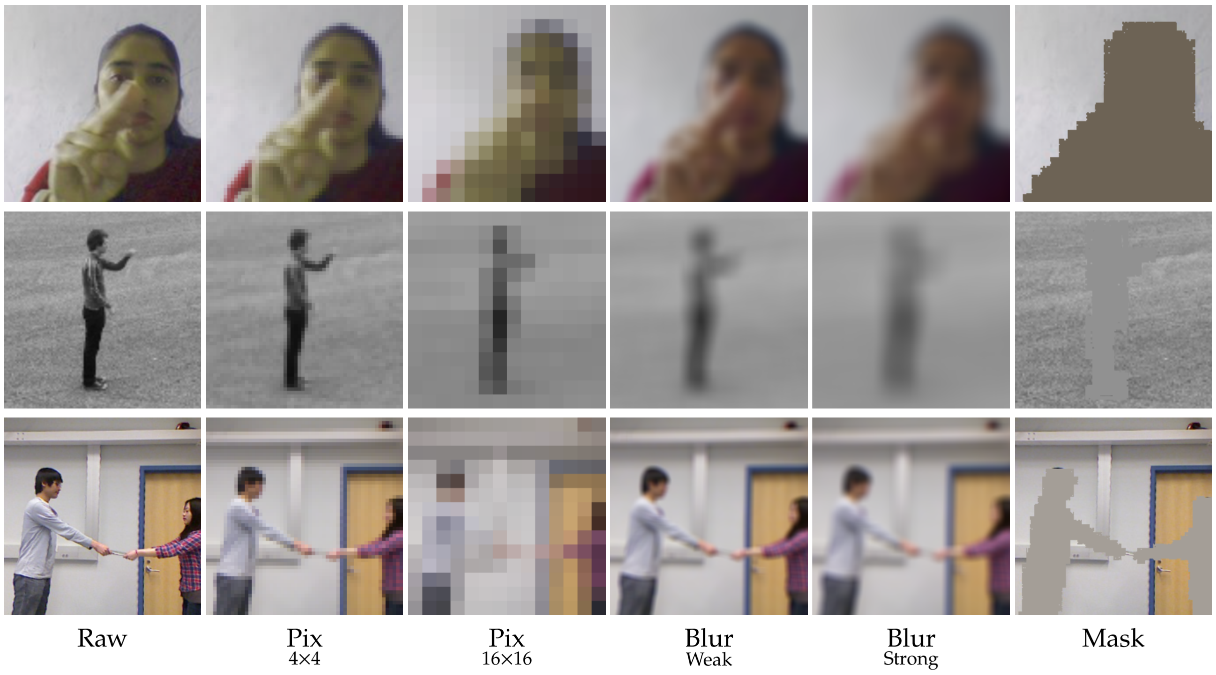


Figure A.5: Qualitative Examples of Naive Baseline Obfuscation Methods. These results augment those in main submission Figure 5.6, which provided similar examples for state-of-the-art methods. Samples top-to-bottom are from IPN, KTH and SBU.

A.3 Appendix to Chapter 5

Implementation details

Motion consistent noise For the creation of motion consistent noise, we follow [217] that uses RAFT [70] for optical flow estimation and nearest neighbor interpolation during warping. We also highlight that using noise that is adjusted to the dataset mean, main submission (5.3), is important, as other initializations reduce classification performance.

Matcher - queries, keys or values? For the template matcher, main submission Figure 5.2b, note that we use the *keys* of the vision transformer. This choice follows related work, *e.g.* [213, 214]; however, in our preliminary experiments we did not find large differences in using either *values* or *queries*, rather than *keys*.

Additional training details

For data augmentation: Videos are randomly resized to introduce scale variation and center cropped to 224×224 pixels. Horizontal flip augmentations are used for SBU and KTH, but not in IPN as the direction of the motion determines the class for some gestures. Colour and temporal jitter are applied, and input sequences are padded with the last frame in case the sequence is not long enough to fit the required clip length of a network. For two-stream approaches, both streams use the same spatial crops as the source video. All spatial augmentations also are applied to privacy networks.

Note that while the action and privacy recognition algorithms are trained on the original datasets used for evaluation [215, 216, 218], they are not trained on our obfuscated videos. In comparison, alternative compared state-of-the-art approaches do train recognition algorithms on their processed videos [189, 200, 205].

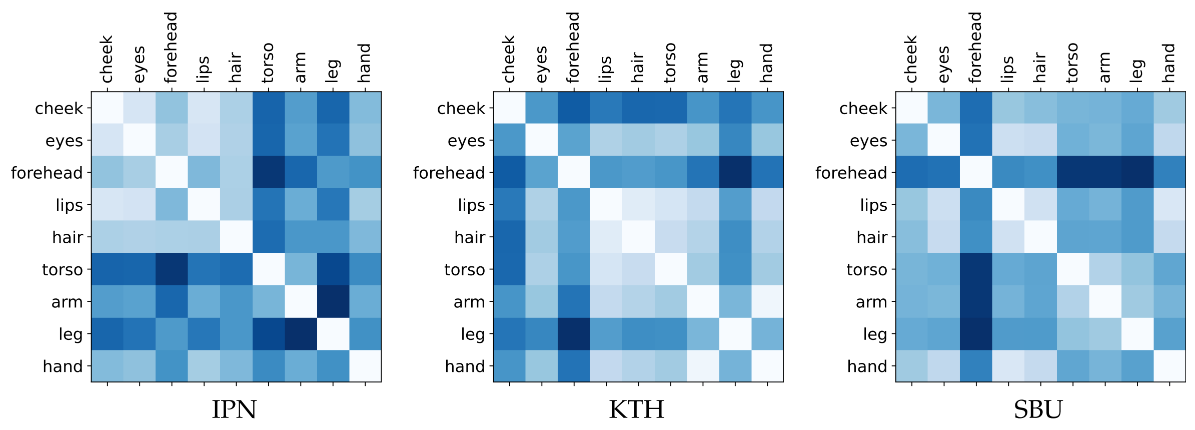


Figure A.6: Mean Average Error Between the Average Template Saliency for Each Dataset. Values plotted so that brighter values indicate that two templates are more similar to each other. The constructed matrices are diagonally symmetric by definition, (A.2).

Additional empirical evaluation

Naive baselines To get a better understanding of the naive baselines, we show samples from each of the three different datasets with all of the naive baseline obfuscation methods in Figure A.5. As noted in the main submission, for the case of mask obfuscation the masking is performed by using the mean of the region covered by the mask.

In the main paper, we used the mean of the region in the original image covered by the mask to fill the region. As a variation, we also considered using black for the masked region. Use of the mean value, rather than black, improved the action recognition results on all three datasets IPN +3.2% points, KTH +1.3% points, and SBU +2.4% points; however, the differences on privacy attributes were small across all datasets: IPN +0.6% points, KTH −0.3% points, and SBU −0.2% points. (We use green and red to indicate desired vs. undesired changes, respectively.)

Ablation over f_λ In the main submission, we introduced f_λ, (5.6), as a way to rank multiple obfuscation methods by combining the two metrics of concern, *i.e.* action recognition and privacy preservation, into a single value. Main submission Figure 5.7 showed the ranking and performance of obfuscation methods based on the importance weighting between action recognition and privacy preservation. The individual dataset ablations show that our method performs best across all datasets, and not just on average as shown in the main submission.

Overlap of individual saliency maps As discussed in the main submission, the selection of the privacy templates is dependent on the task and the videos in the dataset, *e.g.* there is no need to obscure lower body features for cases where only the upper body is visible in the video (IPN) and seeking to obscure detailed facial features when they are not distinguishable due to viewing at a great distance (KTH) also is not merited. Here, we quantify the similarity of selected templates. One way to do this would be to measure the Euclidean distance of the descriptors, but that does not reveal how the obfuscation actually appears in a given dataset. Therefore, to measure the similarity between the templates on a dataset level, we compare the saliency maps directly.

In particular, we compute the average saliency map for each template, τ_k in the template library, T, per dataset, according to

$$\bar{S}_{\tau_k}(\mathbf{p}) = \frac{1}{n}\sum_{i=1}^{n} S_i(\mathbf{p}) \qquad \forall \tau_k \in \mathsf{T}, \tag{A.1}$$

where $\mathbf{p} = (x, y)$ indexes pixels, i indexes saliency maps and n is the number of images in the dataset. (Main submission (5.2) provides the definition of individual salience maps.) These averages allow us to compute the mean absolute error between the salience of all template pairs, τ_i and τ_j, as

$$v_{i,j} = \sum_{\mathbf{p}} |\bar{S}_{\tau_i}(\mathbf{p}) - \bar{S}_{\tau_j}(\mathbf{p})|, \tag{A.2}$$

with **p** ranging over all pixels in the average saliency maps.

Figure A.6 shows the values, $v_{i,j}$, plotted so that greater similarity is visualized as brighter intensity.

From these plots, a number of observations can be made. First, it is seen that for all datasets the largest values appear along the diagonal, as templates should be most similar to themselves, and our similarity plots capture that property. Second, there is a tendency for templates with close spatial proximity to yield more similar saliency maps compared to those that are more distant. For example, templates for fine facial features, "eyes, lips, cheek", tend to yield saliency maps more similar to each other than to more distant body parts, "torso, arm, hand", and vice versa. Third, there is an interaction between spatial support and template similarity: If there is very little spatial support for a template, *e.g.* fine facial features on KTH, then it results in relatively low similarity scores, as there is too little data to capture similarities.

A.4 Publications

First Authorships

I published the following papers in peer-reviewed conferences as a first author during my PhD:

Selective, Interpretable, and Motion Consistent Privacy Attribute Obfuscation for Action Recognition. [247]
[CVPR'24] Conference on Computer Vision and Pattern Recognition. 2024.
Filip Ilic, Thomas Pock, He Zhao, and Richard P. Wildes.

Is Appearance Free Action Recognition Possible? [217]
[ECCV'22] European Conference on Computer Vision. 2022.
Filip Ilic, Thomas Pock, and Richard P. Wildes.

A Study on Robust Feature Representations for Grain Density Estimates in Austenitic Steel. [248]
[OAGM'21] 44th OAGM Workshop 2021: Computer Vision and Pattern Analysis Across Domains. 2021.
Filip Ilic, Marc Masana, Lea Bogensperger, Harald Ganster, and Thomas Pock.

Representing Objects in Video as Space-Time Volumes by Combining Top-Down and Bottom-Up Processes. [249]
[WACV'20] Winter Conference on Applications of Computer Vision. 2020.
Filip Ilic, and Axel Pinz.

Co-Authorships

I also contributed to the following papers as a co-author:

Object Motion Representation in the Macaque Ventral Stream - a Gateway to Understanding the Brain's Intuitive Physics Engine. [250]
[-] Under Submission.2024
Ramezanpour Hamidreza, Filip Ilic, Richard Wildes, and Kohitij Kar.

Score-Based Generative Models for Medical Image Segmentation using Signed Distance Functions. [251]
[GCPR'23] German Conference on Pattern Recognition. 2023.
Lea Bogensperger, Dominik Narnhofer, Filip Ilic, and Thomas Pock.

Lightweight Video Denoising using Aggregated Shifted Window Attention. [252]
[WACV'23] Winter Conference on Applications of Computer Vision. 2023.
Lydia Lindner, Alexander Effland, Filip Ilic, Thomas Pock, and Erich Kobler.

On the Influence of Beta Cell Granule Counting for Classification in Type 1 Diabetes. [253]
[OAGM'21] 44th OAGM Workshop 2021: Computer Vision and Pattern Analysis Across Domains. 2021.
Lea Bogensperger, Marc Masana, Filip Ilic, Harald Ganster, and Thomas Pock.

A.5 Code Repositories

As part of this thesis, several code repositories were created to facilitate the reproducibility of the results. The code is written in Python and uses the Pytorch Deep Learning Framework [50]. The following repositories are available:

Is Appearance Free Action Recognition Possible?

The repository contains code to create the synthetic appearance free dataset on the basis of UCF101. It additionally allows the creating of warping arbitrary noise patterns with optical flow extracted from arbitrary input video. The project page has animated videos of the appearance free data, which is crucial for understanding the importance of this research. The code contains the hyper-parameters of all the networks, data augmentation, and training scripts necessary to reproduce our results.

It furthermore contains the user study UI code that was used to perform the psychophysical user study, in which participants were tasked to classify the actions in the appearance free dataset. All visualization and analysis scripts that were used to create the figures in the paper (and more) are also included in the repository.

- Project Page: `https://f-ilic.github.io/AppearanceFreeActionRecognition`.
- Code Repository: `https://github.com/f-ilic/AppearanceFreeActionRecognition`
- Open Access Paper: `https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136640154.pdf`

Selective, Interpretable, and Motion Consistent Privacy Attribute Obfuscation for Action Recognition

The repository contains code to use a pretrained DINO Vision Transformer, in combination with a GUI application to select templates that ought to be obfuscated for a given input video. It furthermore also includes scripts and tools to visualize the attention of the ViT. Naive baselines and other methods to which comparisons are drawn are implemented or referenced in the repository.

- Project Page: `https://f-ilic.github.io/SelectivePrivacyPreservation`
- Code Repository: `https://github.com/f-ilic/SelectivePrivacyPreservation`
- Open Access Paper: `https://openaccess.thecvf.com/content/CVPR2024/papers/Ilic_Selective_Interpretable_and_Motion_Consistent_Privacy_Attribute_Obfuscation_for_Action_CVPR_2024_paper.pdf`

Representing Objects in Video as Space-Time Volumes by Combining Top-Down and Bottom-Up Processes

As part of the algorithm evaluation we added annotations to the DAVIS2016 [241] dataset. It is available to download on the project page.

- Project Page: `https://f-ilic.github.io/STVI`
- Open Access Paper: `https://openaccess.thecvf.com/content_WACV_2020/papers/Ilic_Representing_Objects_in_Video_as_Space-Time_Volumes_by_Combining_Top-Down_WACV_2020_paper.pdf`

Bibliography

[1] John Jumper et al. 'Highly accurate protein structure prediction with AlphaFold'. In: *Nature*. Vol. 596. 7873. Nature Publishing Group, 2021, pp. 583–589 (cited on page 3).

[2] David Silver et al. 'Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm'. In: *Nature*. Vol. 550. 7676. Nature Publishing Group, 2018, pp. 354–359 (cited on page 3).

[3] Aditya Ramesh et al. 'Zero-shot text-to-image generation'. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8821–8831 (cited on page 3).

[4] Randall Munroe. *Tasks*. XKCD, comic strip. Accessed: 2024-05-29. 2014. URL: https://xkcd.com/1425/ (cited on page 3).

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 'Imagenet classification with deep convolutional neural networks'. In: *Advances in Neural Information Processing Systems*. Vol. 25. 2012, pp. 1097–1105 (cited on pages 4, 48, 52, 85).

[6] Karen Simonyan and Andrew Zisserman. 'Very deep convolutional networks for large-scale image recognition'. In: *International Conference on Learning Representations*. 2015 (cited on pages 4, 29, 52, 85).

[7] Kaiming He et al. 'Deep residual learning for image recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778 (cited on pages 4, 52, 55, 56, 97).

[8] Jia Deng et al. 'Imagenet: A large-scale hierarchical image database'. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 248–255 (cited on pages 4, 48, 97, 103).

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 'U-net: Convolutional networks for biomedical image segmentation'. In: *Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241 (cited on page 4).

[10] Liang-Chieh Chen et al. 'Rethinking atrous convolution for semantic image segmentation'. In: *arXiv preprint arXiv:1706.05587*. 2017 (cited on page 4).

[11] Liang-Chieh Chen et al. 'Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs'. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 40. 4. IEEE, 2017, pp. 834–848 (cited on page 4).

[12] Tsung-Yi Lin et al. 'Microsoft COCO: Common Objects in Context'. In: *CoRR*. Vol. abs/1405.0312. 2014 (cited on pages 5, 6, 109, 110, 114).

[13] Ross Girshick. 'Fast R-CNN'. In: *International Conference on Computer Vision*. IEEE. 2015, pp. 1440–1448 (cited on page 4).

[14] Shaoqing Ren et al. 'Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks'. In: *Advances in Neural Information Processing Systems*. Montreal, Canada: MIT Press, 2015, pp. 91–99 (cited on pages 4, 109).

[15] Kaiming He et al. 'Mask R-CNN'. In: *International Conference on Computer Vision*. IEEE. 2017, pp. 2980–2988 (cited on pages 4, 108–110, 114–116, 118).

[16] Joseph Redmon et al. 'You only look once: Unified, real-time object detection'. In: *Conference on Computer Vision and Pattern Recognition*. 2016 (cited on pages 4, 102).

[17] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics (cited on pages 4, 102).

[18] Nicolas Carion et al. 'End-to-end object detection with transformers'. In: *European Conference on Computer Vision*. Springer. 2020, pp. 213–229 (cited on page 4).

[19] Alexander Kirillov et al. 'Segment anything'. In: *Conference on Computer Vision and Pattern Recognition*. 2023, pp. 4015–4026 (cited on page 6).

[20] C. Feichtenhofer, A. Pinz, and A. Zisserman. 'Detect to Track and Track to Detect'. In: *International Conference on Computer Vision*. 2017 (cited on pages 5, 109).

[21] Paul Voigtlaender et al. 'Siam r-cnn: Visual tracking by re-detection'. In: *Conference on Computer Vision and Pattern Recognition*. 2020, pp. 6578–6588 (cited on page 5).

[22] Xin Chen et al. 'Transformer tracking'. In: *Conference on Computer Vision and Pattern Recognition*. 2021, pp. 8126–8135 (cited on page 5).

[23] Yi Wang et al. 'Internvideo: General video foundation models via generative and discriminative learning'. In: *arXiv preprint arXiv:2212.03191*. 2022 (cited on page 5).

[24] Yuhan Zhu et al. 'Dual DETRs for Multi-Label Temporal Action Detection'. In: *Conference on Computer Vision and Pattern Recognition*. 2024, pp. 18559–18569 (cited on page 5).

[25] Guo Chen et al. 'Video mamba suite: State space model as a versatile alternative for video understanding'. In: *arXiv preprint arXiv:2403.09626*. 2024 (cited on page 5).

[26] Karen Simonyan and Andrew Zisserman. 'Two-stream convolutional networks for action recognition in videos'. In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014 (cited on pages 5, 48, 58, 60, 85, 90–92, 96).

[27] Christoph Feichtenhofer et al. 'SlowFast networks for video recognition'. In: *International Conference on Computer Vision*. 2019 (cited on pages 5, 48, 49, 58, 61, 85, 90, 92, 102).

[28] Christoph Feichtenhofer. 'X3D: Expanding Architectures for Efficient Video Recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2020 (cited on pages 5, 48, 85, 90, 92, 102).

[29] Anurag Arnab et al. 'ViViT: A video vision transformer'. In: *International Conference on Computer Vision*. 2021 (cited on pages 5, 48, 49, 75).

[30] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. 'UCF101: A dataset of 101 human actions classes from videos in the wild'. In: *arXiv preprint arXiv:1212.0402*. 2012 (cited on pages 6, 51).

[31] J. Tinsley Oden and Leszek Demkowicz. *Applied Functional Analysis*. 2nd ed. Textbooks in Mathematics. CRC Press, 2000 (cited on page 10).

[32] Svante Wold, Kim Esbensen, and Paul Geladi. 'Principal component analysis'. In: *Chemometrics and intelligent laboratory systems*. Vol. 2. 1-3. Elsevier, 1987, pp. 37–52 (cited on page 11).

[33] Laurens Van der Maaten and Geoffrey Hinton. 'Visualizing data using t-SNE.' In: *Journal of Machine Learning Research*. Vol. 9. 11. 2008 (cited on pages 11, 75).

[34] Carl Eckart and Gale Young. 'The approximation of one matrix by another of lower rank'. In: *Psychometrika*. Vol. 1. 3. Springer, 1936, pp. 211–218 (cited on page 11).

[35] E. H. Moore. 'On the reciprocal of the general algebraic matrix'. In: *Bulletin of the American Mathematical Society*. Vol. 26. 1920, pp. 394–395 (cited on page 13).

[36] Roger Penrose. 'A generalized inverse for matrices'. In: *Proceedings of the Cambridge Philosophical Society*. Vol. 51. 3. 1955, pp. 406–413 (cited on page 13).

[37] George Cybenko. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of Control, Signals and Systems*. Vol. 2. 4. Springer, 1989, pp. 303–314 (cited on page 17).

[38] John Duchi, Elad Hazan, and Yoram Singer. 'Adaptive subgradient methods for online learning and stochastic optimization.' In: *Journal of Machine Learning Research*. Vol. 12. 7. 2011 (cited on page 19).

[39] Yu Nesterov. 'Gradient methods for minimizing composite functions'. In: *Mathematical Programming*. Vol. 140. 1. Springer, 2013, pp. 125–161 (cited on pages 19, 20).

[40] Ilya Sutskever et al. 'On the importance of initialization and momentum in deep learning'. In: *International Conference on Machine Learning*. PMLR. 2013, pp. 1139–1147 (cited on page 19).

[41] Yoshua Bengio. 'Practical Recommendations for gradient-based training of deep architectures'. In: *Neural Networks: Tricks of the trade: Second edition*. Springer, 2012, pp. 437–478 (cited on page 20).

[42] Leslie N Smith. 'Cyclical learning rates for training neural networks'. In: *Winter Conference on Applications of Computer Vision*. IEEE. 2017, pp. 464–472 (cited on page 20).

[43] Zhouxing Shi et al. 'Fast certified robust training with short warmup'. In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021, pp. 18335–18349 (cited on page 20).

[44] Xavier Glorot and Yoshua Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256 (cited on page 20).

[45] Siddharth Krishna Kumar. 'On weight initialization in deep neural networks'. In: *arXiv preprint arXiv:1704.08863*. 2017 (cited on page 20).

[46] Yanli Liu, Yuan Gao, and Wotao Yin. 'An improved analysis of stochastic gradient descent with momentum'. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 18261–18271 (cited on page 20).

[47] Diederik P Kingma and Jimmy Ba. 'Adam: A method for stochastic optimization'. In: *International Conference on Learning Representations*. 2015 (cited on pages 20, 90, 103).

[48] Seppo Linnainmaa. 'The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors'. In Finnish. Masters. University of Helsinki, 1970, pp. 6–7 (cited on page 20).

[49] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 'Learning representations by back-propagating errors'. In: *Nature*. Vol. 323. 6088. Nature Publishing Group, 1986, pp. 533–536 (cited on page 20).

[50] Adam Paszke et al. 'Pytorch: An imperative style, high-performance deep learning library'. In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019 (cited on pages 20, 136).

[51] Atilim Gunes Baydin et al. 'Automatic differentiation in machine learning: a survey'. In: *Journal of Marchine Learning Research*. Vol. 18. Microtome Publishing, 2018, pp. 1–43 (cited on page 21).

[52] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023 (cited on page 28).

[53] Kunihiko Fukushima. 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position'. In: *Biological cybernetics*. Vol. 36. 4. Springer, 1980, pp. 193–202 (cited on page 29).

[54] Yann LeCun et al. 'Handwritten digit recognition with a back-propagation network'. In: *Advances in neural information processing systems*. Vol. 2. 1989 (cited on page 29).

[55] Daniel E Worrall et al. 'Harmonic networks: Deep translation and rotation equivariance'. In: *Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5028–5037 (cited on page 30).

[56] Rosanne Liu et al. 'An intriguing failing of convolutional neural networks and the coordconv solution'. In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018 (cited on page 30).

[57] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *Nature*. Vol. 323. 6088. Nature Publishing Group, 1986, pp. 533–536 (cited on page 32).

[58] Pascal Vincent et al. 'Extracting and composing robust features with denoising autoencoders'. In: *International Conference on Machine Learning*. 2008, pp. 1096–1103 (cited on page 32).

[59] Andrew Ng et al. 'Sparse autoencoder'. In: *CS294A Lecture notes*. Vol. 72. 2011. Citeseer. 2011, pp. 1–19 (cited on page 32).

[60] Aravindh Mahendran and Andrea Vedaldi. 'Visualizing deep convolutional neural networks using natural pre-images'. In: *International Journal of Computer Vision*. Vol. 120. 3. Springer, 2016, pp. 233–255 (cited on page 35).

[61] Simon Baker et al. 'A Database and Evaluation Methodology for Optical Flow'. In: *International Journal of Computer Vision*. Vol. 92. 1. Springer, 2011, pp. 1–31 (cited on pages 37, 84).

[62] D. J. Butler et al. 'A naturalistic open source movie for optical flow evaluation'. In: *European Conference on Computer Vision*. 2012 (cited on pages 37, 85, 93).

[63] Berthold KP Horn and Brian G Schunck. 'Determining optical flow'. In: *Artificial Intelligence*. Vol. 17. 1-3. Elsevier, 1981, pp. 185–203 (cited on pages 38, 39, 44).

[64] Bruce D Lucas and Takeo Kanade. 'An iterative image registration technique with an application to stereo vision'. In: *International Joint Conference on Artificial Intelligence*. Vol. 2. 1981, pp. 674–679 (cited on page 38).

[65] Christopher Zach, Thomas Pock, and Horst Bischof. 'A duality based approach for realtime tv-l 1 optical flow'. In: *German Conference on Pattern Recognition*. Springer. 2007, pp. 214–223 (cited on pages 39, 44).

[66] Deqing Sun et al. 'PWC-net: Cnns for optical flow using pyramid, warping, and cost volume'. In: *Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8934–8943 (cited on pages 40, 42, 44).

[67] Alexey Dosovitskiy et al. 'FlowNet: Learning Optical Flow with Convolutional Networks'. In: *International Conference on Computer Vision*. 2015 (cited on pages 42, 44, 85).

[68] Anurag Ranjan and Michael J Black. 'Optical flow estimation using a spatial pyramid network'. In: *Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4161–4170 (cited on page 42).

[69] Zachary Teed and Jia Deng. 'RAFT: Recurrent All-Pairs Field Transforms for Optical Flow'. In: *European Conference on Computer Vision*. 2020 (cited on pages 42, 84, 86, 87, 93).

[70] Zachary Teed and Jia Deng. 'RAFT: Recurrent all-pairs field transforms for optical flow'. In: *European Conference on Computer Vision*. 2020 (cited on pages 42–44, 49, 103, 132).

[71] Junyoung Chung et al. 'Empirical evaluation of gated recurrent neural networks on sequence modeling'. In: *arXiv preprint arXiv:1412.3555*. 2014 (cited on pages 43, 49).

[72] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. In: *Neural Comput.* Vol. 9. 8. Cambridge, MA, USA: MIT Press, 1997, pp. 1735–1780. doi: 10.1162/neco.1997.9.8.1735 (cited on pages 43, 49).

[73] Lahav Lipson, Zachary Teed, and Jia Deng. 'Raft-stereo: Multilevel recurrent field transforms for stereo matching'. In: *International Conference on 3D Vision*. 2021, pp. 218–227 (cited on page 43).

[74] Zachary Teed and Jia Deng. 'Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras'. In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021, pp. 16558–16569 (cited on page 43).

[75] Heng Wang et al. 'Dense Trajectories and Motion Boundary Descriptors for action recognition'. In: *International Journal of Computer Vision*. 2013 (cited on page 48).

[76] Corinna Cortes and Vladimir Vapnik. 'Support-vector networks'. In: *Machine learning*. Vol. 20. 3. Springer, 1995, pp. 273–297 (cited on page 48).

[77] I. Laptev and T. Lindeberg. 'Space-time interest points'. In: *International Conference on Computer Vision*. 2003 (cited on pages 48, 111).

[78] Chris Harris, Mike Stephens, et al. 'A combined corner and edge detector'. In: *Alvey Vision Conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244 (cited on page 48).

[79] Soo Min Kang and Richard P Wildes. 'Review of action recognition and detection methods'. In: *arXiv preprint arXiv:1610.06906*. 2016 (cited on pages 48, 84).

[80] Andrej Karpathy et al. 'Large-Scale Video Classification with Convolutional Neural Networks'. In: *Conference on Computer Vision and Pattern Recognition*. 2014 (cited on pages 48, 50).

[81] Du Tran et al. 'Learning spatiotemporal features with 3D convolutional networks'. In: *International Conference on Computer Vision*. 2015 (cited on pages 48, 57, 85).

[82] Jeffrey Donahue et al. 'Long-term recurrent convolutional networks for visual recognition and description'. In: *Conference on Computer Vision and Pattern Recognition*. 2015 (cited on page 48).

[83] Limin Wang et al. 'Temporal segment networks: Towards good practices for deep action recognition'. In: *European Conference on Computer Vision*. Springer. 2016 (cited on page 48).

[84] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. 'Convolutional two-stream network fusion for video action recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1933–1941 (cited on pages 48, 109).

[85] Joao Carreira and Andrew Zisserman. 'Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset'. In: *Conference on Computer Vision and Pattern Recognition*. 2017 (cited on pages 48, 51, 57, 79, 85, 90–92, 94, 102–104).

[86] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. 'Learning spatio-temporal features with 3D residual networks for action recognition'. In: *International Conference on Computer Vision Workshops*. 2017 (cited on page 48).

[87] Xiaolong Wang et al. 'Non-local Neural Networks'. In: *Conference on Computer Vision and Pattern Recognition*. 2018 (cited on pages 48, 85, 90, 92, 102).

[88] Du Tran et al. 'A closer look at spatiotemporal convolutions for action recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2018 (cited on pages 48, 49, 58, 85, 90, 92, 102).

[89] Yi Zhu et al. 'Hidden Two-Stream Convolutional Networks for Action Recognition'. In: *Asian Conference on Computer Vision*. 2018 (cited on page 48).

[90] Bolei Zhou et al. 'Temporal relational reasoning in videos'. In: *European Conference on Computer Vision*. 2018 (cited on page 48).

[91] Saining Xie et al. 'Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification'. In: *European Conference on Computer Vision*. 2018 (cited on pages 48, 57).

[92] Ji Lin, Chuang Gan, and Song Han. 'TSM: Temporal shift module for efficient video understanding'. In: *International Conference on Computer Vision*. 2019 (cited on page 48).

[93] Du Tran et al. 'Video classification with channel-separated convolutional networks'. In: *International Conference on Computer Vision*. 2019 (cited on pages 48, 102).

[94] Yan Li et al. 'Tea: Temporal excitation and aggregation for action recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2020 (cited on page 48).

[95] Haoqi Fan et al. 'Multiscale Vision Transformers'. In: *International Conference on Computer Vision*. 2021 (cited on pages 48, 49, 75, 85, 90, 92, 102).

[96] Zhan Tong et al. 'Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training'. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 10078–10093 (cited on page 48).

[97] Ze Liu et al. 'Video swin transformer'. In: *Conference on Computer Vision and Pattern Recognition*. 2022 (cited on pages 48, 74).

[98] Alexey Gritsenko et al. 'End-to-End Spatio-Temporal Action Localisation with Video Transformers'. In: *arXiv preprint arXiv:2304.12160*. 2023 (cited on page 48).

[99] Yann LeCun et al. 'Gradient-Based Learning Applied to Document Recognition'. In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2324 (cited on page 48).

[100] Andrej Karpathy et al. 'Large-scale video classification with convolutional neural networks'. In: *Conference on Computer Vision and Pattern Recognition*. 2014 (cited on pages 48, 49, 85).

[101] Christoph Feichtenhofer et al. 'What have we learned from deep representations for action recognition?' In: *Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7844–7853 (cited on page 48).

[102] Alexey Dosovitskiy et al. 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale'. In: *International Conference on Learning Representations*. 2021 (cited on pages 49, 97, 98, 102).

[103] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. 'Is space-time attention all you need for video understanding?' In: *International Conference on Machine Learning*. 2021 (cited on pages 49, 75, 76, 79).

[104] Javier Selva et al. 'Video transformers: A survey'. In: *Transactions on Pattern Analysis and Machine Intelligence*. IEEE, 2023 (cited on page 49).

[105] N. Sriastava, E. Manisomov, and R. Salakhutdinov. 'Unsupervised Learning of Video Representations using LSTMs'. In: *International Conference on Machine Learning*. 2015 (cited on pages 49, 85).

[106] Will Kay et al. 'The Kinetics human action video dataset'. In: *arXiv preprint arXiv:1705.06950*. 2017 (cited on pages 49, 76, 97).

[107] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. 'UCF101: A dataset of 101 human actions classes from videos in the wild'. In: *arXiv preprint arXiv:1212.0402*. Nov. 2012 (cited on pages 49, 50, 84–87).

[108] Hildegard Kuehne et al. 'HMDB: A large video database for human motion recognition'. In: *International Conference on Computer Vision*. 2011 (cited on pages 50, 85).

[109] Fabian Caba Heilbron et al. 'ActivityNet: A large-scale video benchmark for human activity understanding'. In: *Conference on Computer Vision and Pattern Recognition*. 2015 (cited on pages 50, 97).

[110] Sami Abu-El-Haija et al. 'Youtube-8m: A large-scale video classification benchmark'. In: *arXiv preprint arXiv:1609.08675*. 2016 (cited on page 50).

[111] Gunnar A Sigurdsson et al. 'Hollywood in homes: Crowdsourcing data collection for activity understanding'. In: *European Conference on Computer Vision*. Springer. 2016 (cited on page 50).

[112] Joao Carreira et al. 'A short note about kinetics-600'. In: *arXiv preprint arXiv:1808.01340*. 2018 (cited on page 50).

[113] Raghav Goyal et al. 'The "something something" Video Database for Learning and Evaluating Visual Common Sense'. In: *International Conference on Computer Vision*. 2017 (cited on pages 50, 85, 94).

[114] Chunhui Gu et al. 'AVA: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions'. In: *Conference on Computer Vision and Pattern Recognition*. 2018 (cited on pages 50, 85).

[115] Yingwei Li, Yi Li, and Nuno Vasconcelos. 'Resound: Towards action recognition without representation bias'. In: *European Conference on Computer Vision*. 2018 (cited on pages 50, 51, 84, 85).

[116] Ali Diba et al. 'Large scale holistic video understanding'. In: *European Conference on Computer Vision*. Springer. 2020, pp. 593–610 (cited on pages 50, 51).

[117] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 'Training Very Deep Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015 (cited on page 52).

[118] Nitish Srivastava et al. 'Dropout: a simple way to prevent neural networks from overfitting'. In: *The Journal of machine learning research*. Vol. 15. 1. JMLR. org, 2014, pp. 1929–1958 (cited on page 53).

[119] Yarin Gal and Zoubin Ghahramani. 'Dropout as a bayesian approximation: Representing model uncertainty in deep learning'. In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1050–1059 (cited on page 53).

[120] Sergey Ioffe and Christian Szegedy. 'Batch normalization: Accelerating deep network training by reducing internal covariate shift'. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 448–456 (cited on page 53).

[121] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *Inceptionism: Going Deeper into Neural Networks*. https://research.google/blog/inceptionism-going-deeper-into-neural-networks/. Accessed: 2023-05-03. 2015 (cited on page 57).

[122] Jason Yosinski et al. 'Understanding neural networks through deep visualization'. In: *arXiv preprint arXiv:1506.06579*. 2015 (cited on page 57).

[123] Christian Szegedy et al. 'Going deeper with convolutions'. In: *Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9 (cited on page 57).

[124] Christian Szegedy et al. 'Rethinking the inception architecture for computer vision'. In: *Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826 (cited on pages 57, 60).

[125] Ashish Vaswani et al. 'Attention is all you need'. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017 (cited on pages 63, 99).

[126] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2nd ed. New York: Springer, 2024 (cited on pages 64, 66).

[127] Alexey Dosovitskiy et al. 'An image is worth 16x16 words: Transformers for image recognition at scale'. In: *International Conference on Learning Representations*. 2021 (cited on pages 66, 69, 70).

[128] Jia Deng et al. 'Imagenet: A large-scale hierarchical image database'. In: *Conference on Computer Vision and Pattern Recognition*. 2009 (cited on page 70).

[129] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. 'On the Relationship between Self-Attention and Convolutional Layers'. In: (2020) (cited on page 71).

[130] Maithra Raghu et al. 'Do vision transformers see like convolutional neural networks?' In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021, pp. 12116–12128 (cited on page 71).

[131] Sinong Wang et al. 'Linformer: Self-attention with linear complexity'. In: *arXiv preprint arXiv:2006.04768*. 2020 (cited on page 73).

[132] Tom Brown et al. 'Language models are few-shot learners'. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 33. 2020, pp. 1877–1901 (cited on page 73).

[133] Ze Liu et al. 'Swin transformer: Hierarchical vision transformer using shifted windows'. In: *International Conference on Computer Vision*. 2021, pp. 10012–10022 (cited on pages 73, 74).

[134] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006 (cited on page 77).

[135] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012 (cited on page 77).

[136] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009 (cited on page 77).

[137] Aditya Dave et al. 'TCLR: Temporal Contrastive Learning for Video Representation'. In: *arXiv preprint arXiv:2101.07974*. 2021 (cited on page 77).

[138] Ashish Jaiswal et al. 'A survey on contrastive self-supervised learning'. In: *Technologies*. Vol. 9. 1. MDPI, 2020, p. 2 (cited on page 77).

[139] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. 'Contrastive Representation Learning: A Framework and Review'. In: *IEEE Access*. Vol. 8. IEEE, 2020, pp. 193907–193934 (cited on page 77).

[140] Maxime Bucher, Stéphane Herbin, and Frédéric Jurie. 'Hard negative mining for metric learning based zero-shot classification'. In: *European Conference on Computer Vision Workshops*. Springer. 2016, pp. 524–531 (cited on page 78).

[141] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 'Distilling the Knowledge in a Neural Network'. In: *arXiv preprint arXiv:1503.02531*. 2015 (cited on page 78).

[142] Mathilde Caron et al. 'Emerging properties in self-supervised vision transformers'. In: *International Conference on Computer Vision*. 2021 (cited on pages 81, 97–99).

[143] Yi Zhu et al. 'A comprehensive study of deep video action recognition'. In: *arXiv preprint arXiv:2012.06567*. 2020 (cited on pages 84, 92).

[144] Liam Hiley, Alun Preece, and Yulia Hicks. 'Explainable Deep Learning for Video Recognition Tasks: A Framework & Recommendations'. In: *arXiv preprint arXiv:1909.05667*. 2019 (cited on page 84).

[145] European Commision. *Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act) and Amending Certain Union Legislative Acts*. European Commision. 2021 (cited on page 84).

[146] Law Council of Ontario. *Regulating AI: Critical Issues and Choices*. Law Council of Ontario. 2021 (cited on page 84).

[147] Tuan-Hung Vu et al. 'Predicting Actions from Static Scenes'. In: *Conference on Computer Vision and Pattern Recognition*. 2014 (cited on page 84).

[148] Yun He et al. 'Human action recognition without human'. In: *European Conference on Computer Vision*. 2016 (cited on page 84).

[149] Jinwoo Choi et al. 'Why can't I dance in the mall? Learning to mitigate scene bias in action recognition'. In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019 (cited on pages 84, 105).

[150] Christoph Feichtenhofer et al. 'Deep Insights into Convolutional networks for video recognition'. In: *International Journal of Computer Vision*. Vol. 128. 2. 2020, pp. 420–437 (cited on pages 84, 85).

[151] He Zhao and Richard P Wildes. 'Interpretable Deep Feature Propagation for Early Action Recognition'. In: *arXiv preprint arXiv:2107.05122*. 2021 (cited on pages 84, 85).

[152] Bela Julesz. *Foundations of Cyclopean Perception.* U. Chicago Press, 1971 (cited on pages 84, 85).

[153] Nayyer Aafaq et al. 'Video description: A survey of methods, datasets, and evaluation metrics'. In: *ACM Computing Surveys*. Vol. 52. 6. ACM New York, NY, USA, 2019, pp. 1–37 (cited on pages 84–87).

[154] Yu Kong, Yunde Jia, and Yun Fu. 'Interactive Phrases: Semantic Descriptions for Human Interaction Recognition'. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 36. 9. IEEE, 2014, pp. 1775–1788 (cited on page 85).

[155] Laura Sevilla-Lara et al. 'Only Time Can Tell: Discovering Temporal Data for Temporal Modeling'. In: *Winter Conference on Applications of Computer Vision*. 2021 (cited on page 85).

[156] Hala Lamdouar et al. 'Betrayed by Motion: Camouflaged Object Discovery via Motion Segmentation'. In: *Asian Conference on Computer Vision*. 2020 (cited on pages 85, 94).

[157] Pia Bideau and Erik Learned-Miller. 'It is Moving! A Probabilistic Model for Causal Motion Segmentation in Moving Camera Videos'. In: *European Conference on Computer Vision*. 2016 (cited on page 85).

[158] Oliver J Braddick. 'Low-Level and High-Level Processes in Apparent Motion'. In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*. Vol. 290. 1038. The Royal Society London, 1980, pp. 137–151 (cited on page 85).

[159] Shin'ya Nishida et al. 'Motion Perception: From Detection to Interpretation'. In: *Annual review of Vision Science*. Vol. 4. Annual Reviews, 2018, pp. 501–523 (cited on page 85).

[160] S Ullman. *The Interpretation of Visual Motion*. MIT Press, 1979 (cited on page 85).

[161] Bolei Zhou, Xiaoou Tang, and Xiaogang Wang. 'Coherent Filtering: Detecting Coherent Motions from Crowd Clutters'. In: *European Conference on Computer Vision*. 2012 (cited on page 85).

[162] Gunnar Johansson. 'Visual perception of biological motion and a model for its analysis'. In: *Perception & Psychophysics*. Vol. 14. 2. 1973, pp. 201–211 (cited on pages 85, 88).

[163] Nikolaus Mayer et al. 'A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation'. In: *Conference on Computer Vision and Pattern Recognition*. 2016 (cited on page 85).

[164] German Ros et al. 'The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes'. In: *Conference on Computer Vision and Pattern Recognition*. 2016 (cited on page 85).

[165] Stephan R Richter, Zeeshan Hayder, and Vladlen Koltun. 'Playing for benchmarks'. In: *International Conference on Computer Vision*. 2017 (cited on page 85).

[166] N. Mahmood et al. 'AMASS: Archive of Motion Capture as Surface Shapes'. In: *International Conference on Computer Vision*. 2019 (cited on page 85).

[167] Forrest N Iandola et al. 'SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size'. In: *arXiv preprint arXiv:1602.07360*. 2016 (cited on page 85).

[168] Kaiming He et al. 'Deep residual learning for image recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2016 (cited on pages 85, 87, 90, 92, 102, 104).

[169] Shuiwang Ji et al. '3D Convolutional Neural Networks for Human Action Recognition'. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 35. 1. 2012, pp. 221–231 (cited on page 85).

[170] David H Hubel and Torsten N Wiesel. 'Receptive Fields of Single Neurones in the Cat's Striate Cortex'. In: *The Journal of Physiology*. Vol. 148. 3. Wiley-Blackwell, 1959, pp. 574–591 (cited on page 85).

[171] Melvyn A Goodale and A David Milner. 'Separate visual pathways for perception and action'. In: *Trends in Neurosciences*. Vol. 15. 1. Elsevier, 1992, pp. 20–25 (cited on page 85).

[172] Christoph Feichtenhofer, Axel Pinz, and Richard P Wildes. 'Spatiotemporal Multiplier Networks for Video Action Recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2017 (cited on pages 85, 93).

[173] Isma Hadji and Richard P Wildes. 'A new large scale dynamic texture dataset with application to convnet understanding'. In: *European Conference on Computer Vision*. 2018 (cited on page 85).

[174] A. Ghodrati, E. Gavves, and C. G. M. Snoek. 'Video Time: Properties, Encoders and Evaluation'. In: *British Machine Vision Conference*. 2018 (cited on page 85).

[175] Joonatan Manttari et al. 'Interpreting Video Features: A comparison of 3D Convolutional Networks and Convolutional LSTM Networks'. In: *Asian Conference on Computer Vision*. 2020 (cited on page 85).

[176] Laura Sevilla-Lara et al. 'On the integration of optical flow and action recognition'. In: *German Conference on Pattern Recognition*. Springer. 2018 (cited on page 85).

[177] Matthew Kowal et al. 'A deeper dive into what deep spatiotemporal networks encode: Quantifying static vs. dynamic information'. In: *German Conference on Pattern Recognition*. 2022 (cited on page 85).

[178] George A Miller. 'The magical number seven, plus or minus two: Some limits on our capacity for processing information.' In: *Psychological review*. Vol. 63. 2. American Psychological Association, 1956, p. 81 (cited on page 87).

[179] Martin Thoma. 'Analysis and Optimization of Convolutional Neural Network Architectures'. MA thesis. University of the State of Baden-Wuerttemberg, 2017 (cited on page 88).

[180] Winand H Dittrich. 'Action Categories and the Perception of Biological Motion'. In: *Perception*. Vol. 22. 1. SAGE Publications Sage UK: London, England, 1993, pp. 15–22 (cited on page 88).

[181] Nikolaus F Troje. 'Decomposing Biological Motion: A Framework for Analysis and Synthesis of Human Gait Patterns'. In: *Journal of Vision*. Vol. 2. 5. The Association for Research in Vision and Ophthalmology, 2002, pp. 2–2 (cited on page 88).

[182] Damien L Crone et al. 'The Socio-Moral Image Database (SMID): A novel stimulus set for the study of social, moral and affective processes'. In: *PloS one*. Vol. 13. 1. Public Library of Science San Francisco, CA USA, 2018, e0190954 (cited on page 96).

[183] Shreyansh Gandhi et al. 'Scalable Detection of Offensive and Non-compliant Content/Logo in Product Images'. In: *Winter Conference on Applications of Computer Vision*. 2020 (cited on page 96).

[184] Agostina J Larrazabal et al. 'Gender imbalance in medical imaging datasets produces biased classifiers for computer-aided diagnosis'. In: *Proceedings of the National Academy of Sciences*. Vol. 117. 23. National Acad Sciences, 2020, pp. 12592–12594 (cited on page 96).

[185] Patrick Schramowski et al. 'Safe latent diffusion: Mitigating inappropriate degeneration in diffusion models'. In: *Conference on Computer Vision and Pattern Recognition*. 2023 (cited on page 96).

[186] Carlos Hinojosa et al. 'PrivHAR: Recognizing human actions from privacy-preserving lens'. In: *European Conference on Computer Vision*. 2022 (cited on pages 96, 97, 101).

[187] Zhenyu Wu et al. 'Towards privacy-preserving visual recognition via adversarial training: A pilot study'. In: *European Conference on Computer Vision*. 2018 (cited on pages 96, 101, 102).

[188] Ishan Rajendrakumar Dave, Chen Chen, and Mubarak Shah. 'SPAct: Self-supervised privacy preservation for action recognition'. In: *Conference on Computer Vision and Pattern Recognition*. 2022 (cited on pages 96, 97, 102).

[189] Sudhakar Kumawat and Hajime Nagahara. 'Privacy-Preserving Action Recognition via Motion Difference Quantization'. In: *European Conference on Computer Vision*. 2022 (cited on pages 96, 97, 101–105, 132).

[190] Yingying Zhu, Nandita M Nayak, and Amit K Roy-Chowdhury. 'Context-aware modeling and recognition of activities in video'. In: *Conference on Computer Vision and Pattern Recognition*. 2013 (cited on page 96).

[191] Alexander John Karran et al. 'Designing for Confidence: The Impact of Visualizing Artificial Intelligence Decisions'. In: *Frontiers in Neuroscience*. Vol. 16. Frontiers Media SA, 2022, p. 883385 (cited on page 96).

[192] Haochen Liu et al. 'Trustworthy AI: A computational perspective'. In: *ACM Transactions on Intelligent Systems and Technology*. Vol. 14. 1. ACM New York, NY, 2022, pp. 1–59 (cited on page 96).

[193] Bart Thomee et al. 'YFCC100M: The new data in multimedia research'. In: *Communications of the ACM*. Vol. 59. 2. ACM New York, NY, USA, 2016, pp. 64–73 (cited on page 97).

[194] Kristen Grauman et al. ‘Ego4d: Around the world in 3,000 hours of egocentric video’. In: *Conference on Computer Vision and Pattern Recognition*. 2022 (cited on page 97).

[195] Nicholas Carlini et al. ‘Extracting training data from large language models’. In: *30th USENIX Security Symposium*. 2021 (cited on page 97).

[196] Bo Liu et al. ‘When machine learning meets privacy: A survey and outlook’. In: *ACM Computing Surveys*. Vol. 54. 2. ACM New York, NY, USA, 2021, pp. 1–36 (cited on page 97).

[197] Vijeta Sharma et al. ‘A review of deep-learning-based human activity recognition on benchmark video datasets’. In: *Applied Artificial Intelligence*. Vol. 36. 1. 2022, p. 2093705 (cited on page 97).

[198] Robert T. Collins, Alan J Lipton, and Takeo Kanade. ‘Introduction to the special section on video surveillance’. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 22. 8. IEEE, 2000, pp. 745–746 (cited on page 97).

[199] Waqas Sultani, Chen Chen, and Mubarak Shah. ‘Real-world anomaly detection in surveillance videos’. In: *Conference on Computer Vision and Pattern Recognition*. 2018 (cited on page 97).

[200] Michael Ryoo et al. ‘Privacy-preserving human activity recognition from extreme low resolution’. In: *AAAI*. Vol. 31. 1. 2017 (cited on pages 97, 101, 102, 104, 132).

[201] Michael Ryoo, Kiyoon Kim, and Hyun Yang. ‘Extreme low resolution activity recognition with multi-siamese embedding learning’. In: *AAAI*. 2018 (cited on pages 97, 103–105).

[202] Jiawei Chen et al. ‘Semi-coupled two-stream fusion ConvNets for action recognition at extremely low resolutions’. In: *Winter Conference on Applications of Computer Vision*. 2017 (cited on page 97).

[203] Zhongzheng Ren, Yong Jae Lee, and Michael S Ryoo. ‘Learning to anonymize faces for privacy preserving action detection’. In: *European Conference on Computer Vision*. 2018 (cited on page 97).

[204] Amin Jourabloo, Xi Yin, and Xiaoming Liu. ‘Attribute preserved face de-identification’. In: *International Conference on Biometrics*. 2015 (cited on page 97).

[205] Zhenyu Wu et al. ‘Privacy-preserving deep action recognition: An adversarial learning framework and a new dataset’. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 44. 4. IEEE, 2020, pp. 2126–2139 (cited on pages 97, 101–105, 132).

[206] Ming Li et al. ‘STPrivacy: Spatio-Temporal Privacy-Preserving Action Recognition’. In: *International Conference on Computer Vision*. 2023 (cited on page 97).

[207] Joseph Fioresi, Ishan Rajendrakumar Dave, and Mubarak Shah. ‘Ted-SPAD: Temporal distinctiveness for self-supervised privacy-preservation for video anomaly detection’. In: *International Conference on Computer Vision*. 2023 (cited on page 97).

[208] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003 (cited on page 97).

[209] Paul-Edouard Sarlin et al. ‘SuperGlue: Learning feature matching with graph neural networks’. In: *Conference on Computer Vision and Pattern Recognition*. 2020 (cited on page 97).

[210] Navneet Dalal and Bill Triggs. ‘Histograms of oriented gradients for human detection’. In: *Conference on Computer Vision and Pattern Recognition*. 2005 (cited on page 97).

[211] Jianbo Shi and Carlo Tomasi. ‘Good features to track’. In: *Conference on Computer Vision and Pattern Recognition*. 1994 (cited on page 97).

[212] David G Lowe. ‘Distinctive image features from scale-invariant keypoints’. In: *International Journal of Computer Vision*. Vol. 60. Springer, 2004, pp. 91–110 (cited on page 97).

[213] Shir Amir et al. ‘Deep ViT Features as Dense Visual Descriptors’. In: *European Conference on Computer Vision Workshops*. 2022 (cited on pages 98, 99, 132).

[214] Maxime Oquab et al. ‘Dinov2: Learning robust visual features without supervision’. In: *arXiv preprint arXiv:2304.07193*. 2023 (cited on pages 98, 99, 132).

[215] Gibran Benitez-Garcia et al. ‘IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition’. In: *Internation Conference on Pattern Recognition*. 2021 (cited on pages 98, 101, 104, 132).

[216] Kiwon Yun et al. 'Two-person interaction detection using body-pose features and multiple instance learning'. In: *Conference on Computer Vision and Pattern Recognition*. 2012 (cited on pages 98, 101, 104, 132).

[217] Filip Ilic, Thomas Pock, and Richard P Wildes. 'Is Appearance Free Action Recognition Possible?' In: *European Conference on Computer Vision*. 2022 (cited on pages 100, 102, 132, 135).

[218] Christian Schuldt, Ivan Laptev, and Barbara Caputo. 'Recognizing human actions: a local SVM approach'. In: *Internation Conference on Pattern Recognition*. 2004 (cited on pages 101, 104, 132).

[219] Simon Baker et al. 'A database and evaluation methodology for optical flow'. In: *International Journal of Computer Vision*. Vol. 92. Springer, 2011, pp. 1–31 (cited on page 103).

[220] Matthew Kowal et al. 'A deeper dive into what deep spatiotemporal networks encode: Quantifying static vs. dynamic information'. In: *Conference on Computer Vision and Pattern Recognition*. 2022 (cited on page 105).

[221] Anil K Jain, Patrick Flynn, and Arun A Ross. *Handbook of Biometrics*. Springer Science & Business Media, 2007 (cited on page 106).

[222] Jason Chang, Donglai Wei, and John W Fisher III. 'A video representation using temporal superpixels'. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2013, pp. 2051–2058 (cited on pages 108, 109, 113–116, 118).

[223] Fuxin Li et al. 'Video Segmentation by Tracking Many Figure-Ground Segments'. In: *International Conference on Computer Vision*. 2013 (cited on pages 108, 109).

[224] Ning Xu et al. 'Youtube-vos: Sequence-to-sequence video object segmentation'. In: *European Conference on Computer Vision*. 2018, pp. 585–601 (cited on page 109).

[225] Qiang Wang et al. 'Fast online object tracking and segmentation: A unifying approach'. In: *Conference on Computer Vision and Pattern Recognition*. 2019 (cited on pages 109, 114–116, 118).

[226] Luca Bertinetto et al. 'Fully-convolutional siamese networks for object tracking'. In: *European Conference on Computer Vision*. Springer. 2016, pp. 850–865 (cited on page 109).

[227] Bo Li et al. 'High performance visual tracking with siamese region proposal network'. In: *Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8971–8980 (cited on page 109).

[228] Amelio Vazquez-Reina et al. 'Multiple hypothesis video segmentation from superpixel flows'. In: *European Conference on Computer Vision*. Springer. 2010, pp. 268–281 (cited on page 109).

[229] Radhakrishna Achanta et al. *SLIC superpixels*. Tech. rep. 149300. EPFL, 2010 (cited on page 109).

[230] Radhakrishna Achanta et al. 'SLIC superpixels compared to state-of-the-art superpixel methods'. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 34. 11. IEEE, 2012, pp. 2274–2282 (cited on pages 109, 113, 114, 116, 118).

[231] Xiaofeng Ren and Jitendra Malik. 'Learning a classification model for segmentation'. In: *International Conference on Computer Vision*. IEEE. 2003 (cited on page 109).

[232] Matthias Reso et al. 'Temporally consistent superpixels'. In: *International Conference on Computer Vision*. 2013, pp. 385–392 (cited on pages 109, 113).

[233] Alex Levinshtein, Cristian Sminchisescu, and Sven Dickinson. 'Spatiotemporal closure'. In: *Asian Conference on Computer Vision*. Springer. 2010, pp. 369–382 (cited on pages 109, 113).

[234] Ivan Laptev. 'On space-time interest points'. In: *International Journal of Computer Vision*. Vol. 64. 2-3. Springer, 2005, pp. 107–123 (cited on page 109).

[235] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Springer, 1994 (cited on page 109).

[236] Lena Gorelick et al. 'Actions as space-time shapes'. In: *Transactions on Pattern Analysis and Machine Intelligence*. Vol. 29. 12. IEEE, 2007, pp. 2247–2253 (cited on pages 109, 112).

[237] Guillaume Seguin et al. 'Instance-level video segmentation from object tracks'. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2016, pp. 3678–3687 (cited on page 109).

[238] Katerina Fragkiadaki et al. 'Learning to segment moving objects in videos'. In: *Conference on Computer Vision and Pattern Recognition*. 2015, pp. 4083–4090 (cited on page 109).

[239] Eddy Ilg et al. 'Flownet 2.0: Evolution of optical flow estimation with deep networks'. In: *Conference on Computer Vision and Pattern Recognition*. Vol. 2. 2017 (cited on page 110).

[240] Xiuming Zhang et al. 'MoSculp: Interactive Visualization of Shape and Time'. In: *The 31st Annual ACM Symposium on User Interface Software and Technology*. ACM. 2018, pp. 275–285 (cited on page 112).

[241] F. Perazzi et al. 'A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation'. In: *Conference on Computer Vision and Pattern Recognition*. 2016 (cited on pages 113, 114, 136).

[242] Chenliang Xu and Jason J Corso. 'Evaluation of super-voxel methods for early video processing'. In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1202–1209 (cited on page 113).

[243] Mark Everingham et al. 'The pascal visual object classes (voc) challenge'. In: *International Journal of Computer Vision*. Vol. 88. 2. Springer, 2010, pp. 303–338 (cited on page 113).

[244] David R Martin, Charless C Fowlkes, and Jitendra Malik. 'Learning to detect natural image boundaries using local brightness, color, and texture cues'. In: *Transactions on Pattern Analysis and Machine Intelligence*. 5. IEEE, 2004, pp. 530–549 (cited on page 113).

[245] MidJourney. *MidJourney: Home*. Accessed: 2024-06-05. 2024. URL: https://www.midjourney.com/home (cited on page 123).

[246] Knowing Machines. *Knowing Machines: The Key to Understanding AI*. Accessed: 2024-06-05. 2024. URL: https://knowingmachines.org/ (cited on page 123).

[247] Filip Ilic et al. 'Selective Interpretable and Motion Consistent Privacy Attribute Obfuscation for Action Recognition'. In: *Conference on Computer Vision and Pattern Recognition*. June 2024, pp. 18730–18739 (cited on page 135).

[248] Filip Ilic et al. 'A study on robust feature representations for grain density estimates in austenitic steel'. In: *Proc. Workshop of the Austrian Association for Pattern Recognition (AAPR/OAGM)*. 2021 (cited on page 135).

[249] Filip Ilic and Axel Pinz. 'Representing Objects in Video as Space-Time Volumes by Combining Top-Down and Bottom-Up Processes'. In: *Winter Conference on Applications of Computer Vision*. Mar. 2020 (cited on page 135).

[250] Hamidreza Ramezanpour et al. 'Object motion representation in the macaque ventral stream–a gateway to understanding the brain's intuitive physics engine'. In: *bioRxiv*. Cold Spring Harbor Laboratory, 2024, pp. 2024–02 (cited on page 135).

[251] Lea Bogensperger et al. 'Score-Based Generative Models for Medical Image Segmentation Using Signed Distance Functions'. In: *German Conference on Pattern Recognition*. Springer-Verlag, 2024 (cited on page 135).

[252] Lydia Lindner et al. 'Lightweight video denoising using aggregated shifted window attention'. In: *Winter Conference on Applications of Computer Vision*. 2023, pp. 351–360 (cited on page 135).

[253] Lea Bogensperger et al. 'On the Influence of Beta Cell Granule Counting for Classification in Type 1 Diabetes'. In: *Proc. Workshop of the Austrian Association for Pattern Recognition (AAPR/OAGM)*. 2022 (cited on page 135).