University of Applied Sciences Ulm

Bachelor Thesis

Spatio-Temporal Feature Extraction for Action Recognition in Videos

Author:
Lucas Mahler
mahler@mail.hs-ulm.de
3125303

Supervisors:
Prof. Dr. -Ing. Herbert Frey
Prof. Dr. -Ing. Philipp Graf

September 28, 2020

Declaration of Authorship

I, Lucas MAHLER, declare that this thesis titled, "Spatio-Temporal Feature Extraction for Action Recognition in Videos" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSITY OF APPLIED SCIENCES ULM

Abstract

Bachelor of Science

Spatio-Temporal Feature Extraction for Action Recognition in Videos

by Lucas Mahler

Autonomous robots and vehicles that primarily act in environments that inhabit humans require the recognition of human action from incoming sensory video data in order to form a complete scene understanding. This scene understanding is necessary for all high level functionality of the autonomous machine.
Motivated by this, this work reviews several methods for human action recognition and detection in videos. To recognize or detect actions, the extraction of spatio-temporal features is a prerequisite. Good performance of 2D CNNs in the domain of action recognition has proven that the spatial dimension contains indications for the present actions. 3D CNNs that symmetrical convolve the spatial and temporal dimension could improve the performance of 2D CNNs only moderately, giving hints that the temporal dimension requires special treatment. This thesis took inspiration from visual processing in mammalian brains to select a very deep two stream architecture, SlowFast, where the two streams represent the spatial and temporal dimension respectively. Furthermore, this thesis empirically demonstrates that the chosen SlowFast architecture has exceptional spatio-temporal modeling capabilities. This is supported by the high parameter utilization of SlowFast. Low runtime cost, high throughput and state-of-the-art accuracy underline the representational power of the architecture. Moreover, the methodical analysis reveals that hierarchical processing in SlowFast is a significant contributor to its performance. The two streams achieve high functional specialization for the tasks of modeling motion and form modeling with the same underlying computational principles. Channel capacity and temporal resolution are shown to have high responsibility in achieving said functional specialization.

Acknowledgements

I would first like to thank my supervisor Prof. Frey for continuous support and guidance. I also want to express my thanks to my supervisor at EDAG, Nathalie Schromm, whose expertise was invaluable in framing the research questions and approaches.

I would also like to acknowledge my colleagues at EDAG for fantastic collaboration. I want to especially thank Alexander Hirschle and Jacek Burger for all the opportunities I was given to deepen my research and for providing me with valuable tools and infrastructure that were required to successfully complete this thesis.

Additionally, I want to voice my gratitude to Patricio Yael Reller Garciy for the numerous insightful discussions, feedback and encouragement. To conclude, I want to thank my family and friends which have been a great source of support.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial Neural Network
AP	Average Precision
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
I3D	Inflated 3D Convolutional Neural Network
iDT	Improved Dense Trajectories
KLT	Kanade-Lucas-Tomasi Feature Tracker
LGN	Lateral Geniculate Nucleus
LTU	Linear Threshold Unit
mAP	Mean Average Precision
MLP	Multi-Layer Perceptron
R2D	2D Residual Network
ResNet	Residual Network
SGD	Stochastic Gradient Descent
STIP	Sparse Trajectories Interest Point
TSN	Temporal Segment Network
V{1..6}	Visual Cortex Area {1..6}

Chapter 1

Introduction

The recent advent of the deep learning revolution sparked massive interest in researching deep neural networks. GPUs get increasingly faster and allow the training of ever more sophisticated and extremely deep architectures, that were, even a decade ago, inconceivable. This research is continuously trickling down into all areas of industry and every day life. Deep learning is now used in a broad spectrum of use cases permeating all areas of business, from production to marketing[129, 83].

With internet and social media gaining widespread popularity, images and videos become the preferred communication media. At the same time the demand for cameras increases steadily and optical imaging technologies get better and better. These advances also inspired the computer vision community in pursuing more research in image and video understanding. The rise of convolutional neural networks showed the powerful learning capabilities in the realm of image recognition[61]. Recent advances in neural network architectures for learning video representations try to pursue the same history as neural networks for images.

1.1 Motivation

Progress in artificial intelligence research also affects the vision of the city of tomorrow. Thus posing significant challenges to the automotive industry, transportation, mobility and civil engineering. As the urbanisation increases, questions on how to realise the smart city of the future arise. Raising awareness to the mega-trends individualisation, digitalization and mobility. The effects of ever growing cities can already be noticed today, raising levels of congestion, smell, dirt, and increasing needs for readily available public transport.

EDAG Engineering GmbH tackles these issues by proposing a new mobility concept based on the *CityBot* Ecosystem. At the core of this ecosystem lies the CityBot which is a swarm-intelligent, fully autonomous, multifunctional vehicle for the city of tomorrow.

The CityBot can be equipped with various modules having integrated tools. CityBot add-ons allow, for example, the carrying of passengers or cargo, or city cleaning and maintenance devices.

In order to operate fully autonomously, the CityBot needs to actively understand its surrounding environment. A key actor in the CityBot's environment is the human, thus the CityBot needs to have a high-level understanding of what a human is doing in its environment to act and interact not only safely but also effectively[31]. An example scenario could be, while the CityBot, equipped with a park maintenance module, is trimming a tree in the city park, a girl is playing with a ball next to it. To ensure the safety of the girl and to predict her danger potential, the CityBot requires a in-depth understanding of its whole surroundings.

A crucial subcomponent of this aforementioned scene-understanding is the recognition of human actions. Human action recognition algorithms determine the present human actions in a video and as a consequence their goal is to correctly classify the displayed action. Main challenges that these algorithms face are, among other things, camera movement, background clutter, occlusions of the actor or shadows and sudden changes in illumination. Now the question arises, how to optimally extract the features present in the space and time dimensions of a given video[25, 23, 122, 123, 124, 94, 117, 118, 120, 112, 113].

1.2 Problem Statement

The core focus of this thesis is the investigation of techniques for extracting spatio-temporal features in videos and the implementation of an action recognition system. Specifically, it shall be addressed how spatio-temporal features are extracted and which architectural decisions need to be made to do so efficiently. As reference architecture serves the *SlowFast Network* from Feichtenhofer et al. [24].

1.3 Structure of this Thesis

This thesis is divided into 9 Chapters. After this introductory Chapter, **Chapter 2** details the fundamentals of learning, artificial neural networks, convolutional neural networks, residual neural networks and also gives an overview of significant datasets. **Chapter 3** serves as a ground of investigation of several methods on how to extract spatio-temporal features for the task of human action recognition. **Chapter 4** gives a brief introduction into visual processing in mammalian brains and draws conclusions for architecture design. **Chapter 5** details the SlowFast architecture and its implementation. In **Chapter 6** the architecture differences for action recognition and action detection are

explored and the focus is on preprocessing, training, inference and the performance of the SlowFast architecture. **Chapter 7** is divided into two parts, part 1 analyses SlowFast from a practical perspective and in part 2 a methodical analysis is carried out. **Chapter 8** summarizes the findings of this thesis and **Chapter 9** discusses the results and gives a brief outlook.

Chapter 2

Fundamentals

This chapter deals with the foundations necessary for this thesis. First learning in general and then artificial neural networks are covered. Perceptrons, Multi-layer perceptrons, optimization and backpropagation are dealt with in detail. The next section covers convolutional neural networks and related concepts and presents the AlexNet architecture. Residual neural networks are detailed and the evaluation of classifiers with accuracy, precision and recall is examined. Finally, relevant datasets for human action recognition and detection are introduced.

2.1 Learning

Learning, as a hypothetical construct in neuropsychology, can only be inferred from observed behaviour and implies a permanent change in behavioural performance. Thus learning as a neuropsychological concept is usually defined as a permanent behavioural change due to experience.[34] This concept of learning is also applied to algorithms that are able to learn from data. Mitchell provides a formal definition of learning that says that 'a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E'[84]

In common language, learning is often itself understood as the task, however Michell's formal definition states that learning is the mean of attaining the ability to perform the task. In machine learning tasks can be understood as how an example should be processed, where a collection of **features** makes up an example.
Typical machine learning tasks are for example classification and regression. In a classification task, the machine learning algorithm shall specify to which of k categories an input belongs to. Solving this task requires the machine learning algorithm to produce a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$ and, if $y = f(x)$ an input described by vector x is assigned to a category with a numeric identifier y[32, p. 96ff.].

For regression tasks, the algorithm is expected to predict a numerical value based on some input. Similarly to the classification task, the algorithm needs to find a function $f : \mathbb{R}^n \to \mathbb{R}$[32, p. 96ff.].

In order to evaluate the performance P of a machine learning algorithm on a specific Task T a quantitave performance measure needs to be defined. For a classification task, this could be the **accuracy** of the model, i.e. the ratio of correct to false outputs. For a regression task, this could be the absolute difference between the models output and the expected output. In principle the focus on performance evaluation lies in determining how well the algorithm performs on a **test set** of the data. This test set is disjoint from the training set[32, p. 96ff.].

Machine learning can be divided into three general categories: supervised learning, unsupervised learning and reinforcement learning. The learning category is determined by the kind of experience the algorithm is allowed to have during the learning process. The proposed algorithms in this thesis are allowed to experience an entire dataset.

Supervised learning or learning from examples is described by Russel et al. to learn a function that predicts the output for new inputs given a collection of input-output pairs.[103, p.693] A supervised learning algorithm thus experiences a complete dataset containing features, where each example is associated with a corresponding label.[32, p. 102] The task of supervised learning is defined as follows[103, p. 695]:

> Given a **training set** of N examples consisting of feature-label pairs
> $$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N),$$
> where each y_j was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f.

Where x and y can take on any value and the function h is a hypothesis. This form of learning will be the main focus of investigtion in this thesis.

Contrasting, in **unsupervised learning** the agent learns patterns in the input without explicitly specified output expectation. Hence the unsupervised learning algorithm experiences the dataset and tries to learn useful properties or tries to model probability densities over the inputs.[103, p. 695]

Reinforcement learning examines how an agent can optimally learn from rewards and punishments. The agent tries to find an optimal policy that maximizes the rewards in order to base its actions upon. Hence the task of a reinforcement learning method is to learn an optimal policy for the environment. Typically an environment is stated as a markov decision process.

2.2 Artificial Neural Networks

In order to create intelligent machines it is only logical to take inspiration from the brains architecture. The most important components of a biological neuron are its cell body, the branching extensions called *dendrites* and one especially long extension called *axon*. At the end of each axon, it splits into many smaller branches called *telodendria* whose ends connect via *synapses*, also called *synaptic terminals*, to dendrites of other neurons. These synapses transmit electrical signals between neurons, if a neuron receives a number of signals within a short period of time, approximately a few milliseconds, exceeding some threshold the neuron fires its own signals. The interconnection of the approximately 10^11 neurons in a human brain enables highly complex computations, eventhough the functioning of individual neurons is rather simple.[37, p. 253ff.] The mapping of neural connections in the brain is still largely undiscovered and an actively researched field but it has been discovered that neurons are organized in consecutive layers as Figure 2.1 shows.

In 1943 McCulloch and Pritts[80] first proposed a mathematical model of how biological neurons work together to perform rather complex operations, giving birth to the first artificial neural network architecture. An artificial neural network (**ANN**) is simply a collection of any number neurons, also called **units**, connected by a directed **link**. These links from neuron i to neuron j propagate **activations** x_i and each link is associated a **weight** $w_{i,j}$ value that determines the strength of the connection. Each neuron j computes a weighted sum given some input:

$$in_j = \sum_{i=0}^{n} w_{i,j} x_i \tag{2.1}$$

In order to obtain an output, the **activation function** g is applied to the previously calculated input in_j:

$$x_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} x_i\right) \tag{2.2}$$

Where the activation function g is usually either a fixed threshold, then the neuron is called a perceptron, or a logistic function. These nonlinear activation functions are crucial because they allow the network of neurons to represent nonlinear functions.[103, p. 727ff.].

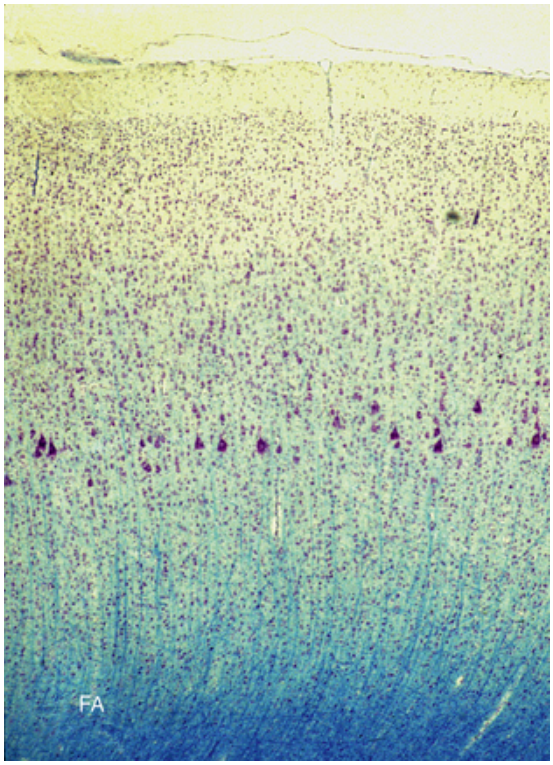


FIGURE 2.1: Cortical columns in the primary motor cortex of non-human primates[114]. The layered structure of neurons is clearly visible.

2.2.1 Perceptrons

In 1957 the perceptron was proposed ba Frank Rosenblatt. It's neurons differed from the ANN proposed by McCulloch and Pitts in that the inputs and outputs of each neuron are numbers instead of booleans and each input has an associated weight. Figure 2.2 shows simple ANN architectures performing logical calculations but by combining these simple networks complex logical expressions can be evaluated.

Rosenblatt called his neurons **linear threshold units** (LTU) because it computes the weighted sum of its inputs *in* and then applies a step function g, like the Heaviside step function, to said weighted sum in order to produce an output, see Equations 2.1, 2.2 and Figure 2.3 .

A perceptron, also called a single-layer neural network, is composed of a single layer of such LTUs and thus the inputs are directly connected to the outputs of the network. Typically the input of a perceptron is represented by special input neurons which just

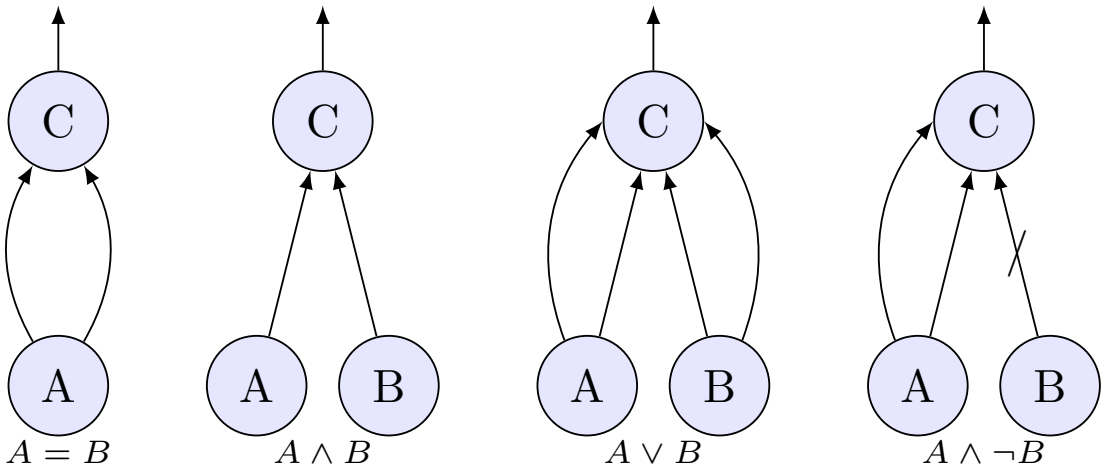


FIGURE 2.2: Boolean operations with neurons having binary inputs, as proposed by McCulloch and Pitts. Reproduced from [37, p. 256].

perform the identity function. Additionally, a bias feature $x_0 = 1$ is added by connecting a bias neuron that always outputs 1 to every LTU in the perceptron [37, p. 257ff.][103, p.729f.].

Since neurons in the ANN proposed by McCulloch and Pitts represent operations described by propositional logic, they argued that these neurons could be combined to perform any complex computational tasks, although they did not know how to train such networks. The computational power arises not from the inherent logic properties of a neuron but from finding the right values for the weights $w_{i,j}$. And since finding these weights by hand is not only extremely tedious but also very time consuming, Rosenblatt took inspiration from Hebbian learning in biological learing. Hebbs found out that when one biological neuron often activates another neuron their connection gets strengthened. He famously stated "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing It, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased"[45, p. 62]. Rosenblatt updated this principle to perceptrons such that connections leading to wrong output are not strengthened. This is done by making a prediction for each fed training instance and afterwards adjusting the weights of the connections from the input neurons such that the output neurons would produce

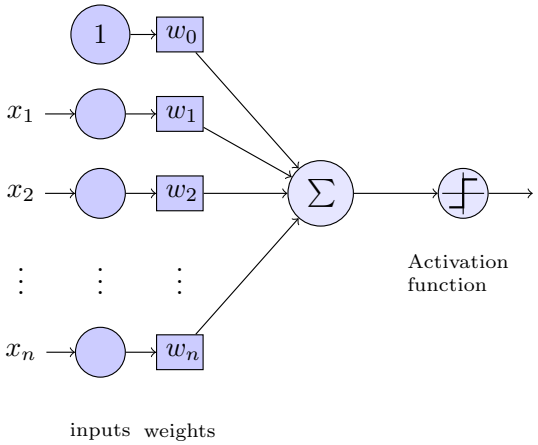


FIGURE 2.3: Linear Threshold Unit (LTU)

the correct result. Equation 2.3 shows this **perceptron learning rule**.

$$w_{i,j} \leftarrow w_{i,j} + \eta(y_j - \hat{y_j})x_i \tag{2.3}$$

Here, $w_{i,j}$ denotes the connection from the i-th input neuron to the j-th output neuron, x_i the i-th input value of the current training instance, $\hat{y_j}$ the output of the j-th output neuron of the current training instance, y_j the target or label to the j-th output neuron of the current training instance and η is the learning rate.[103, p. 729ff.] [37, p. 257ff.]

Rosenblatt also showed that perceptrons are not capable of learning complex patterns due to the linear decision boundary of each output neuron. For linearly separable datasets however, this weight update rule leads to convergence as Murphy et al. have proven in 2017 by using tools from symbolic logic with the proof assistant *Coq*.[86]

2.2.2 Multi-Layer Perceptrons

In 1969 Minsky and Papert demonstrated that perceptrons are incapable of solving simple classification problems, like an Exclusive OR classification, similar to other linear classification models. It was later discovered that these limitations could be overcome by stacking multiple perceptrons on top of each other. This resulting ANN is then called Multi-Layer Perceptron (MLP) or feedforward neural network[37, p. 260].

MLPs aim to best approximate a function f^*. For a classification task this function $y = f^*(x)$ is mapping an input x to a category y, and thus MLPs define mappings $y = f(x; \theta)$ and learn the parameters θ, i.e. the weights w, to best fit the function f^*. Figure 2.5 shows an examplary structure of an MLP. It consists of one input layer with

pass through neurons, one or more hidden layers and an output layer. As can be seen in the figure, every layer has a bias neuron that always outputs 1, hence it has no inputs. Every other neuron of the MLP is fully connected to the next layer and the strength of the connection ij is represented by the weight w_{ij}.
If a MLP consists of more than one hidden layer it is called a deep neural network (DNN). Because information flows from x through the intermediate computations in order to define f to produce an output y, DNNs are also called feed forward networks. MLPs posses no feedback connections, meaning no output of any layer is fed to an earlier layer, if said output is fed back, these networks are called recurrent neural networks.
Feedforward neural networks can be represented as a directed acyclic graph describing the composition of functions. A network composed of e.g. 3 different functions f^1, f^2, f^3 can be linked in a chain $f^3(f^2(f^1(x)))$. Function f^1 represents the 1st layer and similarily, function f^n represents the nth layer of the network. And the lenght of this very chain of functions is called the **depth** of the network, also giving birth to the name *deep learning*. The training data provides noisy samples from $f^*(x)$ that are evaluated at different training points, where each example x has a corresponding label $y \approx f^*(x)$. These training examples tell the output layer how it should look like, thus given an example x the output $\hat{y}$ of the output layer should resemble the label y as closely as possible. Altough the behaviour of the output layer is clearly defined by the label of an example, the behaviour of the other layers is not at all specified and thus learning must decide how best to represent f^*. The name *hidden layers* also stems from this fact that the training data does not specify how the output of these layers should look like.
DNNs are typically vector valued and the dimensionality of the hidden layers determines the *width* of the layer. Each element of the input vector thus corresponds to one neuron in the input layer. The individual layers however do not represent vector-to-vector functions, since each neuron acts in parallel it is rather a vector-to-scalar function. Each neuron in a hidden layer recieves its input from many neurons of the preceeding layer and then computes its own activation based on those inputs. This behaviour of using many vector valued representations of layers and using functions to determine representations of f^* is borrowed from neuroscience and how biological neurons compute their activations. Thus artificial neural networks are not models of the brain but rather methods to produce statistical generalization.[32, p. 163f.]

Linear models have the shortcoming of only being able to directly represent linear relationships. However, it is till possible to represent a nonlinear function f^* with a linear model. To do so, first a nonlinear transformation $\Phi(x)$ has to be applied that transforms the data into a Z-space in which the data is linearly separable. Figure 2.4 shows on the left side a dataset that is cleary not separable by linear models. However if 2nd degree polynomial features are added the data can be transformed to be linearly separable, as

shown on the right hand side of the figure.

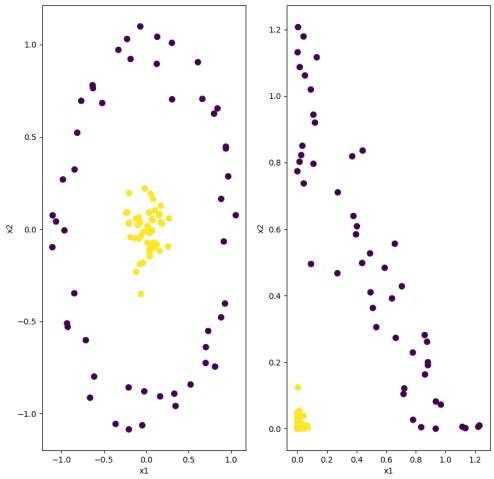


FIGURE 2.4: Nonlinear transformation by adding 2nd degree polynomial features.

Now the question arises how we can select an appropriate transformation Φ? Manually selecting Φ becomes increasingly tedious with larger and more complex datasets. Taking a very generic Φ will lead to poor generalization on the test set. Deep learning takes a different approach and tries to learn Φ by modeling $y = f(x; \theta, w) = \Phi(x; \theta)^T w$. The parameters θ are used to learn Φ, i.e. the hidden layer(s), from a borad class of functions. This is done by parameterization of Φ with θ; $\Phi(x; \theta)$ and then optimizing θ to find the best f resembling f^*. By using a broad family of $\Phi(x; \theta)$ this method be highly generic and also at the same time manually engineered Φ can be added, without needing to precisely select the function but only the general function family. This principle of learning the features of a given dataset applies to the whole field of deep learning and lends deep learning based methods their enourmous power.[32, p. 163.ff]

2.2.3 Optimization

As discussed earlier, the objective of a neural network is to define a function f that minimizes the dissimilarity to some target function f^*. The measure of this dissimilarity is called loss function, cost or error function, an example would be the mean squared

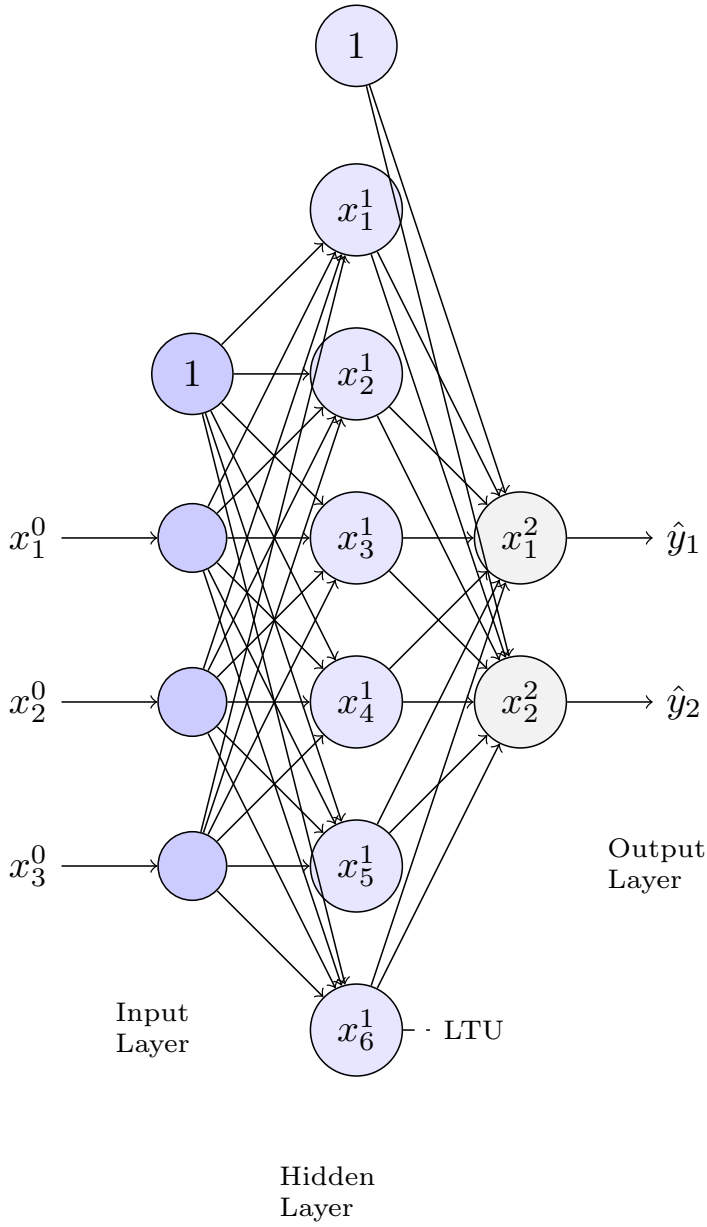


FIGURE 2.5: Multi-layer perceptron with one hidden layer.

error. This loss function maps the output of the network and the corresponding label from the dataset to a scalar value. To reach the objective the loss function has to be minimized for example by using **gradient descent**.

2.2.3.1 Gradient Descent

Gradient descent finds critical points of a given loss function $E(\theta)$ by calculating the gradient vector $\nabla E(\theta) = [\frac{\partial E}{\partial \theta_0}, \frac{\partial E}{\partial \theta_1}, \ldots, \frac{\partial E}{\partial \theta_n}]$. The critical values reside at the points where every element of the gradient vector is zero. Thus to minimize E the direction in which E decreases fastest needs to be found, that is the direction at the negative gradient. Then to obtain $\theta^{'}$, i.e. the parameters for the next step the gradient has to be subtracted from the current parameters θ:

$$\theta^{'} = \theta - \eta \nabla_{\theta} E(\theta) \tag{2.4}$$

Here η specifies the learning rate, that is the size of the step in the direction of steepest descent. This requires the calculation over the full training set for every gradient descent step until a minimum is reached. For linear models with relatively few data points this is feasible but for large datasets this method of calculating the gradient over the whole batch of training data, hence the name **batch gradient descent**, becomes immensly computationally complex and renders this method in practice infeasible. However, due to this property of considering the whole set of trainig examples, convergence is guaranteed and the models parameters θ move somewhat directly to an optimium.[37, p. 114ff.][103, p. 719f][32, p. 82f]

2.2.3.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) tries to tackle this problem of computational cost. SGD picks a random instance in the training set at every optimization step and computes the gradient vector ∇ based on this single instance. This improves the runtime of one optimization step but also due to the stochastic nature of the algorithm, it is less regular, meaning that the loss will bounce and only decrease on average. If a minimum is approximately reached, SGD will still bounce, since the next instance is again randomly picked. Thus optimal values can not be found because SGD can not settle and can, for example, jump out of local minima.
One strategy to mitigate this problem is to use simulated annealing[93]. The learning rate is decreased over time, first it is decreased with larger steps to enable quick progress and to escape local minima, then the steps get smaller and smaller such that the algorithm can settle at local minima. However, if the learning rate is reduced too quickly,

SGD can get stuck in local minima, if it is reduced too slowly, SGD will jump around and never find minima in feasible time.[37, p. 117f]

2.2.3.3 Mini-Batch Gradient Descent

To combat these shortcomings of SGD, nowadays a popular optimization method is Mini-Batch Gradient Descent, which samples a small set of examples from the training set and computes the gradients from these mini-batches. This approach combines the good convergence of batch gradient descent with the faster runtime of stochastic gradient descent. However, it is harder too escape local minima with mini-batch gradient descent than with SGD, due to the mini-batches. On the other hand, mini-batch gradient descent allows for more direct convergence, since the gradients are not only calculated on one instance but on a whole batch of instances, which allows for a better generalization. Furthermore, mini-batch gradient descent yields a significant performance boost on dedicated hardware like GPUs by optimizing matrix operations and the amount of memory scales linearly with the size of the mini-batch. Generally speaking, the larger the mini-batch is, the more accurate the estimate of the gradient becomes, however this does not scale linearly. Typically it is sought after to choose a large mini-batch size, 32 and up, although for large models this can be smaller, and the mini-batch size is typically a power of two in order to allow better runtime on GPUs.[37, p. 119f][32, p. 272]

2.2.3.4 Adam

Another commonly used optimization algorithm is Adam. It provides efficient stochastic optimization and only requires first order gradients with little memory constraints. Adam computes individual adaptive learning rates for different parameters based on first and second moments of the gradients and combines the advantages of AdaGrad[20] and RMSProp[115].
Adam updates exponential moving averages of the gradient and the squared gradient, where hyperparameters control the exponential decay rates of the moving averages. The moving averages are estimates of mean (1st movement) and uncentered variance (2nd raw movement) of the gradients. Since the moving averages are initializes as 0-vectors, the movement estimates are especially 0-biased during initialization and when the decays are very small. The magnetude of the parameter updates is invariant to rescaling of the gradient and the step size of an optimization step is approximately bounded by stepsize hyperparameters. Furthermore, Adam doesnt require a stationary objective, works with sparse gradients, and naturally performs a form of simulated annealing[60].

2.2.4 Backpropagation

As discussed in the section above, optimization algorithms require the computation of gradients. When a neural netwok makes a prediction $\hat{y}$ from an input x, information gets fed forward through the hidden layers until it reaches the output layer. This process of producing an output given an input is called **forward propagation** or forward pass. Forward propagtion continues further until a scalar cost is produced. In contrast, **Backpropagation** lets information flow from the cost backwards through the neural networks layers in order to calculate gradients. This was first proposed in 1986 by D. Rumelhart and G. Hinton[100] and was a turning point in how neural networks can be efficiently trained. Defining an analytical expression for the gradient is rather simple, it is simply the gradient of the chain of functions that the individual hidden layers represent. Evaluation of this expresseion however is computationally very expensive.
Neural networks, as described in section 2.2.2, can be described as a directed graph. Back propagation makes use of this representation and interpretes the neural network as a computational graph, where each **node** in a graph is a variable that takes the form of e.g. a tensor, vector or scalr. In order to formalize this graph, simple functions of one or more variables are understood as **operations** that return a node. If a function that the network implements is more complex than the set of allowed operations, it is broken down into a composition of allowed operations. For example if y is computed via an operation to obtain x, then the nodes x and y are connected with an edge. Figure 2.6 examplifies some functions represented as a computational graph.
The chain rule, in sum or in vector notation, is denoted as follows:

$$\frac{\partial z}{\partial x} = \sum_j \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x_i} \tag{2.5}$$

$$\Leftrightarrow \quad \nabla_x z = (\frac{\partial y}{\partial x})^T \nabla_y z \tag{2.6}$$

Where $x \in \mathbb{R}^m$, $y \in \mathbb{R}^m$, $g : \mathbb{R}^m \mapsto \mathbb{R}^n$ and $h : \mathbb{R}^n \mapsto \mathbb{R}$ and if $y = g(x) \& z = h(x)$, then $\frac{\partial y}{\partial x}$ represents the $m \times n$ Jacobian matrix of the function g.
This makes it easy to derive an algebraic expression by simply performing the Jacobian-gradient product for every operation in the computational graph, however it becomes increasingly more difficult to evaluate when the expression becomes more complex like in the case of neural networks. Due to subexpressions being repeated several times it needs to be decided if said subexpression is stored or recomputed and for complex graphs, like neural networks, there can be exponentially many of those subexpressions which renders inefficient implementations practically unusable. However, sometimes computing

the same subexpression twice may be a feasible tradeoff in order to safe memory. Algoritm 1 shows a pseudocode implementation of the backpropagation algorithm for deep feedforward neural networks.[32, p. 197ff][103, p. 734f]

Algorithm 1: Backward computation of a feedforward neural network wit l layers, an input x and a label y. This yields the gradients on the activations $a^{(k)}$ for each layer k, starting from the output and going backwards to the first hidden layer. From these gradients, the gradients of the parameters , e.g. weights and biases, of this layer can be obtained and can be immediately used to update them as part of an optimization update. Reproduced from [32, p. 206]

After forward computation, compute gradient of output layer:
$\mathbf{g} \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} E(\hat{y}, y)$
for $k = l, l-1, \ldots, 1$ **do**
  Convert gradient on layer's output into gradient into pre-nonlinearity activation (element-wise multiplication if f is element-wise)
  $\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$
  Compute gradients on weights and biases (including regularization term, where needed):
  $\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta)$
  $\nabla_{\mathbf{W}^{(k)}} J = \mathbf{g}\mathbf{h}^{(k-1)^T} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)$
  Propagate gradients with respect to nex lower-level hidden layer's activations:
  $\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)^T} \mathbf{g}$
end

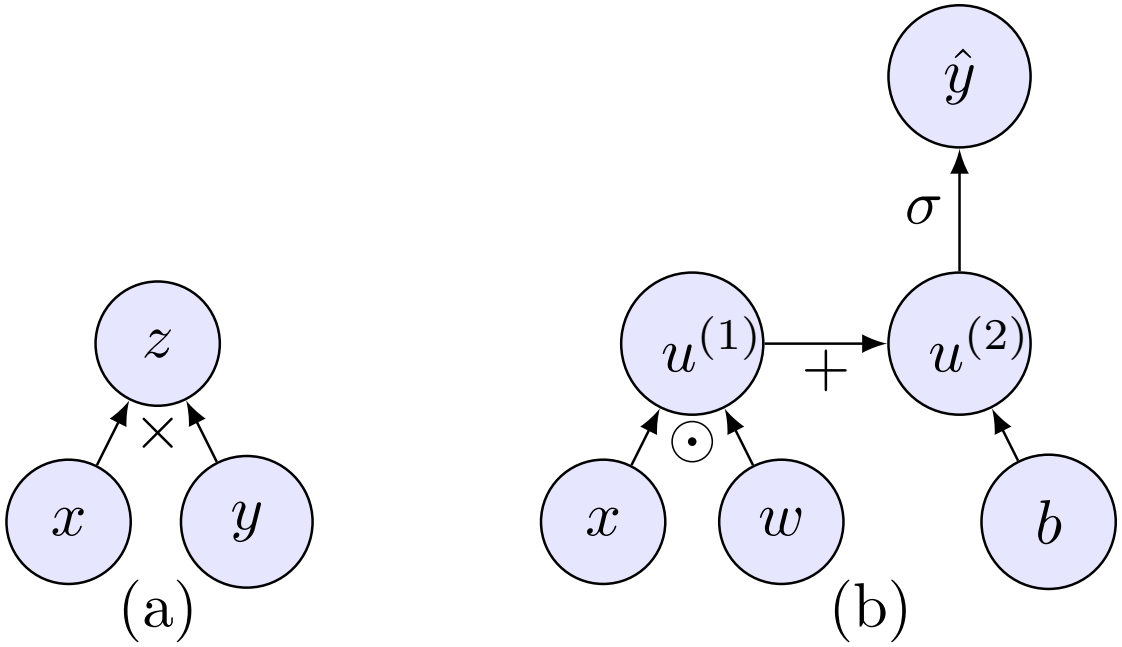


FIGURE 2.6: (a) shows a computational graph to compute $z = xy$ using the $\times$ operator. (b) shows the computational graph for the logistic regression prediction $\hat{y} = \sigma(\mathbf{x}^T\mathbf{w} + b)$. Intermediary nodes $u^{(1)}$ and $u^{(2)}$ are introduced in order to realize the computation. Partially reproduced from [32, p. 200]

2.3 Convolutional Neural Networks

The brains perception bandwidth is mostly occupied by optical information provided to the visual cortex. D. H. Hubel and T. Wiesel found in their 1959 landmark paper that neurons in the visual cortex react only to visual stimuli in a restricted area of the visual field, i.e. they have a small local receptive field. They also found that some neurons are only excited by horizontal edges and others only by vertical edges and furthermore found that these cells are connected in a layered structure where different layers produce different levels of abstraction.[51]
Once again, ideas from neuroscience were borrowed and led K. Fukushima to create the 'neocognitron', a hierarchical self-organizing network that learns to recognize stimuli paterns based on geometrical similarity.[27] This idea was further developed to what is now known as **convolutional neural networks** or in short CNNs.

CNNs were developed to work with grid structured inputs, for example grayscale images, or multiple stacked grids of values, like color images. Using MLPs for image processing is in practice infeasible due to the huge amount of parameters required. An image of size 512×512 already requires the inpput layer to have 262,144 neurons, and if the first hidden layer only has 1000 neurons, which already restricts the amount of information transmitted to the next layer, this would be 262,144,000 connections with their associated weights.[37, p. 356f]
Images display spatial dependencies in that adjacent spatial locations have similar values of individual pixels. Moreover, images are translation invariant, a banana upside down is still a banana. CNNs thus try to create similar feature values for similar input patterns, specializing on grid-like inputs to be able to scale models to very large size.

From 2011 onwards, CNNs led to significant imporvements in image recognition tasks and were able to set standards by lowering the error rate from 25% to 4% on the Imagenet[101] benchmark.

2.3.1 Convolution Operation

At the heart of every CNN lies the convolution operation. Convolution simply combines two functions of the same dimensionality. An illustrative example is taking position measurements with a laser sensor that provides a scalar output value $x(t)$ at each time step t. If we want to reduce the noise of the measurements by applying a weighted average function $w(a)$, wehere a is the age of the measurement, that gives more weight to recent measurements. If this is done at every measured time step, the resulting function $s(t)$ will provide us with a smoothed position estimate. This results in the 1D convolution

operation[32, p. 322]:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.7}$$

In the context of convolutional neural networks, the first argument is referred to as the input, the second argument as the kernel or filter and the ouput is referred to as feature map.

Since we want to apply convolution to grid data such as images that are typically represented as a 2-dimensional array of values, we need to sum over the two spatial axies. Furthermore, images have a finite size so we only sum over the size of the image. And since the images are 2-dimensional, the kernel also needs to be 2-dimensional. Thus the convolution operation for 2D grid-like data looks as follows:

$$\begin{aligned} S(i,j) = (I * K)(i,j) &= \sum_m \sum_n I(m,n)K(i-m, j-n) \\ &= \sum_m \sum_n I(i-m, j-n)K(m,n) \end{aligned} \tag{2.8}$$

Figure 2.8 shows a visualization of a 2D convolution with a sqaure kernel of typical size 3×3. Convolution has interesting properties that make it very suitable for useage in neural networks. Convolution is commutative, thus we can flip the kernel relative to the input, as can be seen in equation 2.8. Convolution has the property of linearity, if the input I is multiplied by some constant c, then the result of the convolution also multiplies by the same constant:

$$(c \times I) * K = I * (c \times K) = c \times (I * K) \tag{2.9}$$

$$\Leftrightarrow \quad (I_1 + I_2) * K = I_1 * K + I_2 * K \tag{2.10}$$

Associativity is also an important property:

$$(I * K_1) * K_2 = I * (K_1 * K_2) \tag{2.11}$$

And due to these properties, convolution is also linearly and x/y-separable:

$$I^{'} = (I * k_x) * k_y \tag{2.12}$$

if $K(i,j) = k_x(i) \times k_y(j)$.[10, p. 100ff.] Figure 2.7 shows how different kernels affect the resulting feature map.

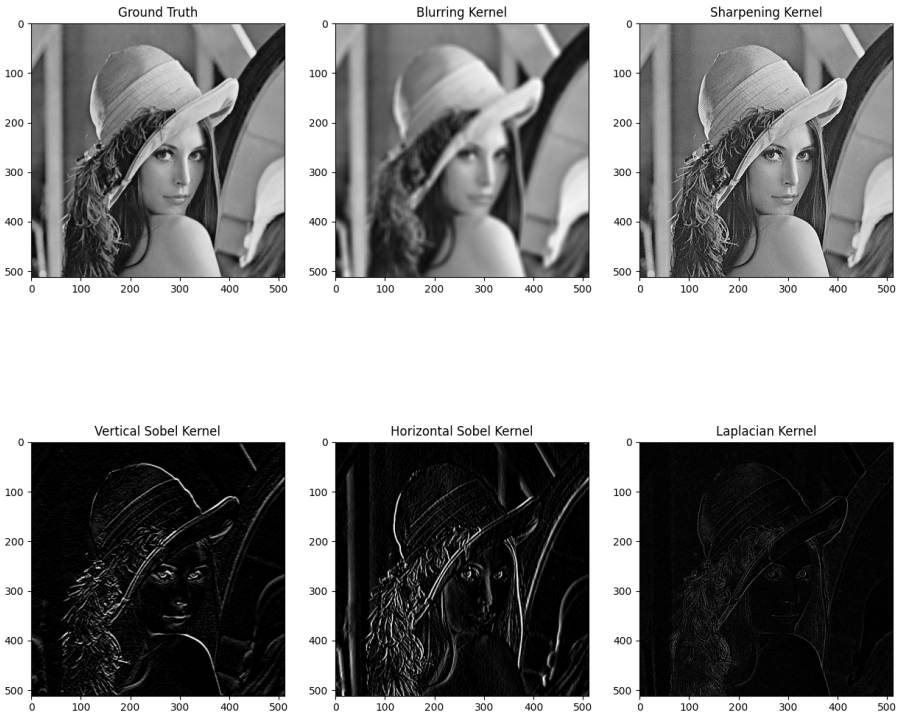


FIGURE 2.7: Different convolution kernels applied to the same input image.

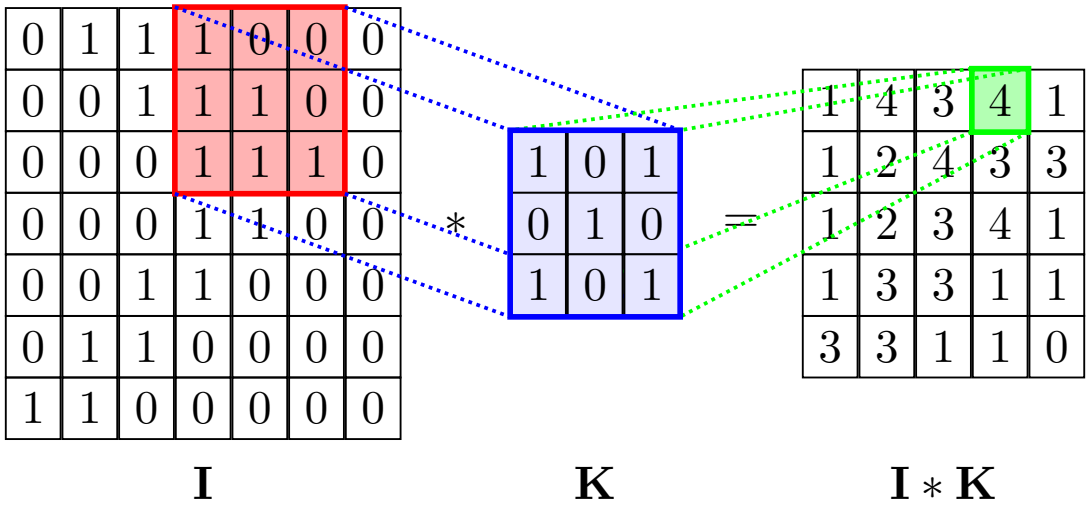


FIGURE 2.8: The input I is convolved with a kernel K of size 3×3 resulting in the output $I * K$. The red grid in the input shows the current convolution window with the associated scalar result in green in the output.

2.3.2 Convolutional Layers

A convolutional layer in a neural network arranges its states similar to its input in a grid structure and inherits relationships from one layer to the next. The basic functionality resembles that of a layer in a feedforward neural network, but unlike every output unit interacts with every input unit, convolutional layers are **sparsely connected.** This sparse connectivity arises from kernel sizes being smaller than the input sizes, and thus each unit in succeeding layer is only connected to $size(kernel)$ units. Figure 2.8 also shows this sparse connectivity. This enables the detection of features like edges even in inputs with millions of parameter, like high resolution images, with kernels that are only hundred pixels in size or even smaller. Hence these sparse connections save memory, improve statistical efficiency and the output computation requires fewer operations.

Another important concept of a convolutional layer is **parameter sharing** which, as the name implies, uses the same parameters for more than one function in the model. So instead of learning a weight for each connection in a convolutional layer, only the weights for the kernel of that layer are learned. The same kernel is applied to the whole input of the layer[32, p. 325ff.].

Let us consider a CNN that processes a 32×32 RGB image, thus its input has the shape $32 \times 32 \times 3$. The input layer L_0 of a CNN always has the same shape as the input

itself, later layers have the same organization in height, width and channels notation, short $H \times W \times C$, but can differ in size. Specifically the spatial dimension $H \times W$ reduces with later layers but the number of channels C increases. Each layer outputs at least one feature map, and feeds it to the next layer, similar to the values in the hidden layers of a MLP. The parameters θ of the Kernels K_q of the q-th layer are commonly organized in a square grid and by convention $H_{Iq} \times W_{Iq} < H_{Kq} \times W_{Kq}$. In a forward pass the 2D convolution is applied to the input of a layer with its associated kernel(s). Thus, the convolution places the kernel at each possible location in the input such that the kernel fully overlaps the input. Then the dot product is performed with the kernel as a vector of size $H_{Ki} \times W_{Ki} \times C_{Ki}$ and the input as a vector of size $H_{Ii} \times W_{Ii} \times C_{Ii}$, as visualized in 2.8. The result of each dot product is the feature at the corresponding position in the resulting feature map. The number of alignments of the kernel in the input defines the size of the next layer:

$$H_{q+1} = (H_q - K_q + 1) \tag{2.13}$$

$$W_{q+1} = (W_q - K_q + 1) \tag{2.14}$$

For our example with the 32×32 RGB image, the size of next layer L_1 is $H_1 = 32-5+1 = 28$ $W_1 = 32 - 5 + 1 = 28$ and thus 32×32 in the spatial dimension if a kernel of size $K_0 \times K_0 = 5 \times 5$ is used. But since the result of this convolution is a feature map with only 1 channel, due to the properties of the dot product, usually multiple kernels with their own sets of parameters are used for a convolutional layer. This is done in order to increase the number of parameters of the total network, which increases its representational capability. More feature maps require more kernels which in turn require more parameters. From this follows that increasing the number of kernels, or filters, in one layer increases the number of feature maps in the next layer.

CNNs processing image data typically use RGB images as inputs having 3 color channels. The number of feature maps generally increases in the later layers and can easily reach over 500 feature maps in the last layer. Each filter identifies spatial patterns in a small region of the image and in order to capture a broad variety of patterns lots of filters are required. The connection to Hubel and Wiesels[51] findings becomes visible again. They found that early visual fields detect simple shapes and late visual fields detect complex shapes, making again the connection to hierarchical feature engineering.

We can now use the concept of convolution to define the convolution layer. The weights of the p-th kernel $K^{p,q}$ in the q-th layer are represented by a 3-dimensional tensor $[w^{p,q}_{i,j,k}]$ and i, j, k indicate the position along the height, width and depth of the filter. $L^q_{i,j,k}$ is the feature map of the q-th layer represented as a 3-dimensional tensor $[l^q_{i,j,k}]$. The computation of the $q+1$-th layer from the q-th layer is thus defined as

follows[3, p. 318ff.]:

$$l_{i,j,p}^{q+1} = \sum_{r=1}^{K_q} \sum_{s=1}^{K_q} \sum_{k=1}^{d_q} w_{r,s,k}^{p,q} \, l_{i+r-q,j+s-1,k}^{q} \tag{2.15}$$

$$\begin{aligned} &\forall i \in \{1 \ldots, W_q - K_q + 1\} \\ &\forall j \in \{1 \ldots, H_q - K_q + 1\} \\ &\forall p \in \{1 \ldots, d_{q+1}\} \end{aligned}$$

2.3.3 Pads and Strides

The above described convolution operation and the resulting convolutional layer have one shortcoming. The resulting feature maps are smaller than the input because the kernels produce an information loss on the borders of the image. However this can be rather easily resolved by adding pixels around the border of the image such that the kernel can overlap the borders to compute the values at the border of the input resulting in a feature map with the same size.

Half-padding adds $(K_q - 1)/2$ 0-valued pixels around the borders of the image, thus increasing the spatial dimension by $K_q - 1$ which is exactly what a convolution with a K_q^2 filter size would decrease. Since the padded regions are filled with zeros, they do not contribute to the dot product and hence do not falsify the convolution with irrelevant features. Figure 2.9 shows a 0-padded matrix with a kernel size of $K_q = 5$.

Full-padding allows the full filter to stick out of the border of the input, and pads $K_q - 1$ 0-valued pixels around the borders. However, this increases the spatial footprint of the resulting featuremap and thus allows for the so called *deconvolution* by convolving the fully padded output of a no-padded convolution.

Another frequently used technique in combination with convolutions are **strides**. A stride determines the movement of the kernel. The basic convolution moves the kernel at each step in increments of 1 and thus the stride for the standard convolution is 1. When a stride of size S is set, the convolution is performed at the locations $\{1, S+1, 2S+1, \ldots\}$ generating a feature map of size $(H_q - K_q)/S_q + 1 \times (W_q - K_q)S_q + 1$.

Strides reduce the level of granularity and the spatial footprint of a layer. Moreover, strides increase the receptive field of each feature in the hidden layer, since more area is covered for a single convolution step and thus allows to capture more complex features in larger spatial regions of the input.[3, p. 322 ff.]

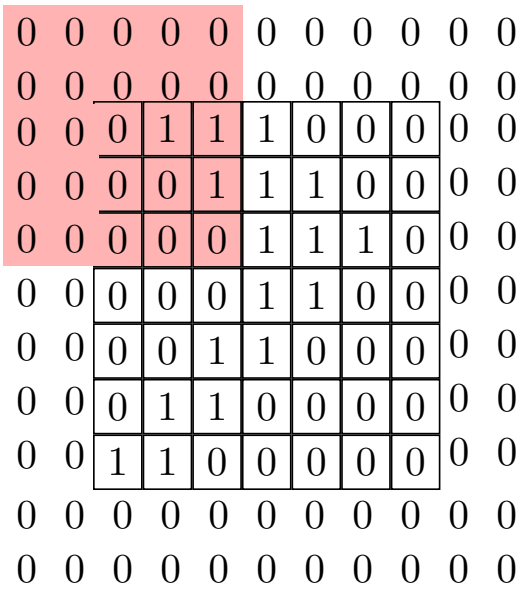


FIGURE 2.9: Half padding on a sample input with kernel size $K_q = 5$. The first position of the kernel window is shown in red.

2.3.4 Pooling Layers

Pooling is a commonly used component when designing neural network architectures. It is used for subsampling of feature maps in order to reduce the computational load, memory and the number of parameters and thus the risk of overfitting. This directly targets the huge memory requirements that processing grid-like inputs, i.e. images, comes with.

Each neuron in a pooling layer is sparsely connected to the outputs of the previous layer. These pooling neurons have no associated weights, they simply perform an aggregation function, for example average, max or L^2. The kernel size, i.e. the size of the receptive field, the padding method and strides can be set as hyperparameters. A pooling neuron only operates on the pixels in its immediate neighbourhood, specified by the kernel size hyperparameter. The max pooling operation is visualized in Figure 2.10 with a kernel size of 2, strides of 1 and with no-padding.

Pooling works on each channel independently and thus the output depth of the feature map of the pooling layer is the same as the input feature map.[37, p.365 ff.]

Another important property of pooling layers are that they make the preceding layers invariant to small translations of their inputs. This is important if we want to know

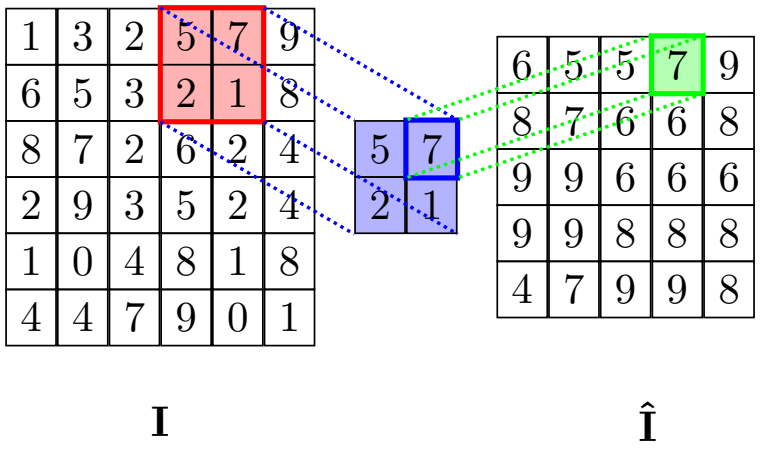


FIGURE 2.10: Max Pooling with a 2×2 kernel strides of 1 and no-padding.

whether a given feature is present but not where it exactly is. If pooling is applied to separately parameterized convolutions, those convolutions can learn which transformation to become invariant to. However, pooling is a rather destructive operation leading to a loss of information and decreases the parameters in the succeeding layers. For example, pooling with a 2×2 kernel and strides of 2 reduces the spatial dimension by a factor of 4.

2.3.4.1 Pooling and Convolution as Prior

Convolutions and pooling can also be understood as an infinitely strong prior distribution. A prior distribution encodes ones own beliefs in the parameters of the network. A weak prior, having high entropy, like a Gaussian distribution with high variance allows the learning algorithm to move the parameters rather arbitrarily. Contrary to that, a strong prior, having low entropy, like a Gaussian distribution with low variance predetermines where the parameters end up. Hence, an infinitely strong prior completely eliminates the possibility to learn some parameters, even if the data says otherwise, thus an infinitely strong prior assigns zero probability to some parameters. Since CNNs are not fully connected, the neural connections are predetermined by the receptive field of their convolutional kernels, a CNN can also be thought of as a MLP with an infinitely strong prior over its weights. Thus the weights of one neuron are the same as its neighbouring neuron, only shifted in space. The weights thus must be 0 for all other connections but this small assigned receptive field. This infinitely strong prior then says that the function

which the layer learns only contains local interactions that are equivariant to translation. Similarly, pooling is then an infinitely strong prior implying that the learned function is invariant to small translations. However if we want to implement a CNN as a MLP with an infinitely strong prior, it would be rather computationally wasteful to do so due to the extremely large number of 0 weights, although it is a nice model to aid thinking about CNNs.[32, p. 330ff.]

Convolutions and pooling may cause underfitting, because the same restrictions apply as with any priors, if the underlying assumptions made are unreasonable, the resulting output will also be inaccurate. So if the task at hand requires the preservation of exact spatial locations, pooling will introduce an unnecessary high error.
Furthermore, it should be noted that due to the aforementioned properties of the infinitely strong prior, convolutional models should only be compared to other convolutional models in statistical benchmarks. A convolutional model could never learn when all the pixels in a given input image are permuted, however a MLP would have no issues.[32, p. 334ff.]

2.3.5 ReLU Activation Function

The last building block required to build powerful CNNs are rectified linear units, simply called **ReLU**. ReLUs are used as an activation function and were introduced by V. Nair and G. Hinton[87]. Given some input z they simply use the max function to determine the activation $g(z) = max\{0, z\}$. This implies that ReLU simply outputs 0 across half of its domain. The derivative is 1 if the unit is active and producing a constant gradient and the second derivative is 0 almost everywhere providing a more useful gradient direction than other activation functions like *tanh* or *sigmoid*. Since it only requires a simple comparison in order to determine the activation, it is much more computationally efficient than the aforementioned sigmoid or hyperbolic tangent. ReLU also outputs true 0s unlike sigmoid and tanh activation functions providing representational sparsity and ReLU also does not require the input to be normalized in order to prevent saturation.

ReLU typically follows an affine transformation, like a convolution, and its good practice to set a low bias, e.g. 0.1 to the preceeding layers in order to allow gradients to flow. ReLU is typically used in virtually every CNN architecture, as A. Krizhevsky showed that it significantly outperforms other activation functions[63]. Since ReLUs are typically used after affine transformations, we use pooling afterwards to acquire the basic layering structure of CNNs. The following subsection will elaborate how this structure lead to the deep learning revolution.

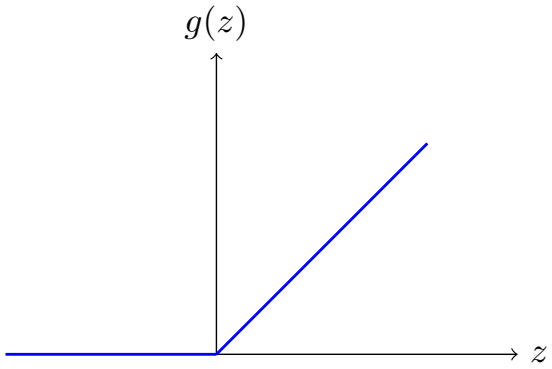


FIGURE 2.11: Rectified Linear Unit Activation Function.

2.3.6 AlexNet

Up until 2010 Deep Learning approaches have not been very successful for large scale image recognition tasks and were widely considered infeasible. This was a direct consequence of the lack of sufficiently large labeled image datasets since up to 2010 datasets with more than ten thousands of images were basically non-existing and also because of lacking computational resources. However, on small datasets like MNIST[69] with 60,000 training examples, CNNs could already achieve human-level performance with Top-1 error rates of $<0.3\%$. In 2009 Deng et al.[16] first presented their *ImageNet* database consisting of over 15 million images associated to 22 thousand classes. The popular *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*[102] used a subset of the ImageNet dataset consisting of 1.2 million examples on 1,200 classes with 1000 instances each. Combined with the more and more powerful GPUs that were available, due to the extreme popularity of computer games, and the extremely efficient implementation of 2D convolutions in general purpose GPU libraries, like NVIDIA CUDA, the major hurdles were being tackled.

These advances allowed A. Krizhevsky, I. Sutskever and G. Hinton in 2012 to train a very deep convolutional neural network on the ILSVRC dataset and considerably outperform the previous state of the art with a network having 60 million parameters and 650,000 neurons. Learning of thousands of objects from millions of training examples requires a model with large learning capacity, like a CNN where its capacity can be controlled by its depth and width. Compared to a standard feedforward neural network, Krizhevsky et al.'s CNN had fewer parameter, due to the aforementioned properties of convolutional layers, but was still too large to fit on the memory of a high end GPU. Utilization the possibility to distribute tasks on multiple GPUs, AlexNet was trained with

Layer	Type	Maps	Size	Kernel Size	Stride	Padding	Activation
IN	Input	3(RGB)	224 x 224	–	–	–	–
C1	Convolution	96	55×55	11×11	4	SAME	ReLU
P2	Max Pooling	96	27×27	3×3	2	VALID	–
C3	Convolution	256	27×27	5×5	1	SAME	ReLU
P4	Max Pooling	256	13×13	3×3	2	VALID	–
C5	Convolution	384	13×13	3×3	1	SAME	ReLU
C6	Convolution	384	13×13	3×3	1	SAME	ReLU
C7	Convolution	256	13×13	3×3	1	SAME	ReLU
F8	Fully Connected	–	4096	–	–	–	ReLU
F9	Fully Connected	–	4096	–	–	–	ReLU
OUT	Fully Connected	–	1000	–	–	–	Softmax

TABLE 2.1: Architecture of AlexNet

cross-GPU parallelization resulting in a significant speed up of the training procedure.

2.3.6.1 Architecture and Implementation

AlexNet's deep architecture, consisting of 5 convolutional layers, 2 max pooling layers and 3 fully connected layers, leveraged the key concepts described in this section and paved the way for modern deep learning methods.

ReLU was used as activation function and yielded a 6 times faster convergence to an error rate of 25% in comparison to the sigmoid and hyperbolic tangent activation functions. After the ReLU activations of the first and third convolutional layer a special normalization was used, *local response normalization*, to aid generalization and to encourage lateral initialization, as proposed by Krizhevsky et al. This specific form normalizes over spatially extending local input regions as follows:

$$z^i_{u,v} = a^i_{u,v}(k + \alpha \sum_{j=j_{low}}^{j_{high}} (a^j_{u,v})^2)^{-\beta} \text{ with } \begin{cases} j_{high} = min(i + \frac{r}{2}, f_n - 1) \\ j_{low} = max(0, i - \frac{r}{2}) \end{cases} \tag{2.16}$$

$z^i_{u,v}$ is the normalized output of the neuron in feature map i, at the current spatial location (u, v). a^i is the activation of the neuron after ReLU. k, r, α, β are tweakable hyperparameters and f_n is the number of feature maps.

Furthermore, data augmentation techniques were used to further enhance the datasets volume by performing label preserving image transformations.

2.4 Residual Neural Networks

The revolution sparked by AlexNet gave rise to deeper and deeper architectures and the importance of depth in neural networks was further strengthened by the hierarchical organization of features. The depth of the network controls the levels of features that can be extracted. But the ever increasing depth of neural networks shone light on the importance of the vanishing gradients problem and the degradation problem.
The deeper the network gets, the smaller the gradients get as they are being back-propagated through the network resulting in large gradients at the later layers and tiny gradients in the earlier layers[49]. Making optimization of these very deep neural networks exceedingly hard. However these vanishing gradients can be mitigated by using normalized initialization or batch normalization[53] but not completely eliminated.
When training neural networks accuracy saturates with increasing network depth and then degrades rapidly but, counterintuitively, adding more layers will result in lower training accuracy as shown by He et al. and Srivastava et al.[41, 110].

Residual Neural Networks (ResNets) try to tackle these problems in order to allow training of extremely deep neural networks.

2.4.1 Architecture and Implementation

Instead of layers directly learning a desired underlying mapping $H(x)$, ResNets should learn the residual mapping $F(x) = H(x) - x$ instead. This recasts the original mapping into $F(x) + x$, which is easier to optimize and if the identity mapping is optimal, i.e the desired mapping, it is easy to push the residual to zero whereas to directly fit an identity mapping by a stack of nonlinear layers.
Recasting into the original mapping is realised by *shortcut connections*[7, 97], where the outputs of one layer are skipped, i.e. copied, to the inputs of any other layer.

If we assume multiple nonlinear layers are able to approximate some complicated function $H(x)$ then they can also approximate their residuals $F(x) = H(x) - x$. He et al. thus let the layers fit the residual mapping $F(x)$ by stacking multiple layers into one residual block:

$$y = F(x, \{w_i\}) + x \tag{2.17}$$

x and y are respectively the input and output of the layers considered and $F(x, \{w_i\})$ represents the residual function to be learned. It is to be noted that the necessary shortcut connections to realize the residual blocks do not introduce extra parameters to the model.
Residual neural network architectures can, in theory, be realized with arbitrarily many residual blocks, but since the input and output of these residual blocks needs to be of

the same dimension in order for the shortcut connections to work, the depth is typically increased after 3 blocks. In order for the identity mapping to work, the extra dimensions are padded with zeros or by a linear projection. Figure 2.12 exemplifies how residual building blocks look like and how they form a residual neural network is shown in Figure 2.13.

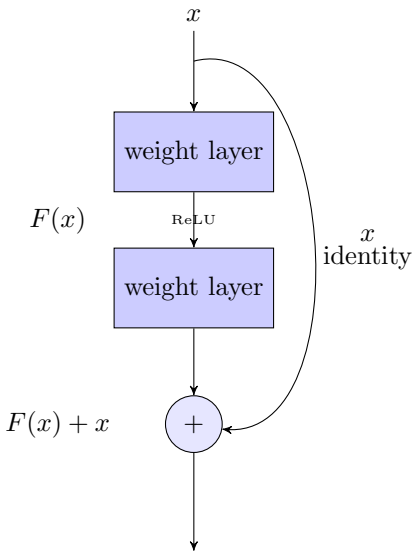


FIGURE 2.12: A residual block consisting of two layers learning the residual mapping $F(x) + x$. Reproduced from [42]

These characteristics allow for extremely deep networks with 152 layers and beyond, something that was previously impossible due to the vanishing gradient and the degradation problem. ResNets were thus able to achieve state-of-the-art performance and won the ImageNet ILSVRC2015 competition, outperforming the second place by 3.57% Top-5 error. Their generalization capabilities was also demonstrated by winning numerous other challenges[42].

2.5 Evaluating Classifiers

Measuring a models performance is of crucial importance not only to determine the error, or loss, of the model during training but also for validating the goodness of fit of a trained model in the validation or testing phases. When a trained model is evaluated the measure of performance is called *metric*.

When solving any given problem with deep learning, commonly, the following steps are followed, data acquisition, data exploration, data preprocessing, selection of suitable models and fine tuning of the selected model. For the last two steps the proper evaluation of the model is crucial.[37, p. 82ff]
It should also be noted that different methods of evaluation should be used for measuring the training error and the testing metrics since the model could simply learn to minimize this measure resulting in a loss of generality. During training, the measure of performance, the error, is calculated for a sample or a batch of samples whereas during testing the average of the metric over the whole test set is taken.

2.5.1 Accuracy

Different choices of metrics are available for different classification use cases however accuracy is the most common one. It is simply the number of correct predictions divided by the total number of predictions, or formally[32, p. 411]:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.18}$$

with

TP: True Positives, the number of correctly predicted positive classes

TN: True Negatives, the number of correctly predicted negative classes

FP: False Positives, the number of incorrectly predicted positive classes

FN: True Positives, the number of incorrectly predicted negative classes

2.5.1.1 Top-5 Accuracy

In many image and video recognition challenges the Top-5 accuracy is used due to multiple possible correct classification in one image or video clip. Thus the algorithm is allowed to return 5 predictions $c_{i1}, ...c_{i5}$ sorted according to the confidence of the algorithm. A prediction is counted as correct, if $c_{ij} \in C_i$ for some j and the set of correct predictions for a given class i is C_i. The error d_{ij} is then computed as follows[102]:

$$d_{ij} = d(c_{ij}, C_i) \begin{cases} 1 \text{ if } c_{ij} \neq C_i \\ 0 \text{ else} \end{cases} \tag{2.19}$$

The error on a test set is determined in a similar manner:

$$\text{Top-5 acc} = \frac{1}{N}\sum_{i=1}^{N} min(d_{ij}) \tag{2.20}$$

2.5.1.2 Top-1 Accuracy

Similar to the top-5 accuracy, the top-1 accuracy regards a list of predictions returned by the algorithm that are sorted according to confidence. If the prediction with the highest confidence is not correct, the algorithm is penalized[102].

$$d_{ij} = d(c_{i,max}, C_i) \begin{cases} 1 \text{ if } c_{i,max} \neq C_i \\ 0 \text{ else} \end{cases} \tag{2.21}$$

2.5.2 Precision & Recall

If a rare event needs to be detected, accuracy is not suitable. Let's assume the rare event has a probability of occurrence of 0.001, if a binary classifier now simply negates all samples it will still achieve an accuracy of 99.9%. Thus accuracy is not a well chosen metric for this task.

Precision p of a classifier measures what proportion of positives was actually correct. That is, precision measures the accuracy of positive predictions and is formally defined as:

$$\text{p} = \frac{TP}{TP + FP} \tag{2.22}$$

Recall r of a classifier measures what proportion of actual positives was predicted correctly. Or in other words, the fraction of true events that were identified correctly:

$$\text{r} = \frac{TP}{TP + FN} \tag{2.23}$$

If the rare event example is considered, a binary classifier that will always predict the event is not present, will have perfect precision but a recall of 0. If, however, the classifier always predicts the event is present, it will have a precision of 0.001 and a perfect recall. Commonly, the precision is plotted against the recall in a PR curve. When the performance of a classifier shall be expressed by a single scalar, the F_1**-Score** is used[32, p. 411f.]. It is simply the harmonic mean of precision and recall.

$$F_1 = \frac{\text{p} \times \text{r}}{\text{p} + \text{r}} = \frac{TP}{TP + \frac{FN+FP}{2}} \tag{2.24}$$

However, it should be considered that the F_1-Score favors classifiers with $p \approx r$ which is not always desired[37, p. 85ff].

2.5.2.1 AP & mAP

Average Precision (**AP**) is a popular metric reported on many pre-trained models and benchmarks. It calculates for every instance i of a class c in a given test set the precision and takes its average:

$$AP = \frac{1}{N} \sum_{i=1}^{N} p(c_i) \tag{2.25}$$

Also often used is the mean average precision, or in short **mAP**. Which is simply the average of the AP scores of all classes and is often used to compare different multi-class classifiers.[22]

2.6 Datasets

Datasets are a key component of the deep learning process. And without a high quality dataset the full potential of a model can not be unveiled. Challenges in developing a dataset include insufficient quantity of training data, nonrepresentatitve training data, poor-quality data and inclusion of irrelevant features[37, p. 22ff.]

In the field of image recognition ImageNet marked a milestone, successfully tackling the challenges and problems posed when composing large-scale image datasets. It is publicly available, large-scale with approximately equal class distributions, diverse set of classes, high variance in images, e.g. lighting conditions, backgrounds, camera positions etc., and high quality image-level annotations. The following presented action recognition datasets strive to pursue the same history as ImageNet but in the video domain and are industry-wide accepted and recognized as benchmarks for testing a models performance as well as for training novel models.

However, it shall be noted that other large scale datasets exist[57, 85, 1, 105]. Despite their size, these datasets are often only used for pre-training due to a lack of the aforementioned quality criteria. And hence, they will not be further discussed.

2.6.1 HMDB51

The large amount of available still image classification datasets [47, 62, 102, 22] and the limitations of action recognition video datasets up to 2011[98, 66, 78, 74, 88], like static backgrounds, no clutter or occlusions, indifference in camera point of view and static

camera motion, single actors and, most importantly, very limited size, inspired the creation of the *Human Motion Database* HMDB. Furthermore, HMDB was motivated by the nonrepresentativity of the aforementioned datasets which lead to very high recognition rates and poor generality.

HMDB51 consists of 51 distinct action classes with a minimum of 101 clips per class, making a total of 6766 clips and each clip has a video level label assigned to it. Meta tags are also supplied, containing information about viewpoint, camera motion, video quality and number of actions in the clip. The height of the clips is fixed to 240px and the width is scaled such that the aspect ratio is kept. All clips are converted to 30 frames per second. Every class is grouped into 5 general action categories: general facial actions, general body movements, facial action with object manipulation, body movement with object interaction and body movement with human interaction. Since Kuehne et al. assumed that camera motion interferes with local motion computation video stabilization by image stitching was applied to $\frac{2}{3}$ of the clips.

Comparisons to earlier datasets showed that HMDB51 has the lowest mean color in HSU colorspace, proving that scenes and actors in HMDB have high variability and offer less clues from lower level features, unlike datasets with constant background and actors. Kuehne et al. also showed that unlike with older datasets HMDB action categories could not be predicted by scene descriptors and low level features. These properties of the data established HMDB51 to a standard benchmark in video action recognition and resulted in high acceptance in the academic community[64]. Figure 2.14 shows some example images from the dataset.

2.6.2 UCF-101

Similar to HMDB51, the UCF-101 dataset tries to solve the 2 major disadvantages of video datasets for action recognition, low number of classes, especially compared to large-scale image recognition datasets like ImageNet, and the presence of controlled environments.
UCF-101, in short for University of Central Florida 101, contains 13320 videos in 101 classes, making it double the size of HMDB51 with almost two times as many classes. Classes can be fit into 5 categories: human object interaction, body motion, human-human interaction, playing musical instruments and sports. Individual clips of these classes are put into 25 groups consisting of 4-7 clips each. Clips in a group share similar feature, like background or number of actors. UCF-101 consists of web videos in unconstrained environments, with camera motion, varying lighting conditions, partial occlusions etc. The mean clip length is 7.2s, each clip has 25 frames per second and a resolution of 320 by 240[109]. UCF-101 was assembled by clips from UCF-11[104], UCF-Sports[98] and UCF-50[95]. UCF-101 is, similar to HMDB51, widely used as a

benchmark for fine tuned models and has big acceptance in the research field of action recognition. Sample images can be seen in Figure 2.15.

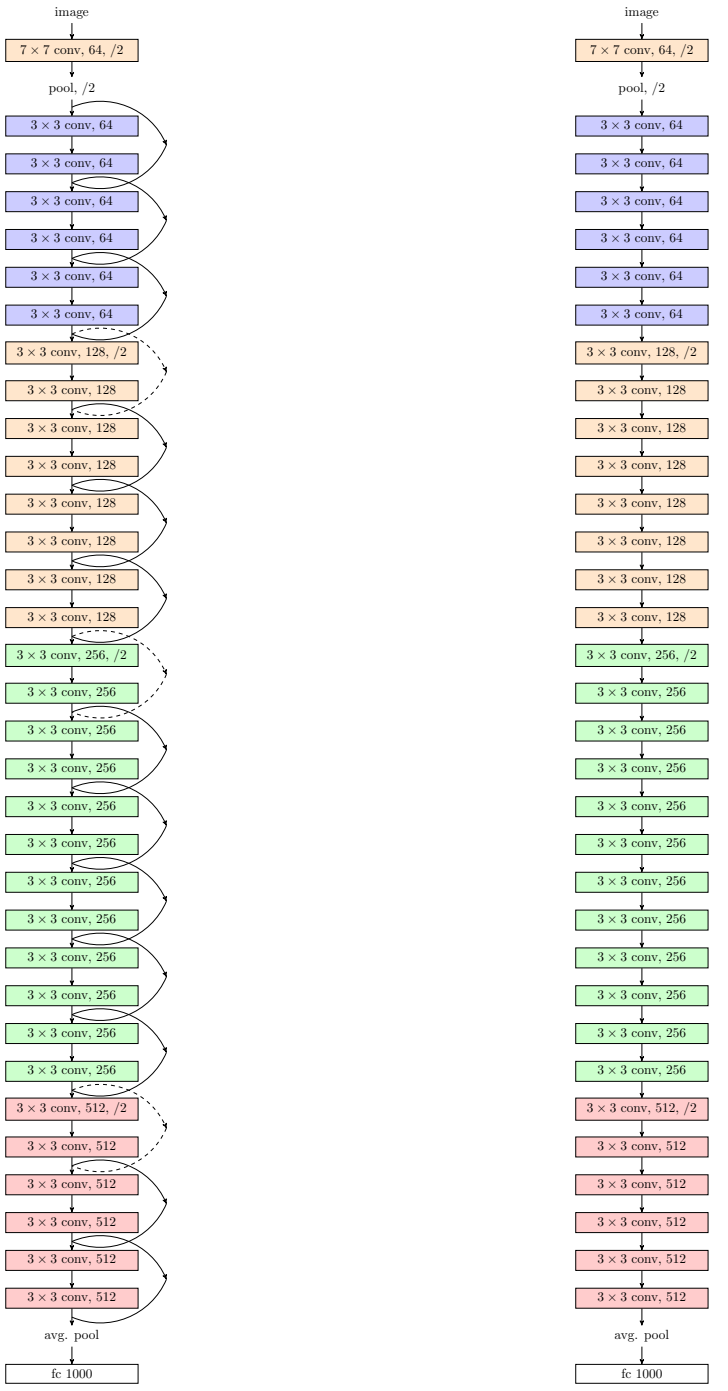
2.6.3 Kinetics

Similar to how ImageNet revolutionized image datasets in 2012, the Kinetics dataset had an equivalent effect to video datasets in 2017. Previous datasets had comparatively few classes and instances, UCF-101 and HMDB51 only 101 and 51 respectively and botch only had only slightly more than 100 instances per class. These small dataset sizes made it practically impossible to train very deep networks from scratch and thus those datasets act mostly as benchmarks or for finetuning. So, Kinetics can be seen as the successor of UCF-101 and HMDB51. The shortcoming of UCF-101 and HMDB51 is its low variance in comparison to image recognition datasets like ImageNet, with only a maximum of 101 actions selected from only 2500 videos. Kinetics selects each of its clips from different videos which are not professionally filmed, have camera motion, illumination variance, shadows, background clutter, and a great variety of performers or actors. The focus of Kinetics is in pure human actions, that fit in the categories single person action, person-person action and person-object action. The first iteration of Kinetics, *Kinetics400*[59], consists of 400 classes with 400-1150 clips each and a mean clip duration of 10s. Each clip contains more than one label, e.g. "texting" while "driving", thus usage of top-5 accuracy is encouraged. The dataset was obtained by acquiring videos from YouTube, selecting the temporal position and manually annotating them.

In 2018, the second iteration of Kinetics was released, *Kinetics600*[13]. The goal was to replicate the success of ImageNet. Kinetics600 follows the same principles as its precursor, with 10s YouTube clips of varying frame rate and resolution. Again, for any class, all clips were taken from different videos. Kinetics600 is an approximate superset of Kinetics400, 368 out of 400 classes are the same but with more examples, resulting in a total of 495,547 clips in 600 classes.

Kinetics700, the next iteration, released in 2019, is also an approximate superset of Kinetics600 but with 700 classes. The amount of clips per class has been increased to at least 600, increasing the number of clips to Kinetics600 by 30%, totalling in 650,317 clips[14]. Some example images can be seen in Figure 2.16.

Furthering the scope of Kinetics to not only provide labels for action recognition but also for action detection/localization, the *AVA-Kinetics*[70] dataset was released in 2020. The dataset contains expensive annotation, every person in a subset of frames gets a set of individual labels in key frames, providing a total of 1.6 million bounding boxes in 238906 clips in 80 classes. AVA-Kinetics was collected by adding clips from Kinetics700 and annotating them to the existing AVA[36] dataset.



(A) A 34-layer ResNet (B) A 34-layer convolutional neural network

FIGURE 2.13: Exemplary architectures for ImageNet classification. A) shows a 34-layer ResNet with 3.6 billion FLOPs, arrows represent an identity mapping, dotted arrows represent identity mappings with dimensionality increase. B) shows a plain 34-layer convolutional neural network. Partially reproduced from [42].

FIGURE 2.14: Randomly sampled frames from HMDB51.

FIGURE 2.15: Randomly sampled frames from UCF-101

FIGURE 2.16: Randomly sampled frames from Kinetics700. The variation in image quality and resolution is clearly visible in the middle frame.

Chapter 3

Comparison of Methods for Action Recognition

The literature about video action recognition can be split into four different approaches. Action recognition with hand-crafted features, 2D CNNs, 3D CNNs and CNNs with decomposed convolutions . All but the last approach use convolutional neural network based methods and since the first CNN achieved state-of-the-art performance hand-crafted feature methods could not keep up. This work explores how to efficiently combine the most successful insights from the first three approaches. Classical algorithmic approaches have long been obsolete and thus are not further mentioned in this work.

3.1 Hand-Crafted Features

Before the deep learning revolution, sparked by AlexNet, methods tracking spatio-temporal points yielded the best results in the field of action recognition. Pre-deep learning, two techniques to do so were prevalent. Sparse point detection and dense point detection.

3.1.1 Sparse Interest Point Detection

Interest points or local image features are abstract representations of structural patterns in a given image. Interesting events in video are characterized by strong variations in data along spatial and temporal dimension. Points with non-constant motion correspond to accelerating local image structures that belong to accelerating objects in the real world. These points contain information about forces acting in the real world changing these local image structures. Harris and Förstner interest points describe significant local variation of image intensities in the spatial dimension and provide high information content and relative stability to perspective transformations[28, 40].

Spatio-temporal interest points (STIP), extend the ideas from Förstner and Harris to the spatio-temporal domain. The resulting local space time features from STIP correspond to events in a given video. Events with different spatio-temporal extents are captured by computing spatio-temporal points in space scale and selecting scales that roughly correspond to the size of the detected events in space and durations in time. An action can then be identified by using k-means clustering and point descriptors by spatio-temporal image derivatives[65, 67]. The tracking of these interest points showed impressive results for action recognition, e.g. [79, 82, 55]. Messing et al. used tracking of Harris3D interest points with KLT trackers[77] for extracting feature trajectories. These trajectories are then clustered, then for each cluster center an affine transformation matrix is computed, where the matrix elements represent the trajectories.

3.1.2 Dense Interest Point Detection

Tracking sparse interest points with a tracker, e.g. KLT, to acquire interest point trajectories is computationally very expensive. Wang et al. investigated if dense sampling at regular points in time could improve runtime and performance and showed that it indeed does outperform the state-of-the-art STIP method at that time[124].

Wang et al. later proposed an efficient way for dense trajectory extraction. Acquisition of trajectories is done by tracking densely sampled points using pre-computed optical flow fields. The robustness of trajectories is ensured by imposing global smoothness constraints on the optical flow fields and the problem of camera motion is overcome by introducing new local descriptors focused on foreground motion[123] This dense sampling technique of feature points in each frame can also be extended by tracking them using optical flow. These improved Dense Trajectories (iDT) had state-of-the-art performance on the task of action recognition and had proven to be a hard competitor for deep learning methods[122].

3.2 2D Convolutional Neural Networks

The easiest and most straight forward approach to leverage deep learning for action recognition in videos is to simply feed consecutive individual frames into a 2D CNN, accumulate the predictions and choose the class with highest frequency of occurrence. Qiu et al. investigated this method with a ResNet-152 on the UCF-101 dataset and achieved competitive results[94]. Another fairly simple approach is, to reshape the 4D input into 3D and then feed the reshaped input into a 2D CNN. But there is one major drawback to these approaches. Vanilla 2D CNNs neglect the temporal dimension and

thus lack temporal context. This fosters Qiu et al.'s insights and is also supported by iDT outperforming the naive ResNet-152 approach on the video action recognition task.

However, other 2D CNN approaches try to take the spatial dimension into account in order to receive better recognition performance. These approaches can be divided into two categories, two-streamed CNNs and Temporal Segment Networks, which will be discussed in the following.

3.2.1 Two-Streamed Convolutional Neural Networks

The basic idea behind two-streamed architectures is that video can be decomposed into a spatial and a temporal dimension. Thus the recognition architecture is divided into two parallel streams. Each stream consists of an individual 2D CNN, one handling spatial information and the other temporal information. The two streams are then merged before formulating a prediction. In principle, the CNN architecture of the two streams can be chosen arbitrarily.

Simonyan et al.[106] proposed this idea in their 2014 paper, where the spatial stream performed action recognition on still frames and the temporal stream shall recognize motion from pre-computed dense optical flow. The advantage of using a still frame image classifier is, that it can be pre-trained on a large-scale image dataset and then finetuned on the vide action recognition dataset. They used a CNN consisting of 5 convolutional layers, 3 pooling layers, 2 fully connected layers and a softmax output layer in both of the streams and fused them at the softmax layer with a support vector machine. As mentioned, the spatial stream is fed still frames, whereas the temporal stream gets optical flow displacement fields stacked between several consecutive video frames. Using this architecture Simonyan et al. were able to achieve state-of-the-art performance on UCF-101 and similar performance on HMDB51 as iDT.

Feichtenhofer et al.[24] however demonstrated that it is better to fuse the two streams at a convolutional layer and not at the output softmax layer. They also showed that fusing spatially at the last convolutional layer and additionally at the softmax layer leads to extra performance boosts. Moreover, pooling of abstract convolutional feature maps over their spatio-temporal neighbourhood further boosts performance. These insights led Feichtenhofer et al. to achieve state-of-the-art performance on both HMDB51 and UCF-101, surpassing hand-crafted feature methods by 4% AP.

3.2.2 Temporal Segment Networks

The two-stream architecture as initially proposed by Simonyan et al. has proven to be very powerful and due to that they have been subject to thorough research. In 2016,

Wang et al. proposed a new architecture based on this two-streamed approach, Temporal Segment Networks (TSN)[125]. They postulate that since consecutive video frames are highly redundant dense temporal sampling is unnecessary and sparse temporal sampling of frames should be more suitable. Their TSNs sparsely samples short snippets of video clips which are uniformly distributed over the temporal dimension, e.g. every other frame is sampled. Allowing the TSN to aggregate long-range temporal information over the whole video. Temporal Segment Networks are composed of spatial and temporal streams and the sampled snippets are individually fed into two-streams respectively. Each spatial model of the spatial and temporal streams gets a single frame input from the corresponding video snippet and each temporal model gets a stacked RGB-difference or alternatively warped optical flow as input. To form a prediction, the consensus of each individual stream pair is taken. As network architecture for each spatial and temporal stream the Inception network with Batch-Normalization is used.[112, 53] This architecture outperformed all existing models in 2016 on the UCF-101 and HMDB51 benchmarks.

Other architectures building upon two-streamed and temporal segment architectures followed, e.g. [120] using long-term temporal convolutions or [134] using temporal relational reasoning, and lead to competitive results. However these architectures prove to be fairly complex and have not got much research attention, hence they will not be further discussed here.

3.3 3D Convolutional Neural Networks

As discussed in the previous section, 2D CNNs lack the explicit ability to model temporal structure. Thus additional information has often to be supplied, i.e. in form of optical flow. A further problem with a 2D CNN approach to video action recognition is that the spatial and temporal dimension are treated separately. 3D CNNs try to eliminate these problems by performing 3D convolutions, that is convolving the spatial and temporal dimension simultaneously.

In 2010 Taylor et al.[113] proposed the first 3D convolutional neural network for action recognition. They used a Restricted Boltzmann Machine[108] with 3D convolutions to tackle the, at that time, hand-crafted features dominated domain of action recognition. Their model was able to extract motion-sensitive features from image pairs. In order to eliminate the expensive annotations problem their model was trained unsupervised.

In 2014 Tran et al. created the novel C3D architecture[117]. It consisted of simple fully convolutional neural network with 8 convolutional layers, 5 pooling layers, 2 fully connected layers and a softmax output layer, similar to the 2D architecture used in both streams of Simonyan et al.[106]. Tran et al. showed with this architecture that

synchronously convolving spatially and temporally with 3D convolutions results in good feature learning of appearance and motion in a synchronous manner. I3D outperformed existing models on UCF-101 and achieved state-of-the-art.

Carreira et al. used proven 2D image recognition architectures and inflated 2D convolutions into 3D[12].Parameters and filters were also bootstrapped from 2D into 3D by implicit pretraining on single image datasets by duplicating single images to produce a static video consisting of the same image. Furthermore, they applied the aforementioned two-stream architecture with their inflated 3D CNNs and were able to outperform all prior methods on the key action recognition benchmarks HMDB51 and UCF-101, marking a new milestone for video models. Carreira et al. also showed that pre-training existing architectures on the newly released Kinetics dataset significantly boosts action recognition performance on the benchmark datasets, thus concluding that there is considerable benefit by pre-training on massive datasets.

3.4 Deomposing Convolutions

3D CNNs provided significant progress to the field of video action recognition. However, a 3D CNN has a quadratic growth of parameters in contrast to a 2D CNN with the same architecture. Thus the memory and runtime constraints grow significantly. Decomposing convolutions aim to reduce the runtime and memory demands posed by 3D CNNs without compromising on the benefits provided by simultaneously convolving space and time.

The novel architecture proposed by Qiu et al.[94] consisted of new bottle neck building blocks that use spatial and temporal filters. These building blocks use $1 \times 3 \times 3$ and $3 \times 1 \times 1$ convolutional layers either in parallel or cascaded. This allowed pre-training on 2D images and thus leveraging important priors. Qui et al. also proposed a novel Pseudo3D ResNet architecture composing each of the previously mentioned building blocks in different position in the ResNet architecture. The building blocks were either placed in parallel in a cascade or in a combination of both. This allowed Qui et al. to achieve state-of-the-art performance in all major benchmarks.

Tran et al. proposed a number of methods on how to best decompose 3D convolutions for action recognition[118]. One approach hypotheses that temporal modeling is particularly useful in the early layers, in late layers, that is, layers with higher levels of semantic abstraction, temporal modeling is not necessary. Thus in their MCx an rMCx architecture they mixed 2D and 3D convolutions and used 3D convolutions in early layers and 2D convolutions in late layers in a 3D ResNet architecture with 5 groups of residual blocks, where the last group is replaced by 2D convolutions.
The counter hypothesis was, that temporal modeling is more beneficial in late layers, thus replacing the earlier layers of the 3D ResNet architecture with 2D Convolutions.

However, both of these approaches lead to unsatisfactory results both in performance as well as memory and runtime constraints.

The other major contribution by Tran et al. was the R(2 + 1)D architecture, based on a 3D ResNet but replacing the 3D convolution residual blocks. Here, spatio-temporal modeling is decomposed into 2 separate steps by approximating the 3D convolution with a 2D convolution followed by a 1D convolution. This does not change the number of parameters in comparison to a 3D convolution, but doubling number of non-linearities in the layer due to the ReLU activations between each 1D and 2D convolution. Factorizing the 1D and 2D convolution also allows for easier optimization, which becomes very noticeable when increasing the networks depth. The R(2 + 1)D architecture achieved best performance on all action recognition tasks, suggesting that decomposing the spatial and temporal dimension leads to a higher accuracy than treating them symmetrically, like 3D convolutions do. Furthermore, the R(2 + 1)D architecture has roughly the same computational complexity as the 3D counterpart but considerably higher accuracy R(2 + 1)D has a roughly linear correlation between the length of the input clip and the computational complexity. Tran et al. also found out that the accuracy of any of their models increased when the length of the input clip increased. This further supports their claim of the importance of motion modeling for video action recognition[118].

Chapter 4

Inspiration from Mammalian Visual Processing

As already shown in the preceding chapters, neural networks are inherently inspired by biological neural systems and computation. Visual processing takes up a significant fraction of the mammals brain area, thus giving hints of its complex and computation intensive nature. This section gives a very brief overview of the mammalian visual processing system and lays the foundation for analysing the proposed neural network architecture in chapter 5.

4.1 Hierarchical Organization of the Visual System

A central question of neuroscience is how does the nervous system handle incoming signals in order to enable the perception of color, structure, motion, and depth. In other words, how is the 2D information on the retina used to obtain a complex set of perceptions about the 3D realworldwith little to no conscious effort? To make sense of these questions, first the structure of the visual system is examined.

When light falls through the lens into the eye, the only organ of the visual system with access to raw information, it hits a light sensitive layer at the back of the eye, the retina. The retina contains cells that are receptive to light, photoreceptors, and cells that are responsible for the initial processing of the captured sensory information[5]. The output of this retinal information processing is received by the first layer of neurons, the retinal ganglion cells, where the information undergoes modification[52]. Ganglion cell layers are still located in the eye, at the innermost layer of the retina, and their axons are bundled into the optic nerve and transmit signals in form of action potentials to the visual processing areas in the brain[90].

Most of the axons from the optical nerve terminate in the lateral geniculate nucleus (LGN) in the thalamus, located in the forebrain near the center of the brain. At the

subcortical level in the LGN, the signals coming from the optical nerve undergo heavy modification. Populations of cells residing in different layers, the magnocellular layer, the parvocellular layer and the koniocellular layer, divide the incoming information into three major streams, see Section 4.2.

The lateral geniculate nucleus directly projects to the visual cortex area V1 and also receives direct feedback from it via feedback connections. The V1, also called striate cortex, takes up 10% of the area of the visual cortex, indicating its complexity and importance in processing visual information. In V1, the three streams from the LGN converge and are reorganized into three new streams. The newly formed Magno-dominated stream, responsible for motion and spatial representation, the blob-dominated stream, responsible for form and color, and the interblob-dominated stream are then projected into the next area of the visual cortex, V2. Again, there are also feedback connection from V2 back to V1. Where these reciprocal feedback pathways begin and end, asymmetries in cellular structure can be cleary seen, which further reinforces the strong hierarchical organization of the individual processing stages in the whole visual system.

Higher stages in the visual cortex, after V2, again reorganize the three incoming streams into a dorsal and ventral stream. The dorsal stream is an extension of the magno-dominated stream and analyses spatial relations. It is then projected to the posterior parietal cortex, where higher level cognitive functions like planning, spatial reasoning, attention and motion control take place[4]. In visual cortex area V4 the blob-dominated stream gets fused with the interblob-dominated stream from V1 and V2 to the ventral stream. This ventral stream projects to the infero temporal cortex, or IT cortex, responsible for representations of objects, places and faces, and thus is also mainly involved in long-term memory[6].

This high level overview of the involved neural processing to achieve high level cognitive functions like memory, planning and reasoning from raw sensory inputs is clearly characterized by the hierarchical organization of the involved processing stages. Each stage furthermore consists of multiple subdivisions. It is also remarkable that not all information is processed by the same mechanisms, but it is contextually divided into different streams that reciprocally connect the different hierarchical stages. In total, there are 32 distinct areas dedicated to vision in the cortex taking up more than 50% of the total area of the neocortex. This further shows the immense computational power that is solely involved in creating visual perception to achieve the high level cognitive functionality from raw signals of the eye[119].

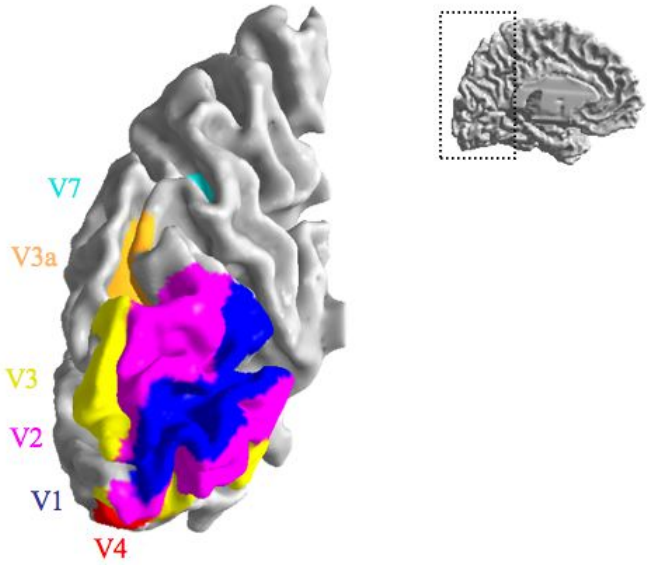


FIGURE 4.1: Areas of the visual cortex, from[46]

4.2 Neuro-Functional Specializations in LGN and V1

As discussed in the section above, the hierarchical processing of stimuli from the retina and the reorganization of said stimuli into contextual streams is an important property of the mammalian visual system. Reorganization into these streams happens through different functional properties and specializations of individual cells and cell groups.

Hubel et al. examined retinal ganglion and lateral geniculate nucleus cells in cats by shining spots of light onto the cats retina and measuring activations of neurons. Said cells were found to respond best to visual stimuli of a very specific size and produced an optimal response to a dot of light of a very specific size with deviations in size leading to non-optimal responses. Thus ganglion and LGN cells register the illumination of a region on the retina and also the difference in illumination between the region and its surround[52]. With the ganglion cells as the first neural processing step in the visual system, these low-level feature activations are projected to next hierarchical step, the LGN.

As mentioned earlier, division into three separate streams is carried out in the LGN by layers of three different cell types. In similar experiments, magnocellular cells (M-cells) in the LGN showed high sensitivity to stimuli of moving grid-like structures but had poor spatial resolution. Whereas parvocellular cells (P-cells) did not respond to stimuli of the same moving gratings. Derrington et al. also showed in these experiments, that

M-cells are very sensitive to contrast and P-cells show hardly any contrast sensitivity. Furthermore, the receptive fields of P-cells is relatively small and that of M-cells is significantly larger. When the retina was stimulated with different frequencies, P-cells responded optimally at 10Hz and the response declined drastically if the stimuli frequency decreased. M-cells on the other hand are not as sensitive to changes in frequency of stimuli and responded optimally at 20Hz with only minor decline when frequencies were altered[17].

These properties of M- and P-cells imply their functional specialization on a cellular level and allow splitting the incoming signal into streams. Magnocellular cells responsiveness to high frequency stimuli and contrast changes indicates their specialization on motion detection. Also the relatively large receptive field on the retina makes sense in context of recognizing motion, since moving objects or are logically travel across the retina. Small receptive fields for P-cells are useful to detect low-level features such as lines and edges suggesting a specialization of P-cells on spatial analysis. Low responses to higher frequency stimuli also support this claim[17]. The function of koniocellular layers is still poorly understood and thus not further examined in this section.

Hubel et al. examined, the so called, simple and complex cells in visual cortex area V1, also known as the primary visual cortex. The diversely structured complex cell groups generally responded best to stimuli from black or white straight lines and to lines separating white and black areas. However, stimuli were only effective when the light was shone onto the retina in an orientation that is specific to the examined neuron. When the orientation of the line was uncharacteristic, i.e. the line was rotated by 90, the neuron did not respond. Thus Hubel et al. were able to identify specific neurons for specific orientations of the stimuli. The placement of the stimuli on the retina had little influence on the response of the examined neuron however. With so called simple cells, in contrast, spatial placement of the stimuli was critical. When spatially displacing the stimuli on the retina without a change of orientation the simple cells did not produce a response. Excitatory and inhibitory regions on the receptive field on the retina thus dictate the output of those simple cells. And since the complex cells disregard spatial placement, they behave as if they receive projections from simple cells. Neighbouring cells in the primary visual cortex also have the same receptive orientation and position characteristics and cells with the same receptive field position are aggregated into a column shape. From this follows that a small region on the retina is mapped to a specific area of neurons in visual cortex area V1[52].

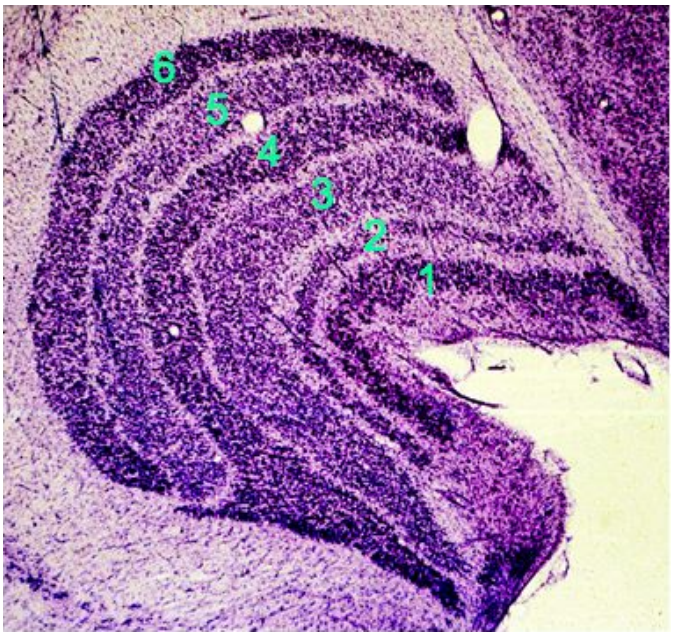


FIGURE 4.2: Laminar organization of the lateral geniculate nucleus from [46]. Layers 1-2 are magnocellular layers, layers 3-6 are parvocellular layers.

4.3 Considerations for Neural Network Design

Taking the aforementioned neuroscientific discoveries in visual processing into account, a few insights and considerations can be useful when designing neural network architectures for processing image data. Since the goal of this thesis is to find a suitable method for extracting spatio-temporal features in videos, this section will only focus on design implications for this specific use-case.

1) *Depth of the architecture:* Since the visual cortex occpupies more than 50% of the neocortex, this gives strong hints about the involved complexity in visual processing. In order to represent such complex processing functions in a neural network, it needs to have a very deep architecture and thus lots of parameters in order to be able to model these complex representations. This large network size is also a requirement of the large datasets at hand, see section 2.6, since large models require large and diverse training datasets in order to prevent co-adaption of feature detectors[48]. That is large models tend to learn noise that is present in the small dataset but not in the real use-case and thus results in poor inference performance.

2) *Hierarchical structure:* Visual processing in the mammalian visual system is, as discussed in section 4.1, hierarchically organized and divides low-level processing into the earlier stages and higher-level processing into later stages. In convolutional neural networks large convolution kernels and large feature maps in the early layers and smaller kernels with smaller feature maps in the late layers resemble this hierarchical organization from a lower to higher level. Dimensionality reduction measures like pooling layers placed between several convolution layers reduce the dimensionality of transmitted data. This further forces the CNN to extract low-level features early in the network since in late layers these features already vanished.
Feedback projections from early processing stages to later stages and back, as present e.g. in V1 and V2, allow for some information to pass through stages without processing. In neural networks, this technique is frequently implemented as shortcut connections, which not only reduce the computational load of the network but also improve convergence significantly[98, 7].
In each hierarchical stage of the visual processing pipeline, there are further subdivision into finer individual steps. This hierarchical subdivision could be realised in a neural network architecture by grouping several layers into conceptual blocks, like the residual blocks in[42].

3) *Specialized neurons:* One major characteristic of cells in the early stage of the visual system, especially in the LGN and V1, is that each cell has a specific receptive field on the retina. This is very similar to CNNs where not every single neuron is responsible for the whole input image. The kernel of the convolutional layer dictates the receptive field of each specific neuron. Thus neurons in a CNN, especially in the early

layers, are have spatial dependencies, as they only get activated if the correct stimulus is within their corresponding receptive field. And hence, similar to neurons in V1, neurons in early layers of a CNN are not invariant to spatial displacement. However, since later layers receive projections from these neurons in the early layers, they show a high degree of spatial invariance.
In V1, neurons with the same receptive field are neighbouring and are aggregated into a columnar shape. Similar to that, neighbouring neurons in a CNN also have roughly the same receptive field. The kernels of convolutional layers only allows activations of neurons if a stimulus has specific property that the kernel is representing. If the kernel is for example a vertical edge detector, see section 2.3.1 and Figure 2.7, a horizontal line will not activate the associated neurons, very similar to what Hubel et al. discovered[52]. Furthermore, Derrington et al. discovered that the LGN and V1 have specialized cells to process temporal and spatial information. Cells specialized to process spatial information are very sensitive to the right frequency whereas cells that process temporal information are not as sensitive in frequency changes. In a 3D CNN this can be realised by large strides in the temporal dimension of the convolution kernel for spatial neurons and small temporal strides for temporal cells.

4) *Spatial and temporal streams:* First division of spatial and temporal information into separate streams happens in the LGN which then in turn projects these streams to the primary visual cortex. Parvocellular layers in the LGN process spatial information whereas magnocellular layers process temporal information. In order to implement such a functionality in a neural network separate streams for the temporal and the spatial dimension need to be present. This is exactly what the two streamed architectures do, see section 3.2.1, by dedicating one network to spatial and one to temporal processing. However, the LGN and V1 use neuro-functional specific cells in each stream, which needs to be handled by the aforementioned differences in temporal strides. Also as Figure 4.2 shows, the magnocellular layers have lower capacity than the parvocellular layers so this also should be considered.

4.3.1 Model Proposal: SlowFast Network

Taking into account the previous methods for action recognition in videos in chapter 3 and considering the insights from the previous sections, choosing a two-streamed approach seems logical. Furthermore, the individual models of the separate streams should be fairly deep and in order to ensure the hierarchical properties and the subdivision of individual stages into smaller blocks, ResNets with at least 34 layers are chosen. In order to guarantee functional specialization of the individual streams, 3D convolutions are required, thus the 2D convolutions in traditional ResNets shall be replaced by their

3D counterpart. These design decisions are all respected by the SlowFast Network architecture proposed by Feichtenhofer et al.[25] and the next section will explain this in detail.

Chapter 5

SlowFast Architecture

In the domain of image recognition the spatial dimensions $I(x, y)$ of an image I are treated symmetrically and thus, as Ruderman and others showed, the orientation of the input image is irrelevant[99, 54]. An image recognizer should still be able to recognize a banana even if it is shown upside down on the input image. This orientation and scale invariance is justified since all orientations in natural images are equally likely, as Ruderman showed[99]. However, when processing videos the temporal dimension is present as well. And an important property of motion is that it is not equally distributed over time, for example, the floor is always stationary and slow motions are more likely than faster motions, e.g. backgrounds in a scene hardly change over time. Thus if not all motions are equally likely to occur, there is no justification to treat the temporal dimension symmetrically to the spatial dimension, like is done in 3D CNNs[117].

Furthermore, the categorical visual semantics in videos often evolve rather slowly. A person doing jumping jacks is still a person over the whole duration of the action. Thus, recognition of said categorical visual semantics can happen at a relatively slow rate. Similar to the characteristics of the parvocellular layers in the LGN, as discussed in section 4.3. Motion, however, happens much faster, i.e. the person who was doing jumping jacks now does squats.

This chapter will focus on the architecture of SlowFast as proposed by Feichtenhofer et al.[25] and give an in-depth view of the individual building blocks of it, first discussing the two-stream approach and then the possible backbones.

5.1 Slow and Fast Streams

The SlowFast architecture proposes a two-stream approach where one pathway captures categorical semantics with a low sampling rate and the other pathway captures rapidly changing motion with high frame rate but with much lighter computational load that takes up roughly 20% of the overall computation. This again shows similarities to the

mammalian visual processing by dividing the processing of spatial and temporal into two streams with the temporal stream occupying significantly less volume than that of the spatial stream. The individual streams have their own expertise and process incoming individually, however since they posses fuse connections, as will be discussed in the following sections, simultaneous training of both streams is made possible.

5.1.1 Slow Pathway

In theory, the slow pathway could consist of any convolutional model that can process spatio-temporal input data. However, as explained in section 4.3, the neuroscientific insights gained suggest that a very deep hierarchical model with further hierarchical subdivisions and specialized neurons for spatial processing shall be selected, a detailed model exploration follows in section 5.2.1.

In order to tackle the cell specialization, large temporal strides are used in the convolution kernels of the input frames. This temporal stride τ lets the layer process only one out of τ frames. Feichtenhofer et al. determined that 16 is a suitable value for τ[25], thus for a 32fps video 2 frames are sampled and input to the network per second of video clip. If T is the number of sampled frames from a clip, the total clip length is $T \times \tau$.

5.1.2 Fast Pathway

The fast pathway runs in parallel to the slow pathway and also consists of a model that regards the considerations made in section 4.3. Again, a very deep architecture with hierarchical subdivisions is chosen. To guarantee a fine temporal representation, a small temporal stride with τ/α is chosen and $\alpha > 1$ represents the ratio between the frame rates of the fast pathway and the slow pathway. Since both pathways shall operate on the same raw input clip, the fast pathway samples α times more frames than the slow pathway. The value of $\alpha > 1$ guarantees that both pathways operate at different temporal speeds, ensures the specialization of neurons in the two pathways and thus is a key concept for SlowFast architectures[25].

In order to not only preserve fine temporal representations in the input layer, SlowFast does not use temporal downsampling layers until the two streams are fused in the global average pooling layer. This implies that all feature tensors until the global average pooling layer have the same temporal dimension of αT frames.

The neuroscientific insights from section 4.3 showed that magnocellular layers have a lower channel capacity than the magnocellular layers in the LGN. SlowFast tries to emulate this property by reducing the channel capacity in the fast layer by a factor of $\beta < 1$. Insights from section 5.2.1 showed that magnocellular layers have a lower channel capacity than the magnocellular layers in the LGN. SlowFast tries to emulate

this property by reducing the channel capacity in the fast layer by a factor of $\beta < 1$. A typical value for β was determined by Feichtenhofer et al. to be 1/8. Common layers computation in terms of floating point operations per seconds, grows quadratically with respect to the channel scaling ratio and thus the fast pathway is has around 20% less computational complexity than the slow pathway, similar to the magnocellular layers in the LGN. This lower channel capacity can be associated with a lower representational capability of modelling spatial semantics[25].

5.1.3 Information Flow via Fusing

In order to make the two separate pathways aware of what representation is learned by the other, information needs to flow between the two. This can be realised by fusing the to streams together after specific layers of the network via so called lateral connections. Using these lateral connections, Lin et al.[73] were able to surpass the state-of-the-art in object detection. Feichtenhofer et al. leveraged them for video action recognition[23, 24]. In Lin et al.'s FPN, lateral connections merge feature maps of the same spatial size from two pathways[73] but since the temporal dimensionality of the slow pathway is not the same as in the fast pathway of the SlowFast architecture, the lateral connections need to perform transformations, as will be detailed in section 5.2.3. However, only the fast pathway will be connected to the slow pathway without any reciprocity. At last, the output of every pathway are fused with a global average pooling layer and the resulting two pooled feature vectors are concatenated to form the output prediction. Figure 5.1 shows an abstract model of the SlowFast architecture and Table 5.1 shows the detailed layer list.

5.2 Backbones

The SlowFast is, in theory, able to use any kind of spatio-temporal convolutional model as backbone in its two streams. Due to the considerations made in section 4.3, this work focuses on specific instantiations of the ResNet architecture in the slow and fast pathway respectively. A 3D ResNet-101 with 101 layers was implemented[39].

5.2.1 Slow Pathway

The slow pathway is implemented by a modified temporally strided 3D ResNet-101. Using the determined temporal stride $\tau = 16$, as suggested by Feichtenhofer et al.[25] and a raw clip length of 64 frames, the input to the slow pathway consists of $T = 4$ sampled frames.

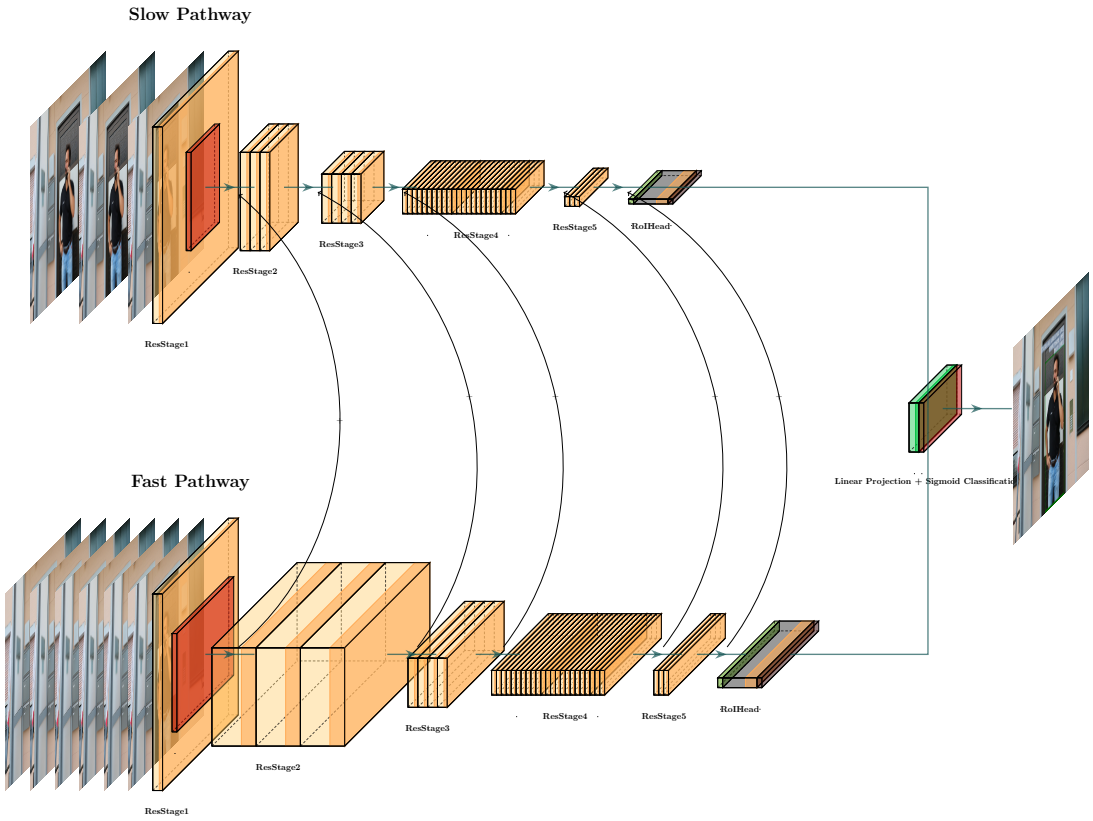


FIGURE 5.1: Schematic diagram of the SlowFast architecture.

3D ResNet Architecture: The SlowFast architecture uses one of the most successful architectures as its backbone, the ResNet. And in order to allow a signal to bypass one layer and move to a next layer in a sequence, shortcut connections are used. These shortcuts pass gradients through the network from early to later layers which allows to train very deep models, in our case with 101 layers. 3D ResNets consist of basic building blocks, similar to 2D ResNets. These represent hierarchical subdivisions of processing stages. Several of these building blocks are grouped into stages, thus this hierarchy is further subdivided into smaller steps of processing. The modified 3D ResNet used in this SlowFast implementation uses so called *bottleneck blocks* as the smallest grouping. Each bottleneck block consists of three 3D convolutional layers, each followed by 3D batch normalization. After an input signal to a bottleneck block has passed the first 3D convolutional layer and the 3D batch normalization, it gets processed by a ReLU activation.

Stage	Slow Pathway	Fast Pathway	Output Size
ResStage1	$1 \times 7^2, 64$	$5 \times 7^2, 8$	$Slow: \ 8 \times 56^2$ $Fast: \ 32 \times 56^2$
ResStage2	$\begin{bmatrix} 1 \times 1^2, 64 \\ 1 \times 3^2, 64 \\ 1 \times 1^2, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 1^2, 8 \\ 1 \times 3^2, 8 \\ 1 \times 1^2, 32 \end{bmatrix} \times 3$	$Slow: \ 8 \times 56^2$ $Fast: \ 32 \times 56^2$
ResStage3	$\begin{bmatrix} 1 \times 1^2, 128 \\ 1 \times 3^2, 128 \\ 1 \times 1^2, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 3 \times 1^2, 16 \\ 1 \times 3^2, 16 \\ 1 \times 1^2, 64 \end{bmatrix} \times 4$	$Slow: \ 8 \times 28^2$ $Fast: \ 32 \times 28^2$
ResStage4	$\begin{bmatrix} 3 \times 1^2, 256 \\ 1 \times 3^2, 256 \\ 1 \times 1^2, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 3 \times 1^2, 32 \\ 1 \times 3^2, 32 \\ 1 \times 1^2, 128 \end{bmatrix} \times 23$	$Slow: \ 8 \times 14^2$ $Fast: \ 32 \times 14^2$
ResStage5	$\begin{bmatrix} 3 \times 1^2, 512 \\ 1 \times 3^2, 512 \\ 1 \times 1^2, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 1^2, 64 \\ 1 \times 3^2, 64 \\ 1 \times 1^2, 256 \end{bmatrix} \times 3$	$Slow: \ 8 \times 7^2$ $Fast: \ 32 \times 7^2$

TABLE 5.1: The used instantiation of the SlowFast architecture. Kernel dimensions are denoted by $\{T \times S^2, C\}$. A residual block is represented by brackets. The backend is ResNet-101. The output layer is omitted for the sake of simplicity. Adopted from [25].

The same happens after the second convolutional layer. The aforementioned shortcut connection skips an incoming signal before it enters the first convolution layer of the block to right after the last batch normalization of the block via identity mapping and zero padding, it then passes a final ReLU activation before producing the bottlenecks output. The kernel size of the first and third convolutional layer are $1 \times 1 \times 1$ and the kernel size of the second convolutional layer is $3 \times 3 \times 3$ in the default bottleneck block. Figure 5.2 shows such a default bottleneck block[39].

Several bottleneck blocks are connected to form a *ResStage*. ResNets, by convention, always have five stages in total. The first stage is always the input stage, and the only stage that does not consist of a bottleneck block. This input stage consists of a 3D convolutional layer with the temporal stride $\tau = 16$, followed by a 3D batch normalization layer that is then activated by ReLU and finally pooled by a max pooling layer. The following four ResStages of the used 3D ResNet-101 contain multiple bottleneck blocks. Stage S_2 has three bottleneck blocks, S_3 has four, S_4 has 23 and the final ResStage S_5 consists of again 4 bottleneck blocks. The convolution kernels of the second convolutional layers of the bottleneck blocks is changed to a *degenerate* kernel, that is the temporal dimension is not convolved and thus the first kernel dimension is changed to one resulting in a kernel of size $1 \times 3 \times 3$. Motivated by the findings of Feichtenhofer et al. that temporal convolutions in the early layers degrades the accuracy, since for a fast moving object and a large temporal stride correlations in temporal receptive fields would be very

little unless spatial receptive fields are sufficiently large, the convolutions in stages up to S_4 are basically 2D convolutions. But since later layers have larger receptive fields, it makes sense to also convolve the temporal dimension and hence, the first convolutional layer in bottleneck blocks of the stages S_4 to S_6 are changed to the size of $3 \times 3 \times 3$. In ResStage S_4, non-local blocks are inserted after bottleneck blocks 6, 13 and 20[25]. After the final bottleneck block, the slow pathway model is fused with the fast pathway model, as will discussed in section 5.2.3.

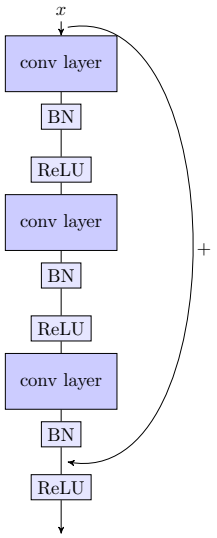


FIGURE 5.2: Schematic diagram of a ResNet bottleneck block.

5.2.1.1 Non-Local Blocks

The aforementioned non-local blocks tackle the problem of convolutions which are only able to process a local neighbourhood at a time. Non-local blocks are, as the name suggests, non-local operations for capturing long-term dependencies. The only possibility for convolutions to capture long range dependencies is when they are repeatedly applied, this has been shown to work in deep stacks of convolutions with large receptive fields[27, 68], however it has been proven to be very computationally inefficient and optimization of these deep stacks is very difficult[50, 42].

The concept of non-local blocks, as proposed by Wang et al., is based on a generalization of the non-local mean operation. Non-local means, in short NL-means, is based on a non-local averaging of all pixels in a given image x. A generic non-local operation

is defined as:

$$y_i = \frac{1}{C(x)} \sum_{\forall j} f(x_i, x_j) g(x_j) \tag{5.1}$$

where i is the index of the position for which the NL-means is to be computed, j iterates over all possible positions, x is the input signal and y is the output of the same size. f is a pairwise function that computes a scalar that represents a relationship between i and all j's. g computes a representation of the input at position j and $C(x)$ is the factor normalizing the response[128].

Different instantiations for f and g are possible, but, for the sake of simplicity, g is only considered n the form of a linear embedding:

$$g(x_j) = W_g x_j \tag{5.2}$$

such that W_g is a weight matrix that is to be learned, implemented by a $1 \times 1^{n-1}$ convolution for n-dimensional inputs. f, in the implementation used in this thesis, is instantiated by a dot product similarity:

$$f(x_i, x_j) = \theta(x_i)^T \phi(x_j) \tag{5.3}$$

Here, θ is the gradient at position x_i and $\phi(x_j)$ is the softmax activation function. In this instantiation the normalizing factor $C(x) = N$, where N is the number of positions in x. Wang et al. discovered that models that implement these non-local operations are not sensitive to the choices of f, since the generic non-local behavior is the source of improvement[128].

To include these non-local operations in neural network architectures, they are bundled into a non-local block:

$$z_i = W_z y_i + x_i \tag{5.4}$$

with y_i is the response from equation 5.1 and $+x_i$ is the residual mapping. Due to this residual mapping, non-local blocks can be inserted into any pre-trained model without breaking the initial behaviour of the model. When the pairwise computation of $\theta(x_i)$ and $\phi(x_j)$ from equation 5.3 is done with feature maps from later layers, as in this implementation in the bottleneck blocks 6, 13, and 20 of ResStage S_4, it is lightweight and computationally comparable to a standard convolutional layer. Furthermore, the channels of the weight tensors W_g, W_θ and W_ϕ are set to be half of the channels of the input x, following the convention of the bottleneck blocks.

The implementation of the non-local blocks is realised by convolving the input x to compute θ, ϕ and g by convolution kernels of size $1 \times 1 \times 1$ and stride 1. The kernels then represent the weight tensors W_θ, W_ϕ and W_g. To further speed up the computation, x is

spatio-temporally pooled after computing θ and before computing ϕ and g to reduce the amount of pairwise computation by 1/4. After calculating the individual components, y_i is determined and also convolved by a $1 \times 1 \times 1$ kernel with strides of 1 to obtain the weights W_z. Finally, y_i is normalized and the residual mapping is added to y_i to obtain the result z_i[128].

These non-local blocks thus allow capturing of long-term dependencies and are a further functional specialization of neuron groups in the network, similar to the complex cells in V1 which respond to stimuli regardless of spatial placement, see section 4.2.

5.2.2 Fast Pathway

In the implementation discussed in this thesis, the fast pathway also consists of a 3D ResNet-101. However, there are minor adjustments to specialize the fast pathway to process motion more efficiently.

For the fast pathway, as discussed in section 5.2.2, a speed ration of $\alpha = 8$ and a channel ratio of $\beta = 1/8$ is used. So for a raw clip with 64 frames as input to the whole network, the input to the fast path is sampled from the raw clip with a sampling rate of $\tau/\alpha = 16/8 = 2$. Thus 32 frames are fed to the input layer of the fast pathway. The channel capacity is reduced by the channel ratio of $\beta = 1/8$ for every channel with respect to its parallel channel in the slow pathway, again inspired by the lower capacity of the magnocellular layers in the LGN.

Bottleneck blocks in all the ResStages of the fast pathway use non-degenerate convoluition kernels, in contrast to the slow pathway. The first convolutional layer in a bottleneck block of the fast pathway has the shape of $3 \times 1 \times 1$, the second $1 \times 3 \times 1$ and the third convolutional layer $1 \times 1 \times 1$. Convolving the temporal dimension as well allows to capture finer temporal resolution, i.e. capture detailed motion. No temporal downsampling further reinforces high temporal resolution and drives the functional specialization to extract temporal features in the fast pathway[25].

5.2.3 Lateral Connections

In order to enable information flow between the two parallel pathways lateral connections are implemented. After every ResStage said lateral connection fuses from the fast to the slow pathway. For a fuse to work, dimensions of the fuse point in the slow and fast pathway need to match, however, the shape of the fuse points in the slow pathway is $\{T, s^2, C\}$ whereas the shape in the fast pathway is $\{\alpha T, S^2, \beta C\}$. In order to match the dimensions, a 3D convolution with a kernel of shape $5 \times 1 \times 1$, $2\beta C$ output channels and stride of α is performed on the tensor at the fuse point on the fast pathway. Then the

result of this convolution is normalized using batch normalization and concatenated with the tensor at the fuse point on the slow pathway. Now the slow pathway continues its processing with the result of this fuse connection.

After the final ResStages of the two pathways, global average pooling is performed on the output tensors of the last stages output respectively. These two output tensors are then concatenated and fed to the final fully connected classification layer.[25].

5.2.4 Loss

The loss function of the network determines its learning behaviour. Thus separate loss functions for the tasks of action classification and detection are used.

5.2.4.1 Classification Loss

For the task of classification, cross entropy is used as loss function. Cross entropy measures the classification performance via the output probabilities which are in the range of $[0, 1]$. It is defined as:

$$H(\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \times log(\hat{y_i}) \tag{5.5}$$

where N is the total number of classes, $\hat{y_i}$ is the confidence for class i, and y_i is a binary indicator if the class is present. Cross-Entropy is always positive but converges towards zero if $\hat{y}$ converges to y. Thus, the cross entropy loss increases as the predictions diverge from the label and highly penalizes false confidence. This is expected due to the logarithmic behaviour of the expression which makes cross-entropy the preferred loss function for multi-class classification tasks. Figure 5.3.

5.2.4.2 Detection Loss

For detection, the binary cross entropy loss, or log loss, is used. It is defined as:

$$H(y) = \frac{1}{N}\sum_{i=1}^{N} y_i \times log(\hat{y_i}) + (1 - y_i) \times log(1 - \hat{y_i}) \tag{5.6}$$

where, similar to cross-entropy, y_i is a binary indicator if the i-th class is present at the evaluated clip, $\hat{y_i}$ is the confidence of the predictor of the i-th class and N is again the total number of classes. Binary cross-entropy differs from the aforementioned cross-entropy in that it is independent for each class, that is, the loss computed for every class is not affected by the loss for another class. This property is especially useful in the detection scenario of multi-label classification, where non-exclusive classes are present.

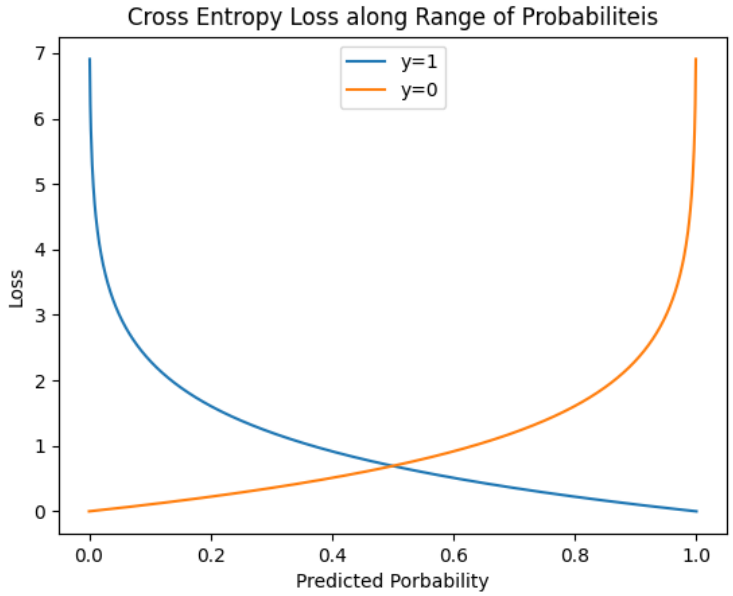


FIGURE 5.3: Range of possible loss values for one class.

5.3 Implementation

The SlowFast architecture was implemented in Python3 with the open source machine learning framework PyTorch[89]. PyTorch provides imperative and pythonic programming style, focusing on usability and speed. It supports easy debugging and error handling, consistent interfaces to popular scientific computing libraries like NumPy, fine grained control of model structures, easy distributed training and efficiently supports acceleration on GPUs.

The *torch.nn.module* serves as base class for each sub component and the architecture as a whole and thus allows the implementation of the SlowFast model in an object-oriented style. All modules need to implement a *forward* method which handles the behaviour of the module, for example a bottleneck block, during a forward pass through the network. Figure 5.4 shows a sample implementation of the *forward* method. The behaviour in the backward pass is then automatically infered and handled by PyTorch. If a special behaviour during the backward pass is to be implemented, it can be easily done by overwriting the *backward* method of the class at hand. The *ResBlock* is the finest grained module, multiple *ResBlocks* form a *ResStage*, multiple *ResStages*, in turn, form a pathway and multiple pathways, combined with fuse connections, form the SlowFast

architecture. The implementation in this thesis uses configuration files to dynamically create architectures by specifying e.g. the number of ResBlocks per ResStage in the configuration file without needing to write additional code.

```python
import torch.nn as nn
class BasicTransform(nn.Module):
    """
    Basic transformation: Tx3x3, 1x3x3, where T is the size of temporal
    kernel.
    """

    def __init__(self, dim_in, dim_out, temp_kernel_size, stride,
    dim_inner=None, num_groups=1, stride_1x1=None, inplace_relu=True, eps
    =1e-5, bn_mmt=0.1, norm_module=nn.BatchNorm3d):
        super(BasicTransform, self).__init__()
        self.temp_kernel_size = temp_kernel_size
        self._inplace_relu = inplace_relu
        self._eps = eps
        self._bn_mmt = bn_mmt
        self._construct(dim_in, dim_out, stride, norm_module)

    def _construct(self, dim_in, dim_out, stride, norm_module):
        # Tx3x3, BN, ReLU.
        self.a = nn.Conv3d(
            dim_in,
            dim_out,
            kernel_size=[self.temp_kernel_size, 3, 3],
            stride=[1, stride, stride],
            padding=[int(self.temp_kernel_size // 2), 1, 1],
            bias=False,
        )
        self.a_bn = norm_module(
            num_features=dim_out, eps=self._eps, momentum=self._bn_mmt
        )
        self.a_relu = nn.ReLU(inplace=self._inplace_relu)
        # 1x3x3, BN.
        self.b = nn.Conv3d(
            dim_out,
            dim_out,
            kernel_size=[1, 3, 3],
            stride=[1, 1, 1],
            padding=[0, 1, 1],
            bias=False,
        )
        self.b_bn = norm_module(
            num_features=dim_out, eps=self._eps, momentum=self._bn_mmt
        )

        self.b_bn.transform_final_bn = True

    def forward(self, x):
        x = self.a(x)
        x = self.a_bn(x)
        x = self.a_relu(x)

        x = self.b(x)
        x = self.b_bn(x)
        return x
```

FIGURE 5.4: PyTorch Code example of the *BasicTransform* module which is for example required to implement skip connections by transforming the shape of the tensor that is input to a ResNet basic block to match the dimension after the last layer of the basic block.

Chapter 6

Action Recognition and Detection with SlowFast

6.1 Action Recognition

The previously proposed SlowFast Architecture with a Non-Local ResNet-101 backend is trained on the Kinetics 400 and 600 datasets. This section will go into detail about the necessary preprocessing to feed the neural network with training and inference data. Finally the specific training scheduleis presented.

6.1.1 Preprocessing

Since the Kinetics datasets consist of YouTube videos, they do not provide a direct download of all the videos. However, a csv-file is provided that includes the URL of a video, a start and end timestamp that mark the relevant 10s clip, and an associated label to that 10s clip. So first, the dataset had to be downloaded with a modified version of the original scripts provided by the Kinetics team. The modified script automatically crops the video to the given 10s and encodes it as mp4[107]. As discussed in the dataset section 2.6.3, Kinetics clips come in a variety of resolutions and framerates. Thus the following preprocessing pipeline was implemented to allow fast, efficient and robust training and inference.

After downloading the dataset a label file is to be created that contains the path to a given video and its label. Now the general data loading procedure is to randomly sample a video from the training set, sample a clip of length $\alpha T \times \tau$ frames from the video and crop it to 224×224pixels, again sample T frames from the clip and feed it to the slow pathway, sample a clip of αT frames from the clip and feed it to the fast pathway, and after the training step is completed, feed the label to the associated clip. This procedure is repeated until an Epoch, consisting of a fixed number of training steps, is finished.

This is then again repeated until a determined number of epochs is reached or the trainig loss saturates. See Figure 6.1

PyTorch again provides a useful framework which tightly integrates the data loading process into the training, evaluation and testing of a given model. First an iterable over the dataset needs to be created. This is done by inheriting from the *torch.utils.data.Dataset* abstract base class which represents a map from keys to data samples. The custom dataset subclass then parses the previously generated label csv-file and fetches a video by index from csv-file. Using python bindings for *ffmpeg*[18], provided by *PyAV*[8], the video is decoded. Now, the individual frames are normalized and T frames are randomly sampled for the slow pathway and αT frames are randomly sampled for the fast pathway. Next, all frames are randomly flipped along the horizontal axis and cropped to the input size of 224×224 pixels. If the initial resolution is too small, the frames are padded with zeros. Finally, the list of frames for each pathway, the according label and the index of the video is returned.

Since the dataset iterable provides access to data by index, the *torch.utils.data.distributed.DistributedSampler* inherited from *torch.utils.data.Sampler*, is used to provide distributed random access to the dataset without losing reproducibility. The *Dataset* class only returns data but does not control what to return, the *DistributedSampler* provides the functionality to split the whole training set into exclusive subsets in order to enable data parallelism during the training procedure. Each process can pass an instance of a *DistributedSampler* to the *DataLoader* as sampler. This *DataLoader* class combines the *DistributedSampler* with the *Dataset* iterable in order to iterate over the whole training set to continuously feed training data to the network. The *DataLoader* is responsible for controlling the suffling of the data, the number of processes used for loading the data and can directly copy batches of training data to CUDA memory before returning them. Making the dataloading process highly efficient and distributable.

6.1.2 Training Schedule

The used pre-trained SlowFast model was trained on a GPU cluster consisting of 128 dedicated GPUs. A mini-batch size of 8 clips per GPU was used, totalling in a batch size of 1024 clips for each training step. Batch normalization is computed for mini-batches of 8 clips instead of the whole 1024 clips.

Initialization of the learnable SlowFast parameters was done according to the strategy proposed by He et al. Here, the weights of the model are initialized by randomly sampling from the normal distribution $N(=, std^2)$, where:

$$std = \frac{gain}{\sqrt{\text{output feature maps} \times \text{receptive field size}}} \tag{6.1}$$

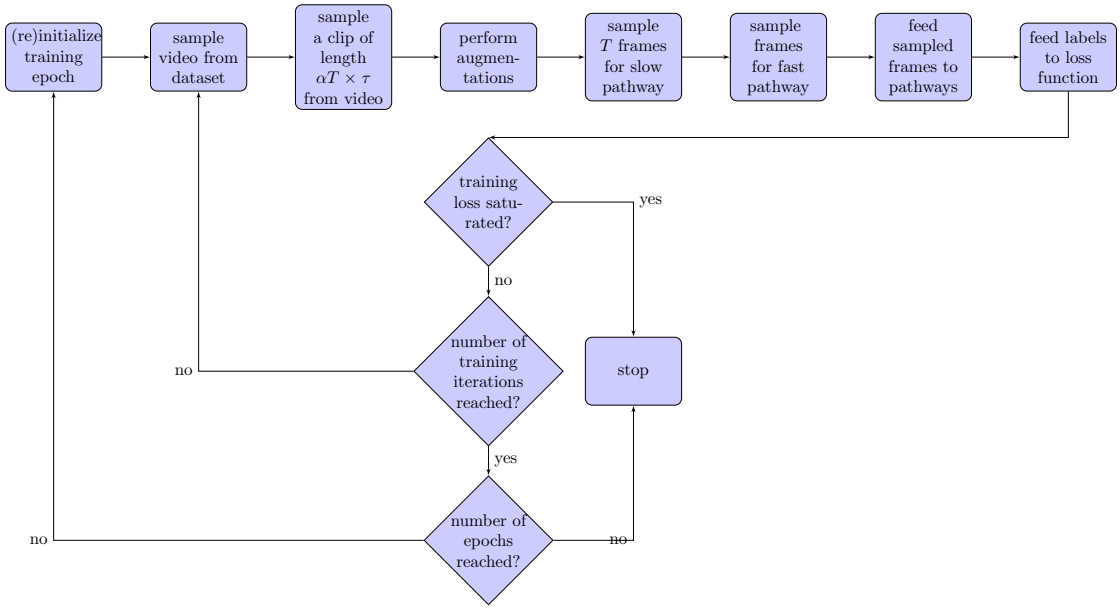


FIGURE 6.1: Flow Diagram of the training process.

and $gain = \sqrt{2}$ for ReLU non-linearities. This so called *He initialization* helps the convergence of very deep models and is considered as the go to initialization method for models with ReLU activations[43].

Distributed training is handled by synchronized mini-batch stochastic gradient descent, following the training strategy proposed by Goyal et al. [33]. The goal is to use very large mini-batches but maintaining training and generalization accuracy. Large mini-batch sizes, without proper strategy, tend to hamper training success, as discussed in section 2.2.3.3. To achieve maximum resource utilization, data parallelism is used without reducing the per worker workload and without loss of training and testing accuracy. Furthermore, a linear scaling rule for the learning rate is applied, that is, the learning rate is multiplied by **k** if the mini-batch size is multiplied by **k**.

Learning rate decay is used to allow faster convergence and better generalization. A half period cosine schedule is implemented to allow so called *warm restarts*. Each warm restart initializes the learning rate to the same value and schedules it to decrease over the next n iterations until the next restart. This improves the rate of convergence and accelerates gradient schemes to deal with ill-conditioned functions that are to be learned. So after n iterations a restart of the training is simulated by increasing the learning rate η_n. The old values x_n of the model are stored as initial solution and the amount η is increased by is controlled via the momentum of SGD[111]. Since these warm restarts

temporarily worsen the performance, not all previous intermediate results are used. In this implementation the cosine learning rule schedule looks as follows:

$$\eta_{n+1} = \eta_n \times 0.5[cos(\frac{n}{n_{max}}\pi] + 1] \tag{6.2}$$

with n_{max} being the maximum number of training iterations and $\eta_0 = 0.1$[75].

The aforementioned linear scaling rule of the learning rate breaks down when training with extremely large mini-batch sizes, especially when the network undergoes rapid changes as in the early stages of training. However, this can be mediated by using less aggressive learning rates only at the beginning of the training. This is then called *warm up*.

Thus a gradual warm up strategy is used, where the learning rate is gradually increased during the first 8k iterations. Instead of starting with a lower learning rate and increasing it after n iterations, a gradual warm up avoids a sudden increase and allows a healthy convergence at the start of the training[33].

In this implementation, the starting learning rate is $\eta_{start} = 0.01$ and gets linearly increased for 34 epochs to the base learning rate of $\eta_0 = 0.1$. The momentum of the stochastic gradient descent optimizer is set to 0.9 and weight decay[76] is set to 10^{-4}.

6.1.3 Inference

After the model has sufficiently converged on the training data, it is used for inference. Actions in a given video can be recognized by sampling 10 clips from it along the temporal axis. These clips are then again cropped or zero padded, if necessary. Then these 10 clips are consecutively forward passed through the network and the softmax scores of each individual clips are taken and averaged. The class with the highest softmax average score is then chosen as final prediction. Finally the label of the predicted class, or, optionally, the labels of the top-k classes are annotated on the video. Figure 6.2 shows sample results from Action Recognition inference.

6.1.4 Performance

The implemented SlowFast architecture with a ResNet101 backend with non-local blocks achieved state of the art performance on Kinetics-400. A top-1 accuracy of 79.8 and top-5 accuracy of 93.9 was achieved. The computational cost per inference is at 7020 GFLOPS, without preprocessing.

On Kinetics-600, a top-1 accuracy of 81.9 and a top-5 accuracy of 95.1 could be achieved with the same model trained from scratch. The inference cost is the same.

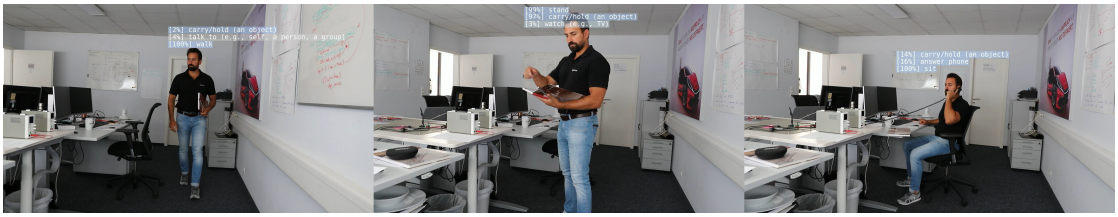


FIGURE 6.2: Recognition inference on a custom made showcase video. Frames are randomly sampled from the full video.

	Top-1	Top-5
Kinetics-400	79.8	93.9
Kinetics-600	81.9	93.9

TABLE 6.1: Performance of SlowFast with ResNet101 backend and non-local blocks.

6.2 Action Detection

Action recognition is rather similar to the task of action detection, however to action detection focuses on the spatio-temporal localization of human actions. Localiztaion of these action is typically represented by the model outputing bounding boxes with associated actions for each person in a given video.

The AVA-Kinetics dataset, see section 2.6.3, is spefically designed for action detection in videos and provides labels with bounding boxes for 1 frame per second of video. This section deals with the necessary model changes for the requirements of action detection.

6.2.1 Model Adjustments

Since the AVA-Kinetics dataset and benchmark require the model to produce possibly multiple bounding boxes with associated labels, the SlowFast architecture is slightly adjusted. First of all, the detection is split into two 2 separate parts, detection of persons in a frame and determining the actions a detected person performs. A separate, off-the-shelf person detector detects people in input clips and hands the bounding box coordinates, or the so called region proposals, to the SlowFast model which then performs action classification for the proposed regions. Due to this division of responsibility and the verified performance of the off-the-shelf person detector, the SlowFast network is trained with the ground truth bounding boxes provided by the AVA-Kinetics dataset and later in the inference, the person detecor is used to provide region proposals.

The SlowFast architecture used for action detection is the same as for action recognition but with minor adjustments. Spatial strides in the last ResStage res_5 are set to 1 instead of 2, and additionally a dilation of 2 for the filters is used. This increases the spatial resolution of filters by a factor of 2. After the last ResStage, a RoI-Head is added as final output stage. This RoI-Head, or Region-of-Intereset-Head, gets the region proposals as input and extracts only features from the relevant regions from the last ResStage res_5.

The RoI-Head has the structure, as depicted in Figure 6.3. For each of the two pathways, the RoI-Head consists of a temporal 3D average pooling layer followed by a RoIAlign layer, followed by a spatial 3D max pooling layer. The two pathways are then merged by a linear projection with dropout[48] and a final sigmoid activation forms the prediction.

Incoming bounding boxes, either from training data or from a person detector, are directly input into the RoIAlign layer without previous intervention. Here they are downscaled to match the dimension of the feature maps of the output of res_5 and then extended into the temporal dimension, since only 1 FPS is provided by the training data. Temporal extension is done by duplicating the incoming bounding box along the temporal dimension matching the temporal dimension of the two pathways of res_5. Each region of interest is then subdivided into so called bins which represent the size of the RoIAlign output feature map. In this implementation the output feature maps are of shape 7×7 and thus 7×7 bins are used. These bins are employed since the downscaled bounding boxes do not directly match the pixels of the feature maps, as shown in Figure 6.4. In order to avoid information loss when quantizing the downscaled bounding boxes to match the feature maps' pixels, each bin in the RoI samples 4 points. Values of the RoIAlign feature maps are then computed by bilinear interpolation. Now the results are aggregated using the following pooling layers. This follows the design proposed by Girshick et al.[30]

Input from slow pathways res_5 feature maps is first temporally pooled, as mentioned above, with a kernel of size $[8 \times 1 \times 1]$. The same happens for the fast pathway but with a pooling kernel of shape $[32 \times 1 \times 1]$. After the RoIAlign layer, the resulting 7×7 feature maps are spatially pooled with a kernel of shape $[7 \times 7]$ for both pathways respectively. After concatenating the resulting tensors, a linear transformation with dropout and 80 output neurons is used, and thus the output of the linear transformation matches the number of classes in AVA-Kinetics. A prediction is formed by a per class sigmoid based classifier for multi label prediction[44].

6.2.2 Preprocessing

Training of the SlowFast action detection on AVA-Kinetics logically first requires the download of the dataset. This is done analoguously to the procedure described in 6.1.1 via the provided csv file. In general, the preprocessing for AVA-Kinetics happens equivalently to the preprocessing for Kinets, however since AVA-Kinetics does not only provide class labels but also bounding boxes, a few additional steps are required.

Again, the first step is to create an iterable over the dataset that delivers the training data and the labels to the dataloader. For AVA-Kinetics this dataset iterable first loads a video according to a given index from the path stored in the dataset csv-file. Then the individual frames and their indices for the two pathways are sampled, cropped or padded, and horizontally flipped with random probability in the exact same procedure as described in section 6.1.1. Using the indices the corresponding action class labels and bounding boxes are loaded. Finally, the frames for the two pathways, the label arrays, the indices of the frames in the original video and the bounbding boxes are returned.

Using this dataset iterable and the same distributed sampler, the dataloader is ready for training, validation and testing.

6.2.3 Trainig Schedule

The training process is adopted from training on Kinetics, however, slight adjustments are made to deal with the difference in size of the training set and the slightly modified architecture. First of all, due to the relatively small size of AVA-Kinetics, especially in comparison to the size of Kinets-{400, 600, 700}, the model for AVA-Kinetics is not trained from ground up. The weights of the SlowFast architecture for action detection of the layers that were adopted from the SlowFast architecture for action recognition are initialized with the weights from the model trained on Kinetics-400. The remaining RoIHead ResStage is initialized using He initialization.

Learning rate scheduling with a half period cosine schedule is used again with a linear warm up for the first 1k iterations in similar fashion to the training schedule for Kinetics but with a weight decay of 10^{-7}.

6.2.4 Inference

After the model has sufficiently convered, an off-the-shelf person detector is used to provide regions of interest for inference, more details will be given in the following section 6.2.4.1. For a given video in which actions shall be detected, the key frame of the 10s long AVA-Kinetics video is determined first. A key frame is the frame with the highest confidence of the person detector. Then a clip of length $\alpha T \times \tau$ is sampled around the keyframe and the individual frames for the slow and fast pathway are sampled whith this

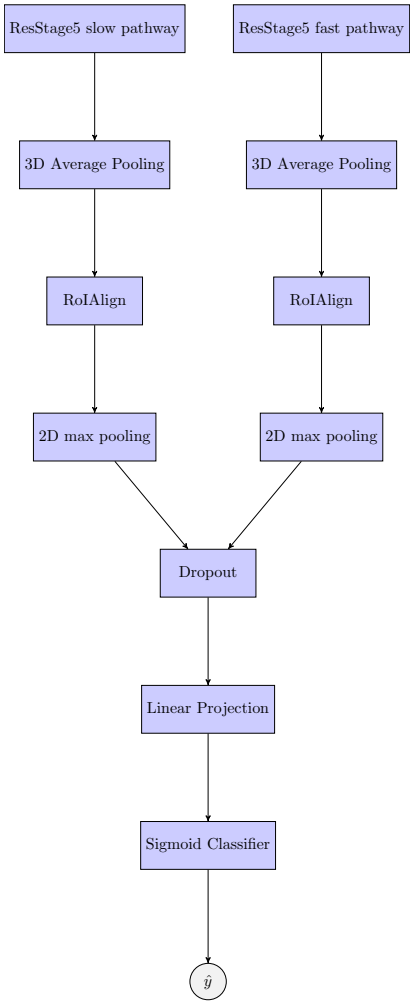


FIGURE 6.3: Diagram of the layout of the RoI Head.

key frame as their center. Frames are then cropped or padded such that the shorter side has 256 pixels.

After sampling of the frames, they are fed to the trained SlowFast model for detection with the bounding boxes from the person detector. The model will then output predictions in the same form as the AVA-Kinetics labels. That is for each bounding box class labels with their associated confidences are returned. Since multiple persons can be in one frame, all the predictions for the corresponding frame are gathered. Bounding boxes from the person detector come in the form of pixel coordinates, which are then used

0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	1	1	0	0	0	0
1	1	0	0	0	0	0

I

FIGURE 6.4: Arbitrary example of a feature map I with a downscaled region of interest to showcase the problem of aligning downscaled bounding boxes with the values from the feature map. The downscaled region of interest is depicted as red rectangle.

to draw bounding boxes on the raw input frames to create a output video via OpenCV. Next, the best suitable position for the class labels for each bounding box are determined by checking for potential overlaps with other bounding boxes and finally the class labels are drawn adjacent to their corresponding bounding box. At last, the all the individual frames of a given input clip are written to a video file using the *OpenCV VideoWriter*. A sample result can be seen in Figure 6.5.

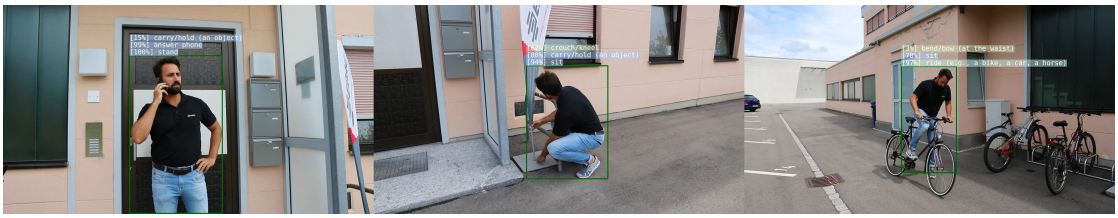


FIGURE 6.5: Detection inference on a custom made showcase video. Frames are randomly sampled from the full video.

6.2.4.1 Person Detection

Since training can be done with pre-computed region of interest proposals, either supplied by the dataset itself or by using an off-the-shelf detector to compute them once, safe them and thus speed up the training time significantly, running inference on the SlowFast detection model requires external region proposals. For this implementation a Faster R-CNN with ResNeXt-101-FPN[73, 131, 96] backbone was used that was not jointly trained with the SlowFast detection architecture for the aforementioned reasons.

The Faster-RCNN architecture uses a region proposal network (RPN) which is a fully convolutional network and shares full-image convolutional features with the detection network and thus is very efficient. However, unlike with the Fast-RCNN architecture[30] where region proposals and the detection network are separate, Faster-RCNN combines a RPN and the detection network into a single network, where the RPN simultaneously predicts object bounds and objectness scores at each position and through the shared features tells the detection network where to look. Specifically, the two-modules of Faster-RCNN share a common set of convolutional layers. From the last shared layer, the RPN takes the input feature maps and outputs rectangular object proposals with an associated objectness score. Then using RoI-Pooling the relevant features of the proposed regions by the RPN are extracted and fed to a multi-label classifier to form a final prediction.

As backend for the Faster-RCNN architecture, a ResNeXt-101-FPN was used, which inherits its layout from ResNets. Here, building blocks of the same topology are stacked to form the architecture of the final network. Each module of building blocks in the ResNeXt architecure performs a set of transformations which are aggreagated by summation at the end of each block, in a similar fashion as the bottleneck blocks in ResNet architectures. However, ResNeXt employs a new concept, cardinality, that does not only stack individual layers of a building block horizontally but also vertically, as depicted in Figure 6.6, without introducing additional complexity. These architecural decisions of the Faster-RCNN architecure with ResNeXt-101-FPN backbones gained wide spread adoption and provide state-of-the-art detection performance.

Implementation in this thesis uses the detectron2 framework[130] for object detection, which provides pre-trained models and an easy to use API. The Faster-RCNN ResNeXt-101-FPN used for detection in conjunction with the SlowFast detection architecure was pre-trained on ImageNet and COCO human keypoint images[72] and then fine-tuned on AVA-Kinetics for person detection. The detector reached an average precision on the AVA-Kinetics validation set of 93.9[25].

6.2.5 Performance

The implemented SlowFast architecture was evaluated on the AVA-Kinetics validation set. Other methods like I3D or ATR [] saw significant improvements by using additional optical flow features combined with pretraining on Kinetics-400. I3D achieved a validation mAP of 15.6 and ATR 21.7, both using optical flow. SlowFast was able to increase the previous state-of-the-art performance from ATR by +7.3mAP, resulting in a validation mAP of 28.2. When pretrained with Kinetics-600, the SlowFast architecure could improve its performance to 30.7mAP on version 2.2 of the AVA-Kinetics dataset, which provides more consistent annotations. Table 6.2 shows a comparison of different models.

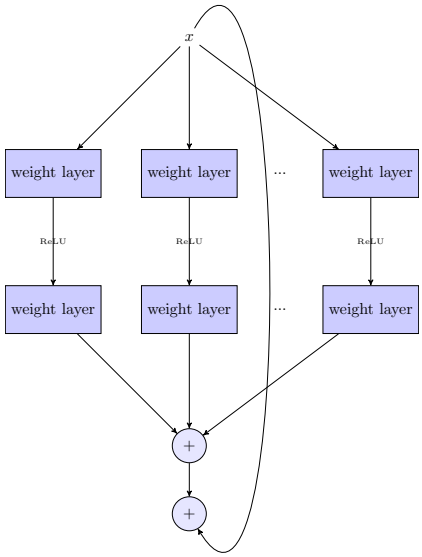


FIGURE 6.6: Diagram of a ResNeXt building block with arbitrary cardinality. The number of additional vertical layers determines the cardinality.

Model	Optical Flow	Pre-Train	Validation mAP
I3D[12]		Kinetics-400	14.5
I3D[12]	✓	Kinetics-400	15.6
ATR R50+NL[126]	✓	Kinetics-400	21.7
I3D[12]	✓	Kinetics-600	21.9
SlowFast		Kinetics-600	**30.7**

TABLE 6.2: Performance of SlowFast with ResNet101 backend and non-local blocks on AVA-Kinetics.

Chapter 7

Model Analysis

This chapter is divided into two main sections. In the first section, a practical analysis of the model is performed, investigating its performance with several different methods. The second section deals with a more methodical approach and tries to explain how features are extracted in the SlowFast architecture by steadily showing parallels or the lack there off to the visual processing system in mammalian brains.

7.1 Practical Analysis

The main goal for most new deep learning methods for video action recognition and detection is to achieve state-of-the-art accuracy on popular benchmarks, like those in Section 2.6. But this practice of evaluation disregards inference times, memory footprint and other factors that are highly relevant when trying to select a model for a practical application. Historically, with increasing accuracy the models have an increasing resource consumption and an incentive for speeding up inference time is missing.

This section will perform a practical analysis of the SlowFast architecture and compare it with relevant architectures with regard to computational requirements and accuracy. It shall provide a decision basis for model selection and optimization for deployment in practical applications. The methods from Canziani et al. [11] is followed for analysis.

7.1.1 Runtime Cost

Runtime cost is a deciding factor when deploying neural networks in practical applications. For example in the automotive domain, an action recognition and detection system needs to perform in real time. To evaluate the actual runtime of the architecture, the cost per spacetime view is regarded. A spacetime view is a temporal clip with spatial crop. The SlowFast architecture used in this thesis, uses a spatial size of 256×256 for 10 temporal clips with 3 spatial crops each to form a prediction and thus has 30 views. The cost per spacetime view is obtained through PyTorch, by feeding the network zero inputs,

and is at 234 GFLOPS per view, totaling in 7020 GFLOPS per inference run. Data pre- and postprocessing steps are disregarded for this analysis, since they highly depend on implementation. Thus only the raw inference cost of the network is regarded, since it is framework and implementation independent. The GPUs under consideration reflect a typical spectrum of devices used in different scenarios of neural network inference. The Quadro RTX 8000 represents the top of the line server GPU, the GeForce RTX 2080Ti is the representative of consumer GPUs, the AGX Xavier is a development board used mainly in the automotive industry and the Jetson Nano is the consumer-line development board, similar to a RaspberryPi with dedicated Tensor-Cores for neural network inference.

Table 7.2 and Figure 7.1 compare the runtime cost per inference run on different CUDA enabled NVIDIA GPUs. Even though the SlowFast architecture consists of two separate 3D ResNet-101 it still has the lowest runtime cost with the second highest accuracy on Kinetics-600. A key contributing factor for the low runtime cost of SlowFast is the low channel capacity of the fast pathway. This allows the model to capture motion without the necessity of a fine grained spatial resolution. Another key factor is the sparse temporal sampling of clips of only 30 views per inference run, other models in table 7.2 use extremely dense sampling of over 100 views during inference, for example R(2+1)D uses 115 views. Figure 7.2 plots accuracy against inference cost. The tendency of more accurate models to also have greater inference costs is clearly visible.

NVIDIA Name	GFLOPS
Qaudro RTX 8000	**16300**
GeForce RTX 2080Ti	13450
AGX Xavier	1410
Jetson Nano	235.8

TABLE 7.1: Typical NVIDIA GPUs used for deep learning inference. GFLOPS according to the technical specification from [15] for 32bit FLOPS.

7.1.2 Memory Cost

Alongside inference cost, memory cost is a deciding factor for deploying models into production environments. Often in embedded use cases, memory is tightly constrained, however recent models tend to require increasingly more memory. Canziani et al. showed, that memory cost for neural networks is initially constant and then raises proportionally to the batch size. The initial memory allocation of the network is a large static part and the memory cost increases with a slope of 1.3 relative to the batch size[11].

Model	GFLOPS/Run	2080Ti	RTX 8000	AGX Xavier	Jetson Nano	Top-1 Acc.
SlowFast	**7020**	**0.52s**	**0.43s**	**4.97s**	**29.77s**	79.8
Non-Local R101[128]	10770	0.80s	0.66s	7.64s	45.67s	77.7
Non-Local R50[128]	8460	0.63s	0.52s	6.00s	35.88s	76.5
R(2+1)D [118] + optical flow	34960	2.56s	2.14s	24.79s	148.26s	73.9
R(2+1)D[118]	17480	1.30s	1.07s	12.40s	74.13s	72.0
Large-scale[127] R(2+1)D-152	8064	0.56s	0.49s	5.72s	34.20s	**82.5**

TABLE 7.2: Performance of SlowFast with ResNet101 backend and non-local blocks.

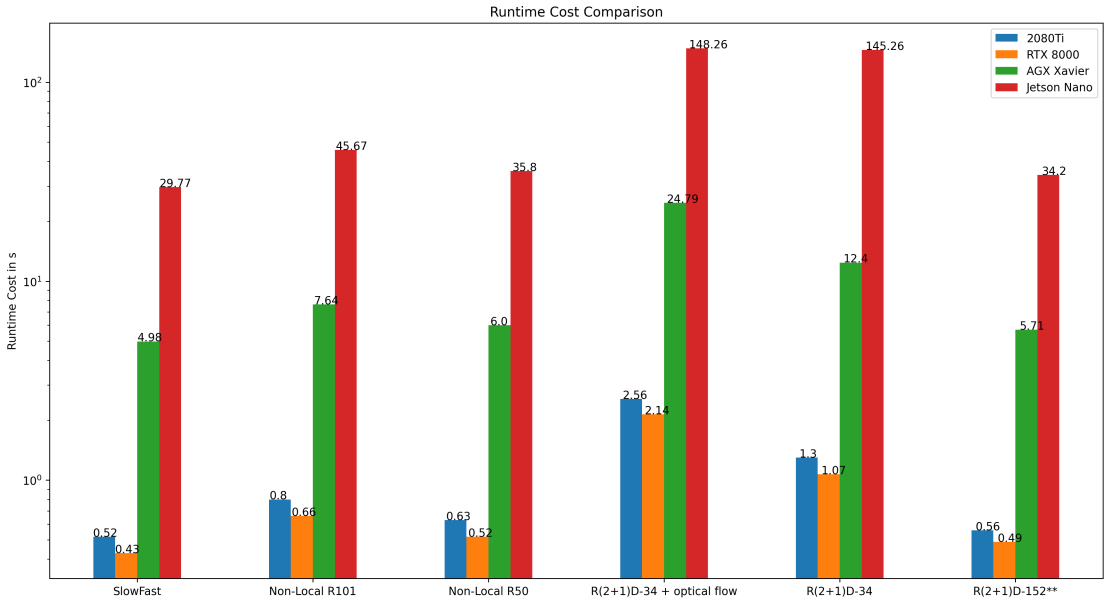


FIGURE 7.1: Runtime cost comparison of selected models on different CUDA enabled GPU devices. (**) is pre-trained on IG-65M[58].

Table 7.3 compares the memory consumption on a System with 2× GeForce RTX 2080Ti, an AMD Ryzen Threadripper 2920X and 64GB of system memory. CPU and GPU memory allocation is determined via helper functions from the *fvcore*[35] library for all models. The implementations of R(2+1)D models were reused from [118], and thus the comparison is constrained. Implementations of the SlowFast model and the Non-Local models were done in the same code base and thus provide a fair ground for comparison.

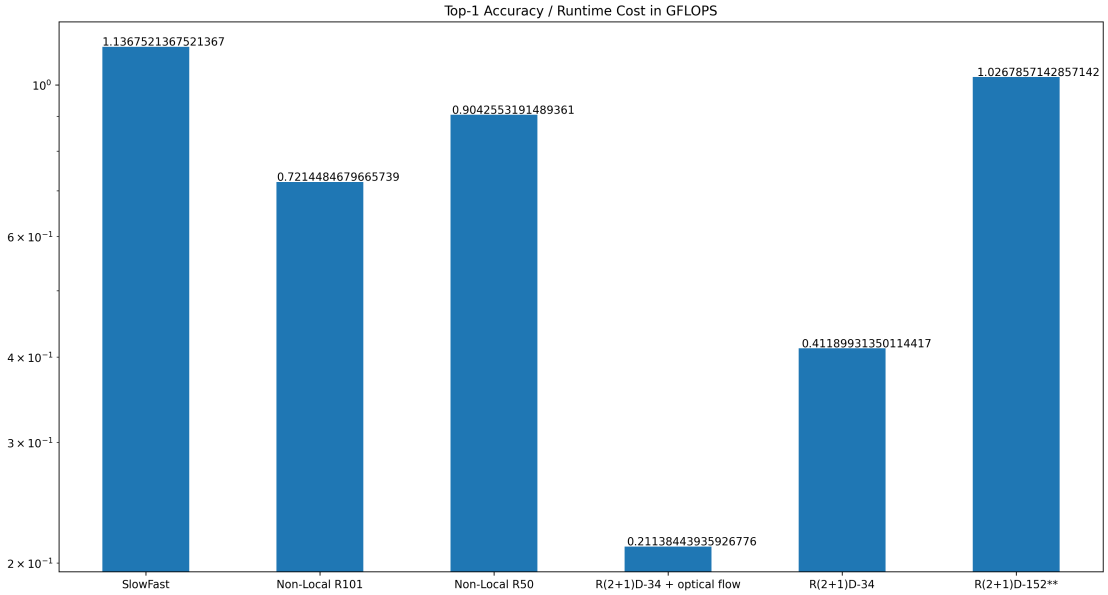


FIGURE 7.2: Accuracy per GFLOPS, higher values are better. (**) is pre-trained on IG-65M[58].

The implementations from the SlowFast code base roughly require the same amount of CPU RAM but differ significantly in GPU RAM. SlowFast requires clearly the most memory but has almost the same number of parameters as Non-Local R101. However, the parameter count only indicates learnable parameters and does not regard parameters for layers like pooling . And since the SlowFast architecture incorporates two individual networks, with the fast network having lower capacity, an almost double amount of required memory in comparison to the single stream Non-Local R101 is logical.

The R(2+1)D implementations from the code base of Tran et al. require significantly less memory than from the code base of this thesis. Even the large R(2+1)D model with 152 layers only requires roughly 1/8 of memory that SlowFast allocates. A significant portion of that difference comes from the modular approach of the SlowFast code base. Furthermore, the additional non-learnable parameters of SlowFast are due to the two-stream approach almost double.

7.1.3 Accuracy and Throughput

How accurate a given model is per floating point operation is a key metric for the evaluation of the practicability of the model. Canziani et al. already report that in their

Model	GPU RAM	CPU RAM	# of Parameters
SlowFast	2.888 GB	4.228 GB	59,342,488
Non-Local R101[128]	1.501 GB	4.193 GB	59,112,656
Non-Local R50[128]	1.122GB	4.191 B	35,401,936
R(2+1)D [118] + optical flow**	0.238 GB	3.153 GB	63,747,500
R(2+1)D**[118]	0.238 GB	3.152 GB	63,747,500
Large-scale[127] R(2+1)D-152**	0.442 GB	3.231 GB	118,277,110

TABLE 7.3: SlowFast memory cost comparison. (**) indicates models that were developed with a SlowFast-foreign codebase.

analysis, the maximum accuracy that can be achieved for a given frame rate is linearly proportional to the frame rate[11]. These findings are also visible in this investigation, see Figure 7.2, however the R(2+1)D with additional optical flow is an outlier, since computation of optical flow is quite expensive and lots of spatio-temporal views are required for an inference run. The other R(2+1)D R34 architecture also requires dense temporal sampling and thus does not fit the linear relationship. Though it shall be noted that the aforementioned architectures were developed to investigate possible building blocks and not to find the best detection architecture. The large R(2+1D) 152 was pre-trained on IG-65M which yields great performance gains, all other methods used models trained from scratch making the comparison unequal.

SlowFast, again, shows considerable performance on this metric with comparatively high throughput due to the lowest temporal sampling required from all the models in comparison and achieves 2nd highest accuracy. Canziani et al. discovered a hyperbolical relationship between accuracy and throughput which is also represented, to some degree, here. However they investigated rather mature models for image classification, whereas the video action classification models presented in this thesis are quite novel. Given a few years time, similar results are to be expected for video classification models. Large-scale pre-training and finetuning on the target dataset also yield significant accuracy gains without needing to change the models architecture and thus not changing the accuracy versus throughput, as [127] showed. SlowFast already showed very promising performance on this metric and potential pre-training is to be expected to boost accuracy versus throughput even further. Furthermore, throughput directly correlates with power consumption thus giving hints that a trade off between accuracy and power might be necessary for applications where power is limited, like in embedded or automotive use cases.

7.1.4 Parameter Utilization

Deep neural networks are known to inefficient in using all available parameters to represent the given training data. Optimization techniques like weight pruning, quantization and others can significantly shrink the network size without sacrificing too much accuracy, as Han et al. demonstrated[38].

Accuracy per parameter can thus be used as an indicative metric of the efficiency of the networks representative capabilities. Denser networks need less parameters for the same accuracy and hence decrease the training time due to fewer parameters needing to be optimized. Also, due to the lower number of parameters, inference time decreases and thus also power consumption decreases. However, due to temporal sampling required for video classification and detection, networks with fewer parameters do not necessarily have faster inference times. Training time also depends not only on the number of parameters to be optimized, but also on how difficult the optimization is, this is clearly demonstrated by Residual Networks[42].

For networks with roughly the same number of parameters, like Non-Local R101, R(2+1)D-34 and R(2+1)D-34 + optical flow, SlowFast has the highest accuracy per parameter with $1.344e^{-6}$. This showcases the great learning capabilities of the SlowFast approach and emphasises the importance of treating the temporal dimension special. Figure 7.3 also shows, that when accuracy per parameter is plotted against runtime cost, SlowFast performs really well, strengthening its aforementioned learning capabilities.

Smaller networks like Non-Local R50 have very high accuracy per parameter, though it should be considered that firstly it comes with a very high runtime cost of 8460 GFLOPS per inference run and that the accuracy per parameter metric highly penalizes a higher number of parameters over a slightly worse accuracy thus smaller networks are always advantageous in that regard. This is also demonstrated by the comparatively low accuracy of the large-scale pre-trained R(2+1)D R152 which only has roughly half the accuracy per parameter of SlowFast with $0.700e^{-6}$. Thus a comparison of these networks with significantly lower and higher parameters to SlowFast is not expedient.

7.1.5 Concluding Remarks

The SlowFast architecture dealt with in this very thesis showed promising results in this practical analysis. SlowFast was able to achieve the lowest runtime cost with state-of-the-art from scratch accuracy. Pre-training of SlowFast and then finetuning on a given target dataset will further improve accuracy without hampering the runtime cost.
Concerning memory cost, the analysis showed, that it is highly implementation specific but in all cases, SlowFast is expected to have higher requirements on memory due to its two streamed approach, which requires two feed the network two temporal sequences

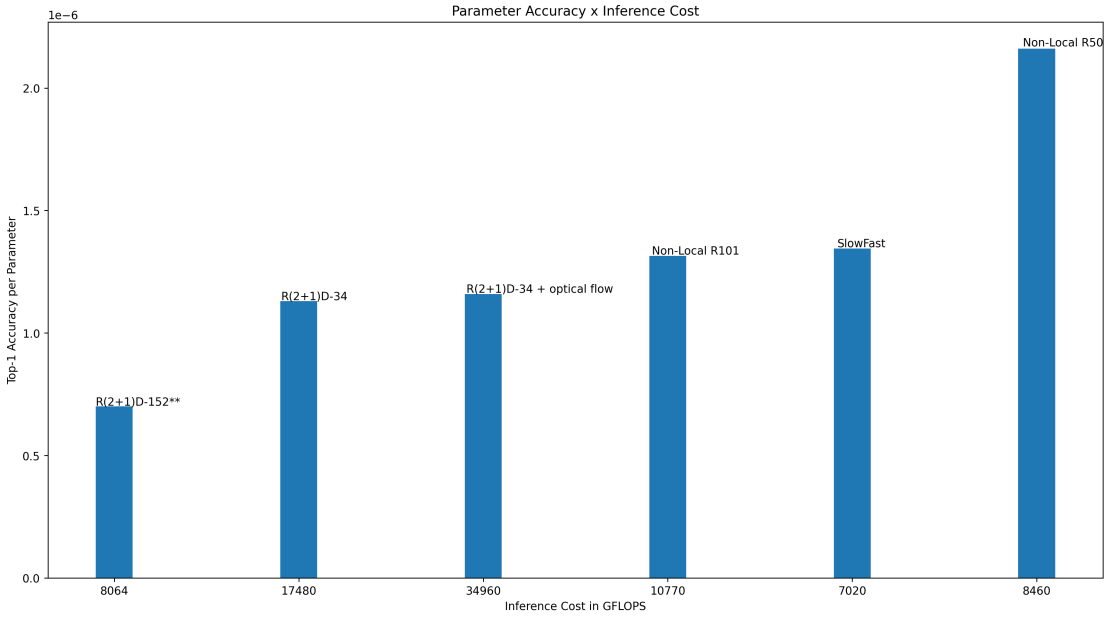


FIGURE 7.3: Accuracy per parameter plotted against runtime cost in GFLOPS , higher values are better. (**) is pre-trained on IG-65M[58].

with different sampling rates. If SlowFast is to be deployed into a practical use-case, these memory requirements could be lowered by employing optimization techniques, like those described by Han et al.[38].
For the analysis of accuracy and throughput, the trained from scratch SlowFast architecture is a clear victor. This shows the efficiency of the two-streamed approach and pre-training will likely boost this performance even more. The effect of pre-training for video action classification has already been demonstrated by Wang et al. on the R(2+1)D architecture and similar results can be expected for SlowFast[127].
Despite SlowFast having more parameters than the two next best performing architectures it has lower inference time cost. This is partly due to the lower density of required temporal sampling but showcases the immense representational learning capabilities of the architecture. Since the slow pathway is very similar to Non-Local R101, the superior temporal modeling capabilities can be exclusively attributed to the fast pathway. Furthermore, SlowFast has two times more layers than Non-Local R101 and four times more than Non-Local R50 due to its two-streamed approach. However th low channel capacity of the fast pathway makes the SlowFast architecture very lightweight and due to the quadratic scaling behavior in terms of channels, the fast pathway only takes about

20% of the total computation. This makes the fast pathway highly beneficial to the representational power of the complete architecture.

Practical analysis of the SlowFast architecture showed the potential of two-streamed approaches for action recognition and detection in videos. The strong motion modeling capabilities of the fast pathway further showed that the temporal domain requires special treatment. Even with a high number of parameters, SlowFast is able to provide low inference time costs and with a bit of optimization, memory requirements could be significantly lowered. The good results in the accuracy and throughput analysis and the high parameter utilization make this architecture a highly viable candidate for a wide range of practical use cases, for example in self driving cars.

7.2 Methodical Analysis

This section will analyse the processes in SlowFast that extract features to form a prediction and shows parallels to the mammalian visual processing system. Though it shall be noted that the goal of SlowFast was not to simulate the visual processing system but rather to look for inspiration and to find methods to transfer them to the application at hand.

7.2.1 Deep Hierarchical Processing

For large recognition and detection dataset challenges, leading models are almost exclusively very deep neural networks with $\geq$ 50 layers[116, 121, 2]. For video action recognition and detection, part of the models performance often stems from pre-training, for example on ImageNet [128, 132, 12]. However, recent architecture for video action recognition also showed promising results without pre-training, but require pre computed motion features in form of optical flow to be somewhat competitive[118].
This necessity of deep architectures is also reinforced by the findings of Hinton et al. [48], providing an empirical as well as a theoretical justification to use very deep networks for the large Kinetics and AVA-Kinetics datasets. SlowFast, with its two very deep ResNet-101, achieved without pre-training state-of-the-art Top-1 and Top-5 accuracy and at the same time having a lower runtime cost than previous models. Thus, a hierarchical subdivision into separate processing of motion and space in the SlowFast architecture is in part responsible for its immense representational capabilities. This is further supported by the fact that the same backends as in [128, 118] were used, but achieving significantly better performance by the introduced hierarchical subdivisions.

Visual processing alone takes up more than 50% of the cortical area in mammals and the visual cortex area 1, where the high level division into separate processing streams happens mostly, occupies again 10% of the size of the visual processing system. The delicate hierarchical structure of cells within the LGN and V1 and their complex interconnection with other processing stages in the visual system also demonstrates the amount of processing power that is dedicated to vision. Serving as "proof by example" that high level visual function, be it artificial or biological, requires a hierarchy of processing stages to give rise to efficient and quality high level visual functions.

7.2.2 2-Streamed Analysis

Chapter 4 laid out the fundamentals of mammalian visual processing, and detailed that motion analysis happens in the magnocellular layers of the lateral geniculate nucleus and in the magno-dominated streams that start in the visual cortex area V1 and lead through the visual processing hierarchy to the posterior parietal cortex. Cells of this layer are functionally specialized and their functional specialization increases when going up the processing hierarchy.

When regarding the feature maps, or activations, of the trained SlowFast model, the feature maps of early convolutional layers in both pathways are relatively similar, see Figure 7.6 and 7.7. Since in both pathways the first ResStage contains only a single convolutional layer with relatively small filter sizes, and thus small receptive fields, only very low level features of the input image can be filtered. This is represented by the high spatial resolution of the resulting feature maps and the skateboarder is still relatively clearly visible. A distinct sensitivity to motion or form in the two pathways is at this point not directly noticeable. Similar to the first processing stages in the LGN, where only very low level features like straight lines or edges lead to activations of its cells[51]. Perceiving such low level clues, however, is crucial to determine high level motion or structure features in the given input clip, since high level features are a complex aggregation of low level features.

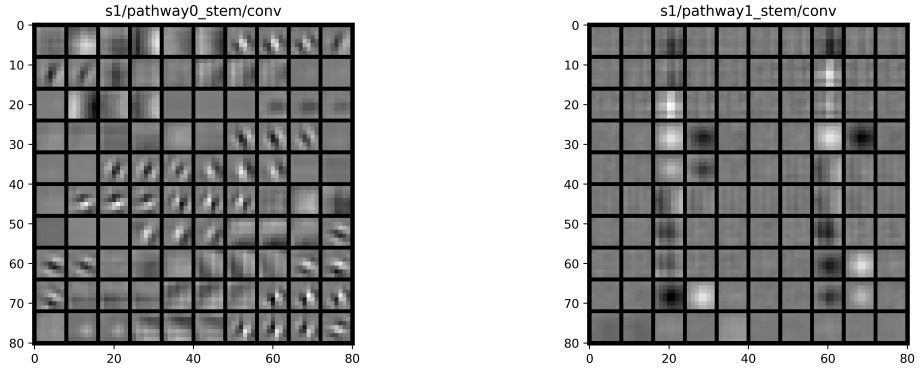
The first convolutional layer in ResStage1 of the slow pathway does not convolve over the temporal dimension, thus no temporal feature aggregation is possible. Forcing the slow pathway to focus on the spatial dimension, and due to the large number of channels, more spatial features can be extracted in this layer and then passed on to the later ResStages. Since ResStage1 of the fast pathway does convolve the temporal dimension, but has much lower channel capacity and higher temporal resolution, it forces the pathway to be motion sensitive. This fine grained temporal resolution of the fast pathway carries through to the deeper layers, where the feature map's sensitivity to motion is clearly visible, as depicted in Figure 7.8 where only the first 8 frames are plotted, for the sake of visibility. Activations of the fast pathway in *ResStage4 & 5* center around moving

humans and fewer activations are present for camera motion or background movements. This visible distinction between local motion, like the trajectory of a human, and the global motion flow fields, i.e. camera movements, is a direct similarity to the MD-stream in visual cortex area V1[119].
From Figures 7.9 and 7.8 it is also clearly visible that specific filters in *ResStage4 & 5* respond to different motion stimuli. Specific filters heavily respond to contracting motion while others respond to expanding motion exclusively. Whereas other filters are, in turn, responsible for detecting slower and faster motion.

In the visual processing system of primates form and color analysis happens first in the parvocellular layer in the LGN and later in the blob dominated and inter-blob dominated streams in V1. The early layers of the slow pathway similarly extract simple features, like horizontal and vertical edges or contours, which is depicted in Figure 7.6. Low level spatial feature extraction is forced by the aforementioned lower temporal resolution and higher channel capacity of the slow pathway. Findings of Hubel et al. could also verify these low level feature extractions in the parvocellular layers in LGN of cats[51]. Movements hardly affect this low level feature extraction in the slow pathway and VanEssen et al. could verify similar behavior in the ventral stream in V1[119].
Again, in the slow pathway there are specific neurons for specific features in the first layers. However the receptive field sizes are very small, leading to a high spatial resolution of the feature maps. In deeper layers in *ResStage4 & 5* feature maps shrink in spatial size but respond to a broader range of stimuli. Since they receive projections from earlier layers, their receptive field size increases dramatically which leads deeper layers to extract higher level features. In these deeper layers it becomes more and more difficult to determine which stimulus activates which feature map, which is in turn another clear indication for the extraction of high level features. In the visual processing system of mammals, later stages in the ventral stream are very hard to analyse, but show very complex receptive fields[26]. In the last *ResStage* pattern recognition is very advanced and only very selective feature maps show any activation and the majority shows basically no activation at all.

The differences of the slow and fast pathways are rather small in the first ResStage and low level feature extraction appears to be relatively similar in the two pathways. With increasing depth the higher channel capacity and lower spatial resolution increasingly lead to more and more complex form and structure analysis in the slow pathway. This is also very visible when the activations of later stages are regarded. Similar to that, the fast pathways lower channel capacity but high temporal resolution leads to a complex motion analysis and excitement of filters is almost exclusively due to motion. Division of form and motion processing stages is thus clearly visible in the activations of the respective

pathways. These findings validate the design decisions of the two stream architecture, the motion capture capabilities of the fast pathway and the representational power for spatial forms and structures of the slow pathway.



(A) Slow Pathway (B) Fast Pathway

FIGURE 7.4: Visualization of the weights from the first convolutional layer of ResStage1. Similarities to filters like the vertical sobel kernel are recognizable.

7.2.3 High Functional Specialization

High level visual functioning requires access to low level visual clues and low level visual clues can in turn contribute to a variety of perceptual tasks. In the visual processing system extensive convergence, divergence and cross talk between processing streams provide the means to perform an almost arbitrary breadth of visual tasks[119].

Filters in the slow and fast pathway are highly specialized due to the great variety of visual clues present in the videos of Kinetics. This specialization can be seen in Figures 7.4 and 7.5 where each filter is activated by a different stimulus resulting in different feature maps. The level of complexity of filters rises with network depth to extract more and more high level features in deeper layers. Division into separate streams for motion and form processing in combination with the hierarchical filter organization gives SlowFast its representational capabilities for the highly complex nature of human actions. This immense filter complexity in all levels of SlowFast processing reflects the complexity of visual processing in the mammalian brain.

Van Essen et al. showed that form and motion analysis in V1 have the same underlying computational mechanisms but are fairly different in functionality[119]. The

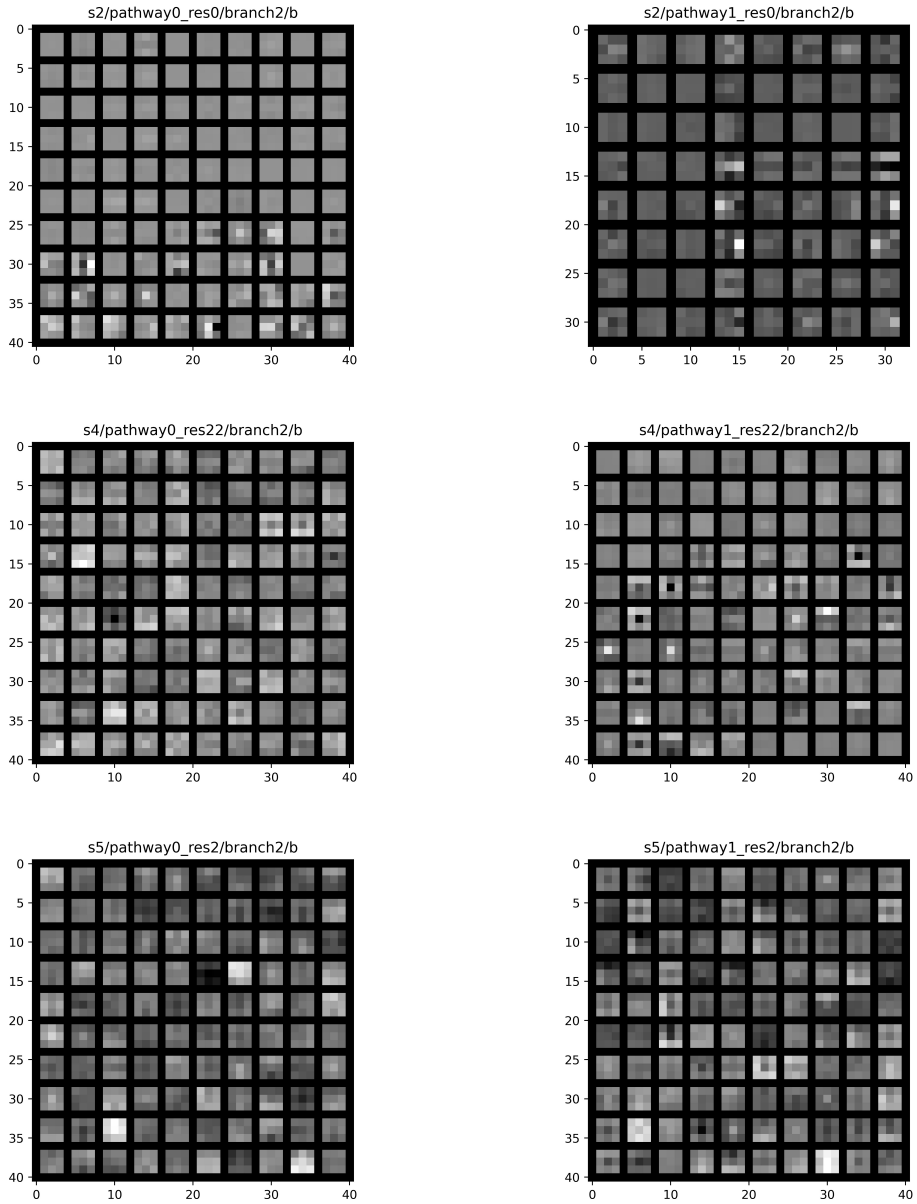


FIGURE 7.5: Visualization of the weights of the second convolutional layer in the last bottleneck block of ResStage 2,4 and 5 for the slow and fast pathway. Only the weights of the first 100 channels are shown. The complexity and diversity of the different filters is a clear indication for a strong focus on high level features.

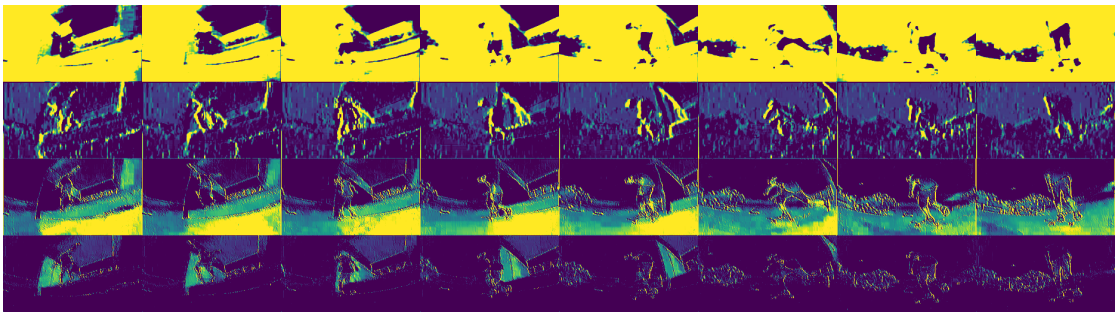


FIGURE 7.6: Visualization of the feature maps from the first convolutional layer of ResStage1 of the *slow* pathway. 4 Channels were selected randomly. The extraction of low level features is very pronounced.

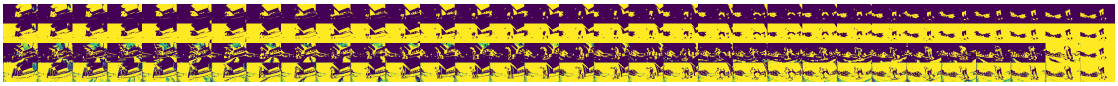


FIGURE 7.7: Visualization of the feature maps from the first convolutional layer of ResStage1 of the *fast* pathway. 4 Channels were selected randomly. (Zooming required)

FIGURE 7.8: Visualization of the feature maps from the second convolutional layer of ResStage2 of the fast pathway. 4 Channels were selected randomly. Feature extraction is already more advanced than in ResStage1.

FIGURE 7.9: Visualization of the feature maps from the second convolutional layer of ResStage4 of the fast pathway. 4 Channels were selected randomly and horizontally stacked. The input video is a skateboarder doing a trick and represents fast movements. Feature detection is very advanced in this stage and activations in regions were motion is present are visible.

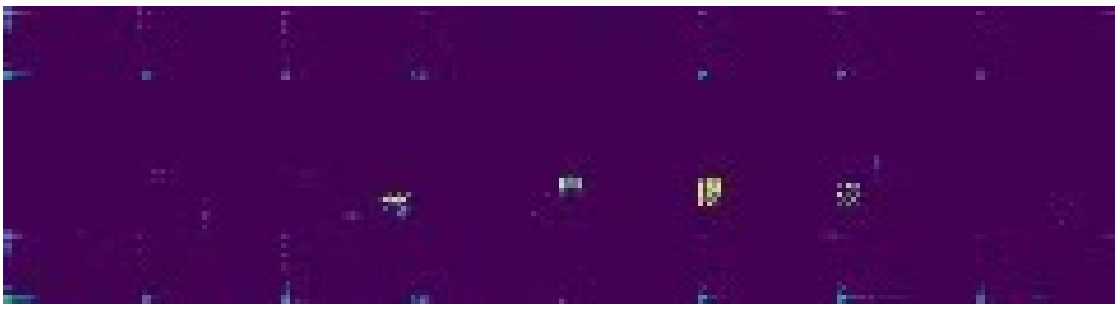


FIGURE 7.10: Visualization of the feature maps from the second convolutional layer of ResStage4 of the fast pathway. 4 Channels were selected randomly and horizontally stacked. Very sparse activations in only a selected few channels can be seen and the hierarchical progression from Figure 7.7 over 7.8 and 7.9 is fairly obvious.

same can be said about the SlowFast architecture, both pathways are governed by the same computational principles, that is convolutions. However slightly different hyperparameters are chosen for the two paths, like the shape of the convolutional filters, which produce a high degree of functional specialization within the two pathways.

In the mammalian brain, neurons act as filters, as Van Essen et al. describe it, which are tuned along a specific dimension. In the early stages of processing, these filters stay relatively simple. However in late stages, filters get increasingly complex and are more selective along more complex dimensions like non-cartesian dimensions in V4. Exactly the same behavior is observable in SlowFast, and similar to the brain, filters in SlowFast are not manually programmed but learned from the training data. Thus feature detection is not implemented as is but rather the hierarchical combination of different filters for different visual clues creates the feature detectors.

Visual processing, be it in the brain or in a deep neural network, is more complex than researchers anticipated two to three decades ago. In mammalian visual processing it is a great challenge to understand how individual stages, components and cells interact with each other to produce high quality visual perception. But even though visual processing is highly complex, with enough effort it is decipherable. This not only holds true for the mammalian visual system but also for deep neural networks.

7.2.4 Concluding Remarks

The implemented two stream approach, inspired by mammalian visual processing, brings a further hierarchical subdivision of the neural network architecture. Splitting of motion and form analysis into two pathways proved to be a great step towards mature video recognition and detection architectures. Separate processing of motion and form gets increasingly more complex with rising network depth and the extraction of high level features is very prominent in the late stages of the two pathways. Varying channel capacity and temporal resolution lead to a strong functional specialization of the two pathways, as anticipated by the design considerations in section 4.3. This functional specialization is further enforced by the increasingly more complex filters that are learned from the training data.

Methodical analysis demonstrated that the spatio-temporal modeling capabilities of the SlowFast architecture do indeed stem from the division into two pathways, the functional specialization of the pathways and the deep architectures of the two processing streams. The inspirations from mammalian visual processing turned out to be highly valuable and the resulting architecture shows lots of similarities that arose from training rather than from explicit programming.

Chapter 8

Conclusion

After covering the necessary foundations, this thesis first examined different methods for spatio-temporal feature extraction. Since the domain of video recognition and detection heavily relies on spatial and temporal features in order to yield accurate predictions most methods stem from this domain. Hand-crafted feature approaches were able to produce relatively fast methods but due to the necessary manual crafting of those features they are highly tedious to build and are not very adaptive to varying use cases. 2D convolutional networks were rather successful in the video action recognition domain, however due to the lack of temporal modeling, more complex features could not be extracted. Despite that, in the early days of action recognition challenges they were able to perform relatively well. 3D convolutional networks however soon turned out to be more powerful, but more resource intensive than the hand-crafted and 2D convolutional methods. 3D CNNs, however, had one big flaw in that they treat the spatial and temporal dimension symmetrical. R(2+1)D architectures tried to tackle this problem by decomposing the 3D convolution into separate spatial and temporal convolutions and showed powerful representational capabilities due to their emphasised focus on the temporal dimension.

By taking inspirations from the visual processing in the mammalian brain, where incoming optical information is divided into 2 streams in the LGN and the V1, further examinations into this visual processing were made. In mammalian visual processing, motion and form information is processed in separate streams. A complex hierarchy with interconnection of different hierarchical stages enables high level visual functions. Crosstalk between streams as well as converging and diverging of these streams in combination with high functional specialization further enhances the performance of the visual processing system.

These insights inspired the architecture that was investigated in this thesis. The architecture under consideration, initially proposed by Feichtenhofer et al., consists of a two streamed deeply hierarchical model, where the two streams are tightly connected to allow information exchange. This architecture was implemented and trained using distributed training via PyTorch.

The findings of this thesis show that the SlowFast architecture has the lowest runtime cost and the highest accuracy of all recent models that were trained from scratch. Only one large-scale pre-trained model was able to surpass the accuracy of SlowFast on the Kinetics benchmark. SlowFast also has the lowest runtime cost, as measured in accuracy per GFLOPS for an inference run. This manifests in a high accuracy per throughput which is due to the low temporal sampling required. Low temporal sampling, in turn, is compensated by the great motion modeling capabilities of the architecture. SlowFast also shows the highest parameter utilization in comparison with models with similar numbers of parameters. Showing the representational capabilities of SlowFast with extremely pronounced spatio-temporal modeling which is a result of spatio-temporal feature extraction.

Furthermore, this thesis showed that the two streamed approach is very powerful with relatively cheap runtime costs, even if the architecture has a large number of parameters. The methodical analysis revealed that this two streamed architecture inherited a lot of functionality from its inspiration, the visual processing system. The hierarchical structure of SlowFast gives rise to powerful modeling capabilities. Also, this hierarchy of processing stages is crucial to obtain high level visual functions and to be efficient in doing so. The two streams allow efficient extraction of spatial and temporal features through functional specialization of the individual streams. On one hand, low channel capacity and high temporal resolution in the fast pathway give rise to high motion modeling capability. On the other hand, high channel capacity and low temporal resolution allow precise form and structure analysis in the slow pathway.
Each stream is highly specialized to the purpose of its task. Each of the filters in a pathway responds to a different visual clue and a breadth of filters is required to represent the variety of visual clues present in the data. The same principles of 3D convolutions in the two streams but with different hyperparameters, like kernel shape, give rise to these specializations.

The SlowFast architecture proved to have outstanding spatio-temporal feature extraction capabilities that are not only present in the intended domain of action recognition but also translate well to other domains. Low runtime cost, high throughput, high degree of parameter utilization and state-of-the-art accuracy also make this architecture suitable for embedded use cases, as for example in an autonomous robot like the EDAG CityBot. Given sufficient optimization, such a two stream architecture can serve as a backbone for the visual perception stack of advanced autonomous systems. More mature two stream architectures in the fields that relate to video recognition are due to the exceptional performance of the SlowFast architecture anticipated.

Chapter 9

Discussion and Outlook

In this thesis, the feasibility of different spatio-temporal feature extraction techniques has been investigated. The chosen SlowFast architecture was able to produce state-of-the-art results on Kinetics and AVA-Kinetics. However, the examined SlowFast architecture was trained from scratch for the recognition task. In Chapter 7 other models were more closely regarded and the pre-trained R(2+1)D-152[127] was able to achieve higher accuracy than SlowFast. Other works [29, 19, 21] have showed that large-scale pre-training and unsupervised pre-training boosts the performance on the target dataset. Thus it is expected that the performance of SlowFast will also benefit from large-scale pre-training.

Furthermore, the 3D ResNet101 was used as backbone for the two pathways in the SlowFast architecture, although other architectures as backbone are entirely possible due to the modular nature of the SlowFast implementation. Thus the question arose if other backbones like R(2+1)D[127, 118] can even further improve the performance of the SlowFast architecture. Also, for embedded use cases for example, shallower backbones could be used to decrease the runtime cost. Further work could explore these paths.

After the start of this thesis, several new research papers in the domains of action recognition and detection were published. For action recognition, methods using attention, additional depth information, multivariate signal sequence encoding or late temporal modeling have been proposed[133, 91, 9, 81, 56, 92], however none of these methods were evaluated on Kinetics. Other models for action detection in videos were also recently proposed[71, 71] but also not evaluated on the AVA-Kinetics challenge. Thus future work could include evaluating these methods on datasets that allow a comparison to the SlowFast architecture and selecting promising models as new backbones for SlowFast. It shall be noted that most of these novel neural networks use pre-training on a large dataset and finetuning on the dataset on which it is to be evaluated.

Bibliography

[1] Sami Abu-El-Haija et al. *YouTube-8M: A Large-Scale Video Classification Benchmark*. 2016. arXiv: 1609.08675 [cs.CV].

[2] Manoj Acharya, Tyler L. Hayes, and Christopher Kanan. *RODEO: Replay for Online Object Detection*. 2020. arXiv: 2008.06439 [cs.CV].

[3] Charu C. Aggarwal. *Neural Networks and Deep Learning - A Textbook*. Berlin, Heidelberg: Springer, 2018. ISBN: 978-3-319-94463-0.

[4] "Dorsal Cortical Pathway". In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 996–996. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_1580. URL: https://doi.org/10.1007/978-3-540-29678-2_1580.

[5] "Retina". In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 3492–3492. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_5101. URL: https://doi.org/10.1007/978-3-540-29678-2_5101.

[6] "Ventral Cortical Pathway". In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 4170–4170. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_6251. URL: https://doi.org/10.1007/978-3-540-29678-2_6251.

[7] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995. ISBN: 0198538642.

[8] Mike Boers. *PyAV*. 2020. URL: https://pyav.org/docs/stable/.

[9] Alban Main de Boissiere and Rita Noumeir. *Infrared and 3D skeleton feature fusion for RGB-D action recognition*. 2020. arXiv: 2002.12886 [cs.CV].

[10] Wilhelm Burger and Mark J. Burge. *Digital Image Processing - An Algorithmic Introduction Using Java*. Berlin, Heidelberg: Springer, 2016. ISBN: 978-1-447-16684-9.

[11] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. *An Analysis of Deep Neural Network Models for Practical Applications*. 2016. arXiv: 1605.07678 [cs.CV].

[12] Joao Carreira and Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2017. arXiv: 1705.07750 [cs.CV].

[13] Joao Carreira et al. *A Short Note about Kinetics-600*. 2018. arXiv: 1808.01340 [cs.CV].

[14] Joao Carreira et al. *A Short Note on the Kinetics-700 Human Action Dataset*. 2019. arXiv: 1907.06987 [cs.CV].

[15] NVIDIA Corporation. *NVIDIA GPU Compute Capabilities*. 2020. URL: https://developer.nvidia.com/cuda-gpus.

[16] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[17] A. M. Derrington and P. Lennie. "Spatial and temporal contrast sensitivities of neurones in lateral geniculate nucleus of macaque". eng. In: *The Journal of physiology* 357 (1984). 6512690[pmid], pp. 219–240. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1984.sp015498. URL: https://pubmed.ncbi.nlm.nih.gov/6512690.

[18] ffmpeg Developers. *ffmpeg*. 2020. URL: https://ffmpeg.org.

[19] Haodong Duan et al. *Omni-sourced Webly-supervised Learning for Video Recognition*. 2020. arXiv: 2003.13042 [cs.CV].

[20] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null (July 2011), 2121–2159. ISSN: 1532-4435.

[21] Dumitru Erhan et al. "Why Does Unsupervised Pre-Training Help Deep Learning?" In: *J. Mach. Learn. Res.* 11 (Mar. 2010), 625–660. ISSN: 1532-4435.

[22] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338. ISSN: 1573-1405. DOI: 10.1007/s11263-009-0275-4. URL: https://doi.org/10.1007/s11263-009-0275-4.

[23] Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. *Spatiotemporal Residual Networks for Video Action Recognition*. 2016. arXiv: 1611.02155 [cs.CV].

[24] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. *Convolutional Two-Stream Network Fusion for Video Action Recognition*. 2016. arXiv: 1604.06573 [cs.CV].

[25] Christoph Feichtenhofer et al. *SlowFast Networks for Video Recognition*. 2018. arXiv: 1812.03982 [cs.CV].

[26] D. Felleman and D. V. Van Essen. "Distributed hierarchical processing in the primate cerebral cortex." In: *Cerebral cortex* 1 1 (1991), pp. 1–47.

[27] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (1980), pp. 193–202. ISSN: 1432-0770. DOI: 10.1007/BF00344251. URL: https://doi.org/10.1007/BF00344251.

[28] W.A Förstner and E. Gülch. "A fast operator for detection and preciselocation of distinct points, corners and centers of circular features". In: *ISPRS* (1987).

[29] Deepti Ghadiyaram et al. *Large-scale weakly-supervised pre-training for video action recognition*. 2019. arXiv: 1905.00561 [cs.CV].

[30] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[31] EDAG Engineering GmbH. *EDAG CityBot – An autonomous transport and working vehicle for the smart city of tomorrow*. 2019. URL: https://www.edag-citybot.de/file/edag-whitepaper-01-2019_en.pdf.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[33] Priya Goyal et al. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2017. arXiv: 1706.02677 [cs.CV].

[34] Richard Gross. *Psychology - The Science of Mind and Behaviour*. Handaya, Tokyo, Hongo Harukicho: Hodder Education, 2010. ISBN: 978-1-444-10831-6.

[35] Facebook AI Research Computer Vision Group. *fvcore*. 2020. URL: https://github.com/facebookresearch/fvcore.

[36] Chunhui Gu et al. *AVA: A Video Dataset of Spatio-temporally Localized Atomic Visual Actions*. 2017. arXiv: 1705.08421 [cs.CV].

[37] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow - Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol: "O'Reilly Media, Inc.", 2017. ISBN: 978-1-491-96226-8.

[38] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2015. arXiv: 1510.00149 [cs.CV].

[39] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. *Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?* 2017. arXiv: 1711.09577 [cs.CV].

[40] C. Harris and M.J. Stephens. "A combined corner and edge detector". In: *Alvey Vision Conference* (1988), pp. 147–152.

[41] Kaiming He and Jian Sun. *Convolutional Neural Networks at Constrained Time Cost*. 2014. arXiv: 1412.1710 [cs.CV].

[42] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[43] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[44] Kaiming He et al. *Mask R-CNN*. 2017. arXiv: 1703.06870 [cs.CV].

[45] D.O. Hebb. *The Organization of Behavior - A Neuropsychological Theory*. Justus-Liebig-Universität Gießen: Taylor Francis, 1949. ISBN: 978-1-410-61240-3.

[46] Prof. David Heeger. *Perception Lecture Notes: LGN and V1*. Accessed July 15, 2020. URL: https://www.cns.nyu.edu/~david/courses/perception/lecturenotes/V1/lgn-V1.html.

[47] Geremy Heitz et al. "Shape-Based Object Localization for Descriptive Classification". In: *International Journal of Computer Vision* 84.1 (2009), pp. 40–62. ISSN: 1573-1405. DOI: 10.1007/s11263-009-0228-y. URL: https://doi.org/10.1007/s11263-009-0228-y.

[48] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [cs.NE].

[49] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: 1991.

[50] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[51] D. H. Hubel and T. N. Wiesel. "Receptive fields of single neurones in the cat's striate cortex". eng. In: *The Journal of physiology* 148.3 (1959). 14403679[pmid], pp. 574–591. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1959.sp006308. URL: https://pubmed.ncbi.nlm.nih.gov/14403679.

[52] David H. Hubel and Torsten N. Wiesel. "RECEPTIVE FIELDS AND FUNCTIONAL ARCHITECTURE IN TWO NONSTRIATE VISUAL AREAS (18 AND 19) OF THE CAT". In: *Journal of Neurophysiology* 28.2 (1965). PMID: 14283058, pp. 229–289. DOI: 10.1152/jn.1965.28.2.229. eprint: https://doi.org/10.1152/jn.1965.28.2.229. URL: https://doi.org/10.1152/jn.1965.28.2.229.

[53] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[54] Jinggang Huang and D. Mumford. "Statistics of natural images and models". In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. Vol. 1. 1999, 541–547 Vol. 1.

[55] Ju Sun et al. "Hierarchical spatio-temporal context modeling for action recognition". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 2004–2011.

[56] M. Esat Kalfaoglu, Sinan Kalkan, and A. Aydin Alatan. *Late Temporal Modeling in 3D CNN Architectures with BERT for Action Recognition*. 2020. arXiv: 2008.01232 [cs.CV].

[57] Andrej Karpathy et al. "Large-scale Video Classification with Convolutional Neural Networks". In: *CVPR*. 2014.

[58] Hirokatsu Kataoka et al. *Would Mega-scale Datasets Further Enhance Spatiotemporal 3D CNNs?* 2020. arXiv: 2004.04968 [cs.CV].

[59] Will Kay et al. *The Kinetics Human Action Video Dataset*. 2017. arXiv: 1705.06950 [cs.CV].

[60] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[61] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[62] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (May 2017), 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[64] H. Kuehne et al. "HMDB: a large video database for human motion recognition". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2011.

[65] Laptev and Lindeberg. "Space-time interest points". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. 2003, 432–439 vol.1.

[66] I. Laptev et al. "Learning realistic human actions from movies". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8.

[67] Ivan Laptev. "On Space-Time Interest Points". In: *International Journal of Computer Vision* 64.2 (2005), pp. 107–123. ISSN: 1573-1405. DOI: 10.1007/s11263-005-1838-7. URL: https://doi.org/10.1007/s11263-005-1838-7.

[68] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1 (1989), pp. 541–551.

[69] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[70] Ang Li et al. *The AVA-Kinetics Localized Human Actions Video Dataset*. 2020. arXiv: 2005.00214 [cs.CV].

[71] Yixuan Li et al. *Actions as Moving Points*. 2020. arXiv: 2001.04608 [cs.CV].

[72] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2014. arXiv: 1405.0312 [cs.CV].

[73] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2016. arXiv: 1612.03144 [cs.CV].

[74] Jingen Liu, Jiebo Luo, and Mubarak Shah. "Recognizing realistic actions from videos in the Wild". In: July 2009, pp. 1996 –2003. DOI: 10.1109/CVPR.2009.5206744.

[75] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2016. arXiv: 1608.03983 [cs.LG].

[76] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2017. arXiv: 1711.05101 [cs.LG].

[77] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, 674–679.

[78] M. Marszalek, I. Laptev, and C. Schmid. "Actions in context". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 2929–2936.

[79] P. Matikainen, M. Hebert, and R. Sukthankar. "Trajectons: Action recognition through the motion analysis of tracked features". In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. 2009, pp. 514–521.

[80] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[81] Raphael Memmesheimer, Nick Theisen, and Dietrich Paulus. *Gimme Signals: Discriminative signal encoding for multimodal activity recognition*. 2020. arXiv: 2003.06156 [cs.CV].

[82] R. Messing, C. Pal, and H. Kautz. "Activity recognition using the velocity histories of tracked keypoints". In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 104–111.

[83] Matiur Rahman Minar and Jibon Naher. *Recent Advances in Deep Learning: An Overview*. 2018. arXiv: 1807.08169 [cs.LG].

[84] Tom M. Mitchell. *Machine Learning* -. New York: McGraw-Hill, 1997. ISBN: 978-0-071-15467-3.

[85] Mathew Monfort et al. *Moments in Time Dataset: one million videos for event understanding*. 2018. arXiv: 1801.03150 [cs.CV].

[86] Charlie Murphy, Patrick Gray, and Gordon Stewart. "Verified Perceptron Convergence Theorem". In: *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. MAPL 2017. Barcelona, Spain: Association for Computing Machinery, 2017, 43–50. ISBN: 9781450350716. DOI: 10.1145/3088525.3088673. URL: https://doi.org/10.1145/3088525.3088673.

[87] Vinod Nair and Geoffrey Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair". In: vol. 27. June 2010, pp. 807–814.

[88] Juan Carlos Niebles, Chih-Wei Chen, and Li Fei-Fei. "Modeling Temporal Structure of Decomposable Motion Segments for Activity Classification". In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 392–405. ISBN: 978-3-642-15552-9.

[89] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[90] Leo Peichl. "Retinal Ganglion Cells". In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 3507–3513. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_5106. URL: https://doi.org/10.1007/978-3-540-29678-2_5106.

[91] Juan-Manuel Perez-Rua et al. *Knowing What, Where and When to Look: Efficient Video Action Modeling with Attention*. 2020. arXiv: 2004.01278 [cs.CV].

[92] AJ Piergiovanni and Michael S. Ryoo. *AViD Dataset: Anonymized Videos from Diverse Countries*. 2020. arXiv: 2007.05515 [cs.CV].

[93] Marcus Pincus. "A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems". In: Brooklyn, NY, USA, 1969. URL: https://pubsonline.informs.org/doi/pdf/10.1287/opre.18.6.1225.

[94] Zhaofan Qiu, Ting Yao, and Tao Mei. *Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks*. 2017. arXiv: 1711.10305 [cs.CV].

[95] Kishore K. Reddy and Mubarak Shah. "Recognizing 50 human action categories of web videos". In: *Machine Vision and Applications* 24.5 (2013), pp. 971–981. ISSN: 1432-1769. DOI: 10.1007/s00138-012-0450-4. URL: https://doi.org/10.1007/s00138-012-0450-4.

[96] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. arXiv: 1506.01497 [cs.CV].

[97] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. DOI: 10.1017/CBO9780511812651.

[98] M. D. Rodriguez, J. Ahmed, and M. Shah. "Action MACH a spatio-temporal Maximum Average Correlation Height filter for action recognition". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8.

[99] Daniel L Ruderman. "The statistics of natural images". In: *Network: Computation in Neural Systems* 5.4 (1994), pp. 517–548. DOI: 10.1088/0954-898X_5_4_006. eprint: https://doi.org/10.1088/0954-898X_5_4_006. URL: https://doi.org/10.1088/0954-898X_5_4_006.

[100] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[101] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575.

[102] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[103] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. 3rd ed. Pearson, 2009.

[104] Christian Schüldt, Ivan Laptev, and Barbara Caputo. "Recognizing human actions: A local SVM approach". In: vol. 3. Sept. 2004, 32 –36 Vol.3. ISBN: 0-7695-2128-2. DOI: 10.1109/ICPR.2004.1334462.

[105] Gunnar A. Sigurdsson et al. *Hollywood in Homes: Crowdsourcing Data Collection for Activity Understanding*. 2016. arXiv: 1604.01753 [cs.CV].

[106] Karen Simonyan and Andrew Zisserman. *Two-Stream Convolutional Networks for Action Recognition in Videos*. 2014. arXiv: 1406.2199 [cs.CV].

[107] Gurkirt Singh and Lucas Mahler. *kinetics-download-prep*. 2020. URL: https://github.com/gurkirt/kinetics-download-prep.

[108] Paul Smolensky. "Information processing in dynamical systems: Foundations of harmony theory". In: *Parallel Distributed Process* 1 (Jan. 1986).

[109] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. 2012. arXiv: 1212.0402 [cs.CV].

[110] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. *Highway Networks*. 2015. arXiv: 1505.00387 [cs.LG].

[111] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1139–1147. URL: http://proceedings.mlr.press/v28/sutskever13.html.

[112] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[113] Graham W. Taylor et al. "Convolutional Learning of Spatio-temporal Features". In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 140–153. ISBN: 978-3-642-15567-3.

[114] Pedro Pasik Thomas P. Naidich Esther A. Nimchinsky. *Cerebral Cortex*. Januar 22, 2016 (Accessed June 25, 2020). URL: https://radiologykey.com/cerebral-cortex/.

[115] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[116] Hugo Touvron et al. *Fixing the train-test resolution discrepancy: FixEfficientNet*. 2020. arXiv: 2003.08237 [cs.CV].

[117] Du Tran et al. *Learning Spatiotemporal Features with 3D Convolutional Networks*. 2014. arXiv: 1412.0767 [cs.CV].

[118] Du Tran et al. *A Closer Look at Spatiotemporal Convolutions for Action Recognition*. 2017. arXiv: 1711.11248 [cs.CV].

[119] David C. Van Essen and Jack L. Gallant. "Neural mechanisms of form and motion processing in the primate visual system". In: *Neuron* 13.1 (1994), pp. 1 –10. ISSN: 0896-6273. DOI: https://doi.org/10.1016/0896-6273(94)90455-3. URL: http://www.sciencedirect.com/science/article/pii/0896627394904553.

[120] Gül Varol, Ivan Laptev, and Cordelia Schmid. *Long-term Temporal Convolutions for Action Recognition*. 2016. arXiv: 1604.04494 [cs.CV].

[121] Chien-Yao Wang et al. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019. arXiv: 1911.11929 [cs.CV].

[122] H. Wang and C. Schmid. "Action Recognition with Improved Trajectories". In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 3551–3558.

[123] H. Wang et al. "Action recognition by dense trajectories". In: *CVPR 2011*. 2011, pp. 3169–3176.

[124] Heng Wang et al. "Evaluation of local spatio-temporal features for action recognition". In: *BMVC 2009 - British Machine Vision Conference*. Ed. by A. Cavallaro, S. Prince, and D. Alexander. London, United Kingdom: BMVA Press, Sept. 2009, pp. 124.1–124.11. DOI: 10.5244/C.23.124. URL: https://hal.inria.fr/inria-00439769.

[125] Limin Wang et al. *Temporal Segment Networks: Towards Good Practices for Deep Action Recognition*. 2016. arXiv: 1608.00859 [cs.CV].

[126] Limin Wang et al. *Appearance-and-Relation Networks for Video Classification*. 2017. arXiv: 1711.09125 [cs.CV].

[127] P. Wang et al. "Large-Scale Unsupervised Pre-Training for End-to-End Spoken Language Understanding". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 7999–8003.

[128] Xiaolong Wang et al. *Non-local Neural Networks*. 2017. arXiv: 1711.07971 [cs.CV].

[129] Xizhao Wang, Yanxia Zhao, and Farhad Pourpanah. "Recent advances in deep learning". In: *International Journal of Machine Learning and Cybernetics* 11.4 (2020), pp. 747–750. ISSN: 1868-808X. DOI: 10.1007/s13042-020-01096-5. URL: https://doi.org/10.1007/s13042-020-01096-5.

[130] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[131] Saining Xie et al. *Aggregated Residual Transformations for Deep Neural Networks*. 2016. arXiv: 1611.05431 [cs.CV].

[132] Saining Xie et al. *Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification*. 2017. arXiv: 1712.04851 [cs.CV].

[133] Can Zhang et al. *PAN: Towards Fast Action Recognition via Learning Persistence of Appearance*. 2020. arXiv: 2008.03462 [cs.CV].

[134] Bolei Zhou et al. *Temporal Relational Reasoning in Videos*. 2017. arXiv: 1711.08496 [cs.CV].