UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

,,,

Video Action Recognition in Basketball

Aslak Niclasen (rql599) & Jonathan Kurish (vgp479)

Master Thesis

Department of Computer Science, DIKU

Supervisor: Kim Steenstrup Pedersen

August 2018

Abstract

Recognizing actions in videos is valuable in a variety of tasks. The surveillance, tourism and medical industry are examples of sectors all benefiting from Video Action Recognition. As actions occur over a period of time, accurately recognizing them is a complex task, because structural information may change over time. The use of Deep Learning has shown promising results in the field of Video Action Recognition.

We implement a two stream model combining two separate ResNet-50 models as presented in [1]. One stream uses spatial information and the other uses temporal information. Besides late fusion, the two streams interact using Multiplicative Gating. Furthermore, Temporal Injections are used to enhance temporal support in the spatial model.

We conduct a number of experiments on UCF101, a benchmark dataset with 101 classes, in order to study the ResNet-50 model. We examine the influence of various hyper-parameters, as well as the depth of the model. Our final two stream model achieved a test accuracy of 80.5%, $\approx$3% higher than using only late fusion to combine the streams.

In addition, we create a new dataset called Basketball Simple Shooting Statistics Dataset, in short B3SD. It is a fine-grained dataset consisting of various basketball videos. We construct multiple versions of B3SD, in order to distinguish videos containing shots from videos without shots, as well as distinguish between shot types. We explore the effect of various input modalities, such as Optical Flow and gray-scaled difference frames. Our final two-part two stream model achieves a test accuracy of 74.7%, using a single RGB-frame as spatial information and a stack of gray-scaled frames as temporal information.

We recommend that future work includes examining the best input modality for temporal information. Additionally, in order to use models trained with B3SD for more practical applications, the quality of the dataset needs further work. Lastly, as B3SD includes bounding box information and multiple actions occurring in a single video, exploration of continuous action recognition and localization models is possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

> *"An action is defined as an act that one consciously wills that may be characterized by physical or mental activity."* [2]

When a basketball player shoots, the action has a specific hand, eye and footwork combination leading up to the shot, providing a motion pattern different from a runner when running. Action recognition in videos describes the task of correctly classifying distinct actions given a video. Similarly, action localization is the task of performing both action recognition, and additionally locating when and where the action occurs in a video.

A video consists of a sequence of individual frames. Each frame contains spatial information (position in space) of objects such as their size, shape and color. In addition, a frame also contains the location of the object in relation to the scene and other possible objects. A single frame does not contain any information about the motion pattern, as it changes over time. However, a sequence of frames does. Because the objects' location, shape or size may change over time, indicating motion, one can utilize this information, and possibly resolve it into an action. This is called temporal information. To solve the task of action recognition and localization in videos both spatial and temporal information is useful.

Our goal is to perform accurate fine-grained continuous action recognition and localization in a realistic amateur video setting. Fine-grained actions are actions with similar motion patterns (e.g tennis vs. squash). Continuous action recognition refers to the task of evaluating the given input frame by frame. A limitation of continuous action recognition is how fast a frame is processed. Some achievements such as YOLO [3] are able to process frames at a speed of more than 25 frames per second.

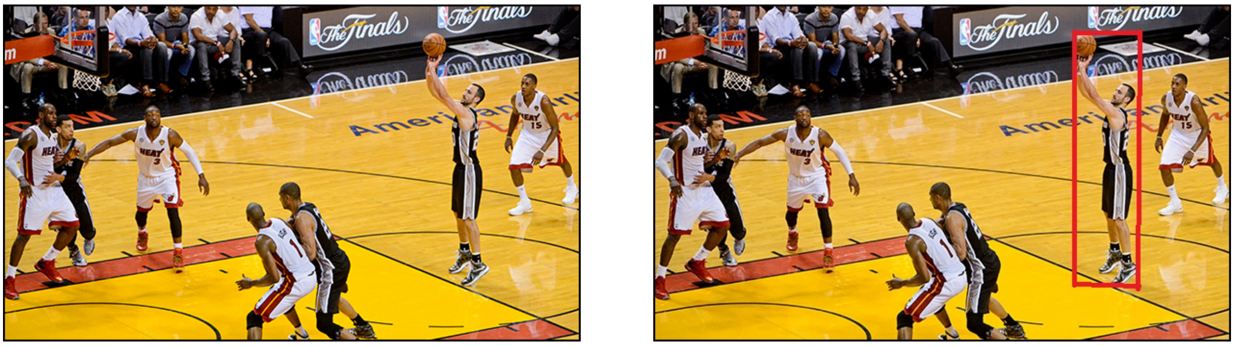


FIGURE 1.1: An example of the two tasks: action recognition and action localization. On the right, a bounding box is used to mark the location of the classified action taking place, whereas the left image only considers the action. Image courtesy of the NBA.

This paper provides theoretical background of Neural Networks used in vision related tasks. We experiment and discuss how to choose hyper-parameters while also performing action recognition on a new dataset called B3SD. The reader is expected to have introductory background knowledge of Machine Learning and Image Analysis. Chapter 2 introduces the various related work within the field of action recognition and localization. Chapter 3 provides the theoretical background of Neural Networks and motion encoding such as Optical Flow. Chapter 4 introduces a collection of datasets used in action recognition. Chapter 5 describes our new dataset, B3SD. Chapter 6 introduces our single stream model based on a 50-layer Residual Network from [4]. Chapter 7 presents a preliminary evaluation of hyper-parameters and the architecture of the model. Chapter 8 introduces our two stream model as presented by Feichtenhofer et al. in [1], while remaining chapters explain the obtained results, discussion and future work.

Chapter 2

Related Work

Action recognition has been an area of interest in the computer vision community for many years due to its possible applications in video surveillance and human computer interaction. Many different approaches to the problem have been proposed and built upon. We first present early recognition methods using feature descriptors and various encoding strategies. Then we present an overview of newer recognition and localization methods surpassing the early methods.

2.1 Feature Descriptors

Early action recognition methods for both images and videos were based on global features. A feature is a pattern or structure which provides relevant information in order to solve a recognition problem. As an example, extracting the human body from the surrounding scene resulted in a silhouette or pose, which could be classified. In addition, a global Optical Flow can be used to separate a dynamic object in a static scene. If global camera motion is present, however, Optical Flow is problematic because the entire scene is dynamic. Instead of using global features, local features were used. Especially for action recognition spatiotemporal features (the combination of spatial and temporal features) were useful. The Harris Corner detector presented in 1988, used to find corners in images, was expanded to 3 dimensions by Laptev and Lindeberg in 2003 [5] as what is known as 3D space-time interest points (STIPs). Another local feature to use is image saliency. The saliency of an image corresponds to a sub region which is perceivable at an instant. As an example consider an image with a small black square on a white background. The saliency is the ability to instantly observe the sub region i.e. the square. The idea was presented in 2001 by Kadir et al. [6] and extended to 3 dimensions by

Ikonomopoulos et al. [7] in 2005.

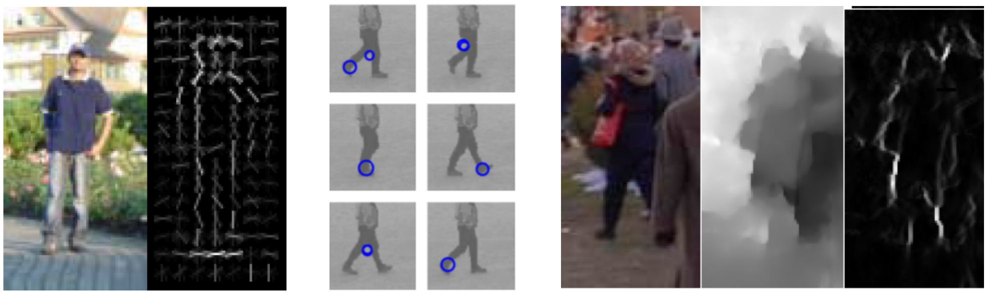


FIGURE 2.1: Example of various feature extraction methods. **Left**: The HOG descriptor on an image of a person. **Middle**: Detected STIP on a person walking. **Right**: The MBH descriptor calculated by taking the gradients on the Optical Flow of an image. Only one gradient direction is shown. Image courtesy of the authors.

Other extraction approaches based on 2D descriptors such as SURF and SIFT were also extended to 3D by Bay et al. [8] and Scovanner et al. [9]. Later work makes use of the Histogram of Oriented Flow (HOF) and Histogram of Oriented Gradients (HOG) by Dalal et al. [10] and Motion Boundary Histograms (MBH) [11], all of which proved to be effective at the time. Finally, another feature was introduced in 2011 by Wang et al., called Dense Trajectories (DT) [12]. Dense Trajectories make use of both HOG and HOF, and were later improved by the same authors [13].

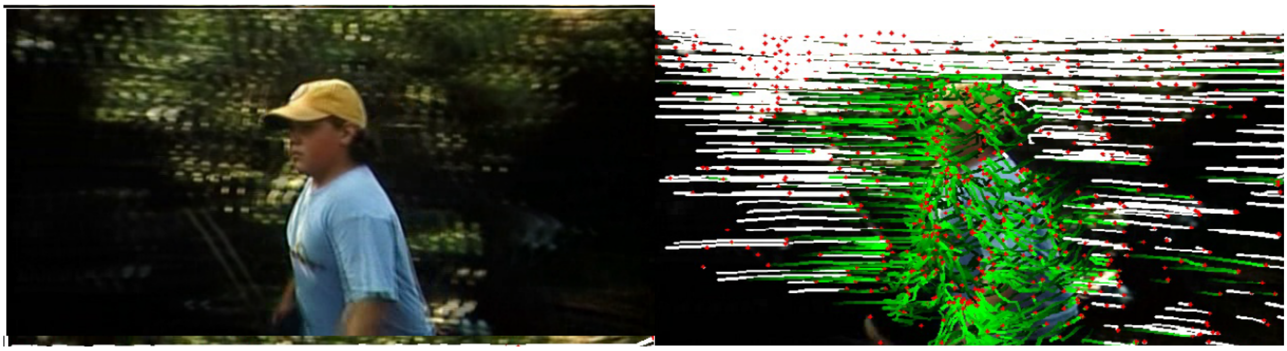


FIGURE 2.2: Example of improved Dense Trajectories (iDT). The trajectories drawn on the image in white are later removed in order to compensate for camera motion providing a better estimation of the boy running. Image courtesy of the original authors.

2.2 Encoding Strategies

It is crucial to represent the information of the feature descriptors in a reasonable way. There exist different strategies on how to encode the data. We briefly mention aspects of a well known encoding strategy called Bag of Words (BoW). This strategy initially extracts features before clustering them to form codewords (using k-means or similar). The centroid of a cluster represents a codeword and a set of codewords is called a codebook. After codebook generation, an assignment of new features from new data is possible through an assignment strategy. Assigning a new feature to a single codeword is called a hard assignment. A popular hard assignment approach, called Vector Quantization (VQ), assigns a feature descriptor to the nearest codeword based on the euclidean distance to the centroid [14]. However, features with similar distance to multiple centroids often change codeword, even if only a small change of centroids occur. To overcome this, soft assignments have proven to be more effective. Features of soft assignments are either described as a combination of the codewords or by their differences. If described as a combination, a membership weight is associated to each codeword based on the feature. Determining the weight can be found by solving an optimization problem, containing a special regularization term. Depending on the regularization term used, different soft assignments are used. Examples include Orthogonal Matching Pursuit (OMP) [15], Sparse Coding (SpC) [16], Local Coordinate Coding (LCC) [17] and Locality-constraint Linear Encoding (LLC) [18].

If instead described by the differences, two popular approaches are used. Vector of Linearly Aggregated Descriptors (VLAD) [19] from 2007 and Fisher Vectors (FV) [20] from 2010. Note that Fisher Vectors require a Gaussian Mixture Model to represent the feature space instead of the most commonly used k-means algorithm.

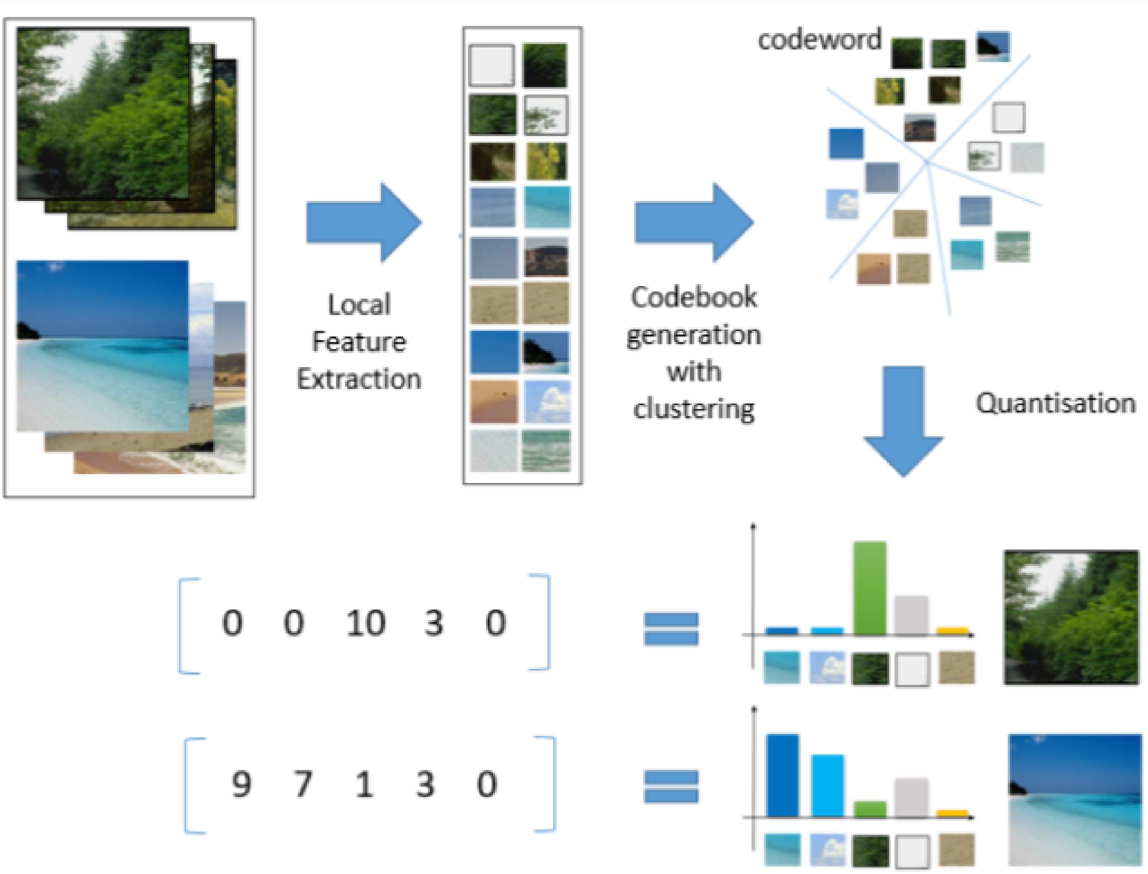


FIGURE 2.3: A Bag of Words approach illustrated with images. First features are extracted and clustered based on similarities. Here, the 5 centroids are encoded as vectors. An image can be mapped to a vector representing the saliency of each codeword as a histogram. Image courtesy of Chee Seng Chan.

When applying a BoW approach, additional encoding might be needed depending on the task at hand. If the available data is incompatible with the algorithm used, pooling is often applied. Pooling has the capability of maintaining important information while lowering the dimensionality of the data. Various types of pooling such as maximum and stochastic pooling exist. Pooling has certain trade-offs between disregarding information and reducing the dimensionality of the data. Typically linked with pooling is normalization of the data. It provides a way to maintain consistency throughout the data. Common normalizations include l_1-, l_2-, power- and intra-normalization. As an example, l_2-normalization is the square root of the sum of the absolute values squared. We will return to pooling and normalization in more detail in chapter 3.

2.3 Action Recognition and Localization in Videos

This section provides an overview of recent methods of action recognition and localization in videos. Action recognition methods can be broadly separated into handcrafted features for classifying actions, or deep learning methods to automatically learn features for classifying actions.

2.3.1 Handcrafted Features

Back in 2013 Wang et al. used FV and iDT in combination with HOG, HOF and MBH achieving accuracies of 57.2% and 85.9% on HMDB51 and UCF101 - two benchmark datasets described in chapter 4 [13, 21]. Recently, in 2017, an unsupervised learning method by Soomro et al. was presented [22]. Their work applies a clustering approach organizing videos in an undirected graph. Each video is initially represented as a supervoxel. A Support Vector Machine is then trained against a dominant subset of each cluster in order to classify new videos. The clustering is based on iDTs being encoded as FVs during supervoxel segmentation.

2.3.2 Deep Architectures

As various Neural Networks (NNs) have been successful in solving image classification tasks, they are a popular choice when trying to perform video action recognition. Varol et al. [23] and Simonyan et al [24] both proposed a two-stream NN for video recognition. A two-stream NN consists of decoupling the spatial and temporal contexts in separate Convolutional Neural Networks (CNNs). At later stages, a fusion of networks using either a Softmax Layer [23] or a linear Support Vector Machine [24] was applied. Another approach based on Recurrent Neural Networks (RNN) was proposed by Bingbing et al. [25], improving fine-grained action recognition on the MPI-II cooking dataset. As an extension to RNNs, networks using Long Short-Term Memory (LSTM) achieved higher accuracy. The state of the art results, however, were achieved using a combination of handcrafted features and a two stream approach [1, 26–29]. Most of them incorporated iDT and FV.

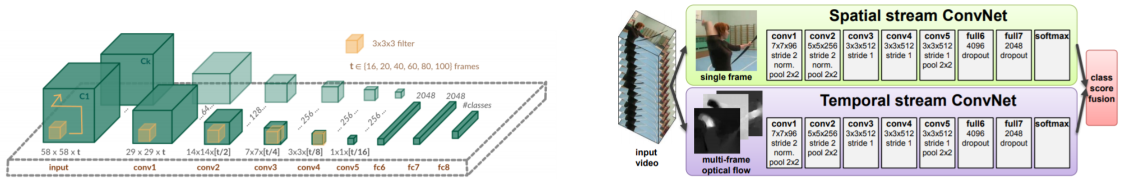


FIGURE 2.4: Different visual representations of two similar two stream network architectures. **Left**: The network presented in [23]. **Right**: The network from [24]. Both networks apply a late fusion strategy.

Feichtenhofer et al. [1] proposed a two-stream architecture with residual connections. The network contains intermediate fusions between the two streams. Incorporating iDTs, they obtain the current 2nd highest accuracy on the HMBD51 and UCF101 datasets with 72.2% and 94.9% accuracy, respectively. One problem of this network is its' number of layers. Zhenyang et al. [30] solves both action recognition and action localization by applying a temporal pyramid representation of a video clip. At the pyramid upper layer, a discriminative model using classifiers for actions is used in a CNN. A scheme using temporal action grouping is devised at lower levels in order to create action localization proposals for the classifiers. They obtain an accuracy of 88.9% and 56.4% on UCF101 and HMDB51, respectively. Finally, Zhenyang et al. [30] introduce an adaptable architecture to fit the video medium itself using an RNN based on Attention-LSTM, that also performs both action recognition and action localization. Zhu et al. [31] proposed to not fuse handcrafted features with deep learning as the first step and instead find key volumes in the video (a section of a video where the action occurs). These sections are often sparse in untrimmed videos, which motivates training classifiers based on these volumes only. They obtain test accuracies of 93.1% on UCF101 and 63.3% on HMDB51. Table 2.1 summarizes the achieved accuracies.

Localization as a separate area of interest has also been explored in various unsupervised and supervised settings. One attempt at solving the localization problem is from 2014 by Oneata et al. [32]. Here, a video is represented as a supervoxel. The approach uses a per-frame based superpixel hierarchical clustering. A superpixel is retrieved using the SLIC algorithm. The authors report the Best Average Overlap (BAO) on the earliest version of UCF101 called UCF-Sports and obtain state of the art results. Hongyuan et al. [33] presented a two-stream architecture in 2017 based on a fusion of Convolutional LSTM and regression based detectors. This combination exploits the power of regression based detectors in object detection in images, while modelling the temporal context with LSTM. The accuracy of the action proposals as bounding boxes achieve state of the art precision at the time compared to existing localization methods [34–36]. Most recently, in 2018 by Ahsan et al. [37], a General Adversary Network (GAN) was shown to outperform existing localization methods by only incorporating spatial information as a restriction. Note that the overall performance is lower than that of [33], but outperforms if only spatial information is included.

The localization approaches presented above are all computationally expensive and thereby not very useful in a continuous setting. To perform fast and accurate action localization, the authors behind You Only Look Once (YOLO) [3] use a single frame, decreasing computational cost. Similar to YOLO, the Single Shot Detector (SSD) [38] also works in a continuous setting by considering single frames. The first edition of the SSD was faster than YOLO, but the current version (version 3) of YOLO, is both faster and more accurate processing up to 30 frames per second.

	UCF101	HMDB51
Soomro et al [22]	61.2% *	N/A
Wang et al. [28]	68.1% **	N/A
Wang et al. [21]	85.9%	N/A
Joe NG et al. [27]	88.6%	N/A
Wang et al. [13]	91.2% ***	57.2%
Simonyan et al. [24]	88.0%	59.4%
Zenyang et al. [30]	88.9%	56.4%
Fernando et al [26]	91.4%	66.9%
Varol et al. [23]	92.7%	67.2%
Zhu et al. [31]	93.1%	63.3%
Feichtenhofer et al. [29]	93.5%	69.2%
Feichtenhofer et al. [1]	94.9%	72.2%

TABLE 2.1: Summary of accuracies with different approaches on UCF101 and HMDB51. *Unsupervised classification. **The authors used the Thumos15 dataset which is the UCF101 dataset. ***This result was on a subset of the UCF101 dataset, the previous version called UCF50.

Chapter 3

Background

Machine Learning, Neural Networks & Deep Learning

Machine Learning is a field of computer science, studying and applying algorithms that aim to learn and recognize patterns from data. Such algorithms learn by repetitively processing information and improving their performance [39]. Machine Learning algorithms have various use cases such as speech recognition, artificial intelligence, video recognition, and medical image segmentation. A large portion of these algorithms are varying types of Neural Networks (NNs).

Neural Networks regained their popularity in the 1980s as a result of Jon Hopfield's work in 1982 [40], even though the fundamental idea was present in the 1940s. Jon Hopfield's contribution is the use of NNs to compute logical circuits. By the start of 1990s, researchers held conferences and contributed to the field of computing developing NNs as introduced by Hopfield. In 1998 and 2004, Yann LeCun showed that CNNs had the capability to solve vision related problems [41, 42]. Despite all of the above, computational requirements were a problem, and only a few institutions and companies had the resources available to use this technology. Over the last 20 years, however, the amount of available resources has increased. Combined with new high-level frameworks such as Tensorflow, Theano and Caffe [43–45], the use and research of NNs has increased drastically.

NNs got their name as inspiration from biology. The interplay between a collection of neurons linked with synapses allows the brain to solve complex recognition tasks. A mathematical representation of this structure is likewise believed to achieve the same. NNs are supervised learning algorithms, represented as oriented graphs with nodes and edges. Each edge has a weight associated to it. The graph is organized in a hierarchical collection of layers with the first layer corresponding to the input data, and the last layer corresponding to the output. Layers in-between are referred to as hidden layers [46]. This terminology is used, since the internal mechanisms used to produce outputs are hidden to the user of the network. The total number of layers, excluding the input layer, is called the depth of the network. The very first NNs called Feed-Forward Neural Networks (FF-NNs) are based on this structure. An example is shown below in figure 3.1. Other types of NNs such as CNNs make use of the hierarchical structure, while the layers themselves are composed differently.

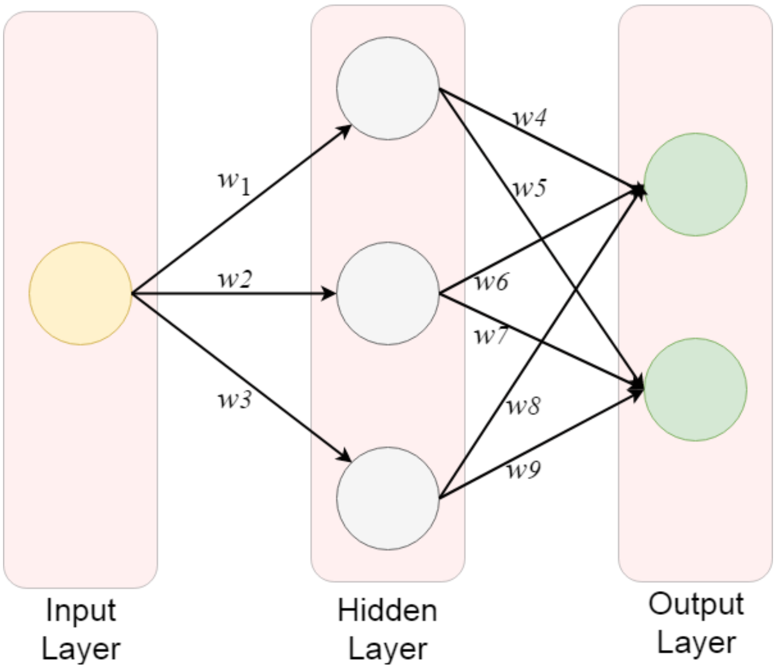


FIGURE 3.1: An example of a Feed-Forward NN, containing 2 layers and 9 weights. Note how each node is fully connected to every node in the previous layer. It is called a Fully Connected Layer. The same is true for the hidden layer.

Most NNs have three phases. A training phase, a validation phase and a test phase. In its training phase, a NN updates the weights of the network to decrease the error between a predicted output and the true labels known beforehand [47]. The error between the prediction and the true label is determined by a loss function. The weight update procedure is performed using an iterative method called backpropagation, which exploits the chain rule of differential calculus. The goal of the update procedure is to minimize the error response of the loss function.

The validation phase is used as a validation of the networks learning ability. The error between the prediction of an input and its true label in the validation phase, is used to check if the network's ability to learn based on the training data is improving [39]. In the test phase, unseen input is presented to the model, and an output prediction from the network is used to calculate the accuracy. This phase does not update weights. Therefore data presented in the training phase, may not be included in the test phase. A full dataset is therefore split into two or three distinct subsets, where each subset corresponds to the phase. The split sizes may vary. Most often the training phase should contain more elements of the dataset, than the testing phase.

Deep Learning is a term used to describe the increased depth of the architecture. Some Deep Learning architectures use NNs with more than 15 or even 50 hidden layers [4, 48], thus drastically increasing the complexity. This leads to the topic of proving why deep learning algorithms are capable of learning complex problems, a task which is not yet fully understood [49, 50].

The remainder of this chapter is divided into 5 sections. The first section provides details of the internal mechanisms used in all NN architectures. In particular, this section introduces the Fully Connected Layer, activation functions, loss functions, gradient descent and backpropagation. The next section introduces Convolutional Neural Networks, including Convolutional Layers and Pooling Layers. An extension of the Backpropagation Algorithm is presented with respect to the new layers. The third section introduces a more recent concept called Residual Connections, used in Residual Networks (ResNets). The fourth section presents overfitting and a number of possible ways to reduce it. The final section provides an overview of how to encode motion in videos.

3.1 Feed-forward Neural Networks

A FF-NN as the one previously shown in figure 3.1 is a very simple example of a NN. Here, information flows from input to output, through each layer. This can be expressed as a sequence of function compositions. Given an input x_l in layer l and a set of learnable weights for that layer w_l outputting x_{l+1}, we express the information flow through the network as

$$f(x) = f_L(\cdots f_2(f_1(x_1, w_1); w_2)\cdots), w_L) \tag{3.1}$$

If all nodes in a layer are connected to all nodes in the previous layer, the layer is referred to as a Fully Connected Layer (FC Layer). Early NNs consisted of a sequence of FC Layers from input to output. Information is passed through each individual node by combining the results of the information flowing to that node from the previous layer. Individual nodes in a FF-NN are called Perceptrons. A Perceptron combines the information from the incoming inputs and the edges to an output. The output of a Perceptron is defined as:

$$\texttt{output} = a\left(\sum_i w_i x_i + b\right) \tag{3.2}$$

Here, a is the activation function, w_i is the i'th weight, x_i the i'th input and b is the bias. Figure 3.2 illustrates this.

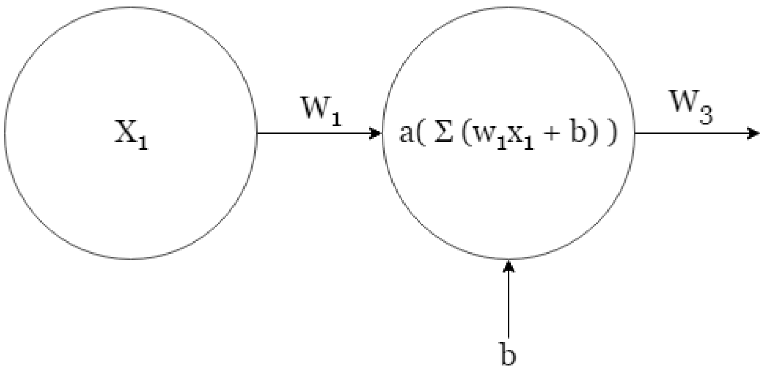


FIGURE 3.2: An illustration of the Perceptron, a single node in a FF-NN. It combines the input and the connected edges to form an output for later nodes. A bias and an activation function are also applied during this computation. Note that the illustration only shows a single input source. Therefore the summation sign is redundant in this case.

3.1.1 Activation Functions

An activation function potentially introduces non-linear properties to a NN. Without them, a NN is a linear function. A linear function can not model a complex mapping from input to output. An activation function could be any type of function as long as it is differentiable, which is needed in order to train the network with backpropagation. Although there are very few limitations when choosing an activation function, there are a number of functions often seen in NNs. One of them is the Step function defined as

$$step(x) = \begin{pmatrix} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{pmatrix} \tag{3.3}$$

A step function can be thought of as a node ”firing”. It returns 1, if the input is positive as opposed to when the input value is negative, resulting in 0. The step-function is not used in practice since the derivative is 0 at all points on the x-axis except for when $x = 0$ when the derivative is undefined and therefore unusable. Additionally, the step function re-scales input to fit within the defined range. A small weight change could lead to a full change from 0 to 1 affecting the final output in ways that are unwanted. The step-function is shown below in figure 3.4. The Sigmoid function [51] is similar to the step function and defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

Here, the output range is between 0 and 1, similar to the step-function, but the function is smooth and continuous. The Sigmoid function is non-linear and is able to describe more complex relations between input and output. However, the Sigmoid function also has limitations. When a weight is very large or very small, the corresponding gradient will be very small. This is known as the vanishing gradient problem [46]. Similar to the Sigmoid function, the Tanh function is defined as

$$tanh(x) = 2\sigma(2x) - 1 \tag{3.5}$$

It is a scaled Sigmoid function with the same characteristics. It also suffers from the vanishing gradient problem, but it is easier to utilize due the steeper curve and zero-centered output. The tanh function is preferred over the Sigmoid function, because the values of the Sigmoid function are between 0 and 1. Due to this an increase of one weight together with a decrease of another, results in a zig-zagging behaviour during learning. In practice this leads to slower learning [52]. Lastly, a different function called the ReLU activation function has emerged [53]. It is defined as

$$ReLU(x) = max(0, x) \tag{3.6}$$

The function avoids the vanishing gradient problem and has been found to have a faster convergence in practice than the Sigmoid and Tanh functions [54], due to the linear gradient. An example of applying the ReLU activation function is shown below in figure 3.3.

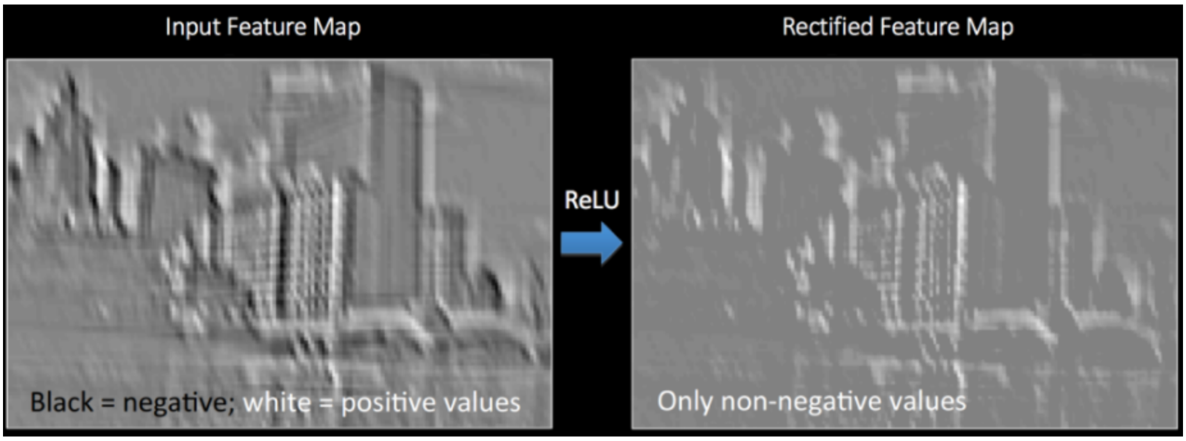


FIGURE 3.3: **Left:** An Input Image. **Right:** The result of applying a ReLU activtion function on the input image. The black pixels in the output correspond to negative values in the input. The result in the right image consists of non-negative values only.

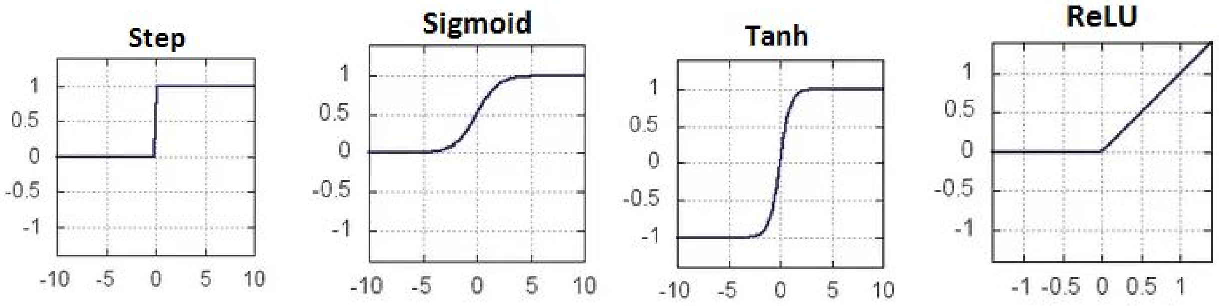


FIGURE 3.4: The four common activation functions presented.

A limitation of ReLU is that gradients may be non existent during training which could potentially decrease the ability to learn. Such a node is said to ”die”, because it does not contribute. In order to adjust for this, Leaky ReLUs can be used. This type of ReLU is able to avoid dying neurons by having a small slope below zero instead of only zero. Note that it might in some cases be beneficial to have dead nodes in a network, if the contribution of such nodes would produce inferior results during the training phase.

3.1.2 The Softmax Function

The Softmax function [55] is another type of activation function which is typically used in the output layer in order to convert the values into a representation that is useful for classification problems. In practice, the value in each output node is converted to a number between 0 and 1, with all node values summing to 1. By using the Softmax function, the value of each output node can be viewed as the probability of each possible output and the node with the highest value can then be chosen as the prediction class. This is an important part of network training, as it can be used to compare the predictions of the model to the true labels. Given S, $S : z \in \mathbb{R} \rightarrow [0; 1]$. The Softmax function is defined as:

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \qquad i = 1, 2 \ldots N \tag{3.7}$$

Here, z_i is a single value in vector z which contains the result of each output node from equation (3.2). e is the exponential function.

3.1.3 Loss Functions & Cross-Entropy Loss Function

A loss function is used to describe the difference between the network output and the ideal output. Consider the actual label of an input being 1. If the result of processing that input is a Softmax score of 0.9, the observed difference indicates, that the network should update its weights, such that the same input processed a second time would be closer to the true label, 1. This observation is key, because the update of the weights depend on the loss function used. The more we can minimize the loss, the closer the processed output is to the true label. The Cross-Entropy loss function is a commonly used loss function for multi-class classification tasks in NNs [43]. For C classes the Cross-Entropy loss with respect to weights in a NN is defined as

$$L(w) = -\sum_{n=1}^{N}\sum_{c=1}^{C} \hat{y}_{nc} ln \left[y_c(x_n; w) \right] \tag{3.8}$$

Here, $\hat{y}_{nc}$ is the true label of the n'th node at the c'th entry in the output vector. The function has the benefit of learning quickly compared to other loss functions. This is because large errors result in large changes, which ultimately minimizes the total loss faster. Note how an actual output from a single node x_n is a number between [0;1], because of the Softmax function. This is because a large set of classes would result in a strong penalty if the target was the class index itself. It is undesirable assuming class ordering is irrelevant for the problem. An encoding method called one-hot-encoding is used for y_{nc}. It converts the true label to a C-dimensional vector of binary values, where the target class is encoded as 1, and all other classes are encoded as 0.

3.1.4 Feed-Forward Neural Network Learning

The key insight into how all types of Neural Networks learn is based on the fact that the loss function L can be minimized using gradient descent [39]. That is, we iteratively seek to find the optimal weights, $\bar{w}$, that minimize the function

$$\bar{w} = \underset{w}{\text{argmin}} \quad L(w) \tag{3.9}$$

Finding the global minimum of the loss function indicates that the prediction would perfectly match the true label and the loss is 0. Visually, consider a random starting position on the loss function. Lowering the loss corresponds to taking a step in the opposite direction of the gradient during an iteration. If the gradient is not 0, another step is taken. In the end, the minimum is reached, and the loss of the function is minimized as in figure 3.5.

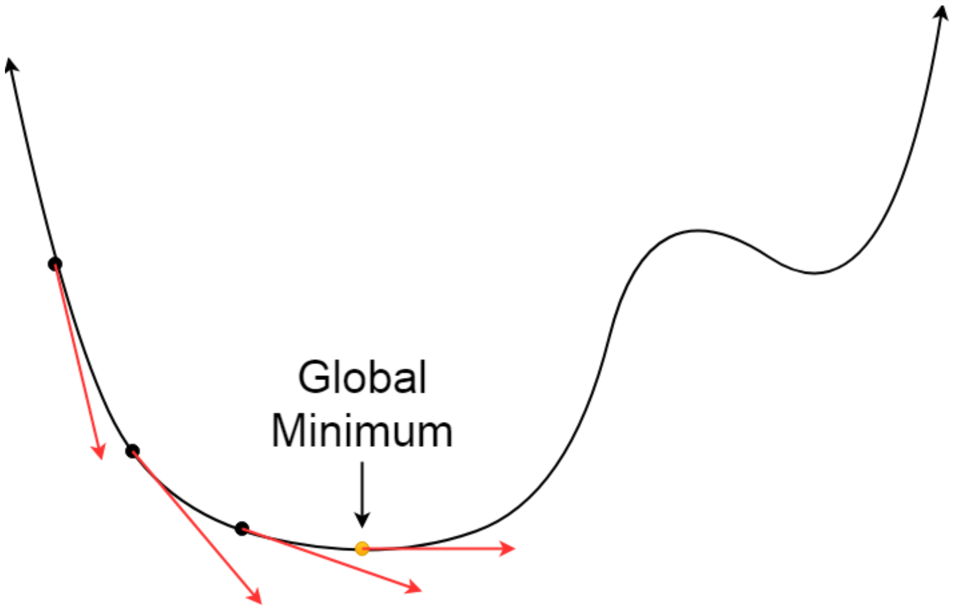


FIGURE 3.5: An example of an iterative update of the total loss performed using gradient descent, by taking a step in the direction opposite of the gradient until a minimum is reached.

The updating procedure of a set of weights with gradient descent is defined as

$$w_t = w_{t-1} + \bigtriangledown w_{t-1} \tag{3.10}$$

Here, the weights at time step t is calculated by adding the weights at the previous steps with their gradients. The algorithm for calculating the gradient of a weight is called backpropagation. As the name suggests, a gradient of the output layer is propagated back through the network.

3.1.5 Backpropagation in Feed-Forward Neural Networks

The terminology and notation used for describing the backpropagation algorithm is outlined below [46, 56].

w_{ij}^k denotes the weight for node j in the k'th layer from the node i in the layer $k-1$

b_i^k denotes the bias for node i in the k'th layer.

a_i^k denotes the weighted sum plus bias for node i in k'th layer

o_i^k denotes the output of the i'th node in the k'th layer

n^k denotes the number of nodes in the k'th layer

f denotes an activation function

Finding $\frac{\partial L}{\partial w}$ gives us the desired result (super and subscripts omitted). In order to calculate the weights, note that the loss function can be decomposed into a sum of the individual losses for each input-output pair (in the end all pairs are summed and then averaged). For an input-output pair, we use to the chain rule and calculate

$$\frac{\partial L}{\partial w_{ij}^k} = \frac{\partial L}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k} \tag{3.11}$$

The expression states, that change of the loss function with respect to a weight, w, is the product of the change of the loss function with respect to the activation, multiplied by the change in the activation due to that weight. The term $\frac{\partial L}{\partial a_j^k}$ is often called the error, not to be confused with the error of the loss function itself. The error term is written as $\delta j^k = \frac{\partial L}{\partial a_j^k}$. Observe that the term a_i^k can be written as

$$a_i^k = \sum_{j=0}^{n^{k-1}} w_{ji}^k o_j^{k-1} \qquad \texttt{if} \quad w_{0i}^k = b_i^k \quad \texttt{and} \quad o_0^{k-1} = 1. \tag{3.12}$$

The above is similar to the Perceptron, except that the information flow has switched direction. It is instead the layer ahead, o_j^{k-1}. rewriting the second term of equation (3.11) with (3.12) yields

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k} \left(\sum_{l=0}^{n^{k-1}} w_{li}^k o_l^{k-1} \right) = o_i^{k-1} \tag{3.13}$$

Combining this with the error term, the change of the loss L with respect to the weight w_{ij}^k gives us

$$\frac{\partial L}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} \tag{3.14}$$

This shows us that the following layers all influence the loss of the current layer being processed. In particular, the following observation of the error term with respect to equation (3.11)

$$\delta_j^k = \frac{\partial L}{\partial a_j^k} = \sum_{l=1}^{n^{k+1}} \frac{\partial L}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k} \tag{3.15}$$

We substitute the term a_l^{k+1} with the definition in equation (3.12), resulting in

$$a_l^{k+1} = \sum_{j=1}^{n^k} w_{jl}^{k+1} f(a_j^k) \tag{3.16}$$

We get the derivative by applying the chain rule

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} \frac{\partial f}{\partial a_j^k} \tag{3.17}$$

Substituting the above into the original expression for δ_j^k gives us

$$\delta_j^k = \sum_{l=1}^{n^{k+1}} \delta_j^{k+1} w_{jl}^{k+1} \frac{\partial f}{\partial a_j^k} \tag{3.18}$$

Finally substituting this into the very first observation in equation (3.14) results in

$$\frac{\partial L}{\partial w_{ij}^k} = \delta_j^{k+1} o_i^{k-1} = \left(\sum_{l=1}^{n^{k+1}} \delta_j^k w_{jl}^{k+1} \frac{\partial f}{\partial a_j^k} \right) o_i^{k-1} \tag{3.19}$$

This term explains that the amount of error in layer k depends on the errors in layer $k+1$. The error is being propagated through the network from the last layer, to the first layer. This is desirable because it is only necessary to calculate the error terms in the final layer based on the output, and propagate this backwards. The gradients found during backpropagation are based on all individual input-output pairs. A single weight update therefore requires the entire dataset to be processed. This is called Batch Gradient Descent, but is time consuming when the number of training samples grow. To reduce the time between updates, a variation called Stochastic Gradient Descent (SGD) can be used [39]. SGD is defined as

$$\frac{\partial L(X)}{\partial w_{ij}^k} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial L_n}{\partial w_{ij}^k} \tag{3.20}$$

Here the weights of the network are updated after each individual sample. The frequent weight updates result in a possibly noisy gradient [57], which is undesirable. To reduce this effect, a random subset of all samples can be selected instead. This is referred to as Mini-Batch Gradient Descent. With Mini-Batch Gradient Descent we also obtain faster convergence than for Batch Gradient Descent, because the update is more frequent. Figure 3.6

shows the three variations of gradient descent. Choosing the optimal mini-batch size, however, is challenging. Setting it too low increases the zig-zagging behaviour as shown by SGD in the figure.

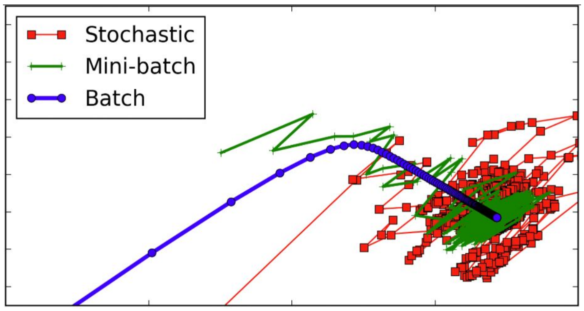


FIGURE 3.6: Optimizing the loss function with different batch variations.

In practice, Mini-Batch Gradient Descent is used to train NNs but is referred to as Stochastic Gradient Descent due to its stochastic nature. The new weight, w^t, is calculated using the gradient, g, calculated in equation (3.20) combined with a term called the learning rate, denoted η, and the previous weight, w^{t-1}.

$$w^t = w^{t-1} - \eta g \tag{3.21}$$

Recall that when updating the weight a step is taken in the opposite direction of the gradient, in hopes of decreasing the loss. Choosing η is hard. If η is set too high, we risk overshooting. Setting η too low, however, results in slow convergence as an increased number of steps are needed to reach the minimum loss. Additionally, by setting η too low, we increase the risk of converging at a local minimum. To improve learning, η is typically adjusted during training. This, however, introduces the challenge of how to select the correct initial η as well as when and how to change it. A common strategy used, called step decay, uses an initially high η (e.g. 0.1), and systematically lowers it. An example is shown in figure 3.7. The idea is to quickly approach the minimum loss, but when getting close, one gradually takes smaller steps to avoid overshooting. Other variations include exponential decay and iteration based decay, known as $\frac{1}{t}$ decay [58].

More complex techniques using adaptive learning rates include RMSprop [59], AdaGrad [60], AdaDelta [61] and ADAM [62].

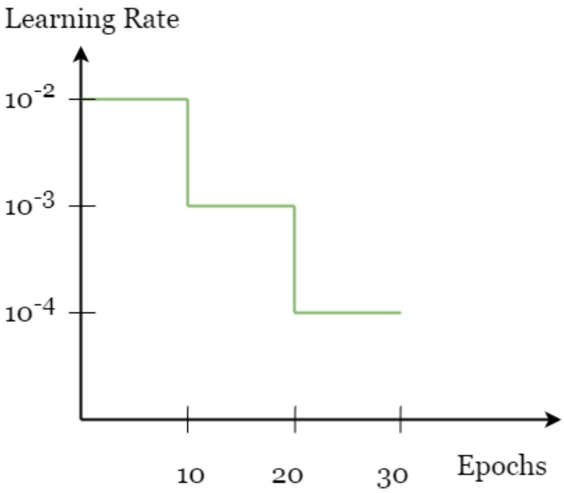


FIGURE 3.7: Simple step decay scheme, lowering η by a factor of 10 per 10 epochs.

ADAM, short for Adaptive Moment Estimation, has seen increased adoption recently. ADAM calculates an exponential moving average of the gradient and the squared gradient. It uses the parameters β_1 and β_2 to control the decay rates of the moving average. ADAM has shown to work well in practice.

3.2 Convolutional Neural Networks

CNNs have a good reputation because they have shown to be able to solve important vision and image based problems [58, 63]. The layers special to a CNN use properties from signal processing theory, which make them good at learning features for vision based problems.

3.2.1 Convolution & Correlation

Convolutions and correlations originate from the field of signal processing. A convolution describes the integral of a function sliding over another function. A correlation is similar but the operations differ because correlation does not obtain the associative property of function composition. In computer vision the input, however, is not a function but an image represented as an N-dimensional matrix. A discrete 2D correlation can be interpreted as a matrix, called a kernel or a filter, sliding across the input. For each position (x, y) in the input, the output is defined as

$$(I \circ f)(x, y) = \sum_{u=0}^{f_w} \sum_{v=0}^{f_h} f(u, v) I(x + u, y + v) \tag{3.22}$$

where f_w and f_h are the width and height of the kernel f and I is the input. The output of the operation produces a new image, I'. In figure 3.8, an example of applying the correlation operation with a 3×3 kernel on a single position is shown. A valuable property of correlations is that they can be implemented efficiently with matrix multiplication as dot products.

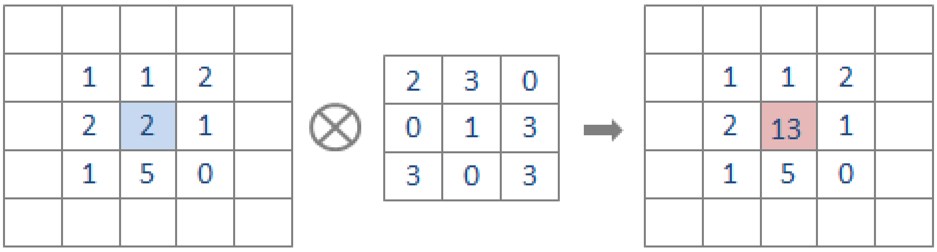


FIGURE 3.8: an example of applying the correlation operation with a 3×3 kernel on a single position for a 2D input. The center pixel in the leftmost grid represents the current position (x,y) which is being evaluated to 13 as the output in the rightmost grid.

An n-dimensional data representation can be encoded as an n-dimensional matrix, often referred to as a tensor. In the case of an RGB image, each pixel intensity at a position (x,y) is a value between [0,255] for each color channel. This can be encoded as a 3D tensor: $I \in \mathbb{R}^{w \times h \times d}$, where w and h denotes the width and height of the image and d denotes the number of channels [64, 65]. For example, if a 2D correlation is applied to an RGB-image, the kernel must have the same number of channels, namely 3, because a correlation slides across the entire input. If the kernel had to slide along the third dimension as well, it would be called a 3D correlation.

The reason for mentioning correlation is quite misleading. In practice, term convolution is used even though actual implementations of CNNs make use of correlation [66]. This confusion may be explained by the fact that a discrete convolution is identical to a discrete correlation if the kernel is symmetrical. For this reason we refer to a correlation operation as a convolution from this point forward. When applying convolutions at the border of I a choice must be made because it affects the width and height of the output. There are four common ways to treat borders: Zero padding, reflection, cloning and completely discarding the convolution at the borders [67]. An example of zero padding, cloning and discarding is shown in figure 3.9.

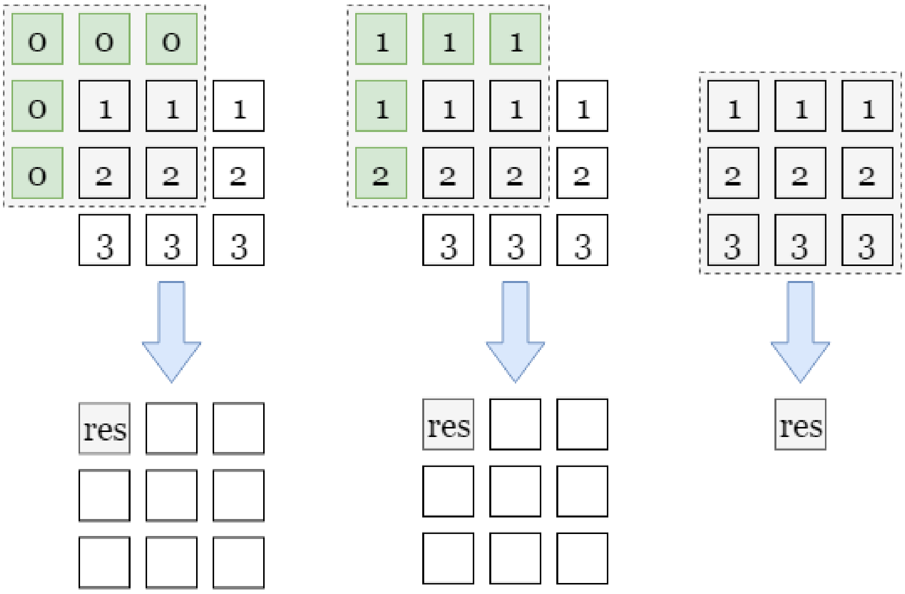


FIGURE 3.9: Various padding methods with a 3 × 3 convolution. **Left:** Zero padding. **Middle:** either cloning or reflection in this case. **Right:** Discarding the convolution at the borders.

In the case of discarding the convolution at the borders, the output image is smaller: $I' \in \mathbb{R}^{(w-f_w+1)\times(h-f_h+1)\times d}$. The other padding methods, on the other hand, maintain the dimensionality. As opposed to cloning or reflecting at the borders, zero padding comes with the added benefit of the borders not affecting the rest of the convolution if larger filters are used. Certain filters are able to highlight specific patterns in the input when applied. These patterns are called features. An example of a feature could be an edge or a corner. Since filters capture features, a very useful naming convention for the output image, is a feature map. The reason for 'map' is two-fold. Firstly, we are 'mapping' a response from a filter onto an input image, creating an output. Secondly, and less intuitive, a map has implicit positional structure. With the use of a filter, features are detected that have no positional dependency, called translation invariance. This property is desirable for classification algorithms, because the position of the object is irrelevant to the classification task. When a filter is applied, the notion of a receptive field is used to express how large of a region of the input affects a specific position in the output. Two filters of different sizes have different receptive fields. A 3×3 filter has 9 values of the original input influencing the value at position (x, y) in the output, while a 5×5 filter has 25. Note that an output, I' can be repeatedly convolved. As an example, two consecutive 3×3 convolutions, may be applied instead of a single 5×5 convolution in order to obtain the same receptive field as shown in figure 3.10. This is crucial for many deep learning approaches because less computations are needed. A convolution with an $n \times n$ filter requires n^2 multiplications and n^2 additions. Thus two 3×3 filters vs. a single 5×5 filter correspond to $2(3^2 + 3^2) = 36$ operations and $5^2 + 5^2 = 50$ operations, respectively.

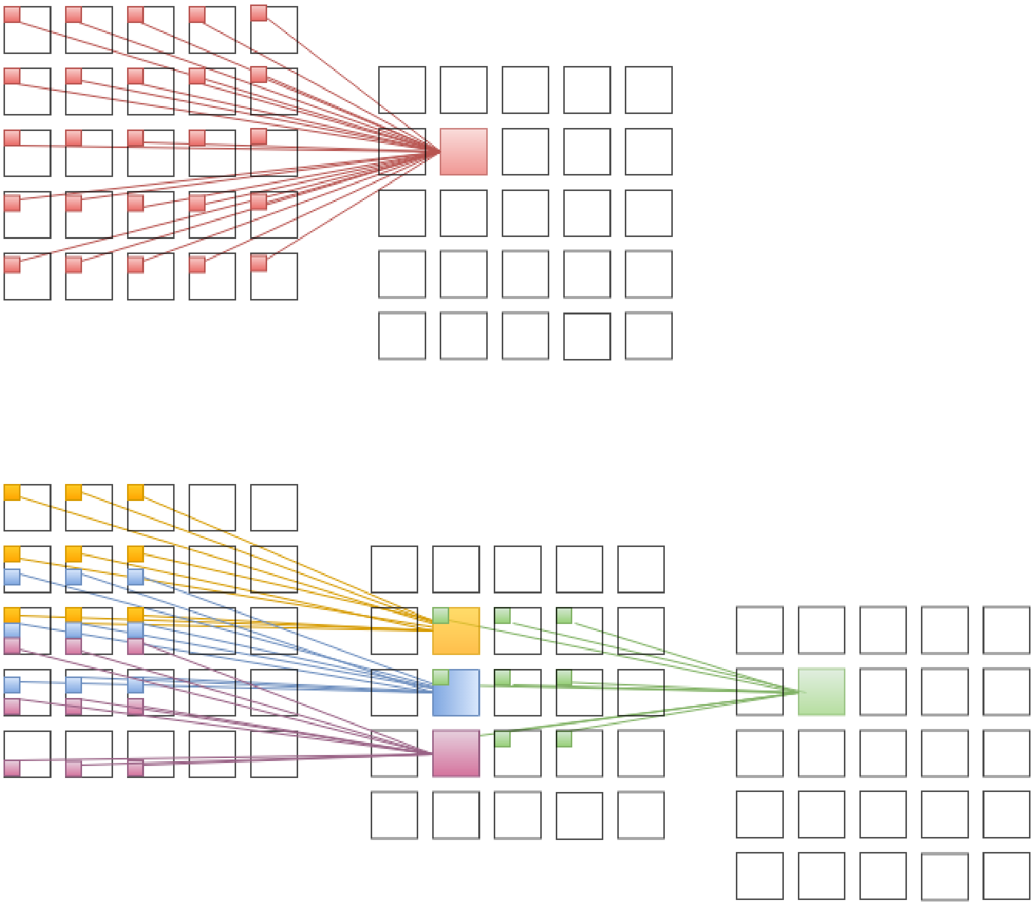


FIGURE 3.10: Two examples of receptive fields. The top convolution applies a 5×5 filter. The bottom uses two 3×3 convolutions. Note how the final receptive field is also 5×5.

3.2.2 Convolutional Layers

A Convolutional Layer is a building block similar to FC layers. With Convolutional Layers the fundamental idea is to exploit the translational invariance of the data. Low-level features as lines, semi circles and corners, can be combined to create more complex features such as houses, planes, traffic signs and so on. While translational invariance is obtainable for simple structures using a single small filter, it is not obtainable with complex structures. By using feature maps as input in following layers, we are able to locate increasingly complex structures in the input. A Convolutional Layer is not limited to a single feature map as input, but does in general receive a set of feature maps, and produces a new set of n feature maps. n is equal to the number of filters in the previous Convolutional Layer. An example of a Convolutional Layer of an RGB image is shown in figure 3.11.

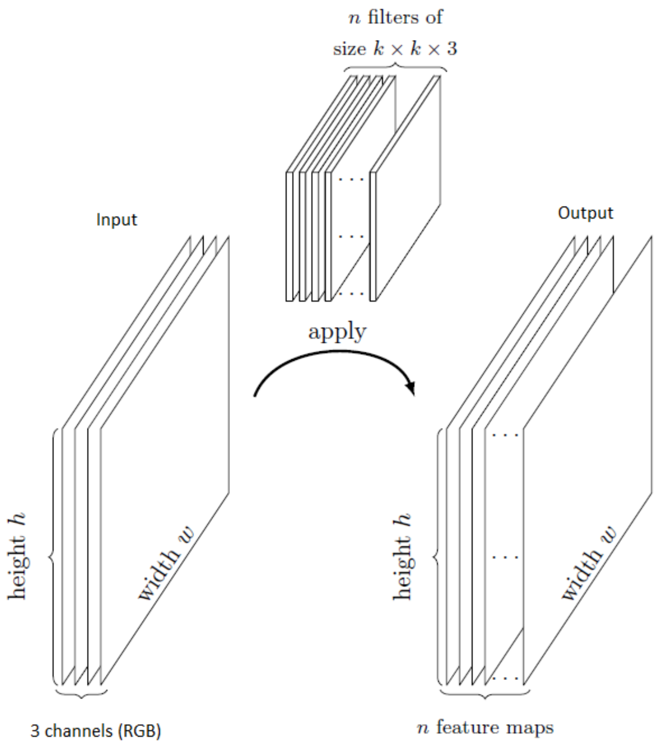


FIGURE 3.11: An RGB image is used as input. It has a width, height and three color channels. The image is convolved with n $k \times k \times 3$ filter. The output is n feature maps of the original width and height assuming borders have been computed. Illustration borrowed from [65].

When applying a filter onto an input, a parameter called stride, s, must be considered. The stride is the step size taken when sliding the kernel during convolution. If $s = 1$ then the convolution operation with a filter maps the input to the output without changing the dimensionality. If $s > 1$, the spatial dimensions of the output are reduced [64]. In figure 3.12, examples with strides of 1 and 2 are shown without padding at the borders. The output size can be calculated as: $\frac{W-F+2P}{S} + 1$, where W is the input size, F is the filter size, P is the padding at the border and S is the stride.

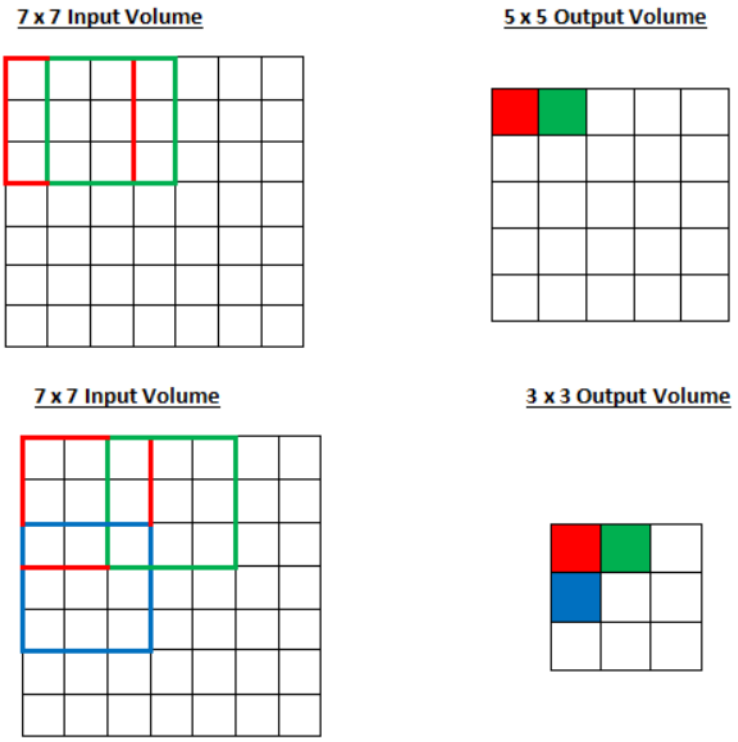


FIGURE 3.12: An example showing how different strides affect the output size. Illustration borrowed from [68].

When a stride of 1 is used, the receptive field overlaps more than with a stride of 2. The increased stride reduces computational cost but also discards information [49]. Usually a stride of 1 is used, however, a stride of 2 is some times applied instead.

3.2.3 Weight Sharing

The concept of weight sharing was introduced in 1989 by Yann Le Cun et al [63]. CNN based architectures have an enormous advantage with respect to the number of parameters needed in order to represent important features in data compared to FF-NNs. In vision based tasks, a feature assumed to be important at one position, is assumed to be important at all positions. Applied in practice this drastically lowers the amount of weights needed. Consider a Convolutional Layer: It has n filters of size $f_w \times f_h \times d$, convolved with $d^{(i-1)}$ feature maps, where i is the current Convolutional Layer. Therefore, the number of output feature maps in the previous layer is the number of input feature maps in the current layer. If each feature map has a size of $w \times h \times d$ there are in total $n \cdot d^{(i-1)} \cdot (w \cdot h)$ parameters excluding the bias term with respect to that layer compared to an FC Layer which has $n \cdot d^{(i-1)} \cdot (w \cdot h)^2$ parameters. A 1D weight sharing example is shown in figure 3.13.

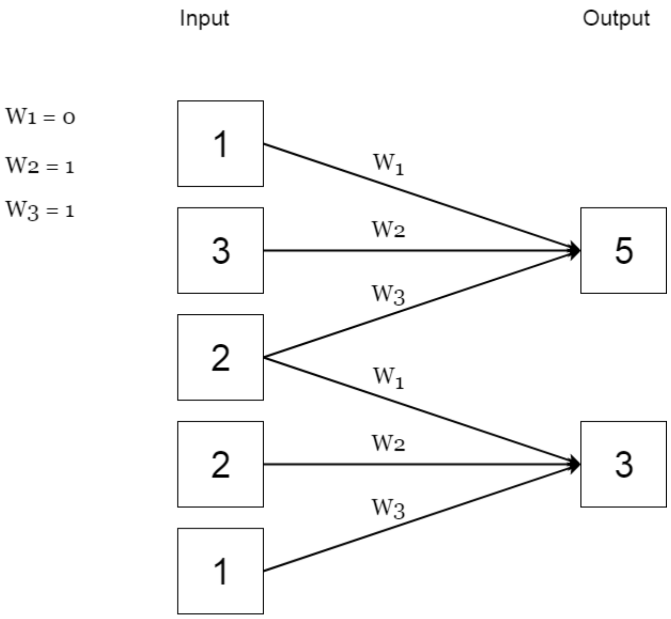


FIGURE 3.13: An example of weight sharing in 1D. A 1×5 input array is convolved with a 1×3 filter using stride $s = 2$. The result is a 1×2 output.

3.2.4 Pooling Layers

A Pooling Layer is a procedure which reduces the number of parameters by downsampling the input data. When reducing dimensionality, one reduces the computational complexity at the cost of losing potentially valuable information. Despite this loss, useful structural information persists with the reduction of data to a different scale space [49]. The trade-off between discarding data, at the cost of applying pooling is highly specific and linked to the task. Pooling is commonly seen in different setups, and has proven to be useful [48]. There exists a variety of different pooling methods where the most common are Max-pooling, Min-pooling, Average-pooling and Stochastic Pooling. Consider Max-poling defined as:

$$s_j = \max_{i \in R_j} a_i \tag{3.23}$$

Here a_i represents the input, R_j represents the region where a_i is located and s_j the resulting output view from applying the operation on all inputs in the region. If we apply a stride $s = 2$ and a pooling filter size of 2×2 we get the result in figure 3.14.

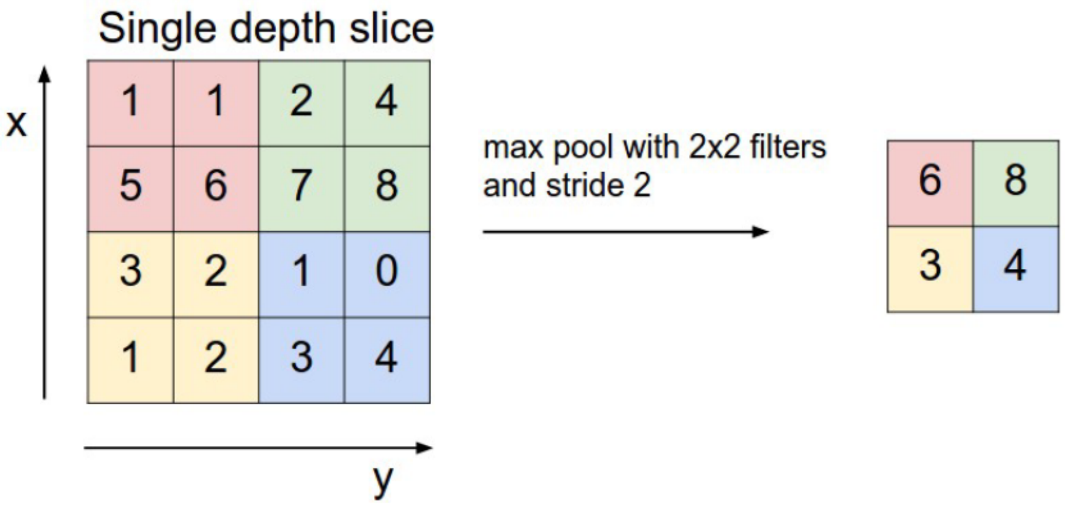


FIGURE 3.14: Example of Max-pooling with a filter size of 2×2, and a stride of $s = 2$. Image borrowed from [58].

Stochastic Pooling is calculated by sampling with probabilities based on the values in the filter. The larger an input is, the higher its probability of being picked is. In figure 3.15, the inputs at the top left and bottom right corner have a probability of 40% and 60% of being sampled for the final output. In this case each region of size 3×3 represents the pooling filter and the calculated probabilities, while the final square represents the selected value.In table 3.1 you can find the different pooling strategies listed.

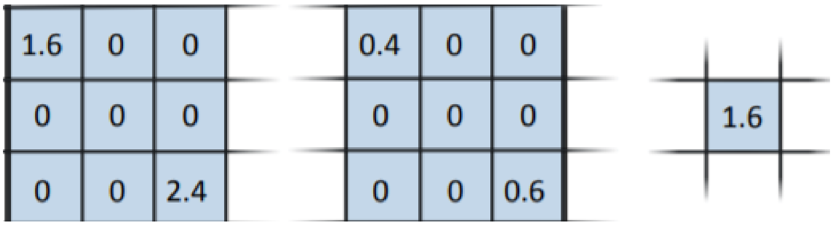


FIGURE 3.15: Example of Stochastic Pooling procedure. Image borrowed from [69].

Method	Definition
Max-pooling	$s_j = \max_{i \in R_j} a_i$
Min-pooling	$s_j = \min_{i \in R_j} a_i$
Average-pooling	$s_j = \frac{1}{\lvert j \rvert} \sum_{i \in R_j} a_i$
Stochastic-pooling*	$s_j = \frac{a_i}{\sum_{i \in R_j} a_i}$

TABLE 3.1: An overview of four common pooling methods and their definitions.

Note that the output of Convolutional Layers and Pooling Layers are multidimensional tensors, which do not match directly with the final output layer as with the case of a FF-NN. Therefore a flattening followed by a FC Layer is often employed prior to the final classification layer.

3.2.5 Backpropagation in Convolutional Neural Networks

The principle of learning for CNN's is identical as with FF-NN. The important difference is what exactly happens, when propagating the error backwards in pooling and convolutional layers. First, consider the Pooling Layer. It takes an input and returns an output without the use of any filters. Hence no learning is present. As a consequence, we only wish to propagate error back to positions that actually had an influence after the pooling operation. If Max-pooling is applied, then we only care about the single position which had the highest value, because it affects the subsequent layers. Therefore, the error signal is only propagated back to that particular position in the previous layer due to backpropagation. The other positions will have an error signal of 0. This happens because the derivative of the activation function is 0 for non maximum outputs of the feature map, and exactly 1 for the maximum neuron [70, 71]. Figure 3.16 illustrates this.

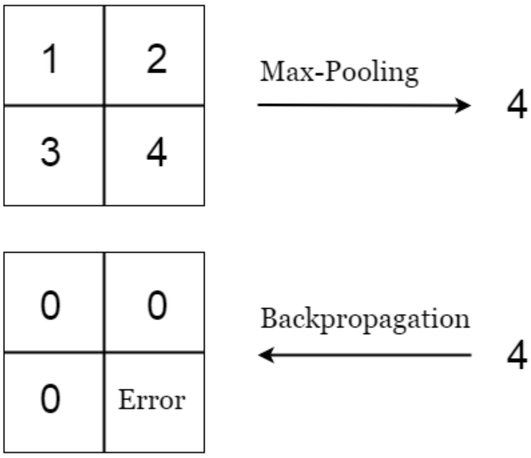


FIGURE 3.16: Example of the error signal being propagated back through the network. The position in the feature map (of size 2×2) determined by Max-Pooling, is the position which influences the output. The other positions should not affect the filters in earlier layers and therefore don't contribute to the error signal.

Now consider the Convolutional Layer. In the forward pass, a single output position (x, y) was determined by a filter convolved with an input. In other words, all values within the receptive field, with (i, j) as the center, had an effect on the single output at (i, j). Furthermore, since weights are shared among all input positions, each individual filter value has an effect on each input where an overlap occurs. The goal is to update the weights in the filter with respect to the loss, L. For a single 2D filter l, we wish to find $\frac{\partial L}{\partial w_{i,j}^{l}}$ where i and j are the indices of the filter's row and column. Recall that the output feature map has a size of $\frac{W-F+2P}{S}+1$. Similar to equation (3.11) for FF-NN, the loss with respect to a weight $w^l_{i,j}$ in a filter l can be written as

$$\frac{\partial L}{\partial w^l_{i,j}} = \sum_{i=0}^{W_{\texttt{Input}}-F} \sum_{j=0}^{H_{\texttt{Input}}-F} \frac{\partial L}{\partial a^l_{i,j}} \frac{\partial a^l_{i,j}}{\partial w^l_{m,n}} \tag{3.24}$$

Here $a^l_{i,j}$ is the product sum and the bias term at a particular position in the l'th feature map. The first fraction is similar to equation (3.12) and is the local error from that position. The second fraction contains the expression, $w^l_{m,n}$, which corresponds to the error at position (i,j) given the weight at position (m,n) in the next layer. It follows that

$$\frac{\partial a^l_{i,j}}{\partial w^l_{m,n}} = o^{l-1}_{i+m,j+n} \tag{3.25}$$

given that $a^l_{i,j} = \sum_0^{m-1} \sum_0^{n-1} w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l$. This is similar to the result of equation (3.13). This can be rewritten as

$$\frac{\partial L}{\partial w^l_{i,j}} = \delta^l_{i,j} \left(\sum_0^{m-1} \sum_0^{n-1} o^{l-1}_{i+m,j+n} \right) \tag{3.26}$$

The error $\delta^l_{i,j}$ is then

$$\delta^l_{i,j} = \frac{\partial L}{\partial a^l_{i,j}} = \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} \delta^{l+1}_{i+m,j+n} \frac{\partial x^{l+1}_{i+m,j+n}}{\partial x^l_{i,j}} = \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} \delta^{l+1}_{i+m,j+n} w^{l+1}_{m,n} f'(x_i, j^l) \tag{3.27}$$

Substituting everything we obtain the result

$$\frac{\partial L}{\partial w^l_{i,j}} = \sum_{i=0}^{W_{\texttt{Input}}-F} \sum_{j=0}^{H_{\texttt{Input}}-F} \left(\sum_{m=0}^{F-1} \sum_{n=0}^{F-1} \delta^{l+1}_{i+m,j+n} w^{l+1}_{m,n} f'(x^l_{i,j}) \right) o^{l-1}_i \tag{3.28}$$

The loss of the weight $w^l_{i,j}$ can be calculated as a convolution of the input feature map and the error signals propagated back to the filter [72–74]. An example of the error being propagated backwards is shown in figure 3.17. Note how the errors are combined directly from the output feature map derivatives which are then convolved with the original input to create a new filter.

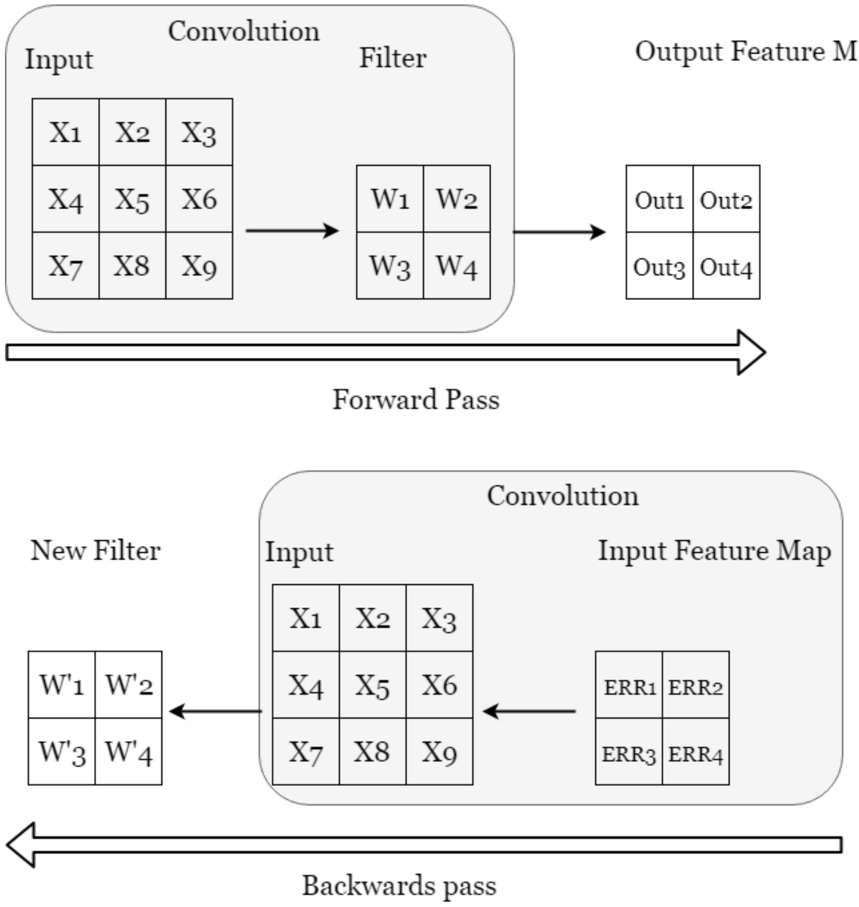


FIGURE 3.17: Example of the backpropagation of error signals in a Convolutional Layer.

For a slightly different and more compact notation of backpropagation in Convolutional Layers using tensor operations see [47, 75].

3.3 Residual Layers

In a Neural Network, the goal of a layer is to learn some function $y = f(x)$. A Residual Network (ResNet) will instead use a Residual Layer and learn the function $y = f(x) + x$, which includes an identity mapping [4]. Here, x is the identity. By including the identity, the layer only needs to find the change from the previous mapping to the next one and add them together. This ensures that the layer doesn't inducing unwanted change.

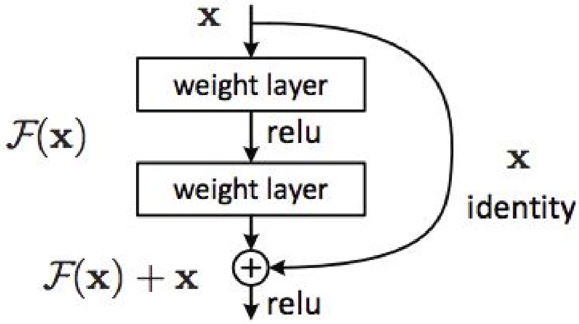

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x})+\mathbf{x}$
relu

FIGURE 3.18: Example of a Residual Layer. The input, x, is mapped with the output of two layers producing a sum $f(x) + x$, which is then fed through an activation function (here ReLU).

A benefit of Residual Connections shown in [76], is how using Residual Layers effectively smooth the landscape of the loss function when training. Smoothing the landscape of the loss function makes it easier to traverse in the optimal direction when locating the minimum (see figure 3.19). Another benefit is that using Residual Layers removes the vanishing gradient problem.

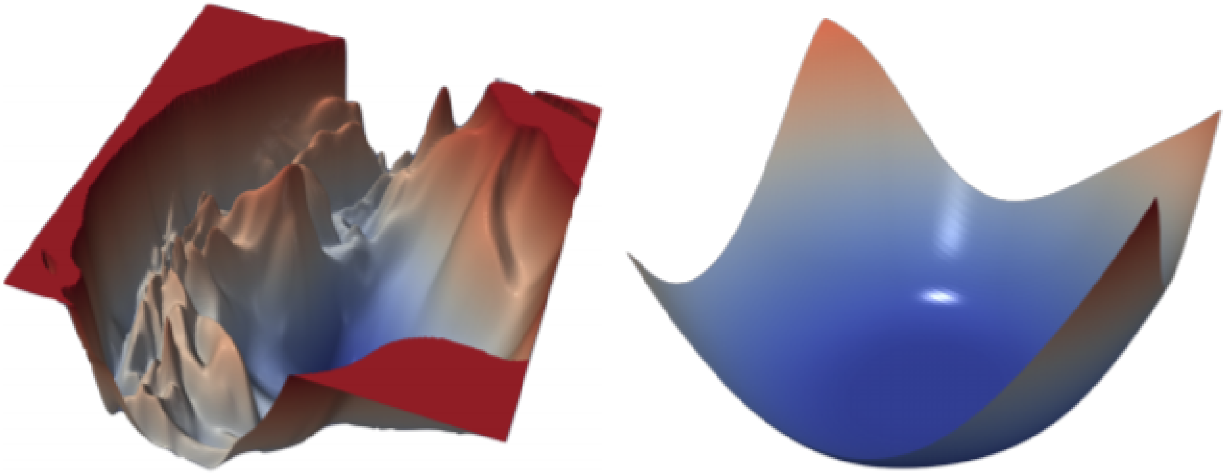


FIGURE 3.19: Examples of loss landscapes with and without the use of Residual Layers. **Left:** Surface of a 110-layer network without residual connections. **Right:** Surface of the DenseNet Neural Network consisting of 120 layers with residual connections. Image is borrowed from [76].

3.3.1 Backpropagation in Residual Layers

Backpropagation in a Residual Layer is slightly different from other layers. The error being propagated is unchanged in the identity mapping. This ensures that the gradient doesn't vanish during training. This might, however, lead to overfitting (explained in section 3.4). To find the loss L, we denote the output of figure 3.18 as $y = F(x) + x$, where x is the input and $F(x)$ is the residual block. Then the loss can be calculated as

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} \tag{3.29}$$

Expanding the second term gives us

$$\frac{\partial y}{\partial x} = 1 + \frac{\partial F}{\partial x} \tag{3.30}$$

We insert the the new value into the original expression and derive the loss

$$\frac{\partial L}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial L}{\partial y}\left(1 + \frac{\partial F}{\partial x}\right) = \frac{\partial L}{\partial y} + \frac{\partial L}{\partial y}\frac{\partial F}{\partial x} \tag{3.31}$$

The obtained result clearly shows why the vanishing gradient problem is diminished since the gradient of the skip connection equals 1. Thus if the gradient vanished from the residual block, we still propagate the loss of the following layer to the previous layer through the skip connection.

3.4 Overfitting

Overfitting is a term used to describe a model that fits training data too closely and therefore doesn't generalize well when classifying unseen data [39].

One reason why the model overfits to the training data is due to the model being too complex for the task.

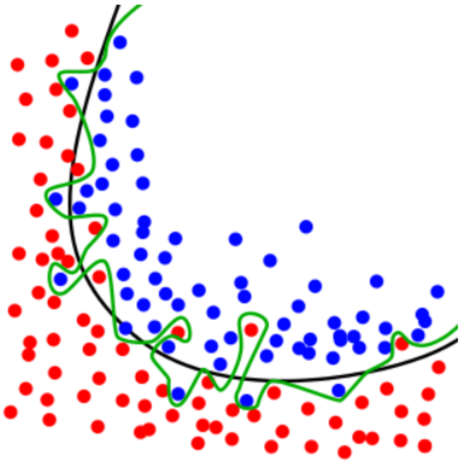


FIGURE 3.20: An example of overfitting. Two models are used to separate the same data. The green line represents a model that overfits. Image borrowed from elitedatascience.

In figure 3.20, the green decision boundary of a complex model able to perfectly classify the data. However, the black decision boundary of a simpler model, presumably represents the true distribution more closely. In contrast to a model being too complex, a model too simple to describe the underlying distribution of the training data is underfitting. A number of techniques designed to reduce overfitting in models and increase generalization exist. These are referred to as regularization techniques [39]. A simple regularization technique against overfitting is called Early Stopping. During training the error of the validation set will saturate and eventually start to increase if the model overfits. At the exact point where the validation error stops decreasing, the training is terminated despite the training error not being saturated.

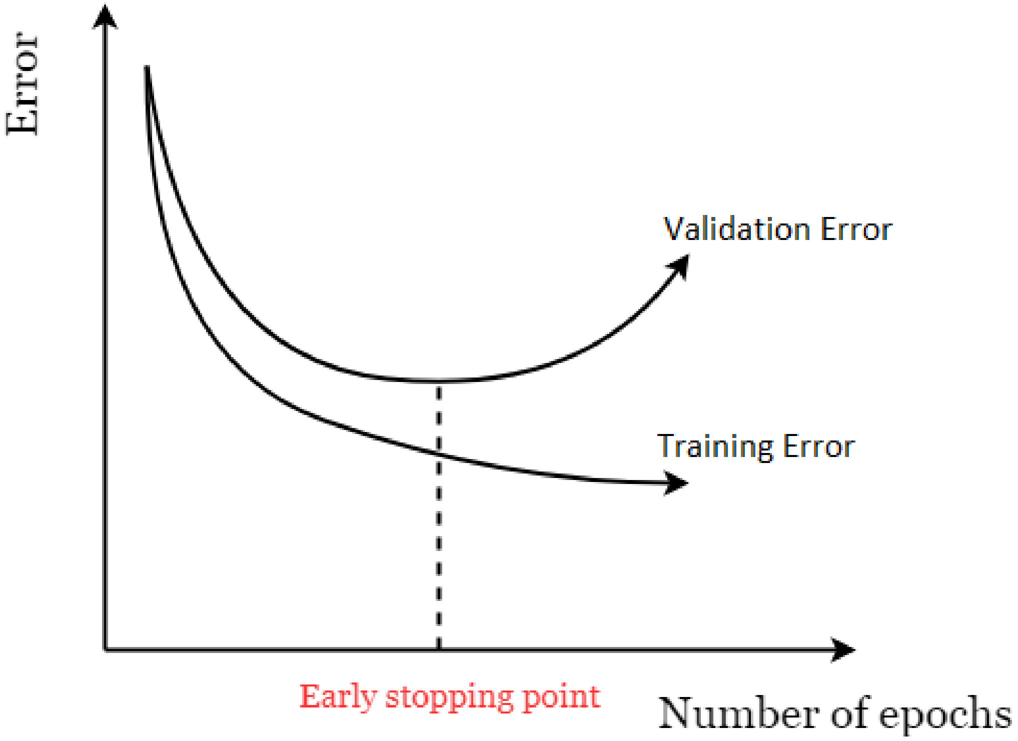


FIGURE 3.21: An example of Early Stopping. At the exact point where the validation error starts to increase, training should terminate in order to avoid overfitting.

Dropout is another regularization technique. It changes a networks activations slightly on each forward pass by discarding varying nodes in the network with a specified probability (see figure 3.22). A typically used probability is 0.5. Dropout is beneficial if particular nodes contribute severely, thereby reducing the influence of others. When applying dropout, the remaining units in the network will be scaled appropriately to take into account the units that were dropped. After the forward pass, the previously dropped units are then inserted back into the network with the previous weights. A variation of dropout, called Adaptive Dropout, keeps or drops units with varying probabilities [77].

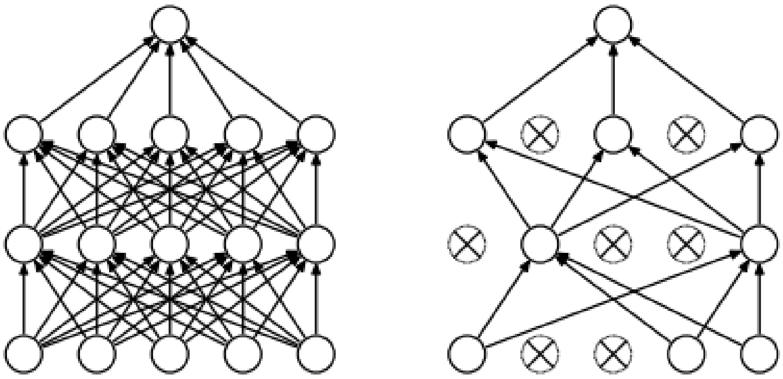


FIGURE 3.22: An example of dropout. **Left:** Fully connected network without dropout. **Right:** Parts of the nodes have been dropped during training.

While dropout is a strong regularization technique, it also involves removing possibly useful information. Batch Normalization is another technique that can be used to increase regularization of a model. It is a way of reducing internal covariance shift, by normalizing and possibly introduce shifting and scaling of the output of the previous layer. For each x^k in a layer with a d-dimensional input $X = (x^1 \dots x^d)$, the following transformation is applied.

$$\hat{x}^k = \frac{x^k - \mathbb{E}[X]}{\sqrt{\texttt{Var}[X] + \epsilon}} \tag{3.32}$$

After normalizing each x^k, a scale and shift is performed based on parameters γ and β.

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{3.33}$$

Typically, γ and β are initialized to 1 and 0, respectively. Furthermore, Batch Normalization is resilient to bad weight initialization, and therefore capable of working with higher learning rates. It has been shown that Batch Normalization can decrease the number of epochs needed for minimizing the loss [78]. Another problem with NNs is a lack of training data for complex problems. If there isn't enough data in the dataset to accurately represent the full variation in the possible data space, then the model won't generalize well to unseen data. In an attempt to solve this problem, Data augmentation can be used [4, 23, 79, 80]. It is the process of altering the input data using various techniques to construct additional data. Multiple techniques can be combined to further increase the size and variation in the training data. Common data augmentation techniques include cropping, scaling, reflection, rotation and channel jittering. When cropping, a subset of the initial input is chosen whereas scaling is the process of zooming the image inward or outward. Typically, cropping and scaling are combined when the dimensions of the input to the model are fixed. These technique increases the variance in the the dataset by representing it at multiple scales. Reflection refers to flipping the input in either direction. Consider a model trained using only right handed tennis players. Applying horizontal reflection would increase its ability to also classify left handed players, effectively doubling the dataset. Rotations refer to rotating the input on the z-axis. It is used to increase viewpoint invariance, referring to scenarios where the orientation of the object is irrelevant to the classification. Usually performed on images, channel jittering changes the values of one or more color channels. High illumination of objects can be removed to some extent by applying this technique [80]. Examples of a few of the techniques mentioned are shown in figure 3.23.

3.5 Estimating Motion

Motion captures the apparent change of an object over time. When performing video action recognition, we wish to accurately capture motion between frames. The most basic estimate of motion involves calculating the difference between two frames. More complex motion estimation techniques exist, such as pixel recursive algorithms, block matching algorithms and Optical Flow estimation algorithms. In recent literature, only Optical Flow based algorithms are used. They provide a good approximation of the true motion, while requiring less resources than block matching algorithms and less computation time than pixel recursive methods. Another useful property of Optical Flow is that it can be pre-computed prior to any training phase, which would be computationally expensive if computed during training. Optical Flow is a displacement vector field where every vector describes the displacement of pixel intensities between frames at time t and $t + \delta t$. It is assumed that the pixel intensities of an object are the same regardless of position when in consecutive frames and also that the motion of pixels close to each other is similar. This is called the Brightness Consistency Constraint and the Spatial Constraint. Optical Flow algorithms are divided into two groups: Sparse Optical Flow and Dense Optical Flow. Sparse Optical Flow only processes parts of the image whereas Dense Optical Flow processes all pixels. This is computationally more expensive, but also more accurate.

We now describe a Dense Optical Flow algorithm by Gunnar Farneback. The key point of the algorithm is to approximate a neighbourhood of pixels using quadratic polynomials as opposed to using derivatives as seen in many approaches [81]. The polynomials are approximated using the polynomial expansion transform. Using the coefficients of this expansion, it is possible to estimate the displacement vector field. The estimation of the entire displacement field can be iteratively updated by configuring the initial estimation. In other words, if one already has an idea of the initial displacement field, the coefficients estimate can make use of this prior information to improve the estimation. Finally, the algorithm is capable of operating at multiple scale spaces. Obtaining a coarse displacement field, at a lower spatial resolution can be used to create a better estimation

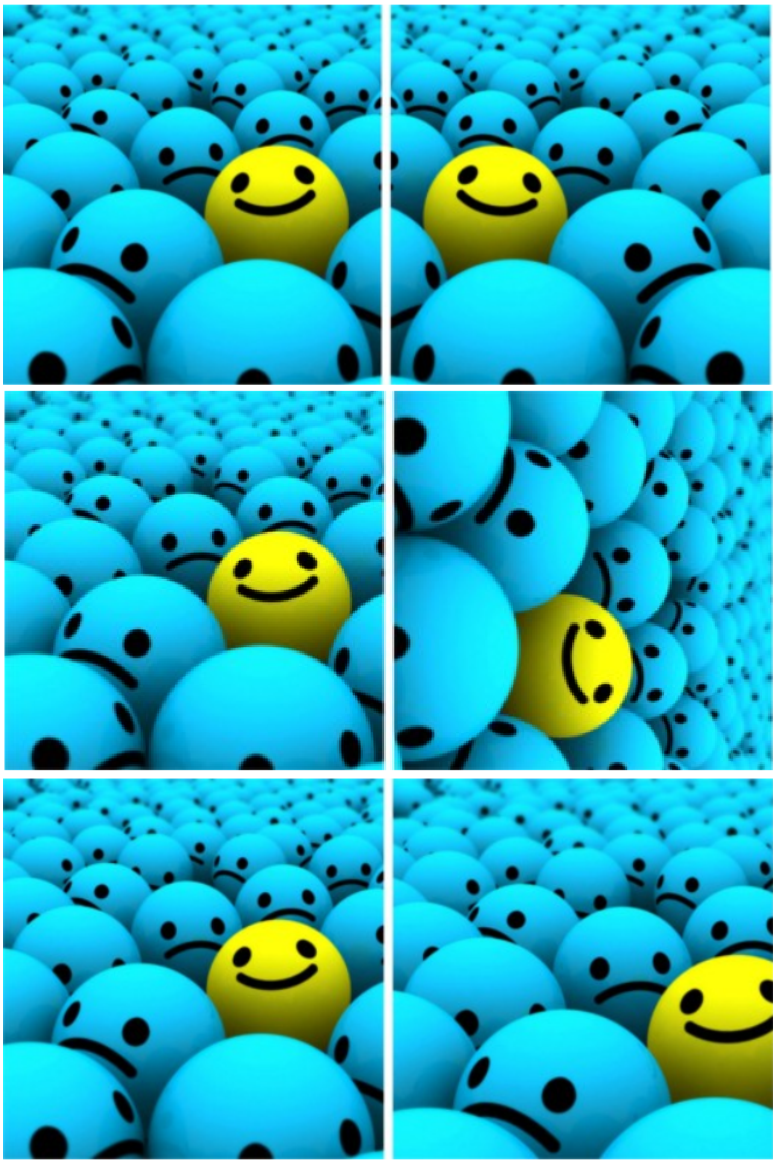


FIGURE 3.23: Examples of data augmentation. **Top:** Horizontal reflection. **Middle:** 90% clockwise rotation. **Bottom:** Random cropping of center rescaled to original size. Images are borrowed from this blog post.

at finer scales. One problem with this is the additional cost of re-computing the estimation of the polynomial coefficients at each scale.

Given an exact quadratic polynomial

$$f_1(x) = x^T A_1 x + b_1^T x + c_1 \tag{3.34}$$

Where A_1 is a symmetric matrix, b_1 is a vector and c_1 is a scalar and a global displacement, d, such that

$$\begin{aligned} f_2(x) &= f_1(x-d) \\ &= (x-d)^T A_1 (x-d) + b_1^T (x-d) + c_1 \\ &= x^T A_2 x + b_2^T x + c2 \end{aligned} \tag{3.35}$$

The solution for the displacement d is given by

$$d = -\frac{1}{2} A_1^{-1} (b_2 - b_1) \tag{3.36}$$

For two images, the polynomial expansion gives the coefficients $A_1(x)$, $b_1(x)$, $c_1(x)$ and $A_2(x)$, $b_2(x)$, $c_2(x)$. Ideally, $A_1 = A_2$ but in practice the following approximation holds:

$$A(x) = \frac{A_1(x) + A_2(x)}{2} \tag{3.37}$$

Furthermore, a constraint is introduced such that:

$$\begin{aligned} \hat{b}(x) &= -\frac{1}{2}(b_2(x) - b_1(x)) \\ &= A(x)d(x) \end{aligned} \tag{3.38}$$

$d(x)$ is now a spatially varying displacement instead of a global displacement. The goal of the algorithm is to find $d(x)$ given a pixel neighborhood I of x. This can be solved as a minimization problem, defined as

$$\begin{aligned} &\sum_{\hat{x} \in I} w(\hat{x}) ||A(x+\hat{x})d(x) - \hat{b}(x+\hat{x})||^2 \\ &\Leftrightarrow \\ &d(x) = \left(\sum w A^T A\right)^{-1} \sum w A^T \hat{b} \end{aligned} \tag{3.39}$$

Consider the following rope-skip example in figure 3.24. Here, two consecutive frames are chosen with the person mid air. The images show the vertical and horizontal displacement field between the two frames. Note how no particular motion in the horizontal direction is apparent, but the displacement from the person jumping makes the vertical displacement apparent.

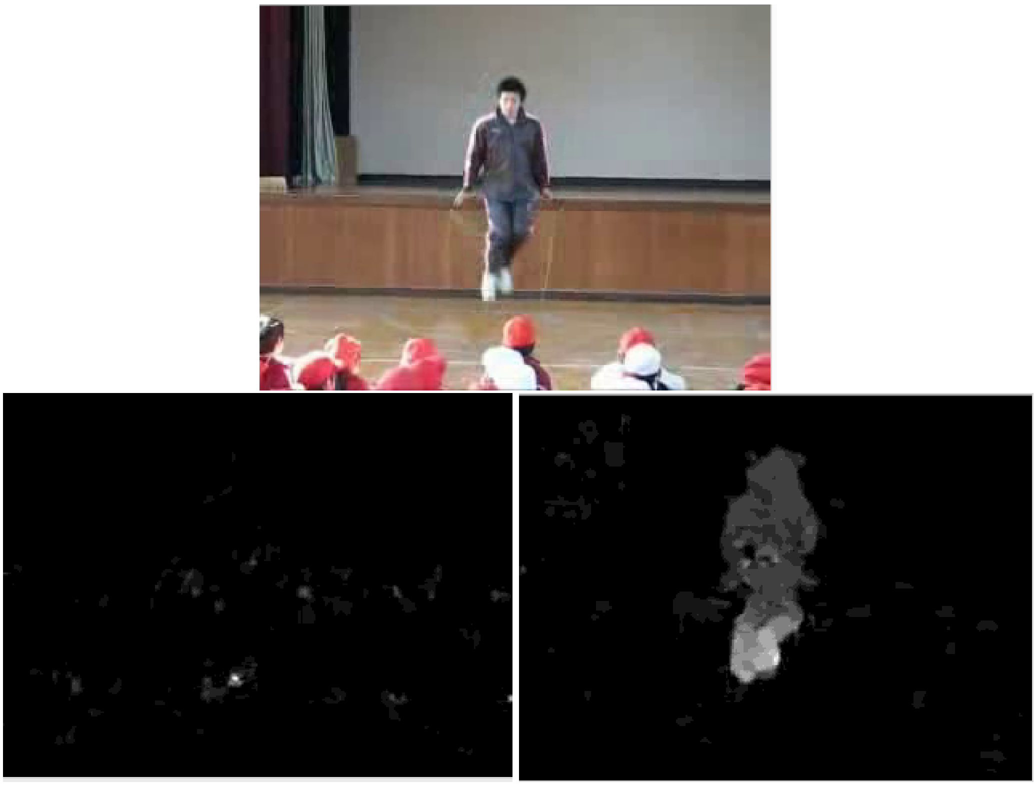


FIGURE 3.24: An example of the displacement field of two consecutive frames in a rope-skip video from UCF101. The subject is captured while in mid-air, making the vertical jump apparent by the estimation of the displacement field.

Chapter 4

Datasets

An important part of solving any form of task in machine learning usually requires a collection of data to train or test a solution. In the field of video action recognition the natural collection of data to pick is a collection of videos. However, the process of collecting the appropriate data is not a trivial task. A series of datasets within the field of action recognition exist today. They capture a variety of human movement events, ranging from Hollywood movies, to various sports. Each dataset was created with a particular goal in mind. As the field grew and solutions improved, the need for more demanding datasets became apparent. In the following two sections we present a summary of well known datasets followed by a description of our own dataset.

One of the first action recognition datasets was the KTH dataset introduced in 2004 [82]. A group from the Royal Institute of Technology in Stockholm created a set of videos consisting of 6 different actions: walking, jogging, running, boxing, hand waving and clapping. Each action was performed by 25 individuals, at 4 different viewpoints, giving a total of 600 video clips. Each video contains the same action 4 times with few exceptions. It is possible to split these into 2391 single actions. Two important features of this dataset are the use of a static camera position, and a static/homogeneous background. This assured that only the intended action is present in the video. As the dataset was used in various works by Ivan Laptev [5, 83, 84] the limitations with a homogeneous background and static camera in regards to recognizing actions in a more realistic setting became apparent. Similar to the KTH dataset, is the Weizmann dataset [85] published the following year. It contains 10 action categories, but is otherwise similar to KTH.

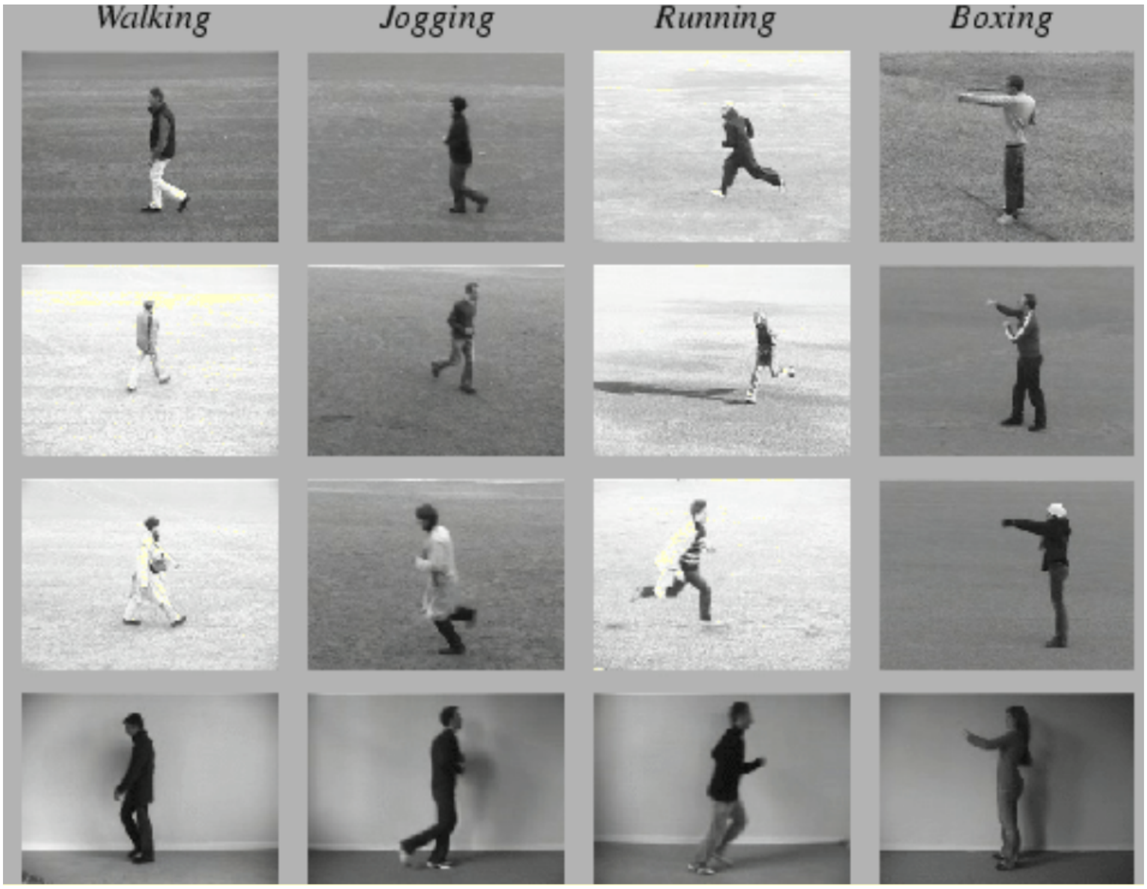


FIGURE 4.1: Still Frames From the KTH Dataset.

FIGURE 4.2: Still Frames From the Weizmann Dataset.

In order to overcome the problem of homogeneous backgrounds, the research group at Carnegie Mellon University (CMU) created a dataset known as the CMU Motion Capture Database [86]. With the intention of placing background motion into videos, this dataset is the first to address this property explicitly. The dataset

consists of 110 videos in total. The limitation for this dataset is the sparse amount of data available.
In order to solve this problem new datasets were constructed. Datasets Hollywood 1 and Hollywood 2 [87] consist of Hollywood movie scenes with human actions. Even though the collection was larger with a total of 3669 clips, every clip was presented in high quality, with optimal scene lighting, making it less useful for realistic video action recognition.

Other datasets include the 1 Million Dataset [88], a web-scrape of YouTube videos annotated by the 'YouTube Topics API', and the MSR-II dataset [89], a series of statically filmed events. Limitations of The 1 million dataset include its automated annotations or the videos no longer being unavailable making it hard to use to compare algorithms. For The MSR-II dataset, a limitation is its use of kinetic depth sensory information instead of videos.
In 2011 a group from Serre Lab at Bron University [90] created a new dataset called HMDB51. With 51 classes and over 100 instances of each action in a mostly realistic environment, it became popular to use as a benchmark. Containing about 6500 videos it was the largest dataset currently made, both in size and complexity.

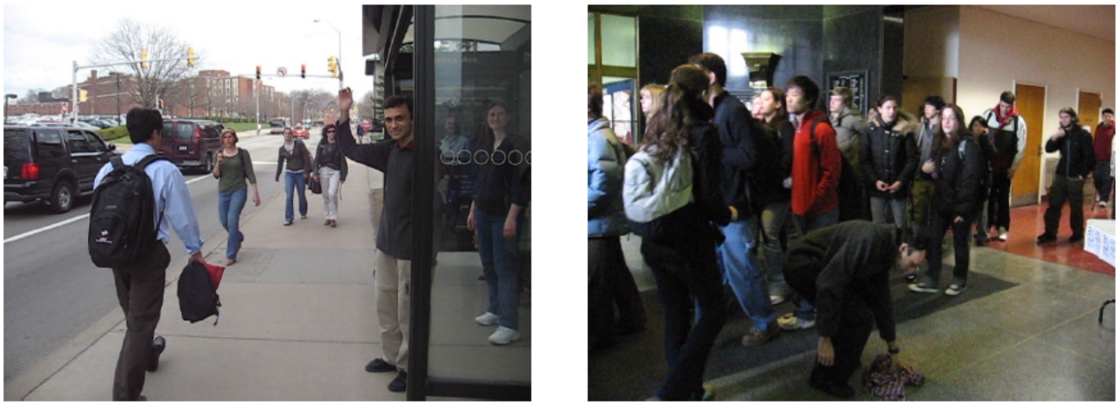
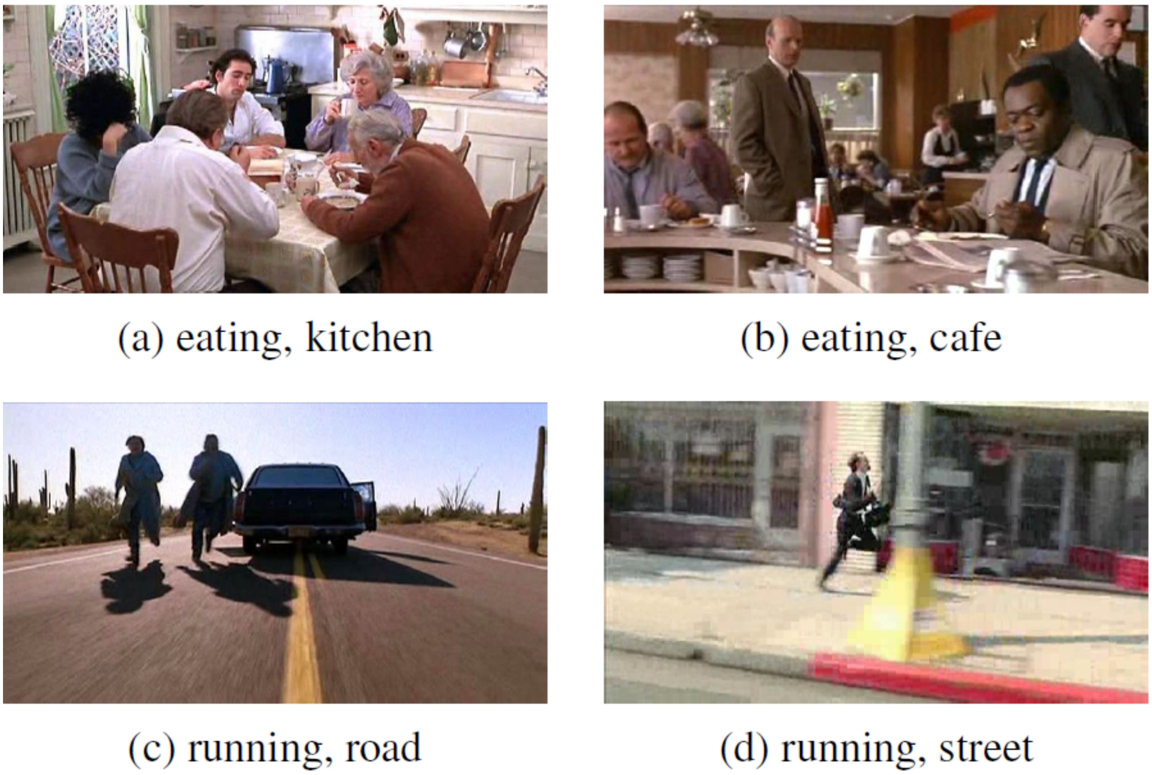


FIGURE 4.3: Still Frames From the CMU Dataset.



(a) eating, kitchen

(b) eating, cafe

(c) running, road

(d) running, street

FIGURE 4.4: Still Frames From the Hollywood-2 Dataset.

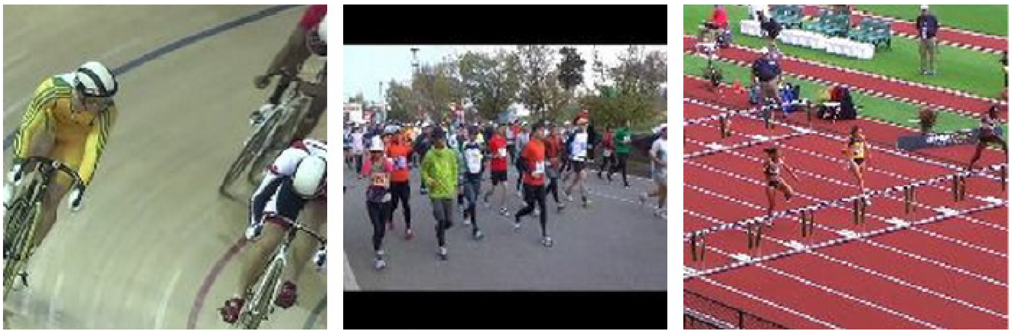


FIGURE 4.5: Still Frames From the 1 Million Dataset.

FIGURE 4.6: Still Frames From the MSR-II Dataset.

A similar dataset of growing interest starting back in 2009 released a third edition in 2012. The UCF101 [91], was introduced (as an extension to is predecessors UCF11 and UCF50). It currently contains 13320 videos, from 101 different action categories. It is a very diverse dataset, with many variations such as viewpoints, camera motion, scale, cluttered background and illumination. It is regarded as a very useful dataset, as the action occurs in natural settings. As with HMDB51, UCF101 is also used as a testing benchmark.

Lastly, a new dataset has emerged, called ActivityNet [92]. It was introduced in 2015 and contains a large collection of annotated YouTube videos for both action detection and classification. One major drawback is that videos must be collected manually, which might not be possible. It has the same properties as the UCF101 and HMDB51, but is not used as frequently.

FIGURE 4.7: Still Frames From the HMBD51 Dataset.

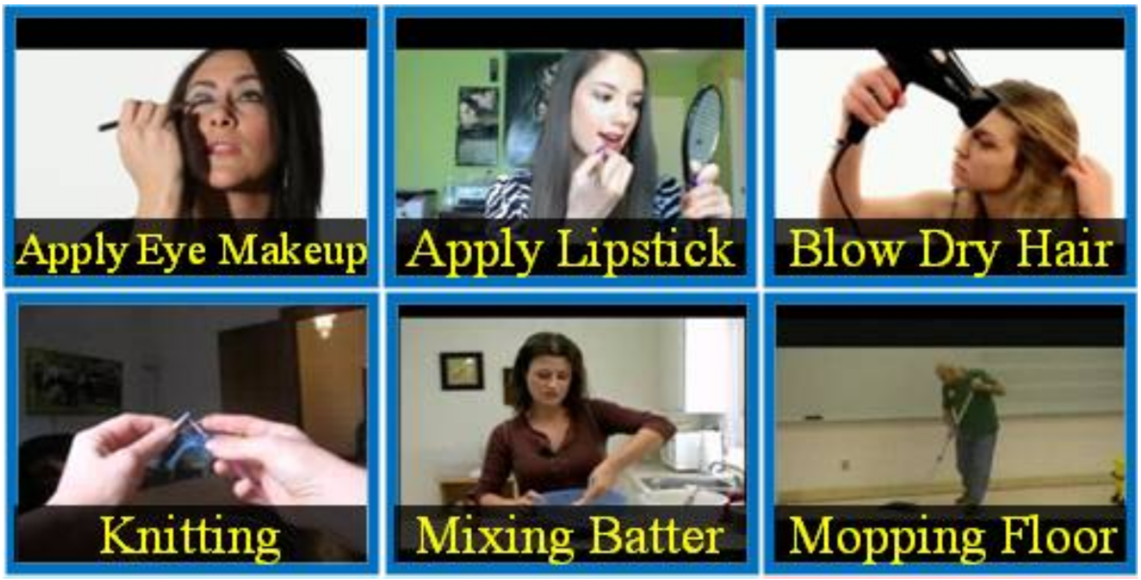


FIGURE 4.8: Still Frames From the UCF101 Dataset.

All datasets mentioned thus far, have consisted of actions with large inter class variance. In 2014, the MPII-Cooking-2 dataset was introduced [93]. It contains a series of actions which required a higher level of fine-grained action localization, because the actions vary very little. The dataset contains only 3 action categories and 54 clips, but have multiple instances of actions in each clip. The dataset was primarily created for localization of actions.

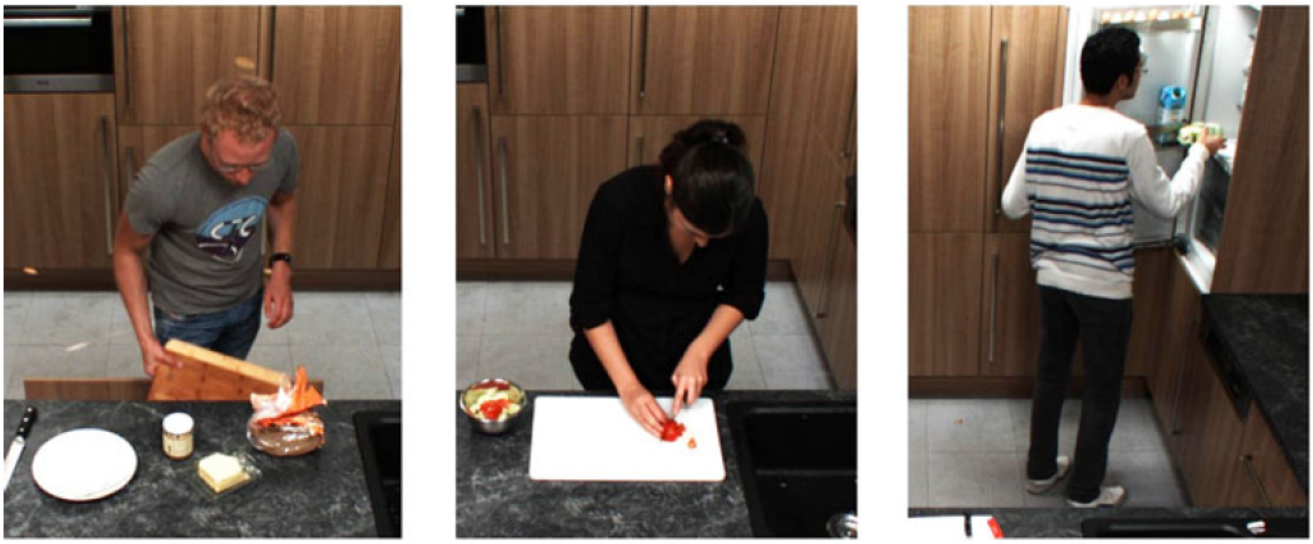


FIGURE 4.9: Still Frames From the MPII-Cooking-2 Dataset.

Chapter 5

Basketball Simple Shooting Statistics Dataset

This chapter describes our dataset, Basketball Simple Shooting Statistics Dataset, in short B3SD. We draw inspiration from the procedure in [94], in which the authors describe the process of creating a dataset for image recognition purposes.

A problem of existing datasets is the lack of temporal and fine-grained information in natural settings. A video may contain several instances of a single action, or it may contain several different actions. To our knowledge, the widely used benchmark datasets only contain a single top level label for the entire video clip. The lack of detail restricts its possible use. The problem of recognizing an action when using this type of dataset is limited to mapping a large amount of information to a single label, which would be unsatisfactory in many scenarios. As an example, consider a video with two distinct actions. One action occurs two times while one action occurs only once. Assume that the overall video classification is the action which occurs the most. However, deep learning algorithms may consider the action which occurs only once, to be dominant, which results in an incorrect classification. If one does not consider the possibility of localization and possibly multiple actions (distinct or not) the problem is simplified to mapping a lot of information to a single label. In recent literature we find that available datasets are mostly used for action classification and not action localization due to this restriction. As we wish to overcome this limitation, it should be possible to use B3SD for both action recognition and localization. Reporting multiple action occurrences in a single video provide users with the freedom to test both continuous and non-continuous action recognition algorithms in a natural setting. In the case of different actions in the same clip, we provide information such as shot type and bounding boxes, which can be useful in fine-grained action recognition. The use of time-stamps for each individual action, can be used to cut longer videos into short clips with a single action.

B3SD is a realistic, fine-grained dataset containing labels and bounding boxes for 4 distinct actions from a variety of basketball game footage collected from YouTube. It consists of 4-second clips, each labeled with a single action. The different actions are: layup, free throw, 2-Point shot and 3-Point shot, with all actions occurring at the frame at exactly 1 second in the clip. We chose this in order to ensure consistency between all generated clips. B3SD also consists of an auto-generated 5th class without shots. Figure 5.1 shows an example from each class.

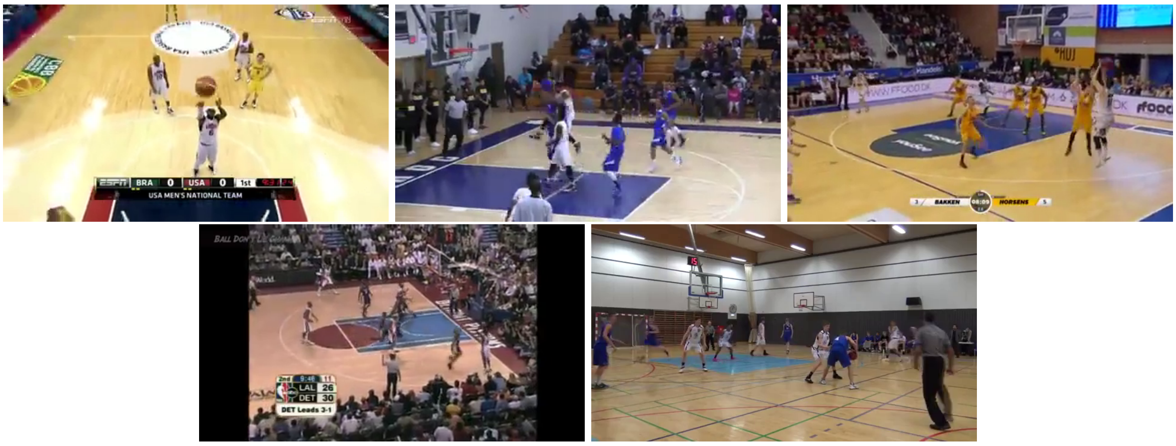


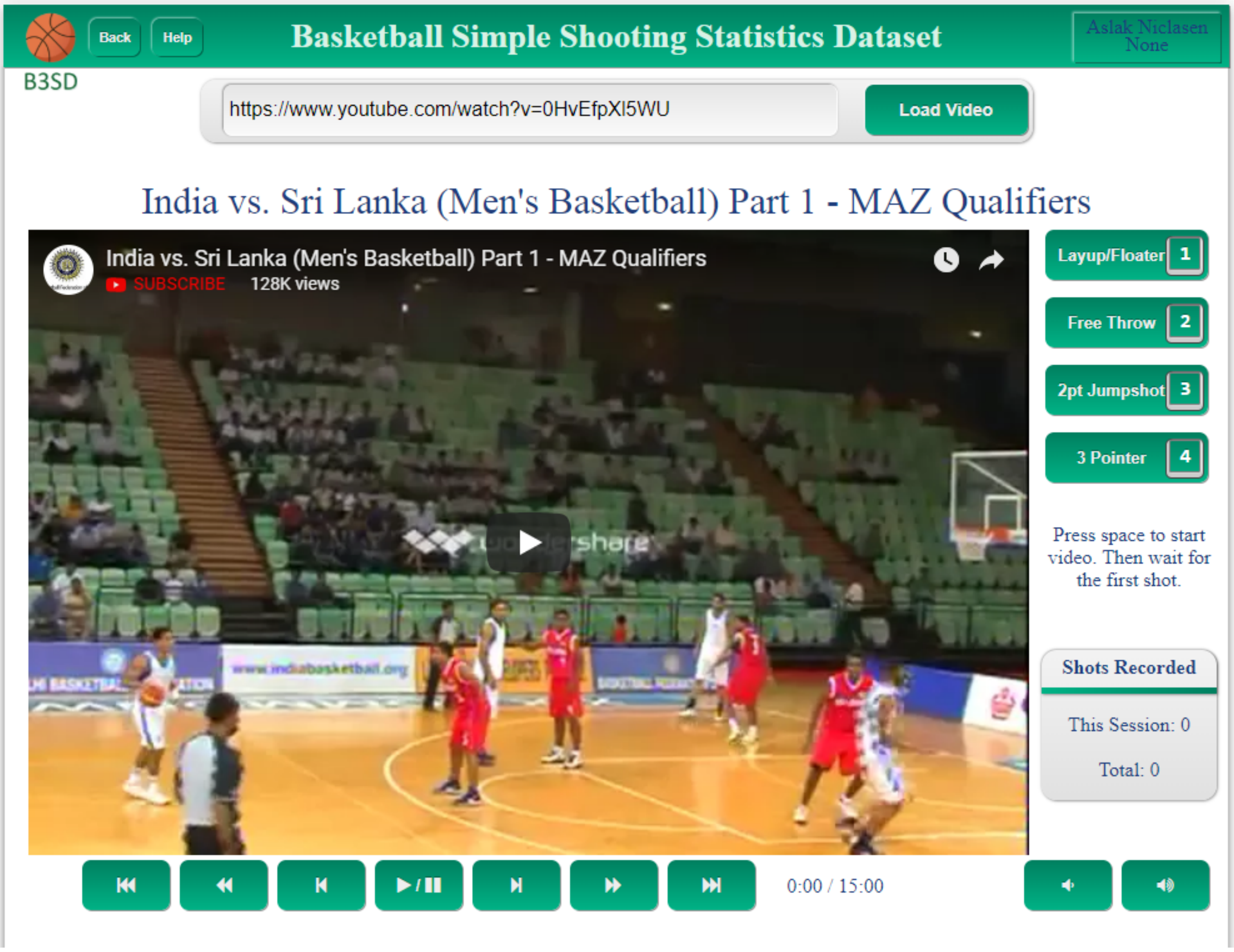
FIGURE 5.1: Examples of still frames from each class in B3SD, taken from different basketball games. From left to right: Free throw, layup, 2-point shot, 3-point shot and no shot.

As can be seen in the figure, a video may contain elite level game footage (NBA) or amateur level game footage (high school or private footage). The reason for this is to capture a large variety of viewpoints, camera motion, background motion and lighting. This choice provides us with a larger possible set of videos, because we are not restricted to a certain type of video quality or level of play e.g. professional vs. amateur. The quality of all videos is either 480p or 360p.

5.1 Creating B3SD

All stats available in B3SD, have been collected with the help of volunteers manually annotating shots. We provided a web interface for annotators to freely pick any full length basketball game in hopes of capturing an increased variety of game footage. We also believe this freedom contributed to the quality of the dataset, because annotator freely selected videos. The incentive of volunteering is different from [94] since the annotators are paid through Amazon Mechanical Turk. Here, the authors filtered away annotators rushing through the process to earn more money instead of providing quality annotations.

Our dataset was constructed by 29 annotators from 3 different countries and 6 basketball clubs. Denmark, Great Britain and Norway. Ages range from 12 to 56, with the average annotator being 24 years old. For the data to be consistent across the annotators and minimize error across individuals, we provided a guided walk-through of the recording program and its various settings. This was followed up by a training procedure of each annotator, although most of them were familiar with the rules of basketball beforehand. The training procedure is a single, annotated 24 second video. The training is used to guide the annotators view of what we consider a proper ”time of shot” and a ”good” bounding box. A bounding box is created by dragging the cursor on top of the action, creating a rectangle. After releasing the cursor it can be moved or reshaped if necessary. We only use 1 training video since the annotators aren't paid and we cannot expect them to keep training excessively before getting started. By letting them begin as soon as possible, we believe that we are able to collect more data. Furthermore, we did not require perfect bounding boxes, because it is more time consuming and we wanted to maximize the dataset size. Initially, only a small group of volunteers was used. This way we were able to receive feedback of the process including the training procedure, and locate any errors. After the initial testing, the web interface was presented in various Facebook groups, to reach a larger audience. A couple of examples of the web interface for creating a new annotation, and validating other annotators' work can be seen in figures 5.2, 5.3 and 5.4.



https://www.youtube.com/watch?v=0HvEfpXI5WU

India vs. Sri Lanka (Men's Basketball) Part 1 - MAZ Qualifiers

Press space to start video. Then wait for the first shot.

This Session: 0

Total: 0

0:00 / 15:00

FIGURE 5.2: An example from the web interface. Buttons for various actions, such as fine-tuning the exact time of shot, and registering the type, can be seen in green.

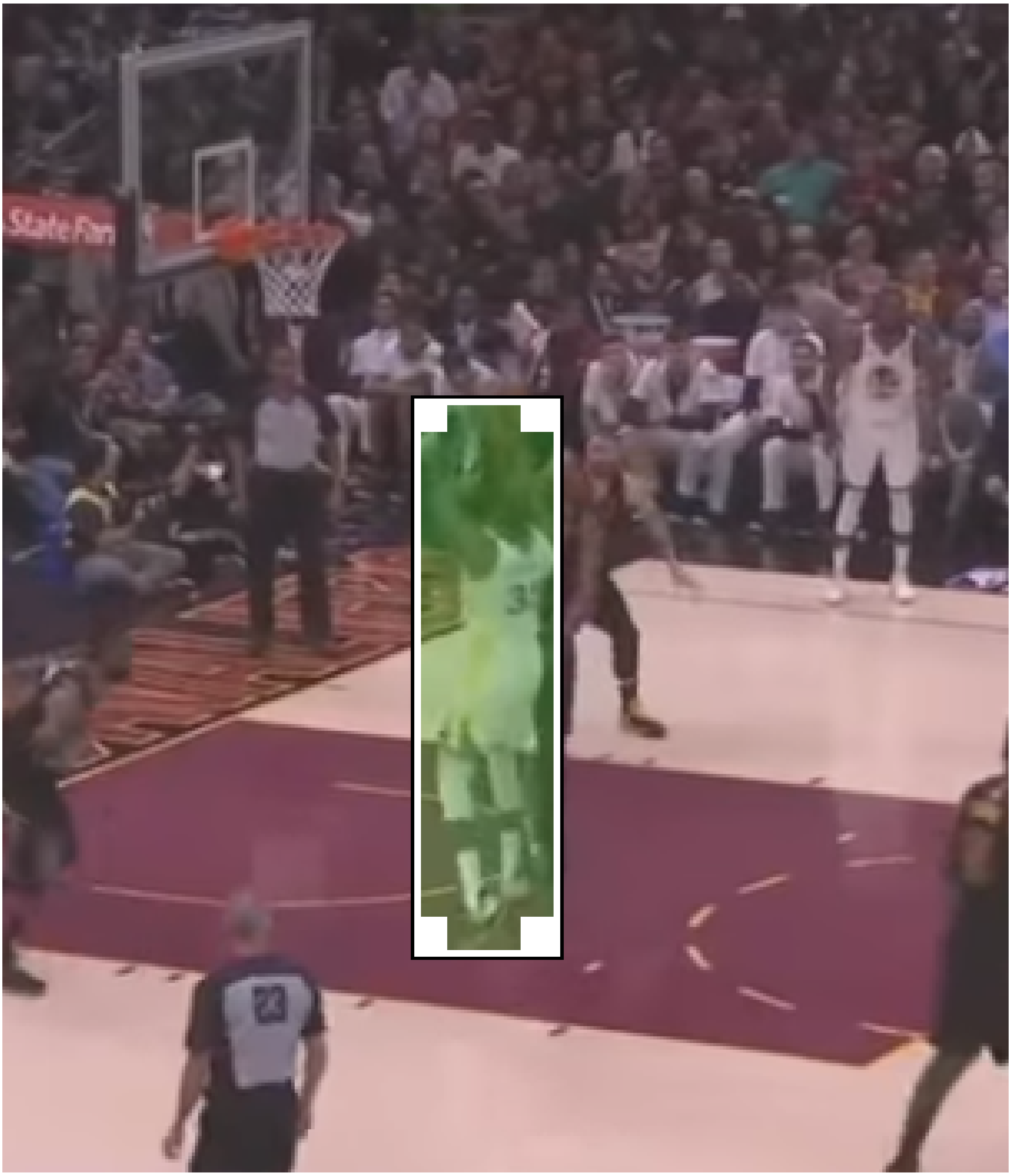


FIGURE 5.3: An example of a bounding box on top of an action.

Each individual annotation is saved under a unique stat id. For each annotation we saved the information shown in table 5.1 below.

Annotation Specifications

Parameter	Example
Stat Id	42
Youtube Video Id	4ksJPxQu-A0
Bounding Box (x1,y1,x2,y2)	123, 120, 200, 218
Time of Shot	86.5
Make	2
Miss	1
Shot Type 1	0
Shot Type 2	3
Shot Type 3	0
Shot Type 4	0
No Shot	0
Garbage	0
Bad Bounding Box	1

TABLE 5.1: Parameters saved for each annotation. The right column shows example values.

To increase data quality, annotators also have the opportunity to verify other annotations. Since the annotators are unpaid volunteers, however, we believe that the stats obtained are of reasonable quality. The web interface for the validation procedure is shown below in figure 5.4.

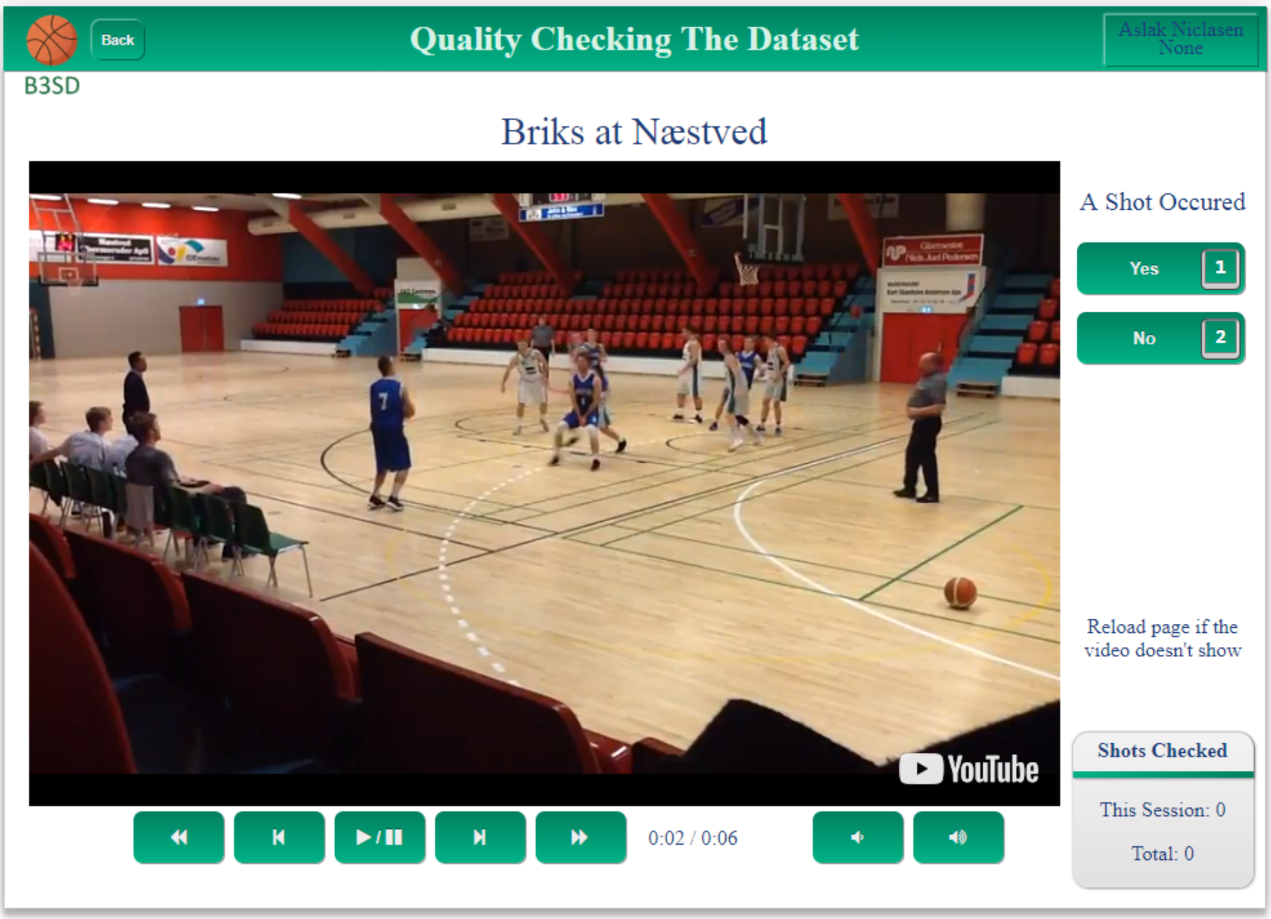


FIGURE 5.4: The web interface for the validation procedure.

In our verification procedure, annotators are presented with a short clip including a single action, selected among the videos with the lowest verification count. The clip is then annotated again, except for bounding box construction. Whether to include the shot as scored or not and the type of shot is determined by a majority vote. As an example based on table 5.1, three annotators have marked a single clip to include a shot of type 2, while disagreeing on whether the shot was made. Since the annotators only mark shots, we create clips containing no shot. Such a stat is simply a video which may contain ball passes between players, timeouts and similar. To create these stats, we assume that annotators have done a good job. The assumption relies on all shots in a full length video to have been annotated. Using the registered time of shots in a video, we compare two consecutive shots, S_1 and S_2, finding the time interval between them. A new clip can be created if the time between two shots is greater than 7 seconds, thereby ensuring that no overlap occurs, even if the time of shot marked is slightly off. The procedure is shown in algorithm 1.

```
Data: $X = [S_1, S_2 \cdots S_n]$

while $length(X) > 1$ do
    if $(S_2 - 3.5) - (S_1 + 3.5) > 7$ then
        create_new_clip();
        $X.remove(S_1)$;
    else
        $X.remove(S_1)$;
    end
end
```

Algorithm 1: Pseudocode of the algorithm used to create clips containing no shot.

The dataset has a collection of 4105 clips in total, of which 1904 of them contain shots and 2201 contain no shot. The number of inputs in B3SD is shown in table 5.2 below.

Class Distribution	
Layup	846
Free Throw	359
2-Point Shot	265
3-Point Shot	434
No Shot	2201
Total	**4105**

TABLE 5.2: Number of inputs for each class in B3SD.

As we are working with a complex fine-grained dataset, we want to investigate multiple classification tasks. We chose to include a high amount of no shot videos because, in addition to the task of distinguishing between all 5 classes, we wanted to distinguish between any shot and no shot. We also construct a version of the dataset without the 5th class in hopes of improving our ability to distinguish between shot types, exclusively. Lastly, we created a version reducing the number of no shot clips, as we believe the over representation of this class will hinder our ability to distinguish between all 5 classes. In total, we created 4 different versions of the dataset as listed below.

- 2 Classes: Shot, No Shot
- 4 Classes: Layup, Free Throw, 2-Point Shot, 3-Point Shot
- 5 Classes Full: Layup, Free Throw, 2-Point Shot, 3-Point Shot, No Shot
- 5 Classes Balanced: Same as above, but with a reduced number of No Shot inputs in order to have 5 somewhat equally distributed classes.

Each version was split 80/20 into training set and test set followed by another 80/20 split of the training set into the final training set and validation set. The size of the splits in each version are shown in table 5.3.

Dataset Versions

Classes	Train Set	Validation Set	Test Set
2	2623	659	823
4	1215	307	382
5	2623	659	823
5*	1497	378	471

TABLE 5.3: The train/val/test splits of the 4 different versions of B3SD used for experiments. *The balanced 5 class dataset.

In figures 5.5, 5.6 and 5.7 below, the number of inputs in each split of the four trimmed versions of B3SD are presented.

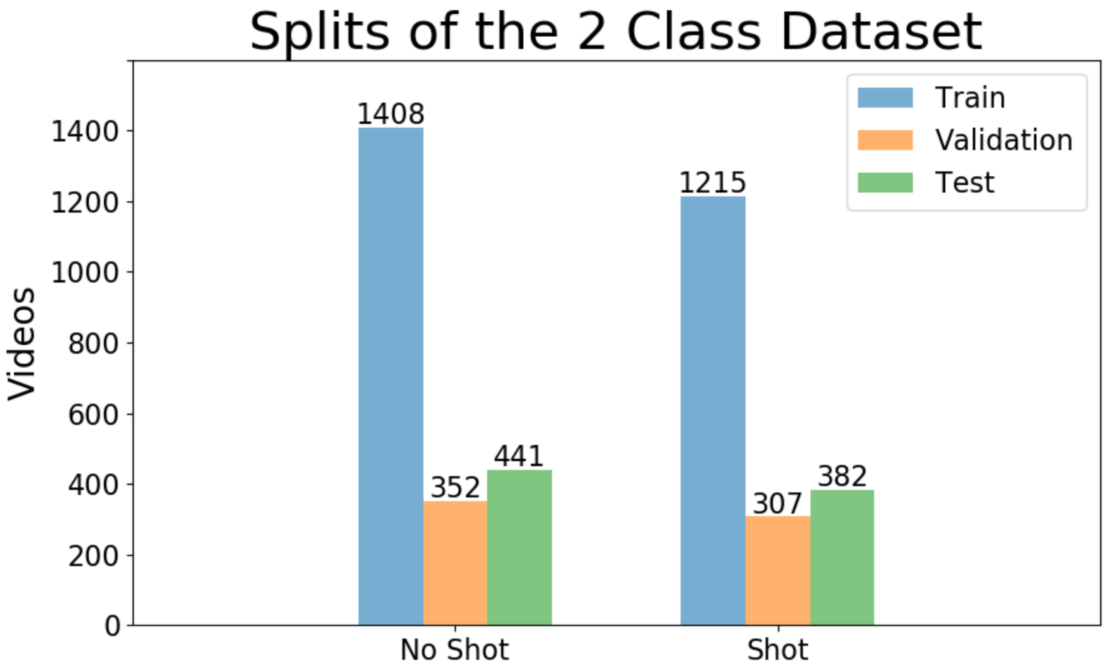


FIGURE 5.5: Train, validation and test split of the 2 class versions of B3SD.

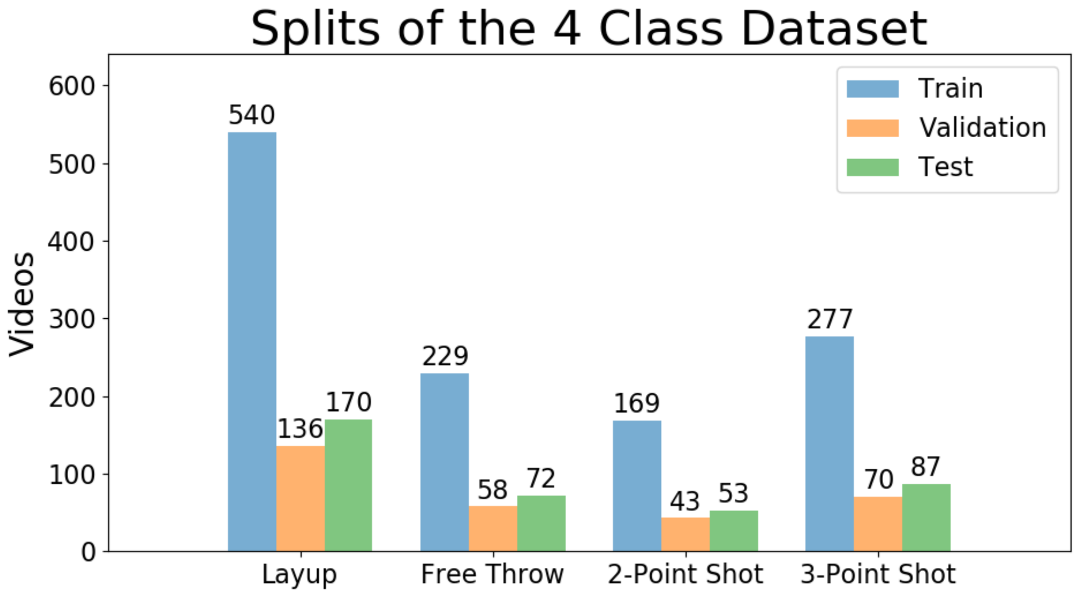


FIGURE 5.6: Train, validation and test split of the 4 class versions of B3SD.

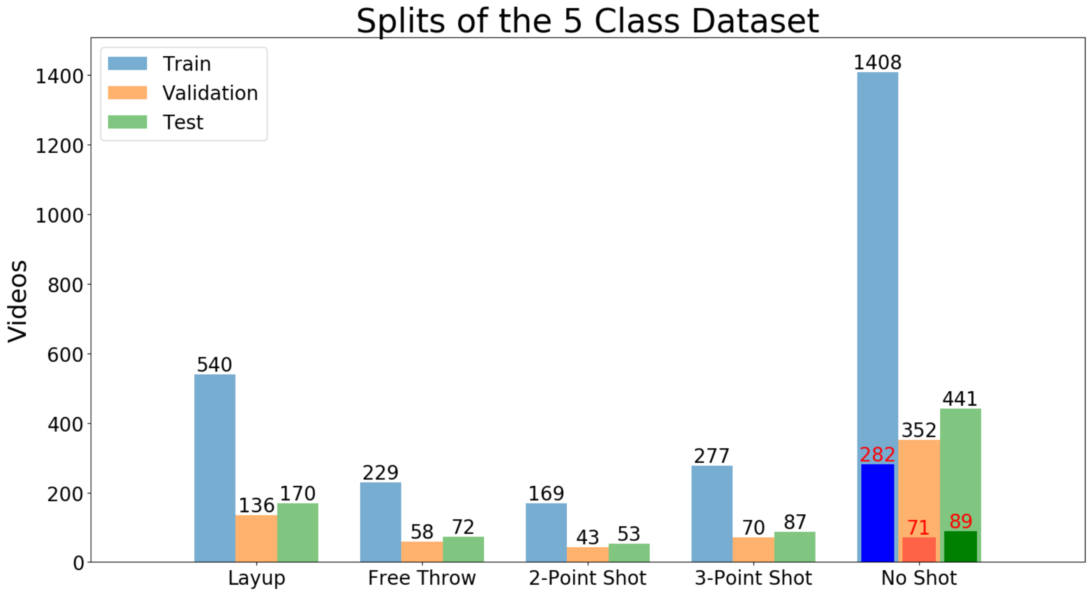


FIGURE 5.7: Train, validation and test split of the 5 class versions of B3SD. The bars with red numbers on top indicate the size of the no shot splits in the balanced 5 class version.

5.2 Limitations

In the following section, we identify a number of limitations with our dataset. Our current strategy when cropping 4 second clips does not account for quick consecutive shots occurring. In some cases, the ball falls very close to the basket after a missed shot and a second shot may occur right after. If the second shot occurs within 1 second after the first, both cropped videos would contain both shots. This is because the action was exactly 1 second in the clip. This is not ideal, since this eliminates the assumption of a cropped video only containing a single action. It is possible to filter out those videos if two stats are too close to each other by altering the annotation file we provide. This would, however, decrease the overall size of the dataset so we decided to keep the videos.

A different type of limitation involves generating clips without shots. We would like the clips to contain footage from an actual basketball game, but this is not guaranteed. The generated clips between annotations may contain game breaks, or ultimately garbage (such as commercials between televised games). With the validation procedure we are able to locate such clips and filter them out. It also enables us to locate annotators who are performing poorly. An annotator may draw bounding boxes too large or too small or chose an incorrect time of shot. Unfortunately, only a small part of the dataset has been verified at the current time. An overview

Dataset Name	Year	Static Camera	Background Motion	Videos	Classes	frame rate (fps)
KTH [82]	2004	Yes	No	600	6	25
Weizmann [85]	2005	Yes	No	600	10	25
CMU-Crowds [86]	2007	Yes	Yes	53	5	24-30
Hollywood-2 [87]	2009	No	Yes	2517	12	23-29
MSRII [89]	2010	Yes	Yes	54	3	15
Olympics [95]	2010	No	Yes	800	16	N/A
HMDB51 [90]	2011	No	Yes	6766	51	30
UCF101 [91]	2012	No	Yes	13320	101	25 and 29
Sports-1M [88]	2014	No	Yes	1130000	487	N/A
MPII-Cooking-2 [93]	2015	Yes	No	273	67	29
ActivityNet [92]	2015	No	Yes	19994	200	30
B3SD*	**2018**	**No**	**Yes**	**4105**	**5**	**N/A**

TABLE 5.4: Overview of the different datasets. Note how some frame rates are not available due to video retrieval from YouTube. For a summary of individual classes in each dataset, see Appendix A.

of the mentioned datasets alongside B3SD can be found in table 5.4. A plot with the number of clips vs. the number of categories for each dataset can be found in figure 5.8.

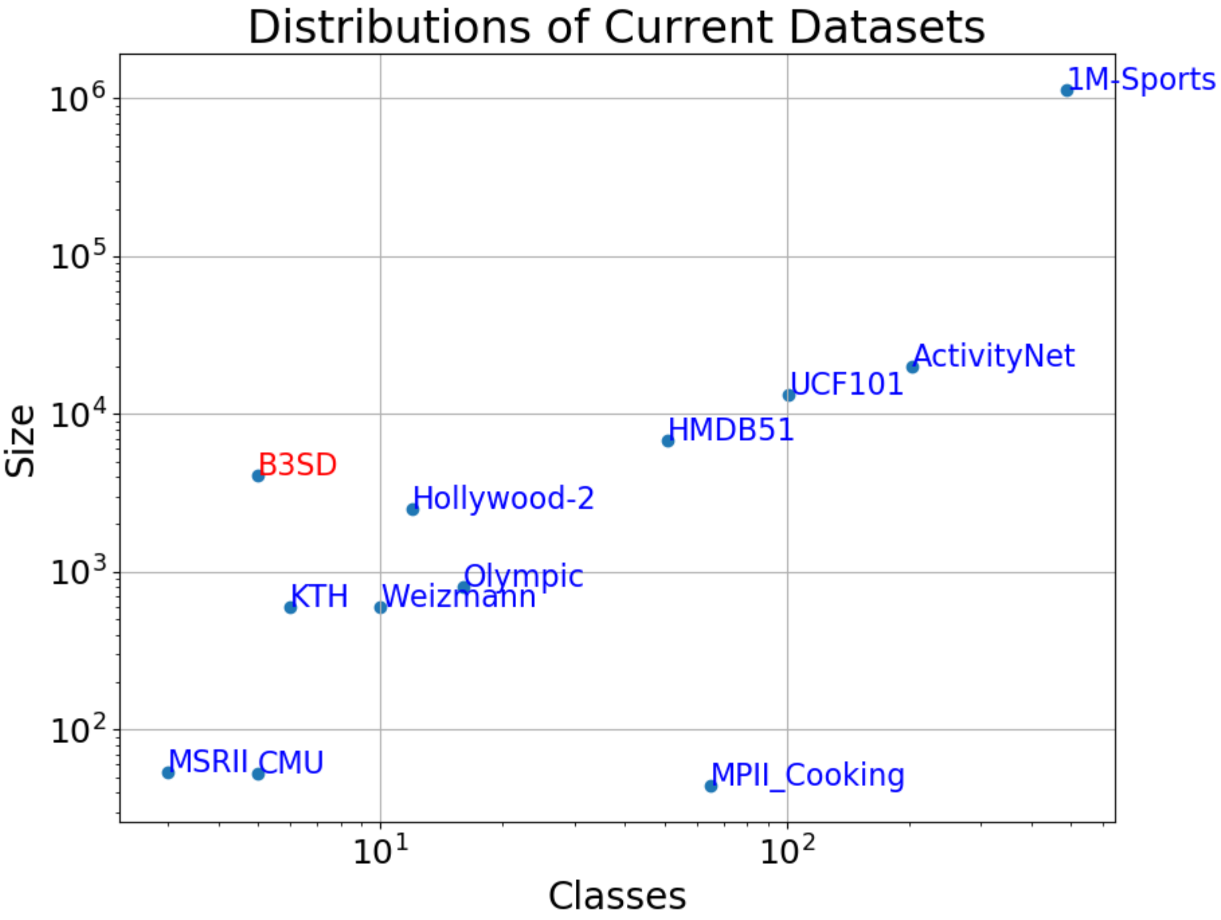


FIGURE 5.8: A summary of the number of clips and classes in the presented datasets. Note that the axis scales are logarithmic.

From the figure we see how the number of classes steadily increases with the number of clips available for most datasets. HMDB51, UCF101 and ActivityNet are popular datasets due to their high number of both classes and videos. B3SD contains only 5 classes, but is the largest fine-grained action dataset to our knowledge. B3SD has more clips than Hollywood 2, Olympic and Weizmann combined, 3 datasets similar to B3SD in the number of classes. It also has far more clips available than MSRII, the dataset most comparable to B3SD, due to its purpose of fine-grained classification tasks.

Should the reader wish to work with B3SD, a guide can be found in appendix B on how to recreate the dataset. The code is available here. If the reader wishes to work with an untrimmed version of B3SD for continuous action recognition, the steps for cropping the videos can be ignored as the annotations hold the time stamps for each action in the full length videos. The web interface for the data collection process is available from: www.kurish.dk. The dataset is currently not available online. Feel free to contact us if you wish to work with B3SD.

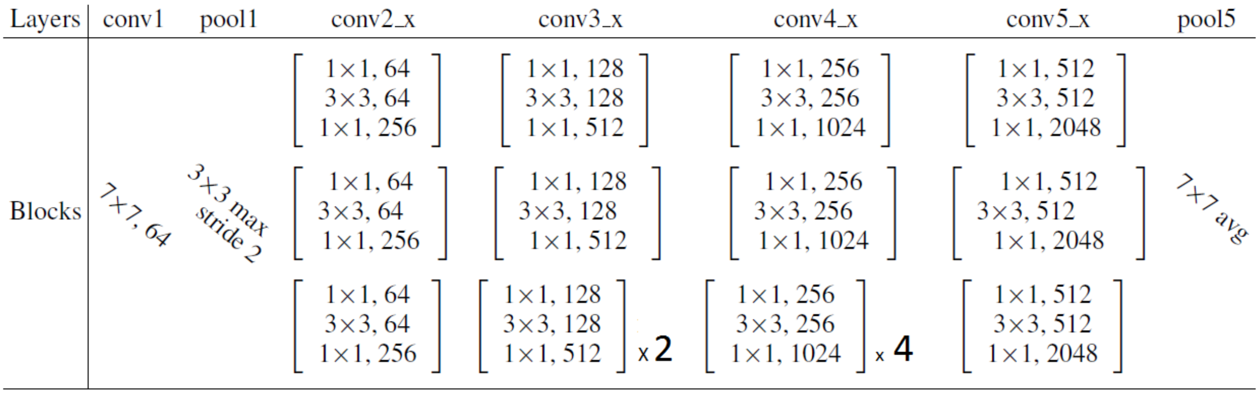
Chapter 6

Single Stream Model

To perform action recognition we use the ResNet-50 model as our single stream model as presented by [1, 4]. The model is implemented in in Keras [96], a high level neural networks API written in Python. We do not employ a 152-layer model as proposed in [1], because the marginal gain in accuracy ($< 2\%$) is not worth the additional training time. This chapter introduces its topology and the input used with both the spatial and temporal streams.

6.1 Topology of the ResNet-50 Model

The model consists of an input layer, a Convolutional Layer, a Max-pooling layer followed by 16 Residual Layers, a late Average-pooling layer and a final FC Layer with a Softmax activation. In total, it includes 50 Convolutional Layers and 2 Max-pooling layers. The architecture is presented in figure 6.1 below.



Layers	conv1	pool1	conv2_x	conv3_x	conv4_x	conv5_x	pool5
Blocks	7×7, 64	3×3 max stride 2	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$	7×7 avg
			$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$	
			$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$ x 2	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$ x 4	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$	

FIGURE 6.1: The ResNet-50 Model as presented by [4].

The initial Convolutional Layer consists of 64 7 × 7 kernels with stride 2 outputting 64 feature maps. Next, each feature map is downsampled to 56 × 56 using a 3 × 3 Max-pooling layer with stride 2. Next, 16 Residual Layers, each consisting of 3 Convolutional Layers and a skip connection are applied. The dimensions of the kernels in the Convolutional Layers are 1×1, 3×3 and 1×1, respectively. A 1×1 convolutional filter creates a feature map by combining the previous output at each spatial position. The number of filters chosen, can be used as a dimensionality increase or reduction [97]. The ResNet-50 model takes advantage of 1×1 convolutions by first reducing the dimensionality before applying the more expensive 3 × 3 convolutions. Another 1 × 1 convolution is then applied in order to increase the dimensionality to fit the skip connection. Following the last Residual Layer, 7 × 7 Average-pooling is applied to further reduce the dimensionality. Lastly, dropout and a FC Layer is added. The FC Layer has the same number of nodes as classes in the dataset followed by Softmax activation. This enables the use of 1-hot-encoding. For models such as ResNet-50, the total number of weights and the memory usage is relevant in order to determine whether or not the model can be trained with the resources available. These values are approximated in the following sections.

6.2 Weights in the ResNet-50 Model

For ResNet-50, the first Convolutional Layer has $3 \times 7 \times 7 \times 64 = 9.408$ weights. The subsequent Pooling Layer downsamples the input requiring no additional weights, followed by 16 Residual Layers. The first Residual Layer has $[64 \times 1 \times 1 \times 64] + [64 \times 3 \times 3 \times 64] + [64 \times 1 \times 1 \times 256] = 57.344$ weights, originating from the 3 Convolutional Layers. A similar calculation is made for the remaining Residual Layers based on the number of feature maps. Lastly, the final FC layer of 101 nodes (for UCF101) with Softmax activation tallies 206.848

weights. In total, the single stream model requires $\approx 9,5$ million weights in contrast to the popular VGGnet with 16 layers containing ≈ 130 million weights [48].

6.3 Memory Usage of the ResNet-50 Model

Considering single RGB-frames as input, a training input contains $224 \times 224 \times 3 = 150.528$ values. The output of the first Convolutional Layer contains $112 \times 112 \times 64 = 802.816$ values. Similarly, the Pooling Layer output contains 200.704 values. The first Residual Layer consists of 2.007.050 values, 602.112 values from each of the first two Convolutional Layers while the 3rd Convolutional Layer adds 802.816 values. All subsequent Residual Layers follow similar calculations, although with a reduced spatial dimensionality and increased number of feature maps. Finally, by converting all values to be stored as 32-bit floating points, the total memory usage estimation in a forward pass of a single training input in ResNet-50 is ≈115MB. An additional backwards pass requires more, because gradients must be stored. This estimate does not include batch normalization or any added bias. Table 6.1 summarizes the number of weights and values we need to store when training with ResNet-50.

Single Stream Model Size

Layer	Weights	Values
RBG Input (Optical Flow Input)	- -	150.528 (1.003.520)
Conv 1	9.408 (62.720)	802.816
Pool 1	-	200.704
Res 1	57.344	1.204.224
Res 2	69.632	1.204.224
Res 3	69.632	1.204.224
Res 4	245.760	602.112
Res 5	278.528	602.112
Res 6	278.528	602.112
Res 7	278.528	602.112
Res 8	983.040	301.056
Res 9	1.114.112	301.056
Res 10	1.114.112	301.056
Res 11	1.114.112	301.056
Res 12	1.114.112	301.056
Res 13	1.114.112	301.056
Res 14	3.932.160	150.528
Res 15	4.456.448	150.528
Res 16	4.456.448	150.528
AvgPool	-	2.048
FC	206.848	101
Total	20.892.864 (20.946.166)	9.435.237 (10.288.229)
Total + Weights	- -	30.328.101 (31.234.405)
Total Memory in MB	- -	115.7 **MB** **(**119.1 **MB)**

TABLE 6.1: Table summarizing the number of weights and memory required for the ResNet-50 model. The final sum, is a sum of all intermediate values multiplied by 4 and divided by 1024^2, when converting to MB.

6.4 Model Input

We separately train the model on the UCF101-dataset for spatial and temporal input. The spatial input consists of RGB-frames while the temporal input consists of 20 Optical Flow frames, 10 in each direction. For both streams the input dimensionality is converted to 224×224 before being fed into the model. In an attempt to better represent the full variation of the input space, we increase the size of the dataset by applying the data augmentations presented for each input modality in [1].

In the case of the spatial stream, we select a single random RGB-frame from each video and augment by randomly zooming up to $\pm 25\%$ in each direction. The resulting image is then randomly cropped up to 25% from each border. The cropped image is randomly flipped horizontally and resized using cubic interpolation. Lastly we subtract the mean value for each channel. A data augmentation example including mean subtraction is shown below in figure 6.2.

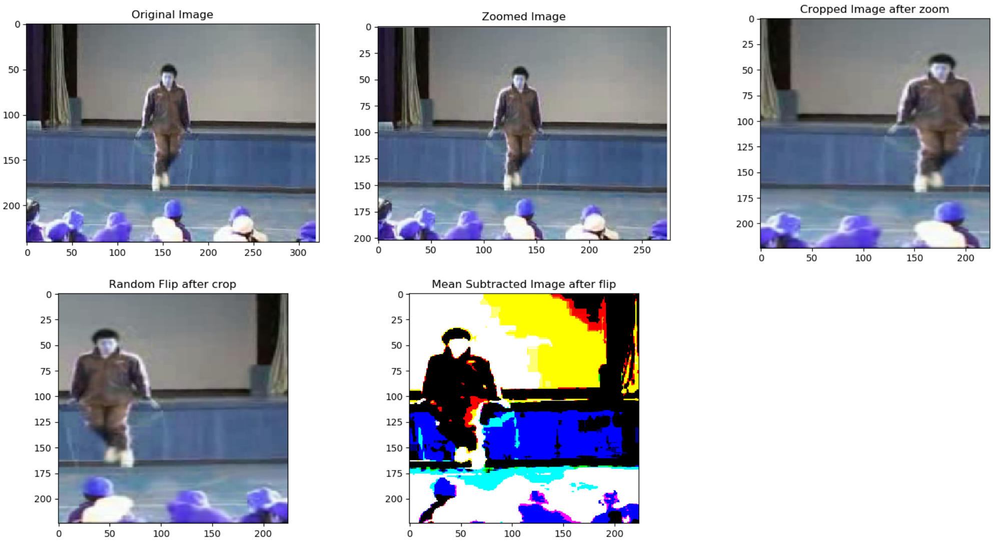


FIGURE 6.2: An example of the data augmentation procedure for spatial data. Images show intermediate steps from the original image (top left) to the final augmented input (bottom right.)

In the case of the temporal network, we select 10 consecutive flow frames in each direction at a random time in each video. For all sampled flows, a random value in the set $\{256, 224, 192, 168\}$ is chosen for each image direction as width and height. Should the random dimensionality exceed the original size, we choose the actual width or height of the image. The cropped image is randomly flipped horizontally and resized using cubic interpolation.

Chapter 7

Preliminary Experiments

We now conduct experiments on the single stream ResNet-50 model for spatial and temporal information, separately. We use the same parameter settings as in [1] in order to recreate their results. Instead of initializing weights randomly, they use weights pre-trained on ImageNet, a large image classification dataset. The use of pre-trained weights have been shown to provide faster convergence and better accuracy when used with new datasets for similar tasks. This is referred to as transfer learning with fine tuning. For the spatial network, the input dimensions are identical using RGB-frames. For the temporal network, however, an input of 20 channels is needed. Therefore we stack the weights of the first layer in the Imagenet weights until we reach 20 channels. In the test phase of each experiment, 25 uniformly distributed samples are chosen from each video and the average Softmax scores are used to classify the video. The testing procedure follows that of [1]. The training, validation and test accuracies for both experiments are shown in table 7.1.

Initial Experiment

Network	Train	Validation	Test	Test[1]
Spatial	96.7%	93.4%	62.4%	82.3%
Temporal	3.9%	5.7%	5.7%	87.0%

TABLE 7.1: Inital ResNet-50 experiment using the same settings as those in [1].

From the table we see, that the test accuracy of both networks is far inferior to those reported in [1]. The temporal stream is unable to classify well, while the spatial stream appears to overfit. The training accuracy is very high compared to the test data, suggesting that overfitting has occured. However, this does not appear to be the case in terms of the validation data. At the time of production, we wanted to construct a validation set since one was not provided by the official website. We decided to split the training data into training and validation sets. As the data splits are used for competitions, we believe that the creators constructed them with the intention of maximizing variance in the test set. This could be the reason why our validation accuracy in our spatial preliminary experiments was much higher than the test accuracy, something we failed to examine.

As our initial results were unsatisfying, we decided to examine the different settings of the model. By doing so we were forced to limit the scope of our work to not include action localization and continuous action recognition as planned.

7.1 Weight Initialization

We wish to examine the effect of the weight initialization. Assuming Optical Flow frames are not related to the visual channels, but to the overall movement of objects, we conduct an experiment where the first Convolutional Layer of the Imagenet weights are gray-scaled. Each of the 64 $7 \times 7 \times 3$ filters is converted to $7 \times 7 \times 1$ by applying the transformation $\texttt{gray} = 0.299r + 0.114g + 0.587b$ at every spatial position before being stacked 20 times to match the input dimensionality. By gray-scaling the weights for each flow frame, we believe that we represent structure better in the first Convolutional Layer. For comparison, we test with random weights for both streams. The validation loss for the weight initialization experiments are shown in figure 7.1.

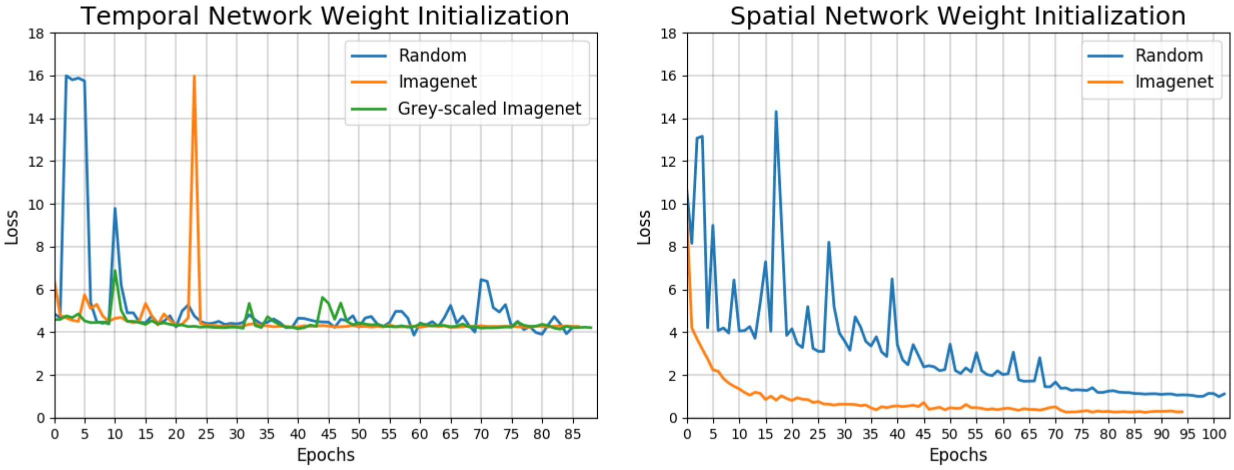


FIGURE 7.1: The influence of weight initialization in the temporal and spatial network.

Setting	Train	Validation	Test
Random	**11.4%**	**8.8%**	**13.4%**
ImageNet	3.9%	5.7%	5.7%
ImageNet*	4.6%	5.7%	7.6%

TABLE 7.2: Results from initializing the temporal network with various weights. *Gray-scaled ImageNet weights.

For the spatial network we observe that the use of pre-trained ImageNet weights, provide higher accuracy and faster convergence than with random weight initialization. We see that the loss after 20 epochs is considerably lower with pre-trained weights. This supports the claim for faster convergence. Another apparent advantage of applying pre-trained weights is the smooth learning curve. The saw-tooth pattern with random initialization is caused by large errors in the training phase, presumably since no basic features are learned yet. For the temporal network, we see a higher loss compared to the spatial network in all three experiments. Despite none of the initializations proving successful in the experiments, the Imagenet weights, both regular and gray-scaled, are less unstable than the random weight initialization. The test accuracy using gray-scaled weights are slightly better than original ImageNet weights.

As none of the results so far have been able to reach those of [1], we wish to experiment with a number hyper-parameters. For our baseline settings we choose to initialize with gray-scaled ImageNet weights, as we believe this should improve performance.

7.2 Hyper-parameter Study

Each experiment examines the influence of a single hyper-parameter modified from the baseline settings as used in [1] (with the exception of the temporal network weight initialization). The chosen parameters are just some of the possible ways to alter the model in order to influence the learning capability. The baseline setting for each of these parameters is presented in table 7.3.

Baseline Network Settings

Hyper-parameter	Spatial	Temporal
Data Augmentation	True	True
Batch size	256	128
Optimizer	SGD	SGD
Dropout Ratio	0.8	0.8
Activation	ReLU	ReLU
Flow Stacking	-	uuvv

TABLE 7.3: Baseline settings for both spatial and temporal networks as specified in [1] (except for the temporal network weight initialization).

7.2.1 Data Augmentation

To explore the result of data augmentation on the UCF101 dataset, we test two different data inputs. First, we provide no data augmentation at all. Second, we apply data augmentation techniques by randomly combining different scale levels and cropping sizes to the input. The validation loss for the spatial network is shown in figure 7.2.

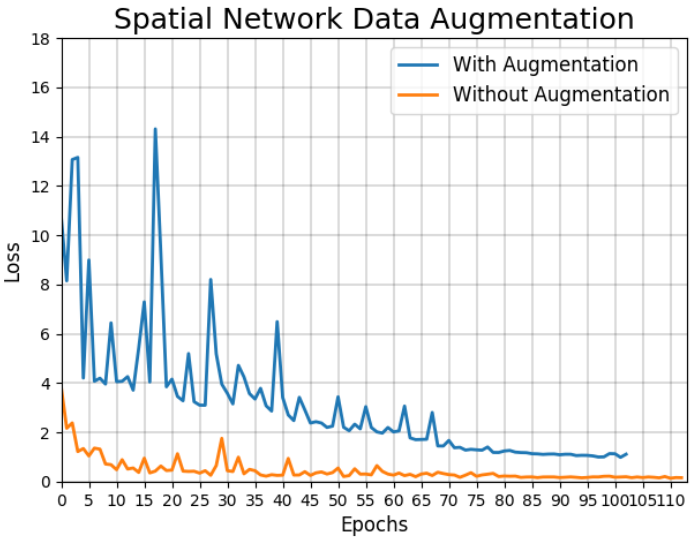


FIGURE 7.2: Validation loss of spatial network with and without data augmentation.

From the figure we observe that the use of data augmentation increased the validation loss, both in terms of convergence as well as overall loss. We also see a more unstable learning curve, suggesting inability to improve the ability to represent the full variance of the data with the type of data augmentation applied. However, table 7.6 shows that the inclusion of data augmentation resulted in better test accuracy.

7.2.2 Batch Size

We wish to find the best batch size for our network, and run experiments with sizes of 64, 128, 256. The validation loss for the spatial network is shown in figure 7.3

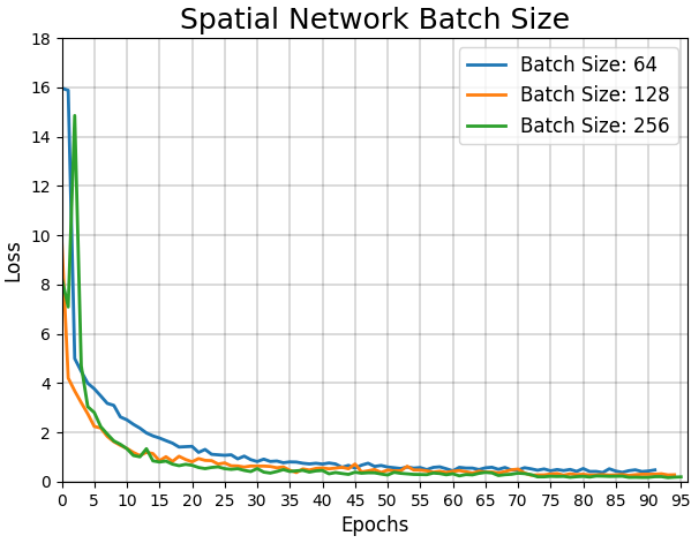


FIGURE 7.3: Validation loss of spatial network with different batch sizes.

We observe that the loss in the end is close to identical. The increased batch size of 256 decreases the validation loss marginally more than the other settings, supporting the claim of better generalization. Additionally, with 101 different classes, it is not possible to contain all classes within a batch size of 64. We expect this to influence the loss negatively. We observe minimal difference in the loss of the temporal network. In this case memory constraints limited us to a maximum batch size of 128. Therefore experiments testing lower batch sizes of 64 and 32 were done but provided no improvement as expected.

7.2.3 Activation Function

Next, we wish to examine whether the use of the Leaky ReLU or regular ReLU activation function obtains better classification accuracy. The validation losses for each of the two activation functions in both network streams are shown in figure 7.4 below.

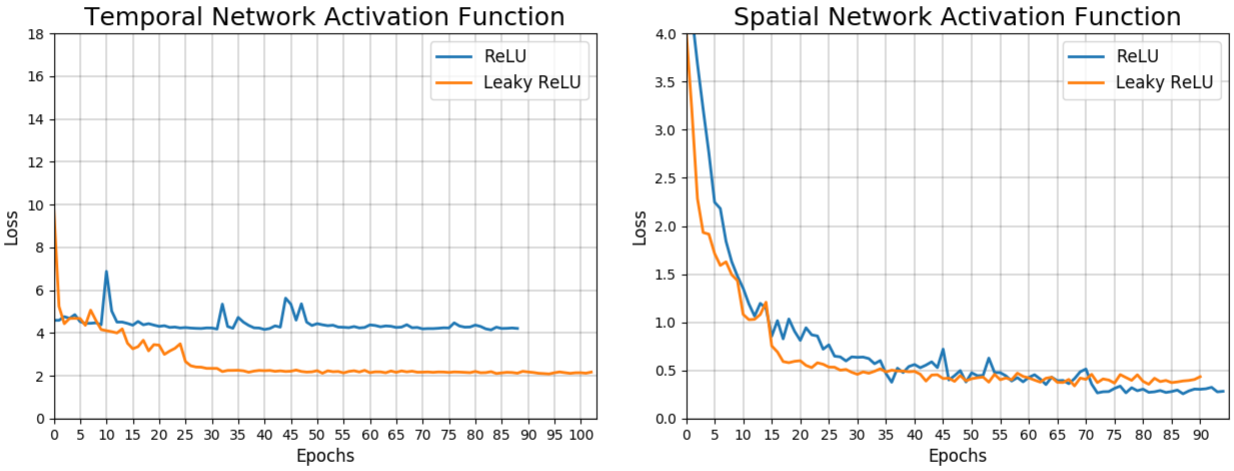


FIGURE 7.4: Validation loss of both temporal and spatial networks comparing the ReLU and Leaky ReLU activation functions.

For the temporal network, we observe that Leaky ReLU lowers the validation loss significantly compared to the regular ReLU activation. For the spatial network, we observe that Leaky ReLU performs slightly worse than regular ReLU.

7.2.4 Optimizer

We examine whether the ADAM optimizer provides any improvements compared to SGD. The resulting validation losses for the spatial network are shown in figure 7.5.

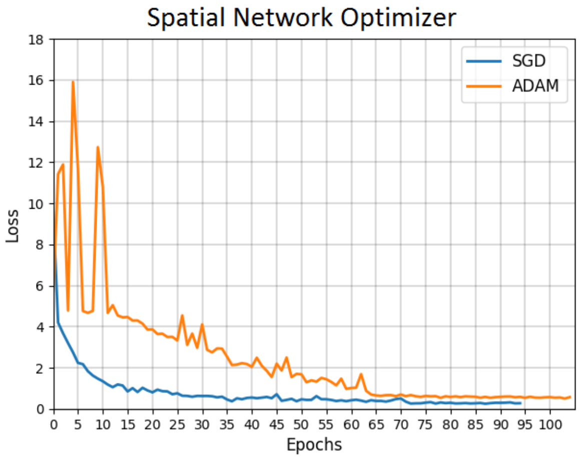


FIGURE 7.5: Validation loss for the spatial network when using SGD and ADAM optimizers.

From the figure we see that using SGD achieves a slightly lower validation loss than ADAM. The figure also shows that it takes much longer for ADAM to converge, suggesting that the recommended parameter initializations ($\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$) are suboptimal. We found no difference between the optimizers when experimenting on the temporal network. We also tested increased values of η for both streams, in an attempt to avoid converging to local minima. This, however, resulted in immediate divergence, causing NaN losses.

7.2.5 Dropout Ratio

We wish to examine if a dropout ratio below 0.8, as used in [1], can improve the network as we suspect this ratio to be too high. We experiment with dropout ratios of 0.5 and 0.2 for both networks while also testing without dropout entirely for the temporal network. Results are shown in fig 7.6.

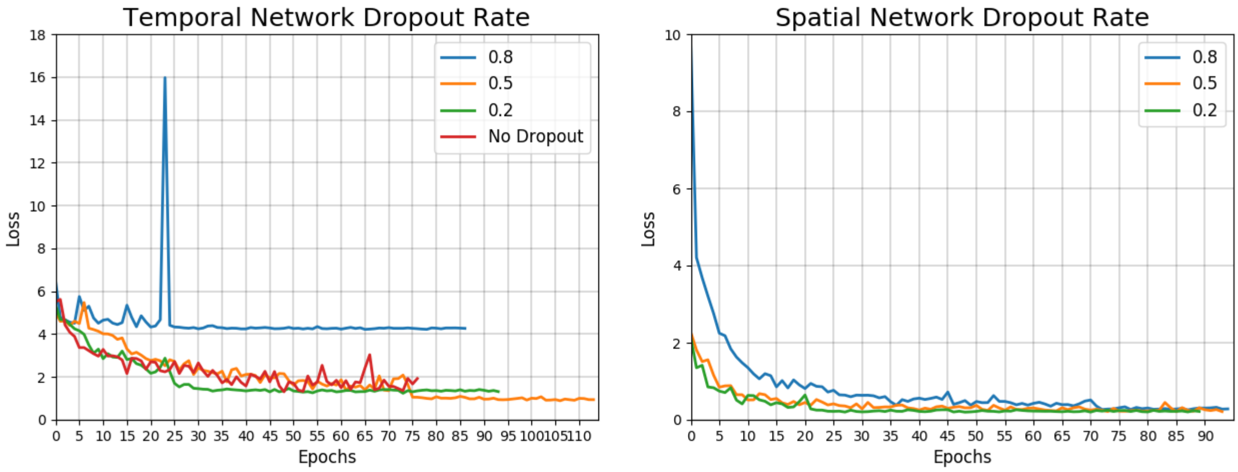


FIGURE 7.6: Validation loss with varying dropout ratios for both networks.

For the temporal network we see that the high dropout ratio hinders the learning ability of the network. All other dropout ratios obtain lower validation loss, with the complete absence of dropout resulting in a less stable learning curve while a ratio of 0.5 obtains the best validation loss. For the spatial network the validation losses are close to identical, with a slower convergence when the dropout ratio is 0.8.

7.2.6 Optical Flow Stacking

Next, we wish to examine how stacking Optical Flow influences performance. From [1] it is not clear how to merge the horizontal and vertical flow and so we explore two ways of stacking them. We examine 10 consecutive frames with horizontal flow followed by 10 frames with vertical flow (uuvv), and alternating the two directions, such that each frame with horizontal flow is followed by a frame with vertical flow (uvuv). The validation loss is shown in figure 7.7. From the figure we see only a marginal decrease in loss using consecutive frames.

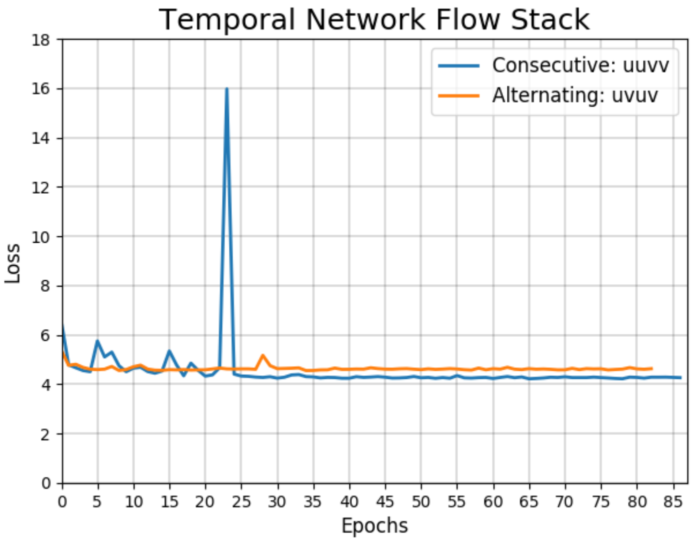


FIGURE 7.7: Validation loss in the temporal network when comparing the flow stack (uuvv and uvuv).

7.2.7 Summary of Hyper-parameter Study

We combine the results and train a new model for both streams using the best parameters achieving the best result. For the spatial network, this increased the test accuracy. For the temporal network, our new model was unable to reach that of our experiment with a dropout of 0.5. We decided to use that model as our final temporal network. The settings are specified in table 7.6. Comparing our final settings to those of [1], we differ with a few variables. Specifically, the dropout ratio in both networks, the data augmentation in the spatial network and the weight initialization in the temporal network.

Results of Hyper-parameter Study

		Spatial Network			Temporal Network		
Setting	Value	Train	Validation	Test	Train	Validation	Test
Baseline Settings	See Table 7.3	96.7%	93.4%	62.4%	4.6%	5.7%	7.6%
Data Augmentation	False	**99.9%**	**96.5%**	25.3%	3.7%	4.5%	4.3%
Batch size	32	-	-	-	3.9%	5.1%	5.9%
Batch Size	64	95.0%	90.3%	46.6%	3.5%	3.9%	4.8%
Batch Size	256	99.0%	96.3%	67.1%	-	-	-
Dropout Ratio	0.5	98.4%	94.1%	67.3%	**79.2%**	**74.8%**	**78.6%**
Dropout Ratio	0.2	98.6%	93.9%	**69.6%**	65.8%	65.9%	74.1%
Dropout Ratio	0.0	-	-	-	75.5%	64.7%	73.2%
Activation	Leaky ReLU	88.7%	89.9%	1.0%	37.8%	44.1%	1.1%
Optimizer	ADAM	83.4%	84.7%	52.8%	4.5%	4.4%	6.0%
Flow Stack	uvuv	-	-	-	1.7%	1.9%	2.4%
Final Settings	See Table 7.4	99.8%	75.1%	77.1%	79.2%	74.8%	78.6%

TABLE 7.4: All accuracy scores from preliminary experiments. Note how these scores are taken at the lowest loss over all epochs. This means that the highest validation accuracy achieved for some settings has taken more epochs than others. Also notice how some results are very low, due to the dropout ratio for all those experiments at 0.8. This is problematic, because it may not show the full potential of each setting.

Final Network Settings

Setting	Spatial Network	Temporal Network
Weight Initialization	ImageNet	ImageNet*
Data Augmentation	False	True
Batch size	256	128
Optimizer	SGD	SGD
Dropout Ratio	0.2	0.5
Activation	ReLU	ReLU
Flow Stacking	-	uuvv

TABLE 7.5: Final settings for both networks based on the preliminary study. * Gray-scaled ImageNet weights.

Final Network Settings

Setting	Two Stream Network
Weight Initialization	Trained Single Streams
Data Augmentation	False
Batch size	128
Optimizer	SGD
Dropout Ratio	0.5
Activation	ReLU
Flow Stacking	uuvv

TABLE 7.6: Final settings for both networks based on the preliminary study. * Gray-scaled ImageNet weights.

7.3 Temporal Flow Stride

Selecting a random RGB-frame as done with the spatial network, is a type of selection strategy. So far, we have used a similar strategy for the temporal network, selecting the flows in each direction of 10 consecutive frames at a random point in the video. We wish to examine if selecting flows with varying temporal stride increases performance. To select frames using a varying temporal stride we consider a random starting point in the video and a random possible jump interval between 5 and 15 frames. If it is not possible to sample flows at an interval of at least 5, the highest possible interval is used instead. An example of flow selection with different temporal strides can be seen in figure 7.8. By using a varying temporal stride, we expect the network to obtain invariance to actions performed at different speeds or recorded with different frame rates, thereby improving performance. When testing varying temporal stride, instead of 25 uniformly distributed tests, we simply ran the test of each input video 25 times. We expect that selecting a random starting point and temporal stride creates equal amount of variance. The validation losses are shown in figure 7.9.

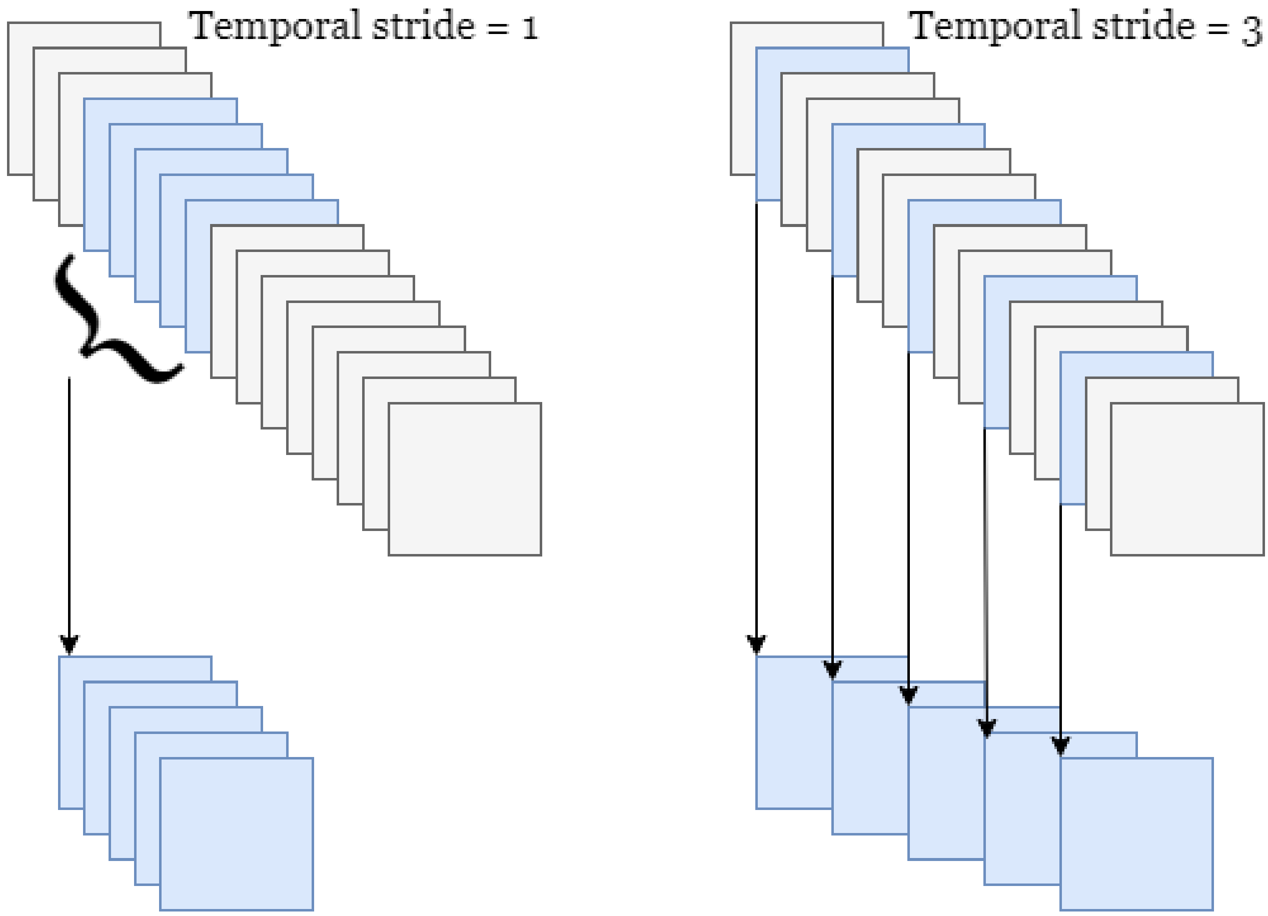


FIGURE 7.8: **Left:** Consecutive flow frames are selected. **Right:** A new temporal stride is chosen between selected flows of each video.

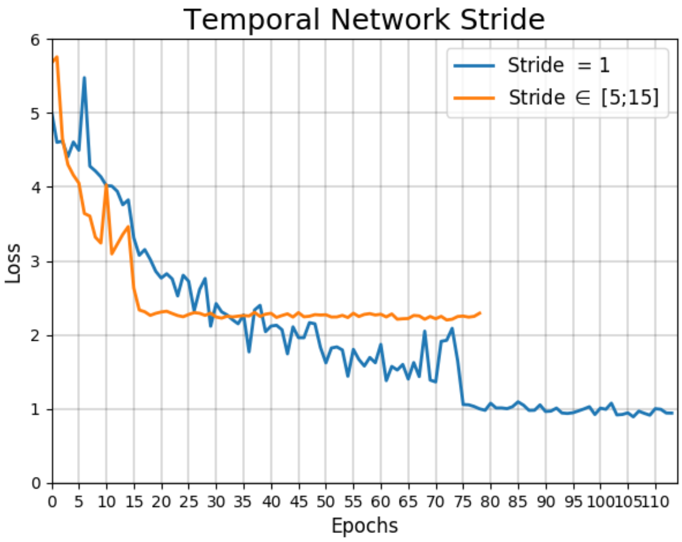


FIGURE 7.9: Validation loss in the temporal network with fixed and varying temporal stride.

Stride	Train	Validation	Test
Fixed at 1	79.2%	74.8%	**78.6%**
Varying	73.5%	43.1%	45.0%
			42.1%*

TABLE 7.7: Results using fixed or varying temporal stride. *Varying temporal stride during testing.

The figure shows that using a varying temporal stride does not improve the validation loss. The learning curves using varying temporal stride is steeper at first but abruptly stops decreasing. This could be due to the loss function reaching a local minimum. As the test accuracy is also higher with temporal stride = 1, we decided to use this setting with all experiments going forward.

7.4 Network Depth

After evaluating which hyper-parameters to apply as well as examining the temporal stride, we experiment with the depth of our network model. Increasing the depth in networks, by adding layers consequently adds to the complexity, memory requirements and training time. For this reason we experiment with removing several intermediate Residual Layers of the current 50-layer model.

Alongside our initial 50-layer model, the spatial stream is tested using 38, 26 and 20 layers. The simplified models are shown in figure 7.10. The results are shown in figure 7.11.

Layers	conv1	pool1	conv2_x	conv3_x	conv4_x	conv5_x	pool5
Blocks	7×7, 64	3×3 max stride 2	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$ $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$ $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$ $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$ $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$ $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$ $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$ $\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$ $\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$	7×7 avg

Layers	conv1	pool1	conv2_x	conv3_x	conv4_x	conv5_x	pool5
Blocks	7×7, 64	3×3 max stride 2	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$ $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$ $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$ $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$ $\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}$	7×7 avg

Layers	conv1	pool1	conv2_x	conv3_x	conv4_x	pool5
Blocks	7×7, 64	3×3 max stride 2	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$ $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$ $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$ $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}$	7×7 avg

FIGURE 7.10: A comparison of different simplified ResNet architectures. **Top**: The 38-layer model **Middle**: 26-layer model **Bottom**: 20-layer model.

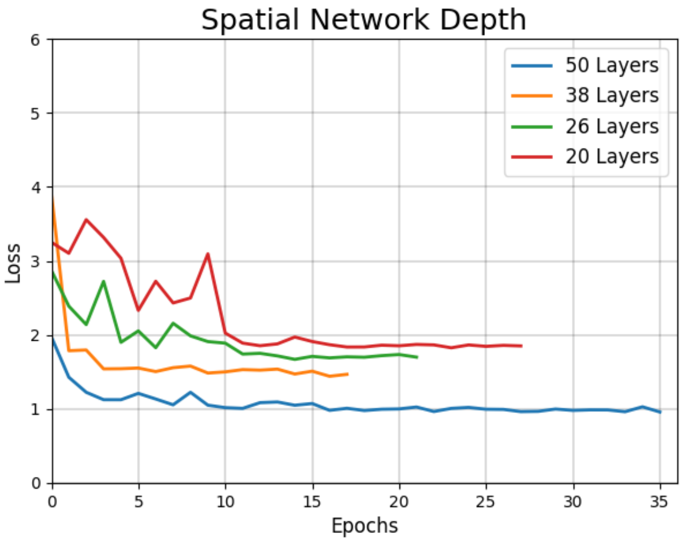


FIGURE 7.11: Comparing the Resnet model at 4 different depths.

The figure shows that the full 50-layer model outperforms all simplified models. Both in terms of final validation loss as well as the convergence. The accuracies of each phase of the model are shown in table 7.8, also showing the large improvement in accuracy using the 50-layer model. This indicates that a high level of complexity is needed for video action recognition. Therefore we will use the 50-layer model for all experiments going forward.

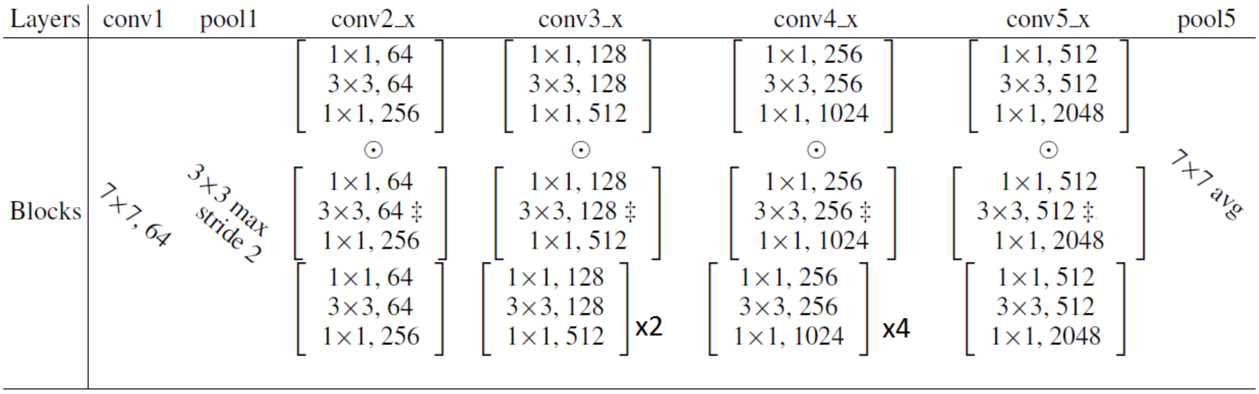
Layers	Train	Validation	Test
50	99.8%	75.1%	**77.1%**
38	99.8%	64.6%	65.9%
26	98.9%	58.2%	25.8%
20	88.0%	52.5%	23.4%

TABLE 7.8: Results using different network depths.

Chapter 8

Two Stream Model

We now present the two stream model combining a spatial stream and a temporal stream as presented in [1], each based on the ResNet-50 models. Each stream is initially trained individually. During training of the two stream model, we select a random RGB-frame and a set of flows centered at the RGB-frame for each video. Both streams are augmented using the augmentation procedure of the single stream spatial network to ensure consistency. Additionally, a batch size of 128, due to memory constraints. We initialize the network with the best parameters of our preliminary study as stated in table 7.6. In order to better learn spatiotemporal features we inject temporal information into the spatial stream by multiplying feature maps, referred to as Multiplicative Gating, in short MG. In order to further increase the use of temporal information, we also add additional $1 \times 1 \times 3$ convolutions in the spatial network, called Temporal Injections (TI). We fuse the streams by averaging the Softmax output for final classification. The model is shown in figure 8.1.



Layers	conv1	pool1	conv2_x	conv3_x	conv4_x	conv5_x	pool5
Blocks	7×7, 64	3×3 max stride 2	1×1, 64 3×3, 64 1×1, 256 ⊙ 1×1, 64 3×3, 64 ‡ 1×1, 256 1×1, 64 3×3, 64 1×1, 256	1×1, 128 3×3, 128 1×1, 512 ⊙ 1×1, 128 3×3, 128 ‡ 1×1, 512 1×1, 128 3×3, 128 1×1, 512 x2	1×1, 256 3×3, 256 1×1, 1024 ⊙ 1×1, 256 3×3, 256 ‡ 1×1, 1024 1×1, 256 3×3, 256 1×1, 1024 x4	1×1, 512 3×3, 512 1×1, 2048 ⊙ 1×1, 512 3×3, 512 ‡ 1×1, 2048 1×1, 512 3×3, 512 1×1, 2048	7×7 avg

FIGURE 8.1: The two stream model as introduced by [1]. ⊙ indicates Multiplicative Gating and ‡ indicates Temporal Injection.

8.1 Multiplicative Gating & Temporal Injections

During training of our model, we inject information from the temporal stream into the spatial stream using multiplication. The multiplicative gate is placed just before the input to a new residual block (see figure 8.2). The information injected is feature maps of the temporal stream. They are multiplied with corresponding feature maps from the spatial stream, which allows the spatial stream to use temporal information during training of the network. The idea of injecting information from one stream to another is that an action is defined by a combination of the available information. Injections, however, might pose a problem, because the input modalities contain different types of information. This was shown in [1] when injecting spatial information into the temporal network. The obtained results were inferior in this case. On the other hand, injecting temporal information into the spatial network improved the overall accuracy. Therefore we see a benefit of injecting information from the temporal stream. TIs are added in order to provide greater temporal support [1]. They are initialized as $[0, 1, 0]$-kernels producing an identity mapping. The are inserted into Residual Layers of the spatial stream. An illustration of the exact placement in a Residual Layer is shown in figure 8.2.

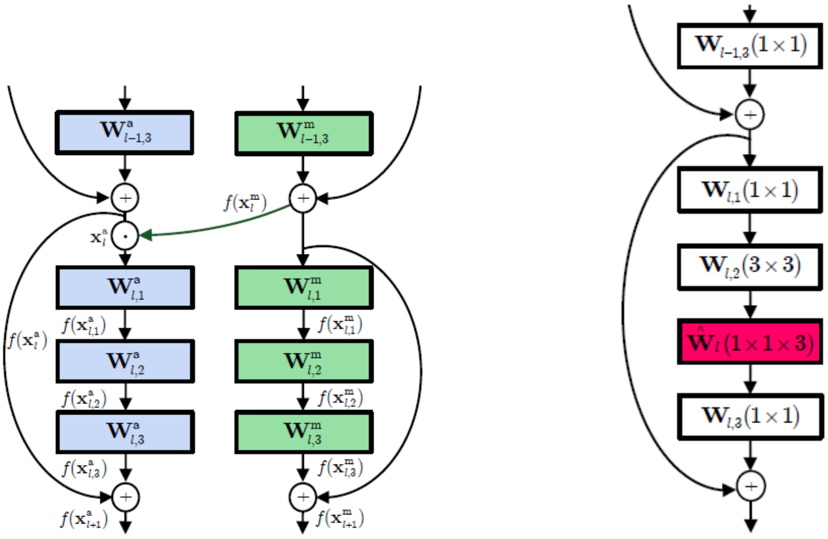

$\mathbf{W}^{a}_{l-1,3}$
$\mathbf{W}^{m}_{l-1,3}$
$f(\mathbf{x}^{m}_{l})$
$\mathbf{x}^{a}_{l}$
$\mathbf{W}^{a}_{l,1}$
$\mathbf{W}^{m}_{l,1}$
$f(\mathbf{x}^{a}_{l,1})$
$f(\mathbf{x}^{m}_{l,1})$
$f(\mathbf{x}^{a}_{l})$
$\mathbf{W}^{a}_{l,2}$
$\mathbf{W}^{m}_{l,2}$
$f(\mathbf{x}^{a}_{l,2})$
$f(\mathbf{x}^{m}_{l,2})$
$\mathbf{W}^{a}_{l,3}$
$\mathbf{W}^{m}_{l,3}$
$f(\mathbf{x}^{a}_{l,3})$
$f(\mathbf{x}^{m}_{l,3})$
$f(\mathbf{x}^{a}_{l+1})$
$f(\mathbf{x}^{m}_{l+1})$
$\mathbf{W}_{l-1,3}(1\times 1)$
$\mathbf{W}_{l,1}(1\times 1)$
$\mathbf{W}_{l,2}(3\times 3)$
$\hat{\mathbf{W}}_{l}(1\times 1\times 3)$
$\mathbf{W}_{l,3}(1\times 1)$

FIGURE 8.2: **Left**: An illustration detailing Multiplicative Gating. A set of feature maps from the temporal stream (green) are multiplied into the spatial stream (blue). **Right**: Temporal Injection (highlighted in red) within a Residual Layer.

8.2 Number of Weights and Memory Usage

The total number of weights in the two stream model is the number of weights needed and memory usage of MG and TIs in addition to that of the two single stream models combined. When applying MG of the two stream model, new feature maps are constructed, requiring additional memory for storing values, while no weights are added. The dimensionality is dependent on the layers being multiplied together. There are 4 multiplicative gates, where the 1st gate requires $56 \times 56 \times 256 = 802.816$ values. The next 3 gates require half the number of values as the previous, tallying 401.408, 200.704 and 100.352. In total, MG requires $1.505.280$ values.

When applying TIs, each with a size of $1 \times 1 \times 3$, we only require 12 additional weights, 3 for each injection. We do require additional memory when storing the result of the convolution. The resulting output has the same dimensionality as the input. Thus for the 1st injection we obtain $56 \times \ 56 \times 64 = 200.704$. Similarly, we obtain 100.352, 50.176 and 25.088 for the remaining injections. In total the number of new values stored is 376.320. Combining these numbers with the previous from the single stream model estimation, we obtain a total number of weights equal to $41.839.030$ and values in memory equal to $9.435 + 10.288.229 + 376.320 + 1.505.280 = 21.605.066$. Thus the total memory usage estimation in a forward pass of a single training input in the two stream model is $\approx$ 242.0MB, excluding biases and batch normalization. The estimates are shown in table 8.1.

Two Stream Model Size

Model	Weights	Values	Memory
Spatial Single Stream	20.892.864	9.435.237	115.7MB
Temporal Single Stream	20.946.166	10.288.229	119.1MB
Two stream	**41.839.042**	**21.605.066**	**242.0MB**

TABLE 8.1: Table summarizing the number of weights and memory required for the two stream model.

Chapter 9

Experiments & Results

9.1 Two Stream UCF101 Experiments

Testing the two stream model on the UCF101 dataset, we achieve an improvement in performance of $\approx 2\%$ compared to the individual single stream models as shown in table 9.1. In order to examine the influence of Multiplicative Gating and Temporal Injections, we also conduct an experiment using only late fusion between the two streams. The results show that including MG and TI improves accuracy by 2.9%. Additionally, we observe that all models excluding the single stream model with temporal information overfit.

Two Stream Network Results

Network	Train	Validation	Test
Single Stream Spatial	99.8%	75.1%	77.1%
Single Stream Temporal	79.2%	74.8%	78.6%
Two Stream	**99.9%**	75.8%	77.6%
Two Stream with MG & TIs	99.5%	**76.2%**	**80.5%**

TABLE 9.1: Results on the two stream model with UCF101 compared to the best spatial and temporal single stream model.

We now conduct a number of experiments looking to successfully classify shots in basketball videos. This task is different from that of UCF101, as it requires a higher level of fine-grained recognition abilities. We wish to explore if the ResNet-50 model and the two stream model can effectively solve this task.

In the UCF101-dataset, the action is typically placed in the center of the frame or at least in the vicinity. This is not the case for B3SD and basketball game footage in general. Although the ball is the most important object to keep inside the camera frame, the spectator typically wants to be able to see all 10 players on the floor. This means that the person handling the ball may be at the edge of the frame. If a person at the edge decides to take a shot, we risk cropping him or her out during our data augmentation process. As the variation between the classes in B3SD is less than in the UCF101 dataset, we presume the shooter being cut out to be too influential to classify accurately. Therefore, we decided to remove cropping and zooming from the data augmentation on B3SD experiments and only perform mean subtraction and horizontal flipping. We start by examining the spatial and temporal models on B3SD, individually, before using the two stream model to fine-tune.

9.2 B3SD Experiments

As the classes of B3SD are not evenly distributed, our first experiment compares classification ability of the spatial network with and without scaling the weights per class frequency on the 4 class version of the dataset containing only shots. We chose this version due to its smaller size allowing for quicker training procedure. The results are shown below:

Scaling Weights Per Class Frequency

Weight Scaling	Train	Validation	Test
With	78.2%	54.7%	**58.4%**
Without	60.4%	41.4%	41.4%

TABLE 9.2: Results on the spatial network using 4-class B3SD with and without the use of weight scaling.

The results show that the use of weight scaling increases the overall accuracy. The results are quite significant, increasing the test accuracy from 41.4% without weight scaling to 58.4% with it. We therefore applied weight scaling on the rest of the experiments.

As a next stage in the experiments, we wanted to explore different input modalities for the network. As the shooting motion and the shot itself is only a small part of each video in B3SD, we wanted to determine how much of an impact the chosen frame had. We therefore conducted experiments using both a random frame (rf) from the clip as done with UCF101 but also always using the frame at the time of the shot (tos). The time of the shot is specified to be the frame at second 1 in all B3SD videos. We conducted these experiments with both the spatial stream and the temporal stream on the 4-class version of the dataset. In addition to the frame selection, for the spatial stream we wanted to explore whether the use of gray-scaled frames would provide results as good as when using all three RGB-channels. By using gray-scaled images we are able to reduce the total size of the dataset. The results are shown in table 9.3 below.

Spatial Network 4 Classes

Input	Train	Validation	Test
RGB rf	78.2%	54.7%	58.4%
RGB tos	91.9%	**59.0%**	**58.6%**
Gray rf	49.0%	45.6%	47.6%
Gray tos	**97.7%**	51.8%	53.1%

TABLE 9.3: Results on 4-class B3SD with RGB and gray-scaled frames with both random frame (rf) selection as well as time of shot (tos) frame.

From the table see that overfitting occurs in 3 of the 4 cases. Especially for the two input modalities using time of shot frames. We also observe that using gray-scaled frames decreased the accuracy with both random frames as well as the time of shot. Using time of shot increased the accuracy in both cases.

For the temporal stream we wanted to explore a number of different input modalities that challenge the use of Optical Flow. Particularly, we tested stacked gray-frames at 3 and 20 frames per stack as well as difference frames with stack sizes of 1, 3 and 20. With these input modalities, the network is tasked to learn the motion within the videos implicitly as opposed to using Optical Flow, where the motion in the videos has been calculated beforehand. These modalities are simpler to implement and require less computational effort than Optical Flow. As with the spatial network we tested both with a random frame selection as well as at the time of shot. The results are shown in table 9.4 below.

Temporal Network 4 Classes

Input	Channels	Train	Validation	Test
Flow rf	20	42.7%	39.1%	39.3%
Flow tos	20	41.2%	44.6%	44.5%
Diff rf	1	46.0%	42.7%	44.8%
Diff tos	1	55.9%	49.8%	44.5%
Diff rf	3	48.4%	41.0%	47.6%
Diff tos	3	59.1%	47.2%	52.6%
Diff rf	20	49.2%	45.3%	49.0%
Diff tos	20	56.4%	45.9%	51.6%
Gray rf	3	45.7%	44.6%	44.5%
Gray tos	3	44.0%	44.3%	44.5%
Gray rf	20	72.3%	55.0%	56.3%
Gray tos	20	**95.3%**	**58.3%**	**58.1%**

TABLE 9.4: Results on 4-class B3SD with different input modalities: Optical Flow (Flow), stacked gray difference frames (Diff) and stacked gray frame (Gray). All are trained with the middle frame as both a random frame (rf) selection as well as the time of shot (tos) frame.

From the table we see that, on average, using the time of release of the shot increased accuracy. The exceptions are using a single difference frame and when using 3 stacked gray-frames. We also see that both difference frames and stacked gray frames achieved similar or higher accuracy than Optical Flow. The highest accuracy was achieved by 20 stacked gray frames at the time of shot. As a result of the experiments on the 4 class version of the dataset, we decided to conduct all following experiments using the time of shot as the point to collect both RGB-frame for the spatial network and 20 gray-frames for the temporal network. This was done for the other 3 versions of B3SD and the accuracies of all 4 versions are presented below in table 9.5:

Single Stream Network Results

	Spatial Network			Temporal Network		
Classes	Train	Validation	Test	Train	Validation	Test
2	90.9%	77.5%	72.5%	98.7%	88.9%	85.4%
4	91.9%	59.0%	58.6%	95.3%	58.3%	58.1%
5	89.8%	66.5%	61.7%	89.1%	61.9%	53.7%
5*	88.4%	54.0%	51.0%	68.1%	56.3%	52.7%

TABLE 9.5: Results using single stream network on different versions of B3SD. *Balanced version of dataset.

Next, we want to see if a more relevant weight initialization of the 5-class network could improve the accuracy. We therefore initialize the 5-class network with the best 2-class weights recorded and fine-tune these to the 5 classes. The results are shown in table 9.6 below.

5 Class Single Stream Network Using 2 Class Weight Initialization

		Spatial Network			Temporal Network		
B3SD Size	Weight Init	Train	Validation	Test	Train	Validation	Test
Balanced	ImageNet	88.4%	54.0%	51.0%	68.1%	56.3%	52.7%
Balanced	2-Class	76.3%	50.5%	52.7%	84.8%	57.4%	**57.1%**
Full	ImageNet	89.8%	**66.5%**	61.7%	89.1%	61.9%	53.7%
Full	2-Class	**91.3%**	64.5%	**63.9%**	**91.8%**	**69.2%**	54.1%

TABLE 9.6: Results using 2 class weight initialization to train the 5 class model compared to ImageNet weights.

From the table we see that using 2 class weight initialization for 5 class networks improve the test accuracy for all experiments. Therefore, we use the single stream 5 class networks with 2 class weight initialization when training two stream 5 class models. The results using the two-stream model on all versions of B3SD are shown in table 9.7.

Two Stream Network Results

Classes	Train	Validation	Test
2	98.8%	92.3%	88.6%
4	94.7%	59.3%	62.8%
5	90.4%	65.3%	65.9%
5*	92.3%	56.9%	55.0%

TABLE 9.7: Results using two stream multiplier network on different versions of B3SD. *Represents the 5 class model trained on the balanced 5 class dataset.

The results show that the two stream model increases the test accuracy in 3 of the 4 models, with the exception being the balanced 5 class model. Furthermore, we see all 4 models overfit to the training data.

9.2.1 Two-Part Two Stream Model

In an attempt to further increase accuracy on the 5 class models, we also constructed a two-part two stream network. Initially, we filter shots from no shots by predicting with the 2 class model. Next, the videos classified as containing shots are then predicted using the 5 class model. By using the 5 class model instead of the 4 class model we also have the capability of correcting no shot videos that were incorrectly classified to start with. An illustration of the model is shown in 9.1 and the test accuracies are shown in table 9.8 below.

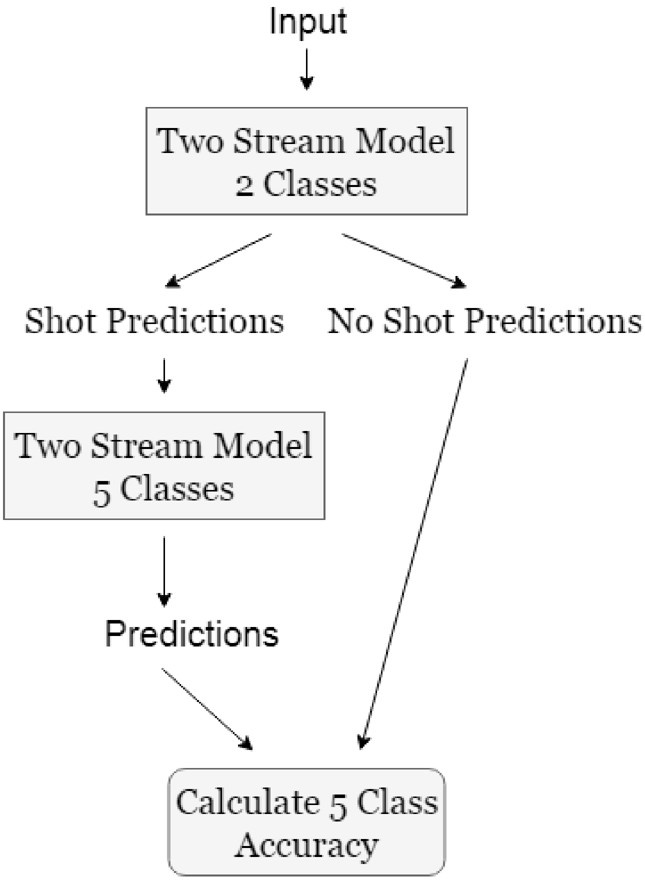

Input
Two Stream Model 2 Classes
Shot Predictions
No Shot Predictions
Two Stream Model 5 Classes
Predictions
Calculate 5 Class Accuracy

FIGURE 9.1: The two-part two stream model used for classifying B3SD.

Two-part Two Stream Results

Model	Test Accuracy
5	68.3 %
5*	74.7%

TABLE 9.8: Results using the two-part two stream network when training on both the full 5 class model and the balanced 5* class model. Both models are tested on the full test set.

With the two-part two stream model using the balanced 5 class model weights we achieve the highest recorded accuracy on the 5 class version of B3SD. The accuracy of 74.7% outperforms the two-part two stream model using the full 5 class model weights by 6.4%. The result of both two part models show, that being able to divide the task into sub-components can improve accuracy.

Chapter 10

Discussion

In this chapter we will discuss topics specific to the results of our experiments on both UCF101 and B3SD. We end the chapter with a brief discussion of understanding how networks learn.

10.1 UCF101 Discussion

Based on preliminary experiments, we train two individual single stream models on the dataset with the optimal parameter setting. We analyze our results, by initially examining the class variance within the dataset. As multiple classes may have similar motion patterns or spatial structure, the network might incorrectly distinguish between them. The UCF101 dataset contains similar classes. To locate these, we plot the network predictions against the true labels in a confusion matrix. A confusion matrix places all classes along both axes, and colors the corresponding $N \times N$ grid where a prediction occurs. A lighter grid color corresponds to many predictions. If all predictions of the test set are classified correctly, only the diagonal of the confusion matrix would be colored. This allows us to quickly identify dominant grid positions not on the diagonal. Such positions reveal that a particular class is incorrectly classified as a different class. In figure 10.1 most of the classifications lie on the diagonal, indicating that the network has classified well. However it also shows certain outliers not at the diagonal. We found that many outlier classes contain a dominant green background. This confuses the spatial network. This happens since only a single RGB-frame is used to classify the action of an entire video. Without the explicit temporal information, the background influences the classification more than wanted, which results in wrong classifications. Several more confusion matrices with various model parameters can be found in Appendix C.

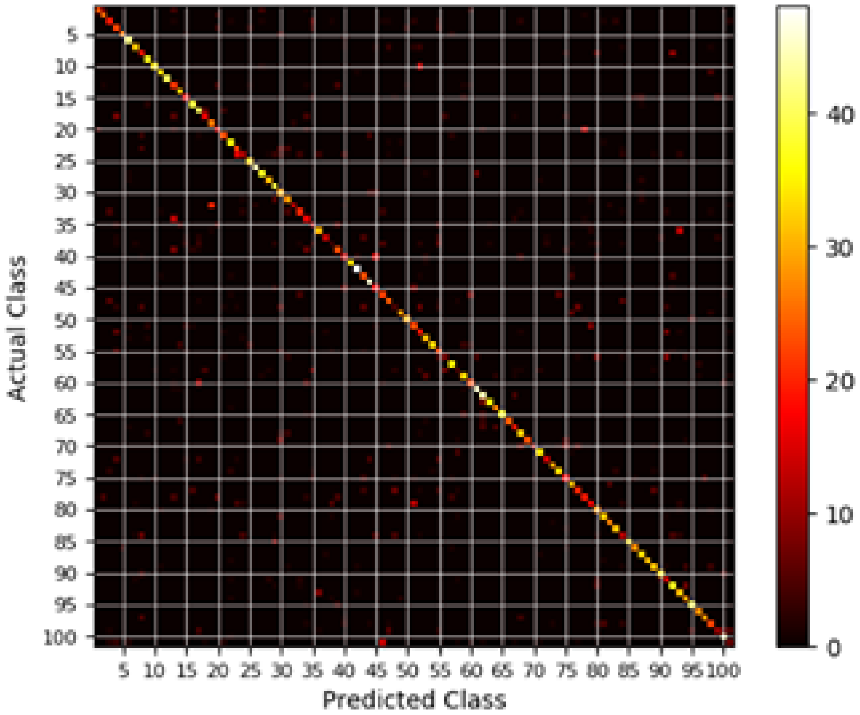


FIGURE 10.1: UCF101 confusion matrix of a preliminary test experiment for the spatial network. We see that a reasonable amount of guesses are correct. However some classes, are incorrectly predicted due to a green background.

Class similarity may not be the only problem with the UCF101 dataset. Intra-class variance poses a big challenge, when the number of training examples is small. Too much variance within a single class, result in not learning the important features. This is to some extent a problem when using the UCF101 dataset, since the authors have explicitly placed high variability within some classes, making the dataset more realistic, but at the same time more challenging. This is one of the reasons for applying data augmentation for training. Lastly, we also investigate the number of videos within each class. If most of the data is labelled to a few classes, then they would overrepresent the dataset. This is, however, not a problem in the UCF101 dataset as shown in figure 10.2.

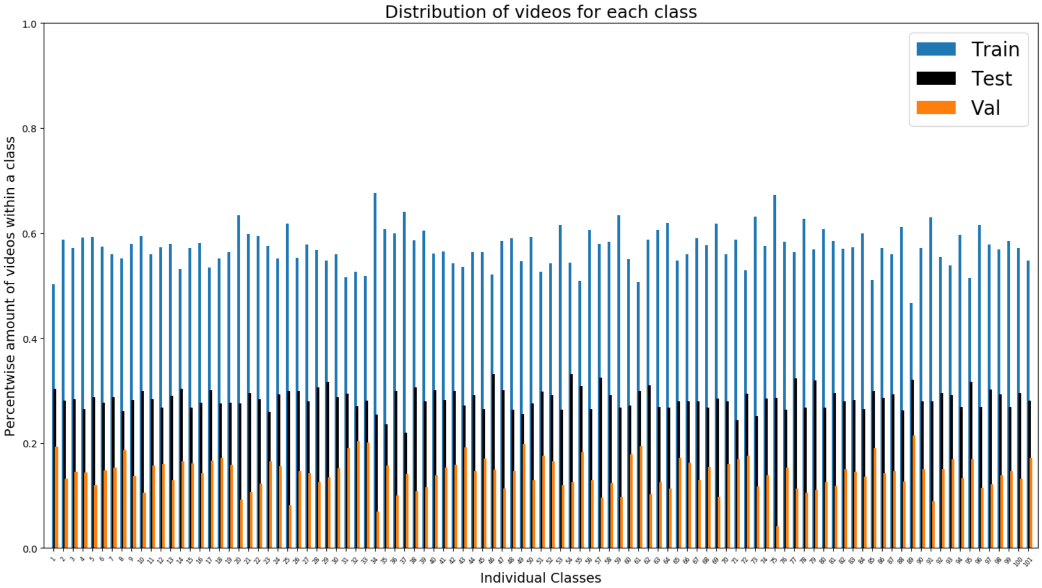


FIGURE 10.2: The train/val/test distribution of classes on the 1st split of the UCF101 dataset. As expected we see a reasonable distribution of classes within each set.

Another way of examining why incorrect classifications occur, relies on the information of the feature maps of the model. They can be used to locate whether different classes produce similar responses. Early layers identify basic shapes, while later layers operate at a coarser scale due to downsampling in pooling and Convolution Layers. Critical information separating the classes may be lost in this process. It is, however, not obvious in our case to understand the information contained in the feature maps, because of the model complexity. By visualizing a small subset of the 2048 feature maps, we can gain insight into some of the features learned. However, unless we examine all 2048 feature maps, we can not conclude what combination of features determine the class.

As part of our preliminary experiments on the Optical Flow frames of the UCF101 dataset, we examined the influence of using consecutive frames, referred to as having a temporal stride = 1, as opposed to using a varying temporal stride. However, our model unexpectedly produced inferior results. We believe the results should improve using a varying temporal stride as explained by the following example. Consider the task of distinguishing between a man walking and a man running using consecutive Optical Flow frames. Since the actions look rather similar if represented as a single RGB-frame, the easiest way to distinguish the two using both spatial and temporal data is to classify the increased amount of movement between Optical Flow frames. However, if we trained the model using only videos with a high frame rate, and we wanted to classify a video with a lower frame rate showing a man walking, the Optical Flow might indicate he was running because of the increase movement between frames. Instead, the model should be trained such that the difference between a man walking and a man running is not only influenced by the increased movement in the Optical Flow frames, but also include how a single RGB-frame of a man running differs from that of a man walking. In summary, using temporal stride = 1 makes the model frame rate dependent. We later realized, that all videos in the UCF101 dataset used frame rates of either 25 or 29, making the frame rate dependency somewhat irrelevant to the classification. As a result, we believe that the decrease in accuracy could instead be caused by the variance of each class increasing too much, making it more difficult to distinguish between the classes altogether.

The use of pre-computed Optical Flow frames are useful in order to decrease training time, but also limits their power to precisely represent the movement of the videos. If we use consecutive frames to pre-compute Optical Flow, we reduce the models ability to be frame rate independent during training. Even though we are still able to somewhat modulate different frame rates by varying the temporal stride, the actual information stored in each Optical Flow frame only represents the movement between consecutive frames from the input video. This could be interpreted as ”lag” between frames. Instead of the full movement being captured, only a fragment is captured. A way to avoid losing the information between frames with a varying temporal stride is to instead use Optical Flow as a data augmentation step during the training of each batch. However, as Optical Flow is expensive to compute, this greatly increases the training time.

When training both streams, we initially used ImageNet weight initializations. As our preliminary experiment on the temporal stream using ImageNet weight initialization was far from successful, we decided to use gray-scaled ImageNet weights instead. This was done because we expected that using the combination of all three color channels for a single flow frame, would represent the overall structure in the frame better than a single color channel on each flow frame. Instead, using a completely different initialization for the temporal stream with weights specifically designed for Optical Flow or motion patterns might improve the overall performance. In addition, the initialization of the bias parameter in ImageNet is set to 0. This might be a problem for the network, since it can lead to unwanted dead neurons at early training epochs, slowing convergence.

A major challenge of the UCF101 dataset is that videos are untrimmed and with different lengths. We do not know when the action occurs, and it is therefore plausible to use an unimportant part of a video. The problem is apparent for both spatial and temporal streams because the maximum number of frames used is 1 for the spatial stream and 11 for the temporal stream (resulting in 20 flows). 11 frames in a video with a frame rate of 25 frames per second is only half a second. Half a second might not be enough to distinguish actions. Additionally, consider brushing teeth and applying lipstick, two classes present in the UCF101 dataset with similar spatial appearance. If the frames selected involve the person picking up the object (toothbrush or lipstick) the

FIGURE 10.3: An example of an Optical Flow frame. The action of interest is found, but additional movement is captured, adding unwanted noise.

distinction of the two classes becomes harder when performing the actual action.

A different kind of problem when performing video action recognition, is how to tell precisely what the algorithm has learned during training. We can not know if a specific feature, unrelated to the actual action, is used to identify the class. As an example if the same actor is used in all videos of a class, we may end up classifying the person and not the action. This could occur even though the environment might be different or if data augmentation is applied. In UCF101, the same actor sometimes occur in multiple videos of the same class.

A practical limitation occurs when training on larger datasets. The training time increases as the number of examples grows. Training the two stream model in our case, required ≈10 hours for a single epoch. This limited our ability to continue experimenting with different hyper-parameters and input modalities. The settings optimal for the single stream model are not guaranteed to be optimal for the two stream model. We believe, that additional experiments with the two stream model could have increased the accuracy.
During our experiments, the data augmentation applied may be too aggressive, as it is possible to scale independently in both directions, with up to ±25%, before cropping and rescaling. For some augmentations, this makes the resulting output not realistic as the example in figure 10.4 shows. We should have used less aggressive data augmentation, to better represent the possible input space. We believe this could reduce overfitting.

The results of the two stream models show that overfitting is present. Furthermore, when comparing them to the single stream models, we only observe a marginal improvement with the additional implementation of MG and TIs. Additionally, our experiment using only late fusion indicate that the averaging of the information of

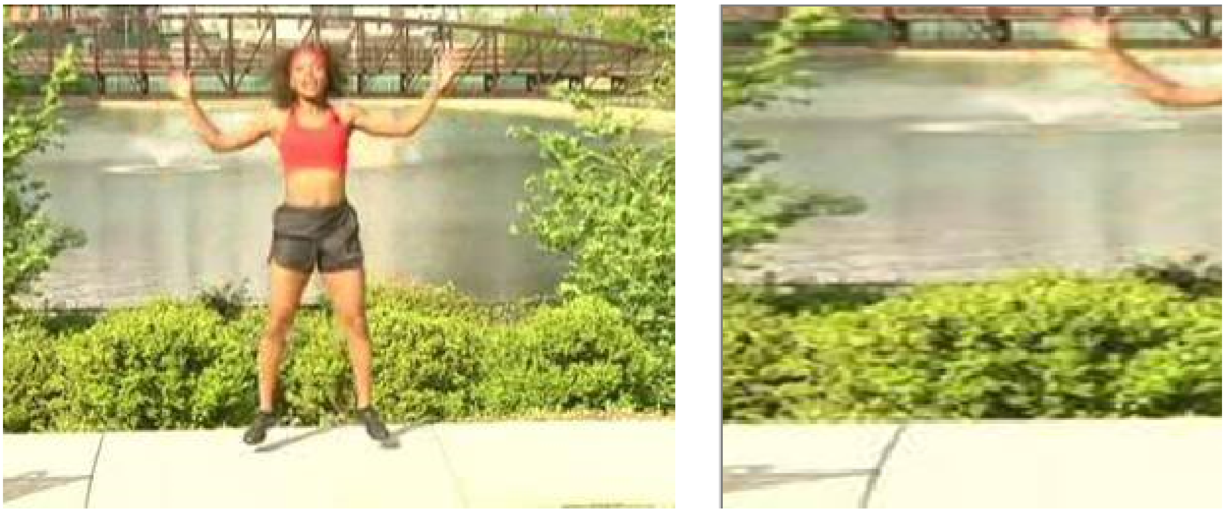


FIGURE 10.4: Example of an aggressive data augmentation. **Left:** The original frame. Note that the dimensionality is not 224 × 224 here. **Right:** The frame is zoomed inward and outward for x and y axis with 25% before randomly flipping, cropping and rescaling back.

the two streams is not complementary for all inputs, since the accuracy is below that of the single stream model with temporal information. In [1], various types of interaction between the two streams are examined. Among other things, they found that adding spatial information to the temporal stream reduced accuracy as opposed to adding temporal information to the spatial stream. Our experiments show that adding temporal information to the spatial stream earlier in the model with MG improve accuracy, supporting their findings. After further inspecting the temporal injections, we believe that the interaction between 3 consecutive feature maps is irrelevant compared to the subsequent full 1 × 1 convolution, where all feature maps interact. Furthermore, the addition of only 12 learnable parameters has limited effect in a network with more than 40 million.

10.2 B3SD Discussion

The results of the experiments on our own dataset show that classification of B3SD is a complex task. In this section we discuss some of the challenges with the dataset as well as possible ways to solve them.

One of our initial experiments on B3SD showed, that the use of Optical Flow (just 44.5% on 4 classes) did not obtain results as high as when using stacked gray frames or stacked difference frames. A possible cause is that the Optical Flow version used in [1] was unavailable, leading us to implement a different Optical Flow algorithm. Although we have examined the algorithm, finding the optimal combination of parameters is no trivial task. By using multiple Optical Flow calculations of B3SD, we might have been able to improve our performance. The movement pattern discrepancies in B3SD classes are more fine-grained than UCF101, and as a result, we might simply be better suited if we focus more on spatial information. Fast camera movement, such as panning or zooming, greatly reduces the flow estimation of actual objects moving in a scene [81]. As an example consider the optical flow in figure 10.3. As the action of interest is not moving independently in a video, we obtain a distorted view containing unwanted flow, which doesn't reflect the important motion that defines the action. In figure 10.5 below, we see the confusion matrices for the single stream spatial and temporal networks that achieved the highest test accuracy (58.6% and 58.1%, respectively) on the 4 class version of B3SD. Table 10.1 shows the number used to represent each class in the confusion matrices.

Number of Each Shot Type

Shot type	Number
Layup	0
Free Throw	1
2-Point Shot	2
3-Point Shot	3
No Shot	4

TABLE 10.1: The 5 classes of B3SD. The number used to represent each class in the confusion matrices.

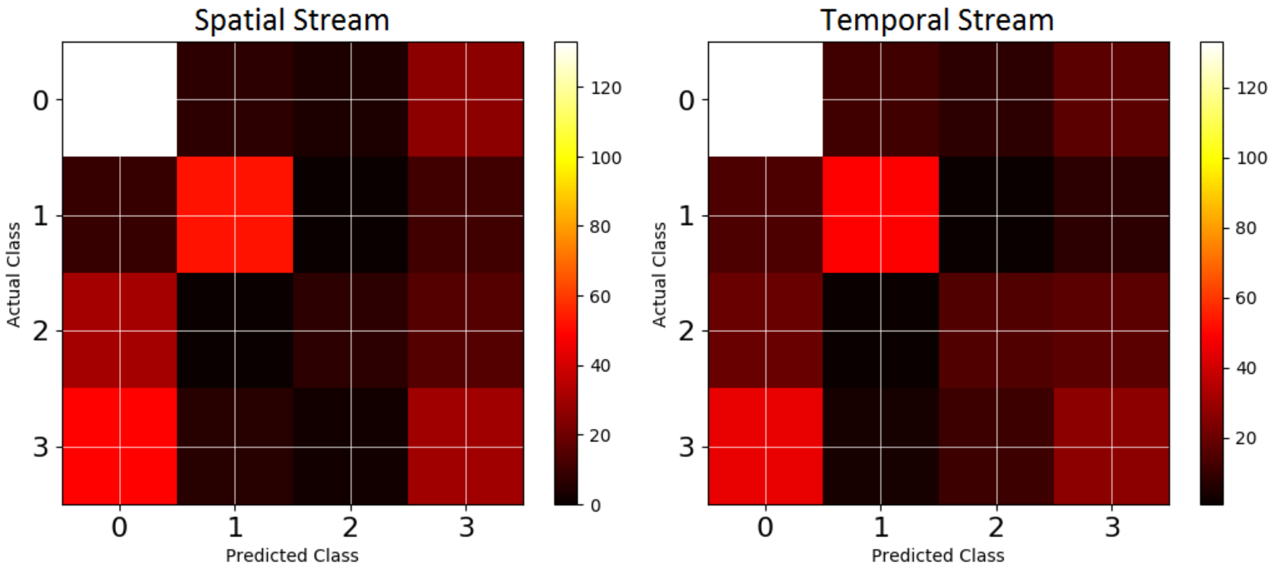


FIGURE 10.5: Confusion Matrices of the spatial and temporal network with 4 class version of the dataset

.

The matrices show that when discriminating between shots, both streams provide very similar results while still having many incorrect classifications. This could indicate that, even for the temporal stream, as we use a stack of gray-frames, the appearance information of the frames dominate the motion information that we presume they are able to capture. On the contrary, this perception of temporal information could also be used to infer that the application of Optical Flow or difference frames, despite being less accurate on its own, is relevant for capturing features where both spatial and temporal information is needed. In other words, we may only need the temporal information complementary to the spatial information. Using a model where the information of the appearance stream is weighted higher than the temporal stream could potentially provide better classification ability.

When investigating the results using difference frames in table 9.4, we see that using only 3 difference frames at the time of shot instead of 20 provide similar results (52,5% and 51,6%, respectively). Although the slight increase could be caused by randomness in the learning process, it could also be an indication that using 20 stacked difference frames adds too much information. By using 20 difference frames we effectively lower the impact of the exact frame of the time of shot to only be 5% of the information inserted, as opposed to 33% with 3 frames. Despite the assumption that actions are longer than 3 frames, the increased stack may include too much information, thereby reducing the variance between the classes. A possible solution to this could employ a weighting scheme increasing the influence of the frames closer to the time of shot. This way the most important frames have the biggest impact, while avoiding to discard information.

In figure 10.6 below, we see a comparison of the temporal networks with the 5 class models. Despite the accuracy being higher for the full 5 class dataset, the model has not learned to discriminate between classes and only predicts no shots. This is due to the full version having a very high frequency of no shot inputs. As everything is classified as a no shot, this results in a deceptively high test accuracy. This tells us that the weight scaling strategy supposed to handle an uneven class distribution is unsuccessful. This could be due to the high number of clips increasing the intra class variance of the no shot class, and as a result reduces the overall inter class variance. As we do not apply a high amount of data augmentation, this obstacle is hard to overcome.

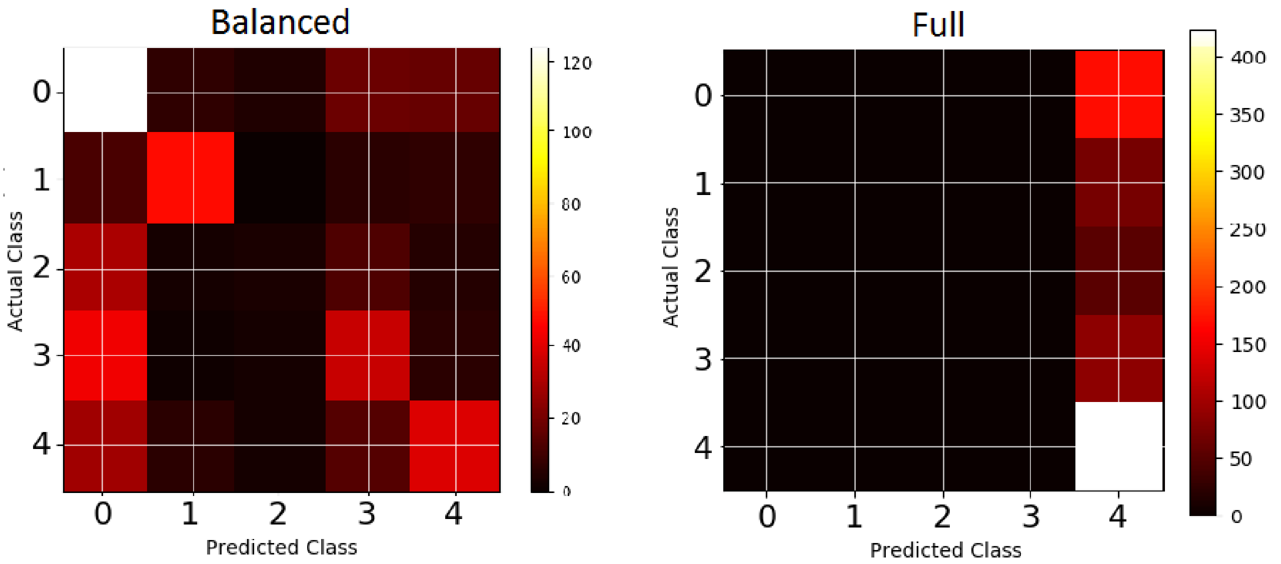


FIGURE 10.6: Confusion Matrix of the temporal network on the 5 class versions of B3SD.

For the balanced model we also see from the figure that, despite the overall performance being poor, it distinguishes well between free throws and other classes. This point is also apparent from figure 10.5 on the 4 class version.

Although the models are able to classify free throw shots at a higher accuracy than all the other classes, another limitation of our dataset revolve around the specific situation of a free throw attempt. As opposed to the other three shot types that occur during live action, the free throw is done in a dead-ball situation, where mostly everyone is standing still in very specific positions. This could introduce an increased number of incorrect classifications in the no-shot clips where a player is getting ready to shoot a free throw but doesn't actually do it within the clip. Everyone will be standing still and the distinction between the no-shot clips that vary greatly and the free throw clips that look similar, will tilt the classification towards the free throw class in presumably every scenario. A possible solution is to construct a separate network with two classes classifying free throw setup and free throw shot to use as a part of a classification pipeline on the clips initially classified as free throws. In addition, footage from professional games typically uses multiple cameras, which leads to sudden scene cuts, changing the view angle. This often occurs during free throws. We believe this makes the model prone to classify free throws based on the sudden change of appearance. The simple solution involves manually dropping such clips, which is undesirable as we wish to have as much data as possible. The matrices in figure 10.5 also show that the 2-point shot is a rarely predicted class. This could be a result of the class being the least attempted type of shot in a basketball game and therefore under represented in B3SD. In addition, 2-point shots typically look similar to either a 3-point shot where the only difference is the distance to the basket, or a layup, where the defense and the shooter are close to each other. Another shortcoming of B3SD is the limited number of validated videos. As a result, the variance between the authors' views of different classes increases. The authors could have different views of what is a layup and what is a 2-point shot since both award 2 points.

In many of our experiments, the model overfits to the training data, as seen by the training accuracy being close to or above 90% while the validation and test accuracies are closer to or below 50%. Our decision to always choose the same frame in each video as well as limiting data augmentation presumably contributes to the model overfitting.In addition the annotators have different views of exactly when to mark the time of shot, having a small shift of the input frame chosen during training might improve the models performance and reduce overfitting. Additionally, synthetic data could also be constructed in order to increase size of the dataset. However, synthetic data of high quality is hard to create and even more so for a fine-grained dataset like B3SD.

In basketball, the viewpoint of the camera is very crucial when performing action recognition, because there are many players on a small court. Often the player attempting a shot is occluded by other players. In these cases, the model attempts to classify based on other factors, such as the position and movement of the defense or the reaction of the audience.

Classifying an action based on the reaction of the surroundings complicates the task. With enough data, however, we believe a model should be able estimate if the action has occurred. The most plausible solution to this, however, probably involves using multiple cameras, either to infer depth between viewpoints using template matching or even better, avoid occlusion. A different solution involves tracking the ball. Since the size of the ball and the courts measurements are fixed, if we know the position and viewpoint of the camera, we should be able to find the balls trajectory. Assuming the basket is always visible, this opens up the possibility of backtracking, thereby predicting the shot type.

10.3 Examining How Networks Learn

In the previous sections we discussed a number of challenges in our work many of which involved model input. However, examining how networks learn in general is an ongoing area of research. Using alternative methods using visual aids, the machine learning community tries to better understand how networks learn in general. It is possible to visually represent the feature maps in NNs in an attempt to understand the features learned during training [98]. However, these attempts give limited insight as the data may be high-dimensional making it not only hard to visualize, but it is the author's decision to show what they consider important.

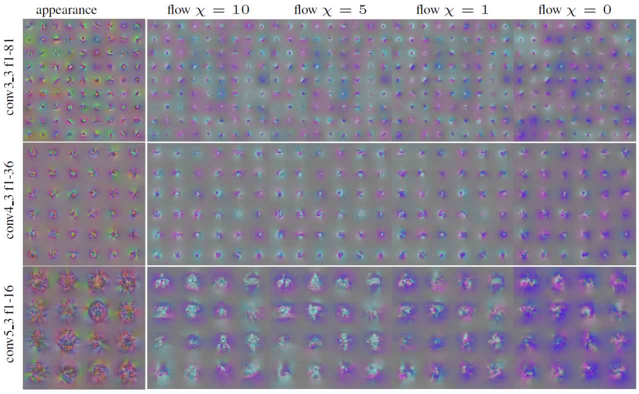


FIGURE 10.7: A visualization from [98]. Here anisotropic regularization is used to highlight activation responses in output feature maps of different Convolutional layers in a network. They model the activations of time. The original article presents this as clickable GIFs.

Understanding how to effectively train any type of NN to generalize well and learn optimally for a particular problem is one of the most sought answers of the field. Currently, much empirical research provides a starting point. Different problems require different combinations of settings, and so an optimal solution is often obtainable through mostly trial and error.

Chapter 11

Conclusion & Future Work

In this work we introduced a new dataset for video action recognition called Basketball Simple Shootings Statistics Dataset, in short B3SD. It consists of 5 classes, 4 of which are types of shots in basketball game footage and a 5th class without shots. Alongside B3SD, we presented a two stream model based on [1] with multiplicative gating and temporal injections, trying to solve the task of video action recognition in both UCF101 as well as multiple version of B3SD. Using 50-layer Residual Networks in various setups, preliminary results show that the use of both spatial and temporal information streams individually can be used for solving video action recognition. Although Optical Flow provides a good estimation of true motion, it might not be the best input modality for some video action recognition tasks. Optical Flow is expensive to compute during training, and if pre-computed introduces potential limitations. Additionally, our results show that using stacked gray-scaled images improved performance.

The classification accuracy on three of the four versions of B3SD increased when using the two stream model, the exception being the balanced 5 class version. Similarly, the classification accuracy using the two stream model on UCF101 improved the accuracy by $\approx 2\%$. Furthermore, the addition of Multiplicative Gating is beneficial as it improves on the accuracy of a simple late fusion strategy.

Lastly, our results show that splitting the B3SD classification task into two parts of first distinguishing between shots and no shots before selecting particular shot types improves test accuracy. Based on our results, we can conclude that the use of both temporal and spatial information for action recognition problems are valuable but hard to combine effectively.

A series of extensions to this work exist. The first extension we consider, is to extend the solution to include action localization. As B3SD also contains localization data in the form of bounding boxes for all registered shots, experimenting with an existing action localization algorithm is a natural next step. Furthermore, B3SD has the capability of being used for continuous action recognition and real time action localization. Both of these tasks are increasingly difficult. Real time action localization using our model is currently not feasible in a standard setting. Variants such as YOLO [3] and SSD [38] only use spatial information, and so a clever extension adding temporal information may boost performance. Further work refining B3SD is needed, due to its limitations. Increasing the size of the dataset as well as multiple validations of all clips, would increase the quality significantly. Providing additional versions of B3SD for new recognition tasks would be highly valuable for future work. As an example, we do not currently have a version distinguishing scores and misses. Solving the tasks of action recognition and localization may require more information than both spatial and temporal. A third stream with audio information could be added to improve learning. At the start of this project we decided to implement the model of [1] as it achieved the highest UCF101 test accuracy we could find. Since then, we discovered other models with a higher accuracy. Joao Carreira and Andrew Zisserman obtained a test accuracy of 98%, using transfer learning of a large video dataset [99], outperforming the 94.9% achieved by [1].

When a person does basketball statistics manually, there are often a couple of mistakes along the way due to the complex and fast-paced nature of the game. A few mistakes here and there is an accepted part of basketball and our results on B3SD provide promising classification accuracy on many of the different settings.

Acknowledgements

We would like to thank Kim Steenstrup Pedersen for his support and guidance throughout this project.

Bibliography

[1] Christoph Feichtenhofer, Axel Pinz, and Richard P Wildes. Spatiotemporal multiplier networks for video action recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[2] Dictionary.com. Definition of the word: Action. http://www.dictionary.com/browse/action?s=t, 2018. Visited: 2018-06-29.

[3] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[5] I. Laptev and T. Lindeberg. Space-time interest points. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 432–439 vol.1, Oct 2003.

[6] Timor Kadir and Michael Brady. Saliency, scale and image description. *Int. J. Comput. Vision*, 45(2):83–105, November 2001.

[7] A. Oikonomopoulos, I. Patras, and M. Pantic. Spatiotemporal salient points for visual recognition of human actions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(3):710–719, June 2005.

[8] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th ACM International Conference on Multimedia*, MM '07, pages 357–360, New York, NY, USA, 2007. ACM.

[9] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[11] Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part II*, ECCV'06, pages 428–441, Berlin, Heidelberg, 2006. Springer-Verlag.

[12] H. Wang, A. Kläser, C. Schmid, and C. L. Liu. Action recognition by dense trajectories. In *CVPR 2011*, pages 3169–3176, June 2011.

[13] H. Wang and C. Schmid. Action recognition with improved trajectories. In *2013 IEEE International Conference on Computer Vision*, pages 3551–3558, Dec 2013.

[14] J. Sivic and A. Zisserman. Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2, Oct 2003.

[15] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, Dec 2007.

[16] Jianchao Yang, Kai Yu, Yihong Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1794–1801, June 2009.

[17] Kai Yu, Tong Zhang, and Yihong Gong. Nonlinear learning using local coordinate coding. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2223–2231. Curran Associates, Inc., 2009.

[18] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3360–3367, June 2010.

[19] H. Jhuang, T. Serre, L. Wolf, and T. Poggio. A biologically inspired system for action recognition. In *International Conference on Computer Vision (ICCV)*, 2007.

[20] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 143–156, Berlin, Heidelberg, 2010. Springer-Verlag.

[21] Heng Wang. Lear-inria submission for the thumos workshop. 2013.

[22] Khurram Soomro and Mubarak Shah. Unsupervised action discovery and localization in videos. In *ICCV*, pages 696–705. IEEE Computer Society, 2017.

[23] Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[24] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014.

[25] B. Ni, X. Yang, and S. Gao. Progressively parsing interactional objects for fine grained action detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1020–1028, June 2016.

[26] B. Fernando, P. Anderson, M. Hutter, and S. Gould. Discriminative hierarchical rank pooling for activity recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1924–1932, June 2016.

[27] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. 03 2015.

[28] Limin Wang, Luke L Daemen, Yuanjun Xiong, and Yu Qi Qiao. Cuhksiat submission for thumos15 action recognition challenge. 2015.

[29] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[30] Zhenyang Li, Kirill Gavrilyuk, Efstratios Gavves, Mihir Jain, and Cees G.M. Snoek. Videolstm convolves, attends and flows for action recognition. *Comput. Vis. Image Underst.*, 166(C):41–50, January 2018.

[31] W. Zhu, J. Hu, G. Sun, X. Cao, and Y. Qiao. A key volume mining deep framework for action recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1991–1999, June 2016.

[32] Dan Oneata, Jerome Revaud, Jakob Verbeek, and Cordelia Schmid. Spatio-temporal object detection proposals. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 737–752, Cham, 2014. Springer International Publishing.

[33] H. Zhu, R. Vial, and S. Lu. Tornado: A spatio-temporal convolutional regression network for video action proposal. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5814–5822, Oct 2017.

[34] Nannan Li, Dan Xu, Zhenqiang Ying, Zhihao Li, and Ge Li. Searching action proposals via spatial actionness estimation and temporal path inference and tracking. In *ACCV*, 2016.

[35] J. C. van Gemert, M. Jain, E. Gati, and C. G. M. Snoek. Apt: Action localization proposals from dense trajectories. In *British Machine Vision Conference*, 2015.

[36] G. Yu and J. Yuan. Fast action proposals for human action detection and search. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1302–1311, June 2015.

[37] Unaiza Ahsan, Chen Sun, and Irfan A. Essa. Discrimnet: Semi-supervised action recognition from videos using generative adversarial networks. *CoRR*, abs/1801.07230, 2018.

[38] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[39] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.

[40] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[42] Yann LeCun, Fu-Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

[43] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow - large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[44] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[45] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[46] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/index.html.

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

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[49] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui, and J. Collomosse. Everything you wanted to know about deep learning for computer vision but were afraid to ask. In *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, pages 17–41, Oct 2017.

[50] Soo-Min Kang and Richard P. Wildes. Review of action recognition and detection methods. *CoRR*, abs/1610.06906, 2016.

[51] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *Proceedings of the International Workshop on Artificial Neural Networks: From Natural to Artificial Neural Computation*, IWANN '96, pages 195–201, London, UK, UK, 1995. Springer-Verlag.

[52] A Vehbi Olgac and Bekir Karlik. Performance analysis of various activation functions in generalized mlp architectures of neural networks. 1:111–122, 02 2011.

[53] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[54] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.

[55] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[56] Brilliant.org. Backpropagation. https://brilliant.org/wiki/backpropagation/, 2018. Visited: 2018-03-16.

[57] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[58] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. course notes. http://cs231n.github.io/, 2017.

[59] Geoffrey Hinton. Neural networks for machine learning lecture 6d a separate, adaptive learning rate for each connection. University Lecture, 20.

[60] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[61] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[62] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, December 2014.

[63] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural net chips and automatic learning. In F. Fogelman, J. Herault, and Y. Burnod, editors, *Neurocomputing, Algorithms, Architectures and Applications*, Les Arcs, France, 1989. Springer.

[64] J.C. Kao. Convolutional neural networks. University Lecture, 2018.

[65] Martin Thoma. Analysis and optimization of convolutional neural network architectures. Master's thesis, Department of Computer Science Institute for Anthropomatics and FZI Research Center for Information Technology, 2017.

[66] tensorflow team. Api r1.8 neural network - convolution. https://www.tensorflow.org/api_guides/python/nn#Convolution, 2018. Visited: 2018-05-16.

[67] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, 2016.

[68] Adit Deshpande. A beginner's guide to understanding convolutional neural networks part 2. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/, 2016. Visited: 2018-05-15.

[69] Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.

[70] Leonardo Araujo dos Santos. Pooling layer. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pooling_layer.html, 2017. Visited: 2018-04-02.

[71] abora (username). Backprop through max-pooling layers? https://datascience.stackexchange.com/questions/11699/backprop-through-max-pooling-layers?noredirect=1&lq=1, 2016. Visited: 2018-04-02.

[72] Jefkine Kafunah. Backpropagation in convolutional neural networks. http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/, 2016. visited: 2018-03-15.

[73] Sujit Rai. Forward and backpropagation in convolutional neural network. https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa
2017. visited: 2018-03-15.

[74] Druhv Batra. Lecture 5: Backpropagation in convnets. https://www.youtube.com/watch?v=BvrWiL2fd0M, 2015. visited: 2018-04-20.

[75] Ian J. Goodfellow. Technical report: Multidimensional, downsampled convolution for autoencoders. Technical report, Université de Montréal, 2010.

[76] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *CoRR*, abs/1712.09913, 2017.

[77] Lei Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3084–3092, USA, 2013. Curran Associates Inc.

[78] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456, 2015.

[79] Yan Xu, Ran Jia, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. Improved relation classification by deep recurrent neural networks with data augmentation. *CoRR*, abs/1601.03651, 2016.

[80] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[81] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg, 2003. Springer-Verlag.

[82] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ICPR '04, pages 32–36, Washington, DC, USA, 2004. IEEE Computer Society.

[83] I. Laptev and T. Lindeberg. Velocity adaptation of space-time interest points. In *Proc. Int. Conf. Pattern Recognition (ICPR'04)*, Cambridge, U.K, 2004.

[84] I. Laptev. *Local Spatio-Temporal Image Features for Motion Interpretation*. PhD thesis, Department of Numerical Analysis and Computer Science (NADA), KTH, 2004.

[85] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1395–1402 Vol. 2, Oct 2005.

[86] Yan Ke, Rahul Sukthankar, and Martial Hebert. Event detection in crowded videos. *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007.

[87] M. Marszalek, I. Laptev, and C. Schmid. Actions in context. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2929–2936, June 2009.

[88] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, June 2014.

[89] L. Cao, Z. Liu, and T. S. Huang. Cross-dataset action detection. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1998–2005, June 2010.

[90] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: A large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563, Nov 2011.

[91] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012.

[92] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–970, June 2015.

[93] Marcus Rohrbach, Anna Rohrbach, Michaela Regneri, Sikandar Amin, Mykhaylo Andriluka, Manfred Pinkal, and Bernt Schiele. Recognizing fine-grained and composite activities using hand-centric features and script data. *CoRR*, abs/1502.06648, 2015.

[94] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[95] Juan Carlos Niebles, Chih-Wei Chen, and Li Fei-Fei. Modeling temporal structure of decomposable motion segments for activity classification. In *Proceedings of the 11th European Conference on Computer Vision: Part II*, ECCV'10, pages 392–405, Berlin, Heidelberg, 2010. Springer-Verlag.

[96] François Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

[97] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[98] C. Feichtenhofer, A. Pinz, R. P. Wildes, and A. Zisserman. What have we learned from deep representations for action recognition? *ArXiv e-prints*, January 2018.

[99] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750, 2017.

[100] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 25–36, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

Appendices

Appendix A

Individual Dataset Categories

This page is intentionally left blank

Name	Classes	Class Names
KTH	6	walking, jogging, running, boxing, hand waving, hand, clapping
Weizmann	10	running, walking, jumping-jack, jump-forward, jump-in-place, galloping-sideways, waving-one-hand, bending
CMU-Crowds	5	pick up, one hand wave, two hand wave, push button,jumping jack
Hollywood-2	12	answer phone, drive car, eat, fight person, get out of car,handshake, hug person, kiss, run, sit down, sit up, stand up
MSRII	3	box, wave, clap
Olympics	16	high jump, long jump, triple jump, pole vault, discus throw, hammer throw, javelin throw, shot put, basketball layup, bowling, tennis serve, diving platform, diving springboard, weightlifting snatch, weightlifting clean jerk, vault gymnastics
HMDB51	51	brush hair, cartwheel, catch, chew, clap, climb, climb stairs, dive, draw sword, dribble, drink, eat, fall floor, fencing, shoot bow, shoot gun, flic flac, golf, hand stand, hit, hug, jump, kick, stand, kick ball, kiss, laugh, pick pour, pullup, punch, push, pushup, ride bike, ride horse, run, shake hands, shoot ball, shoot bow, shoot gun, sit, situp, smile, smoke, somersault, swing baseball, sword exercise, sword, talk, throw, turn, walk, wave
UCF101	101	apply eye makeup, apply lipstick, archery, baby crawling, balance beam, band marching, baseball pitch, basketball shooting, basketball dunk, bench press, biking, billiards shot, blow dry hair, blowing candles, body weight squats, bowling, boxing punching bag, boxing speed bag, breaststroke, brushing teeth, clean and jerk, cliff diving, cricket bowling, cricket shot, cutting in kitchen, diving, drumming, fencing, field hockey penalty, floor gymnastics, frisbee catch, front crawl, golf swing, haircut, hammer throw, hammering, handstand pushups, handstand walking, head massage, high jump, horse race, horse riding, hula Hoop, ice dancing, javelin throw, juggling balls, jump rope, jumping jack, kayaking, knitting, long jump, lunges, military parade, mixing batter, mopping floor, nunchucks, parallel bars, pizza tossing, playing guitar, playing piano, playing tabla, playing violin, playing cello, playing daf, playing dhol, playing flute, playing sitar, pole vault, pommel horse, pull ups, punch, push ups, rafting, rock climbing indoor, rope climbing, rowing, salsa spins, shaving beard, shot put, skate boarding, skiing, skijet, sky diving, soccer juggling, soccer penalty, still rings, sumo wrestling, surfing, swing, table tennis shot, tai chi, tennis swing, throw discus, trampoline jumping, typing, uneven bars, volleyball spiking, walking with dog, wall push-ups, writing on board, yo-yo
MPII-Cooking 2	65	add, arrange, change temperature, chop, clean, close, cut apart, cut dice, cut off ends, cut out inside, cut stripes, cut, dry, enter, fill, gather, grate, hang, mix, move, open close, open egg, open tin, open, package, peel, plug, pour, pull apart, pull up, pull, puree, purge, push down, put in, put lid, put on, read, remove from package, rip open, scratch off, screw close, screw open, shake, shape, slice, smell, spice, spread, squeeze, stamp, stir, strew, take apart, take lid, take out, tap, taste, test temperature, throw in garbage, turn off, turn on, turn over, unplug, wash, whip, wring out

TABLE A.1: A table with the classes in a number of noteworthy datasets, except for the 1-Million Dataset and ActivityNet.

Appendix B

Creating B3SD

The relevant code for recreating the dataset is contained in a collection of python files. The B3SD_main.py is the entry from which all other files are used. The data files containing video information and annotations are located in videos.csv and validation.csv.
The dataset is created in 5 main steps.

- Create folders
- Download data from YouTube
- Crop videos
- Create frames from cropped videos
- Create flows from frames

First, all available full length videos are downloaded. If a video is already downloaded (at the source destination provided), it will not be downloaded again.
Next we crop all shot occurrences recorded into 4 second clips containing a single action (while still keeping the original videos). The actions' annotated timestamp occurs exactly 1 second into the video. This way one can later find the exact frame corresponding to the time of shot. The cropped videos are created using the open source program ffmpeg easily installed with pip package manager. Creating a video with ffmpeg requires python starting a separate sub-process to handle the request. If a clip already exists ffmpeg will ignore the new request, keep the old clip and continue.
After cropping, each 4 second video is converted to a sequence of .png RBG images each with a unique name

of statId_frameNumber.png. We do not convert videos to .jpg format due to compression loss.

Finally, for all consecutive pairs of frames for all cropped videos, we calculate Optical Flow in each direction and store it as gray-scale images in .png format. The Optical Flow is estimated using Gunnar Farneback's algorithm, whereas [23] uses Thomas Brox's Optical Flow [100]. This is because Gunnar Farneback's algorithm is easily available for the Python OpenCV module used in the scripts. For a collection of 100 RBG frames, we would obtain 99 vertical (u) and horizontal (v) flow images. Each step can be run independently by simply not calling the main module, but instead calling the actual functions individually. The resulting folder structure will look the following

```
Dataset/
    - videos.csv
    - validation.csv
    - B3SD_main.py
    - B3SD_utils.py
    - B3SD_download_videos.py
    - B3SD_make_clips.py
    - B3SD_make_frames.py
    - B3SD_make_flows.py
    - videos/
        - clips/
            - shotType_statId.mp4
        - original/
            - <url>.mp4
    - frames/
        - shotType_statId/
            - shotType_statId_frameNumber.png
    - flows/
          - u/
             - shotType_statId/
                - shotType_statId_frameNumber_u.png
          - v/
             - shotType_statId/
                - shotType_statId_frameNumber_v.png
```

Appendix C

Confusion Matrices for Class Predictions

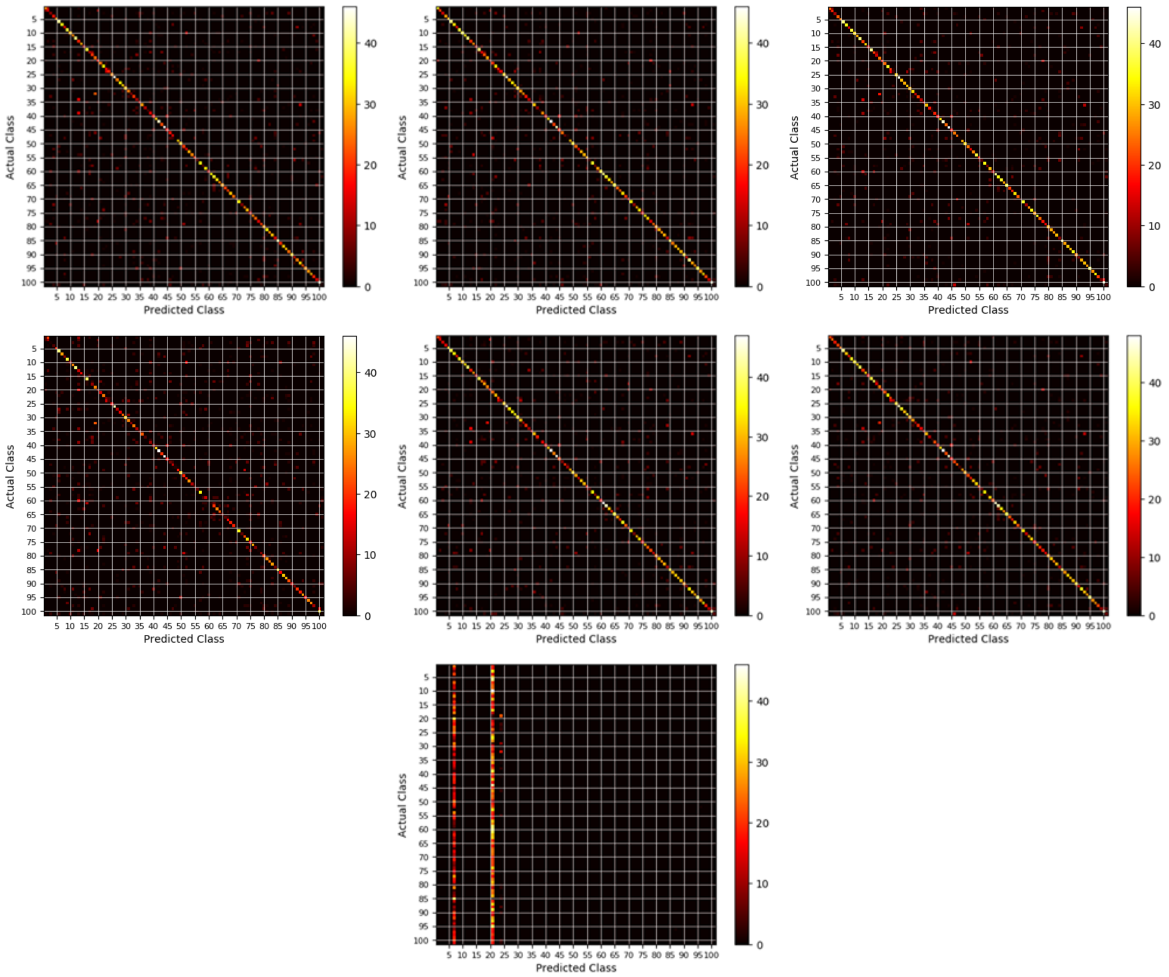


FIGURE C.1: Confusion matrices for various single stream spatial experiments. From top left to bottom right: Weight Initializtion: ImageNet, Data Augmentation: False, Batch Size: 256, Batch size: 64, Dropout Ratio: 0.5, Dropout Ratio: 0.2, Activation Function: Leaky ReLU.

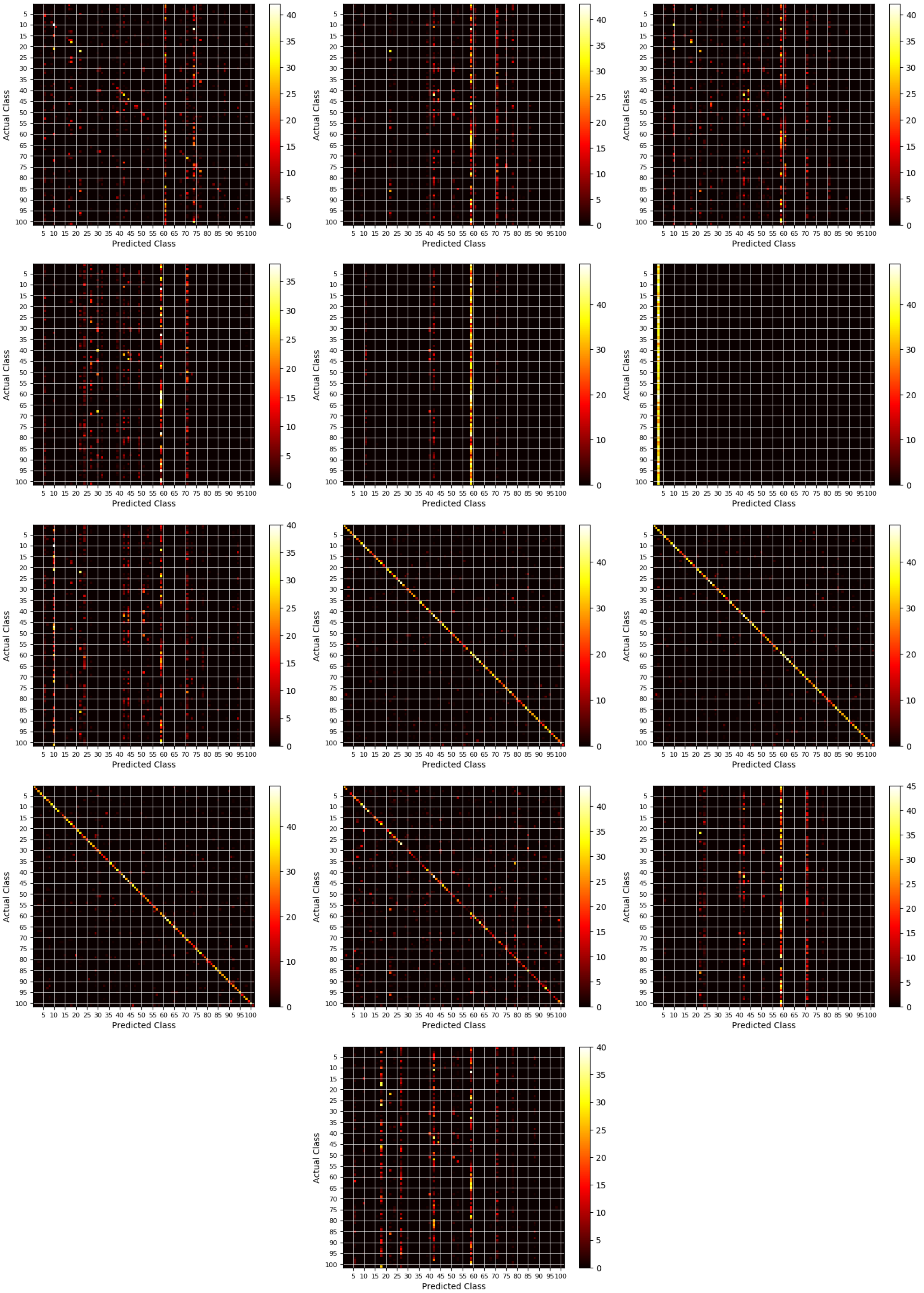


FIGURE C.2: Confusion matrices for various single stream temporal experiments. From top left, to bottom right: Weight Initialization: Random weights, Weight Initialization: ImageNet, Weight Initialization: Gray-Scaled ImageNet, Data Augmentation: False, Flow Stack: uvuv, Activation Function: Leaky ReLU, Optimizer: ADAM, Dropout Ratio: 0 Dropout Ratio: 0.2, Dropout Ratio: 0.5, Flow Jumps: True, Batch Size: 64, Batch Size: 32

Appendix D

Loss Graphs for all Preliminary Experiments

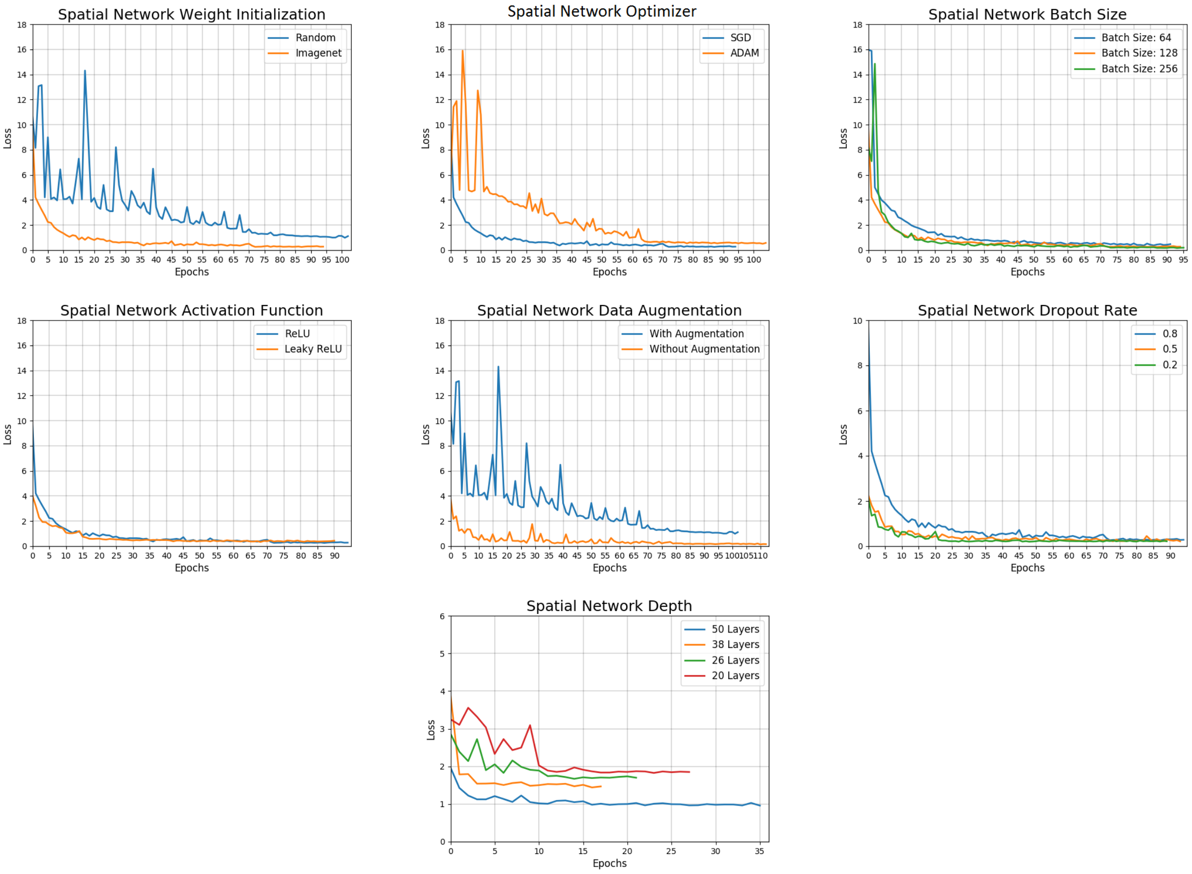


FIGURE D.1: Different loss graphs from all preliminary experiments on the single stream spatial model.

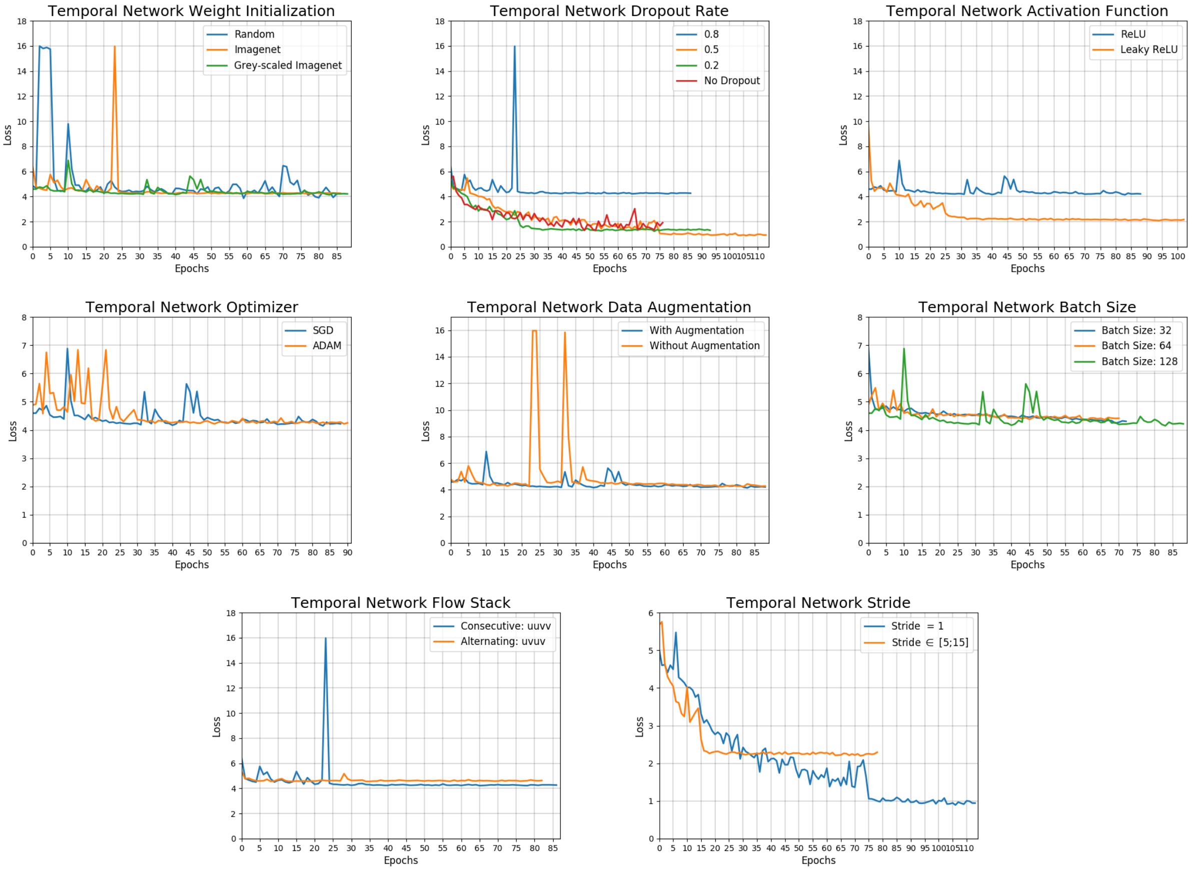


FIGURE D.2: Different loss graphs from all preliminary experiments on the single stream temporal model.