Video classification using deep learning

by

Gregory Newman

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Applied Mathematics) in the Faculty of Science at Stellenbosch University

Supervisor: Prof. W.H. Brink
Co-supervisor: Prof. B.M. Herbst

March 2020

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2020

Abstract

To help analyse, classify, and monitor video data we need scalable algorithms that can handle video sequences of various lengths. Existing approaches tend to be both computationally expensive and restricted to classifying sequences of a fixed length, making them ill-suited for real-world use.

For video classification we explore using convolutional neural networks to learn the spatial features relevant to each frame of a video, and several transfer learning approaches to leverage the InceptionV3 architecture with weights pre-trained on ImageNet. With Grad-CAM we show that CNN models alone primarily rely on detecting class specific objects within images, and perform poorly on classes that have similar spatial features to other classes.

To learn the temporal features of a video and to accommodate variable length sequences, we train LSTM and GRU networks. We show that without down-sampling the frames the parameter space of the networks explodes, quickly becoming computationally infeasible to train over, but that downsampling techniques cause too much information loss. We also find comparable performance between the two types of recurrent networks, despite the GRU network having fewer parameters.

We go on to propose an architecture that uses InceptionV3, with pretrained weights, to learn representations of the frames to be used when training a GRU network. After experimenting with different transfer learning approaches we show that we can achieve a top-5 classification accuracy of 91.8% on the UCF-101 test set, which is 6.2% less than the state-of-the-art while having half as many parameters and an architecture that can accommodate variable length inputs.

Uittreksel

Om die analise, klassifisering en monitering van video's met veranderlike lengtes te verbeter, het ons algoritmes nodig wat kan skaleer. Bestaande benaderings is tipies berekeningsintensief en beperk tot die klassifisering van video's van vaste lengtes, wat hulle ongeskik maak vir gebruik in die regte wêreld.

Ons ondersoek die gebruik van konvolusionele neurale netwerke vir die klassifisering van video's, om ruimtelike kenmerke van elke videoraam te leer. Ons kyk ook na verskeie benaderings van oordragsleer, om voordeel te trek uit die InceptionV3-argitektuur se gewigte wat vooraf op ImageNet afgerig is. Ons gebruik Grad-CAM om te wys dat konvolusionele modelle op hul eie hoofsaaklik op die opsporing van klas-spesifieke voorwerpe in beelde fokus, en sleg vaar op klasse waar die ruimtelike kenmerke soortgelyk is aan dié van ander klasse.

LSTM en GRU netwerke word afgerig om tyd-afhanklike kenmerke te leer, en om die veranderlike lengtes van die video's te akkommodeer. Ons wys dat sonder om die prente te reduseer, ontplof die parameter-ruimte van die netwerke, en maak dat praktiese afrigting vinnig onmoonlik word. Die reduksie-tegnieke veroorsaak wel te veel dataverlies. Ons vind vergelykbare prestasies tussen die twee tipes terugkerende netwerke, ten spyte van die feit dat die GRU netwerk minder parameters het.

Ons stel dan ook 'n argitektuur voor wat die InceptionV3 met vooraf-afgerigte gewigte gebruik om voorstellings van die rame te leer, en dan daardie voorstellings gebruik om die GRU netwerk af te rig. Eksperimentering met verskillende oordragsleer-tegnieke wys dat ons 'n top-5 akkuraatheid van 91.8% op die UCF-101 toetsstel kan behaal. Hierdie akkuraatheid is 6.2% minder as die huidige beste metode, maar benodig omtrent die helfte soveel parameters en kan video's van verandelike lengtes hanteer.

Acknowledgements

I would like to thank my supervisor, Willie Brink, for the endless patience, encouragement and willingness to help out, and the motivating conversations. I would also like to thank my co-supervisor Ben Herbst for the guidance, inspirational chats, and nudges to always seek a deeper understanding.

Contents

Chapter 1

Introduction

Videos have become an integral part of our daily lives; from sharing experiences on social media to observing scientific discoveries on Mars through a robot. It is becoming increasingly easier to both consume and produce videos, and with video camera technology becoming better and cheaper we can only expect this trend to continue. Social media and content streaming platforms like Facebook, YouTube, Twitch and Netflix are making it simpler to upload, share, archive, and consume videos. There is simply too much data for humans alone to analyse and discern. To better enable ourselves, it is crucial to develop scalable automated algorithms to assist with the analysis and classification of videos in order to benefit from this data as well as make its consumption safe.

Enabling accurate video classification can help analysis in numerous fields where video capture is the primary source of information; such as being able to assist doctors and nurses out in the field with echocardiography. Classification of cardiovascular diseases could help health professionals in developing countries administer low cost interventions and treatment plans to patients who need it most (Carapetis, 2007).

Automatic video classification can help with the organisation of the data being generated (Wang and Nandan Parameswaran, 2004). With better organisation, new task-specific datasets can be created more readily.

Automatic monitoring of the content that is being shared for sensitive material is also something that video classification can help with. Crimes shared on platforms like Facebook may achieve millions of views before a moderator is able to remove them (Martin, 2017). Algorithms could help catch these incidents before they reach a wide audience.

1.1 What is a video

To understand what a video is, we first need to understand what a digital image is. An image is essentially a 3-dimensional matrix structured with a

height, width, and number of channels. Each (row, column) coordinate contains a vector, called a pixel, that represents information about the image at that location. Pixels most commonly contain information about red, green, and blue intensity, to produce what are called RGB images, although other standards do exist depending on the use case. Each value in the pixel is typically represented as an unsigned 8-bit integer between 0 and 255. Individual pixels do not mean much to the human eye, but once arranged spatially they collectively reveal the scene (as with Figure 1.1).

Figure 1.1: An example of a digital image. The pixels are arranged to show a surfer riding a wave.

Videos are essentially temporally ordered sequence of images, where each image is called a frame. Now, instead of having three dimensions we have four; the number of frames, the height and width of each frame and, the number of channels for each pixel. A visual representation of a video can be seen in Figure 1.2 below. When we (humans) watch a video, each frame is displayed for a short amount of time, inversely proportional to what is known as the frame rate. Our brains fill in the blanks to create the illusion that the objects in the video are moving in a continuous space, but in reality it is discrete.

Video classification requires models that can handle high-dimensional data. The information is both spatially structured within each frame, and temporally ordered throughout the sequence of frames. The spatial structure in the arrangement of pixels gives an image meaning, for example a violin or football in a frame. This information is specific to each frame, and does not require any of the other frames in the sequence to be identified. The temporal information is how those spatial features change over time, from one frame to the next. It

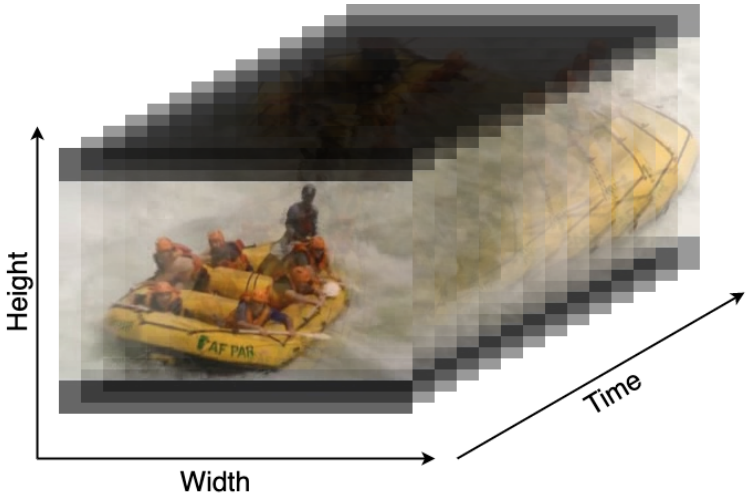
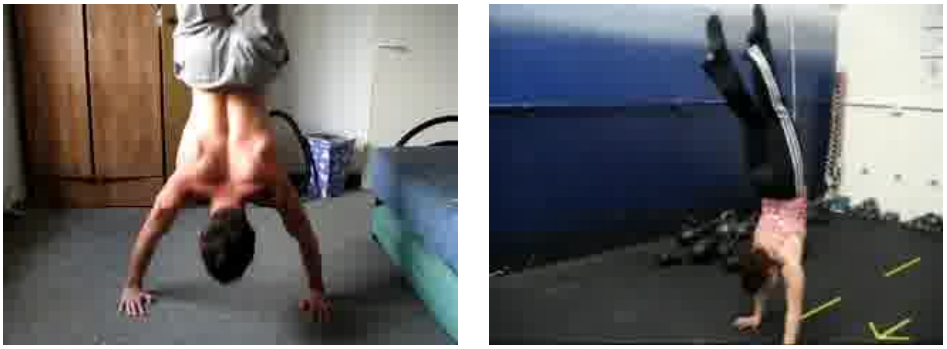


Figure 1.2: A video is a temporally ordered sequence of images, referred to as frames.

provides context of what is happening in the scene; that the musical instrument is being played and not just held up by the musician, or the direction in which the football is travelling.

Often an individual frame is not enough to determine what is happening in a video. Consider the two frames in Figure 1.3 taken from the UCF-101 dataset (Soomro *et al.*, 2012). One is of a person walking on his hands and the other is of a person doing handstand push-ups. It can be difficult to tell which frame belongs to which label, without watching more of the video sequences. The spatial features in these frames alone are not enough to determine which is which, but with the rest of the frames from each video sequence it would be more simple to determine which class each belongs to.



HandstandPushup HandstandWalking

Figure 1.3: Example of visually similar classes that cannot easily be classified without the context of how the individuals are moving.

1.2 Related work

Video classification is a challenging problem, requiring the handling of large amounts of high-dimensional and very noisy data. Large variations in camera motion, subject viewpoint, video quality and occlusion make working with video sequences difficult. For example, the subject matter may not always be in the centre of the frame, or even within a number of frames; additionally the camera might be in a fixed position watching the scene, or it may be tracking an object or person centering on them through changing backgrounds. Consider the frames in Figure 1.4. In the bottom row the camera is tracking the person swinging from frame to frame, while in the top row the camera is fixed.

Figure 1.4: A comparison of different camera motions for the same video class. In the top row the camera is fixed and the subject moves from frame to frame. In the bottom row the camera tracks the subject from frame to frame.

Previous approaches to video classification were mostly discriminative, using a visual bag of words (Csurka *et al.*, 2004) from local spatial features extracted with methods like SURF (Bay *et al.*, 2008). The classes would then typically be determined with a classifier such as a support vector machine (Duong *et al.*, 2012). The few attempts at using generative modelling commonly followed a topic modelling approach, such as Fei-Fei and Perona (2005) who developed an unsupervised method for categorising local features into "themes" or classes, and then went on to learn a distribution over these themes for classification using a Bayesian hierarchical model.

Since then, a few things have come together to enable new approaches. Not only has a plethora of image and video data been generated and made available, but developments in deep learning (LeCun *et al.*, 2015), hardware, software libraries and infrastructure such as TensorFlow, Google Cloud Platform and Amazon Web Services, have appeared that facilitate the scaling of these models to large datasets. Convolutional neural networks (CNNs), in particular, have been shown to be effective in leveraging large datasets to learn millions of

parameters and achieve state-of-the-art results across multiple benchmarks (Krizhevsky *et al.*, 2012).

Karpathy *et al.* (2014), inspired by the effectiveness in of CNNs in image classification, attempted to use deep learning to train a model on a large dataset; 1 million sports videos extracted from YouTube. They proposed an architecture that extends the CNN into the time domain by selecting various frames throughout the video sequence and fusing them together before feeding them into the model. See Figure 1.5 for a visual depiction. The approach outperformed the then state-of-the-art classifier by Soomro *et al.* (2012) and achieved 63.9% top-1 and 82.4% top-5 accuracy on the UCF-101 dataset.

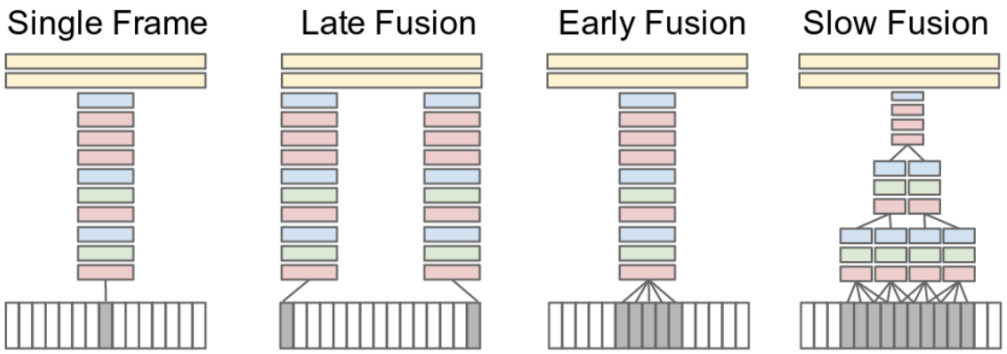


Figure 1.5: Various fusion strategies for extending CNNs into the time domain, demonstrated in Karpathy *et al.* (2014). Red, green and blue boxes indicate convolutional, normalisation and pooling layers, respectively. In the slow fusion model, the depicted columns share parameters.

Tran *et al.* (2015) demonstrated the use of 3D convolutional layers to learn spatio-temporal features, achieving an 85.5% top-5 accuracy. Carreira and Zisserman (2017) extended on this by creating an additional input stream that uses both optical flow and RGB frames (see Figure 1.6). They leveraged transfer learning from the Kinetics dataset (Carreira *et al.*, 2019), ultimately achieving a 98% top-5 classification accuracy. Their model is referred to colloquially as the I3D model.

While these models work well in terms of classification accuracy, they are limited computationally to work with shorter, fixed sized video sequences and can typically classify 50 to 100 frames at a time. This poses a problem for longer videos or where there may be more than one class within the video, such as in the YouTube8M dataset (Abu-El-Haija *et al.*, 2016), where a model may need to capture long term dependencies in order to classify various classes correctly. These architectural decisions are likely a by-product of the datasets that are being used for benchmarking, where each video only contains a single class. Architectural flexibility and scalability need not be considered when the

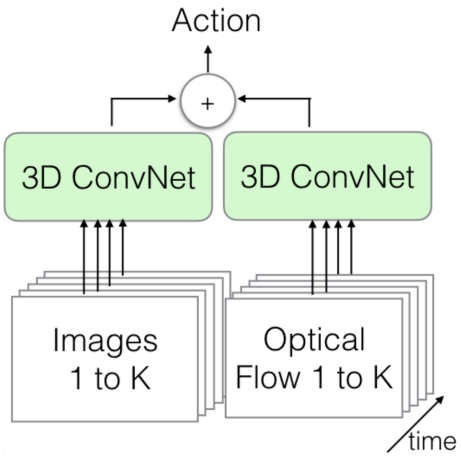


Figure 1.6: The two-stream approach used by Carreira and Zisserman (2017).

benchmarks do not require them and the researchers have large amounts of compute available to them. The YouTube8M dataset is a good step towards correcting for this. It is, however, still out of reach for most researchers due to the scale of data and the amount of compute required.

Wang *et al.* (2017) tried to solve the problem of fixed length inputs by creating temporal segment networks (TSNs). Instead of fusing frames together using the heuristics shown in Figure 1.5, they developed a sampling strategy for selecting frames, and after classifying the frames, a consensus mechanism to consolidate the classifications. They too used a two-stream approach in their network, consisting of a spatial CNN for RBG frames and a temporal CNN for stacked optical flow frames. The spatial CNN takes in each frame and extracts spatial features local to each frame, while the temporal CNN takes in the precalculated optical flow frames. They achieved a 93.2% top-5 classification accuracy on the UCF-101 dataset. The architecture is visualised in Figure 1.7.

Although the model of Wang *et al.* (2017) performs well, it is computationally expensive and complex. The optical flow features have to be precalculated before being fed into the network, and the number of network parameters is very large. Each input stream in the network is an Inception network, meaning that the resulting network would have at least 46 million parameters (23 million for each Inception network) just for the convolutional layers, plus some parameter overhead for the frame sampling and the classification tasks.

1.3 Objective of the study

We seek to develop an approach to video classification that is both flexible and scalable, using deep learning. By flexible, we mean that the model should not be limited to processing fixed length sequences of data, or outputting only a

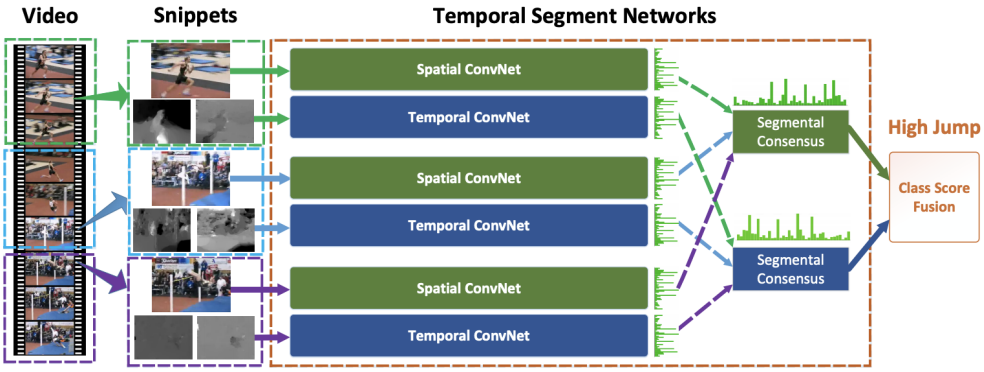


Figure 1.7: The architecture of temporal segment networks (Wang *et al.*, 2017). The network has a sampling strategy for selecting frames from a video sequence, and passes RGB frames to the spatial stream and stacked optical flow images to the temporal stream.

single class for the entire sequence. Instead, it should be able to work with variable length sequences and have the option of classifying each frame in the sequence, or the video as a whole. By scalable, we mean that the model should be adaptable to other, potentially larger datasets and have few enough parameters that it can still be used in real-world applications.

1.4 Thesis outline

Chapter 2 surveys the available datasets commonly used for video classification and highlights some practical considerations. We present some detail of the UCF-101 dataset, which is used for exploration and model training in this thesis.

Chapter 3 provides some theoretical building blocks to better understand our proposed approach, details of the software implementation and the hardware requirements, as well as the performance metrics used.

Chapter 4 looks at how convolutional networks can be used to learn the spatial features in video frames, and how transfer learning can be used to leverage feature maps learned from image classification datasets with a larger number of classes.

Chapter 5 evaluates how recurrent models can be used to learn the temporal components of a video, and looks at some shortcomings of using this approach alone.

Chapter 6 demonstrates how to effectively combine the models from Chapter 4 and Chapter 5 to leverage the best of both. We also compare our combined approach to two state-of-the-art models: the I3D model and temporal segment networks.

Chapter 7 summarises our findings and proposes possible extensions and further research.

Chapter 2

Datasets

Video classification is a computationally expensive task. Training new models may not only result in a slow turnaround time and high monetary costs, but also a high engineering overhead to enable scale to distributed training strategies like asynchronous stochastic gradient descent (ASGD) (Lian *et al.*, 2015). While leveraging larger datasets for this thesis would be ideal, it is not really feasible. Here we consider the available datasets to select one that will allow training on a single GPU and still be sufficiently general to be representative of real-world videos.

2.1 Potential datasets

Of the video datasets available, there exist two that are commonly used for researching and benchmarking new classification models: HMDB-51 (Kuehne *et al.*, 2011) and UCF-101 (Soomro *et al.*, 2012). HMDB-51 has 7,000 video clips across 51 action classes. The video sequences are extracted from YouTube, Google Photos, and movies, where each frame was scaled to 240×240. Motion stabilisation and normalisation were done as preprocessing steps to the video data.

UCF-101 was collected from YouTube and consists of 101 different activity categories totalling 13,320 videos and 3 million frames, each scaled to 240×320. Having been collected solely from YouTube, without any preprocessing or normalisation, gives the dataset the advantage of having more realistic videos than most other datasets in that it is collected from real-world settings and features video classification challenges such as camera variations, changing backgrounds, and varying scales.

We opted to use UCF-101 given that it has more videos, a larger number of classes, and more natural videos while still being small enough to allow training on a single GPU.

There are other datasets collected from YouTube that can be used for video classification. The two most popular are the DeepMind Kinetics dataset (Carreira *et al.*, 2019), which consists of 650,000 videos and 700 classes, totalling approximately 650 Gigabytes of video data, and the YouTube8M dataset (Abu-El-Haija *et al.*, 2016), which has 8 million videos and 3,863 classes. The frame level features of YouTube8M take up 1.53 Terabytes of hard drive space once downloaded, and the raw frames would be much larger. Since it is infeasible to train on these datasets without considerable compute budget, these larger datasets are not as well adopted yet, making benchmarking against other models tricky as they would require implementing and training of their architectures on the dataset in question.

2.2 Characteristics of UCF-101

The activity classes within UCF-101 are useful for benchmarking the spatial and temporal components of proposed architectures to determine where a model might be under-performing. A performant model would have to leverage both of these sources of information to make nuanced classifications between classes that are similar. The dataset contains classes with similar motions but very different objects, such as PlayingViolin compared to BrushingTeeth (see Figure 2.1). Other classes in UCF-101 have dissimilar motions but similar objects, such as HandstandPushups compared to HandstandWalking. Some classes also have very similar contextual settings, like the swimming classes SwimmingCrawl and SwimmingBreaststroke.

2.3 Dataset splits

Each activity category is split into 25 groups, and each group has many samples. Samples within a group can share similar settings or objects, and it is therefore vital to do the train/test splits according to the groups, rather than just the samples, in order to prevent data leakage.

There are three available splits that are typically used when benchmarking models on UCF-101 (Soomro *et al.*, 2012). We use TensorFlow-datasets to manage the dataset and splits. The training set consists of 9,537 videos, and the test set consists of 3,783 videos. During training iterations 10% of the videos in the training set are reserved as a validation set. If the loss calculated on the validation set starts to increase after an epoch of training, the training is stopped. This method of regularisation is called early stopping (Bishop, 2007).

(a) HandstandPushups

(c) PlayingViolin

(e) SwimmingCrawl

(b) HandstandWalking

(d) BrushingTeeth

(f) SwimmingBreaststroke

Figure 2.1: Example frames from six UCF-101 classes. Notice how the classes in (a) and (b) would have similar spatial features but dissimilar temporal features, while (c) and (d) would have dissimilar spatial features but similar temporal features. Classes in (e) and (f) show similar contextual information.

Chapter 3

Background theory

In this chapter we supply some theoretical building blocks to better understand deep learning and why it is a good fit for video classification. We also have a look at some of the practical aspects of the implementation of our experiments and some metrics for model evaluation.

3.1 Deep learning

We mentioned in Section 1.2 that deep learning is effective in leveraging large datasets to fit millions of parameters, but what really makes deep learning attractive for video classification is its modularity. Various components of neural networks can be removed from one network and added to another, or components can be trained on one dataset and applied to another.

This allows us to develop components for the spatial and temporal aspects of video classification separately and bring them together later.

3.1.1 Layers

Deep learning is essentially stacking many neural network components together to form a hierarchy of layers. Each layer takes in some input tensor $\mathbf{x}$, transforms it with a set of weights $\mathbf{w}$, and outputs the transformed value z, to serve as one of the input elements for the next layer. The most fundamental layer is the fully-connected layer, developed by Rosenblatt (1957). These layers consist of neurons, where each element x_i of the layer input is multiplied by the corresponding weight w_i and summed along with a bias b. To make the model nonlinear we apply a nonlinear activation function g to the output. The operation in one neuron can therefore be expressed as

$$z = g(b + \sum_i w_i x_i). \tag{3.1}$$

Layers of neurons can be composed together by supplying the output from one layer as the input to another layer. As we will see in the next chapters, different types of layers have different purposes and strengths.

3.1.2 Activation functions

There are many activation functions that have been researched (Nwankpa *et al.*, 2018), even some where the activation function itself is learned (Agostinelli *et al.*, 2014). The ones particular to our research is the rectified linear unit (ReLU) (Nair and Hinton, 2010), the hyperbolic tangent (tanh) (LeCun *et al.*, 1998), and the softmax activation function (Bridle, 1990). These functions are defined as follows:

$$\text{ReLU: } g(z) = \begin{cases} 0 \text{ if } z \leq 0 \\ z \text{ if } z > 0 \end{cases} \tag{3.2}$$

$$\tanh\text{: } g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.3}$$

$$\text{softmax: } g(\mathbf{z})_n = \frac{e^{z_n}}{\sum_{i=1}^{N} e^{z_i}} \text{ for } n = 1, \ldots, N \tag{3.4}$$

ReLU units have become popular in deep learning models since they do not saturate (Goodfellow *et al.*, 2016) and enable deeper models before running into the vanishing gradient problem (Hochreiter, 1998). Tanh is the preferred activation for recurrent models (more on this later). Anti-symmetric activation functions like tanh may lead to faster convergence during training than non-symmetric activation functions (LeCun *et al.*, 1998). The softmax activation function is typically used in the output layer for classification problems with N classes. This function ensures that the output values of $g(\mathbf{z})$ are real values in the range (0, 1) and the sum of the components is 1. The output vector is often treated as a probability distribution, where the nth element corresponds to the probability that the input belongs to class n.

3.1.3 Cost functions

Since our problem is a classification problem it makes sense to use categorical cross-entropy as our loss function (Goodfellow *et al.*, 2016). This loss function is tied together with the softmax activation in the output layer. The softmax output is a vector where each element corresponds to the probability of the input sample belonging to that class. We one-hot encode our labels, meaning that each of them is then a vector of length N with only one non-zero value

corresponding to the class label. The categorical cross-entropy loss function is then given by:

$$L(\mathbf{t}, \mathbf{s}) = -\sum_{i=1}^{N} t_i \log(s_i), \tag{3.5}$$

where $\mathbf{t}$ is a one-hot encoded label, and $\mathbf{s}$ is the softmax output.

3.1.4 Transfer learning

Transfer learning is a technique for reusing what has been learnt in one application and applying it to another. In deep learning, this typically means reusing a model, or layers of a model, that was trained on one dataset as a starting point for a new and related problem. It has been shown to help models train faster (Goodfellow *et al.*, 2016) as well as improve model generalisation (Yosinski *et al.*, 2014). There are many strategies for leveraging these pretrained weights; some require fine-tuning, where the weights are updated with a small learning rate value, while others freeze the weights so that they are not updated at all.

3.1.5 Optimisers

When updating the parameters of a network (the weights and biases of all the layers) using an optimisation algorithm like stochastic gradient descent (Goodfellow *et al.*, 2016), a parameter called the learning rate controls the size of the steps to reach a local minimum. The most popular is stochastic gradient descent (SGD), which uses, at every iteration, a sample of the training set to estimate the loss function and its gradient. SGD performs frequent updates with a high variance that can cause the loss to fluctuate.

There are many additional optimisers that adapt the learning rate during training to reduce the variance between updates and speed up model convergence. Adaptive moment estimation, or Adam (Kingma and Ba, 2014), is one such algorithm. It keeps track of an exponentially decaying average of past square gradients as well as an exponentially decaying average of past gradients. These are estimates of the first moment (the mean) and the second moment (the uncentred variance) of the gradients, respectively (Ruder, 2016).

We make use of Adam's adaptive learning rates while training models from scratch, but not when transferring weights learned from one application to another. With high learning rates, the previous knowledge can be forgotten too quickly (Yosinski *et al.*, 2014). Instead, when making use of pretrained weights, we apply standard SGD with a fixed learning rate that is very low to carefully update the weights.

3.1.6 Computational graphs

Neural networks are typically implemented in software packages like TensorFlow (Abadi *et al.*, 2016), where the underlying implementation is a computational graph. These graphs offer a convenient abstraction for mathematical operations. Each variable and operation is a node in the graph. The graphs are directed to provide instruction on how to get to the final result, which might be the neural network's output, or the loss function.

The abstraction makes computing gradients of the loss function during backpropagation simple, since the derivatives are obtained by traversing the graph in reverse order and utilising the chain rule. Another benefit of this abstraction is that it allows one to pick and place various neural network components easily, making transfer learning or model combinations very easy.

3.2 Implementation

3.2.1 Software libraries

All our experiments are implemented in TensorFlow 2.0, which offers a simple Python API to manage the computational graph implemented in C++. TensorFlow offers good tooling for model diagnostics and evaluating performance bottlenecks.

3.2.2 Hardware

We pointed out in Chapter 2 that video classification datasets are large. Even the UCF-101 dataset requires use of a GPU to enable an acceptable turnaround time between experiments. Our model training was done either locally on an Nvidia GeForce 1070 for small experiments, or on Google Cloud Platform using an Nvidia Tesla K80 for larger experiments.

Model inspection and evaluation are performed in the Google Colaboratory environment which offers free GPU usage for interactive sessions in a Jupyter Notebook-like environment.

3.2.3 Performance metrics

Video classification performance is usually reported with top-1 and top-5 classification accuracy (Karpathy *et al.*, 2014; Carreira and Zisserman, 2017; Wang *et al.*, 2017). Top-1 classification accuracy is determined as follows:

$$\text{Top-1: } f(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 1 \text{ if } y_i = \hat{y}_i \\ 0 \text{ if } y_i \neq \hat{y}_i \end{cases} \tag{3.6}$$

where N is the number of samples, y is the set of predicted labels and $\hat{y}$ is the set of true labels.

The top-5 classification accuracy is given as follows:

$$\text{Top-5: } f(S, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 1 \text{ if } \hat{y}_i \in S \\ 0 \text{ if } \hat{y}_i \notin S \end{cases} \tag{3.7}$$

where N is the number of samples, S is a set containing the 5 predicted labels with the highest probabilities with the highest output probabilities, and $\hat{y}$ is the set of true labels.

Chapter 4

Learning spatial features

In this chapter we use convolutional neural networks to learn the spatial features in each frame of a video. We also evaluate some existing state-of-the-art architectures for image classification, and experiment with using pretrained weights learnt from ImageNet.

4.1 Convolutional neural networks

In order to learn the spatial features required for video classification, we will make use of convolutional layers. As the name suggests, these layers summarise information from structured data by combining (or convolving) information with a kernel. The frames within videos are in essence images which have highly structural spatial features, and thus it makes intuitive sense that convolutional neural networks would make a good fit. A useful side effect of convolutional layers is that they scale very well by having low bandwidth requirements to the GPU and the convolution operations can be parallelised, which is desirable when handling video datasets.

To understand how convolutional layers summarise information we must first understand convolutions, filters, and pooling layers. A convolutional is a mathematical operation that combines two sources of information (A and B) to output a third (C) that describes how the two relate to each other. To learn spatial features we are interested in 2-dimensional convolutions to summarise the information in each frame. Here, an input image A is convolved with a filter B (also called a kernel) and outputs the summarised information C, as illustrated in Figure 4.1. The filter is moved across the image in search of features, where the step size of the movement is called the stride length.

Each convolutional layer can consist of many such filters. During training, the filters are optimised to extract information most important to the classification task. The output of these filters are known as feature maps, and can be visualised by maximising the activation of each of the convolutional outputs

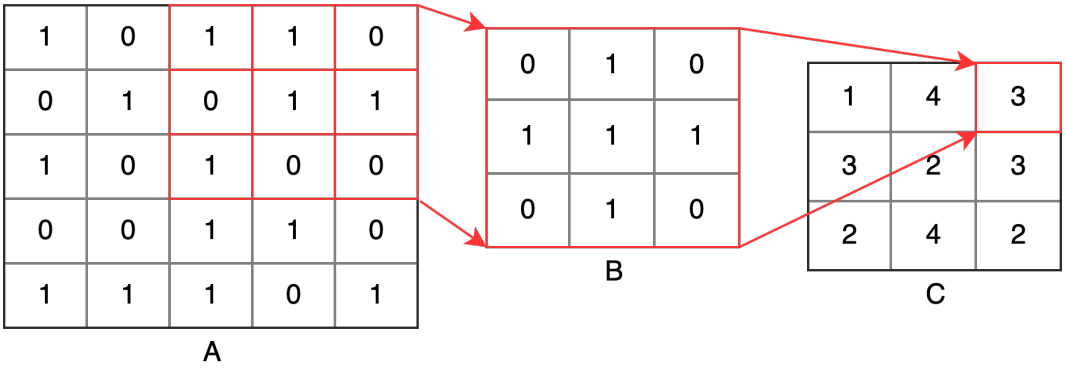


Figure 4.1: Example of a convolutional operation. The input data is convolved with the kernel to output a convoluted feature that summarises the information in the input data.

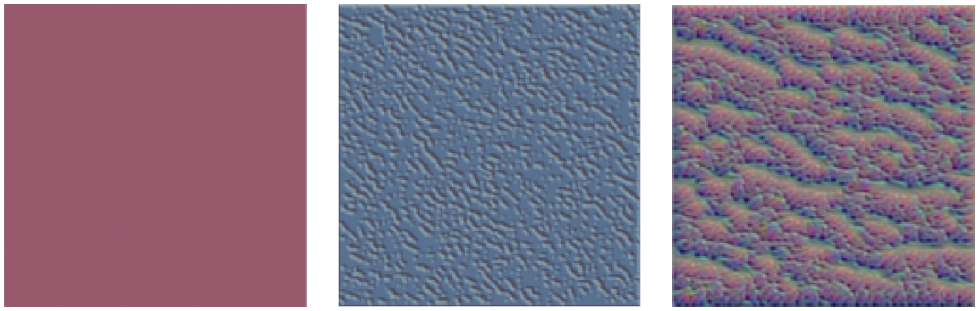
(Erhan *et al.*, 2009). Examples of feature maps can be seen in Figure 4.2 below.

While the filters are good at summarising information, they can be sensitive to small features within their receptive fields. To address this, pooling layers are introduced to downsample the feature maps by further summarising the presence of features. These pooling layers are typically very simple, and the most popular are average pooling and max pooling. As the names suggest, average pooling looks for average activation within regions of the feature map, and max pooling finds the maximum activation within regions of the feature map.

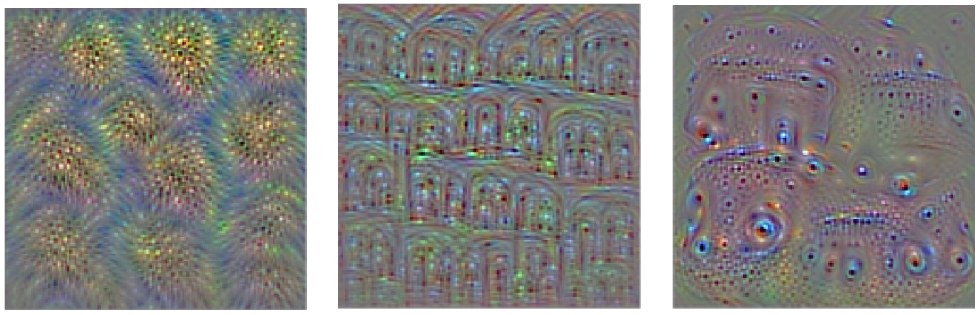
Once multiple convolution and pooling layers are stacked they can learn hierarchical features (Goodfellow *et al.*, 2016), where the complexity of the features learned increases with depth in the hierarchy. Figure 4.2 provides an example. Layers closer to the input tend to learn simple features such as colours or textures. Layers closer to the output tend to learn more complex features of the spatial relationships that describe specific objects within the images.

Convolutional neural networks that can handle a large number of classes, like the 1,000 classes in ImageNet, tend to have a large number of parameters. Training these parameters from scratch would require a lot of compute and time. To speed up experimentation and iterations, and help the model generalise (Yosinski *et al.*, 2014), it is simpler to transfer pretrained weights from a network already well suited to image classification.

There are many flavours of pretrained convolutional networks available. To narrow the search, we only consider the most researched and battle-hardened networks. The networks considered are AlexNet (Krizhevsky *et al.*, 2012), VGG (Simonyan and Zisserman, 2014), InceptionV3 (Szegedy *et al.*, 2015), and ResNet (He *et al.*, 2015). For each we consider the number of parameters and classification accuracy on ImageNet (Deng *et al.*, 2009), as listed in Table 4.1 below. Although ResNet has the best performance in terms of classification



(a) Layer 1 Block 1 (c) Layer 1 Block 2 (e) Layer 2 Block 2



(b) Layer 3 Block 2 (d) Layer 5 Block 3 (f) Layer 5 Block 5

Figure 4.2: Examples of hierarchical features visualised from the VGG network (Simonyan and Zisserman, 2014) using activation maximisation. The features learned increase in complexity as the network goes deeper.

accuracy, InceptionV3 has the best trade-off with only 23.6 million parameters and top-5 classification accuracy of 93.30%, compared to the 61.5 million parameters and 95.51% top-5 accuracy offered by ResNet.

Architecture	**Year**	**Parameters**	**Top-5 accuracy (ImageNet)**
AlexNex	2012	60M	84.70%
VGG	2014	138M	92.30%
InceptionV3	2015	23.6M	93.30%
ResNet	2016	61.5M	95.51%

Table 4.1: The convolutional architectures considered, their number of parameters, and their classification accuracy on ImageNet.

4.2 InceptionV3

The InceptionV3 network was designed to perform well under memory and compute constraints (Szegedy *et al.*, 2015). The authors achieved this by fol-

lowing a number of general design principles that prevent bandwidth bottlenecks that would slow down training and inference of the network, and result in better features from the convolutional filters themselves. Some of these efficiency principles can be summarised as factorising the convolutional filter sizes and stacking those into modules. For example, factorising a 5×5 filter into two stacked 3×3 filters is shown to be 28% more computationally efficient (see Figure 4.3). In later layers this is taken even further by factorising the 3×3 filters into stacked 3×1 and 1×3 filters, resulting in a 33% gain.

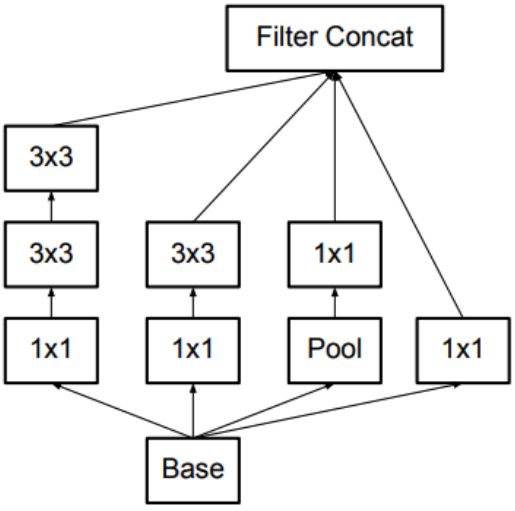


Figure 4.3: An example of an InceptionV3 module where a 5×5 filter is factorised into two stacked 3×3 filters.

InceptionV3 stacks these factorised convolutional layers, increasing the level to which the filters are factorised deeper into the network (see Figure 4.4 below). Szegedy *et al.* (2015) argue that this allows the earlier layers to learn richer representations before being reduced further. The computational efficiency makes this network an attractive choice for learning the spatial features in video classification since it will scale well to large datasets.

4.3 Training methodology

4.3.1 Dataset preparation

To train and fine-tune the convolutional layers for the UCF-101 dataset, each video is split into frames, and each frame is labelled according to the video class. To classify the video in its entirety, the mean label of the classified frames is taken to be the label of the video.

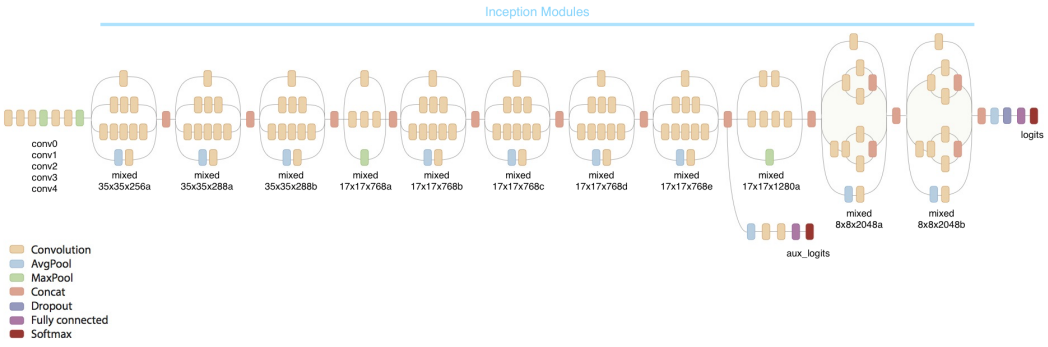


Figure 4.4: The InceptionV3 model architecture. The Inception modules are repeated throughout the network. Image source: Szegedy *et al.* (2015).

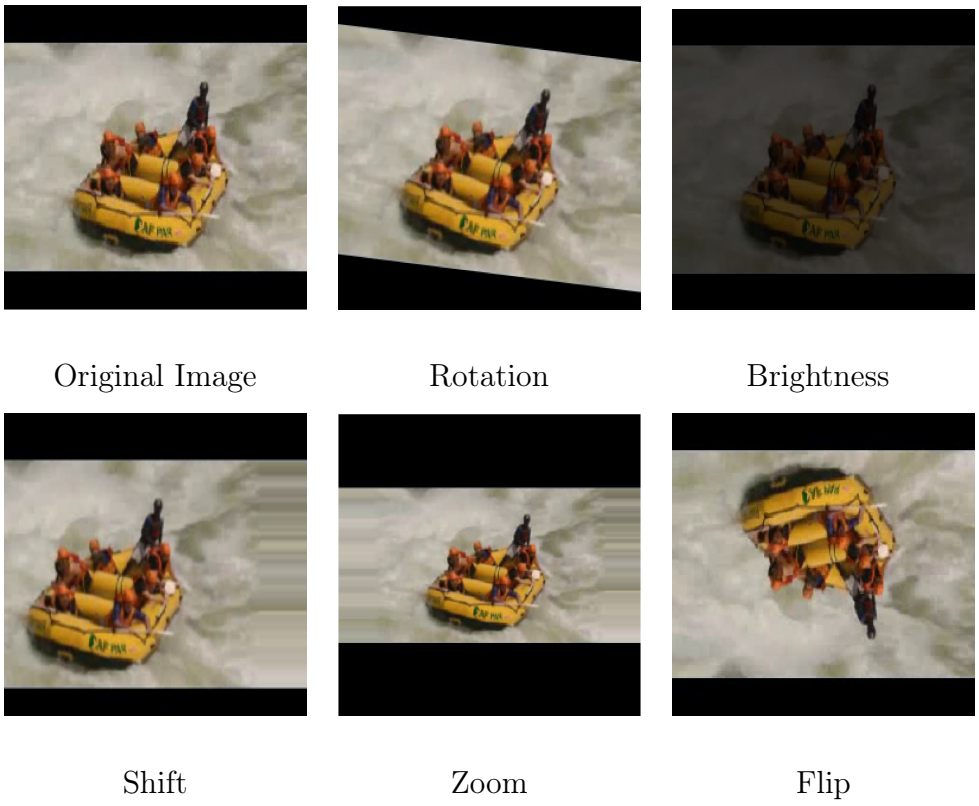


Figure 4.5: Examples of some of the transformations applied to the training images.

A data augmentation step is added to the training pipeline that applies combinations of randomised colour and geometric transformations to the images, such as crops, rotations, brightness adjustments, shifts, zooms and flips. Examples are shown in Figure 4.5. These transformations artificially increase the size and variance in the dataset to help the model generalise and to reduce overfitting.

4.3.2 Models and hyperparameters

Yosinski *et al.* (2014) showed that there are many trade-offs when adapting pretrained weights to a new classification problem. We first need to add new input and output layers to the model to fit the input shape and number of output classes of the UCF-101 dataset. We do this with fully-connected layers and a softmax activation on the output layer. What remains are the convolutional filters that have been learned from ImageNet, capable of extracting useful features from images.

To get the most out of these filters we experiment with some suggestions by Yosinski *et al.* (2014). The training is split into two phases. First, we only apply the gradient updates to the newly added layers, and fix the InceptionV3 layers, meaning that they are not updated during training. This is done for one epoch only, with the Adam optimiser (Kingma and Ba, 2014). We refer to the resulting model as InceptionV3-Pretrained.

Next we train the top two Inception modules (the ones closest to the output layer) with a very low learning rate (0.001) and standard SGD optimisation for a few epochs, and stop training before overfitting occurs. This helps retain the information in the filters learnt from ImageNet while still allowing for some adaptation to the new images. If the learning rate is too high, those weights would forget the information gained from ImageNet and replace it with features specific to the UCF-101 training dataset. We refer to this model as InceptionV3-Finetuned.

To verify that we are obtaining benefits from using transfer learning, we also train the InceptionV3 network from scratch (with random initial weights for all the layers), and refer to it as InceptionV3-Scratch. Models were selected where the training and validation loss diverged, and the hyperparameters used to train the three models are provided in Table 4.2.

Model	**Epochs**	**Optimizer/Learning rate**	**Batch size**
InceptionV3-Scratch	100	Adaptive (Adam)	128
InceptionV3-Pretrained	1	Adaptive (Adam)	128
InceptionV3-Finetuned	10	0.001	128

Table 4.2: The hyperparameters used to train the variants of the InceptionV3 architecture.

4.4 Results

In Table 4.3 we compare the three approaches to training the InceptionV3 architecture: InceptionV3-Scratch which is the InceptionV3 model trained from

scratch (no pretrained weights), InceptionV3-Pretrained which uses the pretrained weights from ImageNet but does not update them further, and InceptionV3-Finetuned which uses the InceptionV3-Pretrained model as a base and updates the top few convolutional layers as described in Section 4.3.2. The table also lists results from two state-of-the-art models: temporal segment networks (Wang *et al.*, 2017) and the I3D network (Carreira and Zisserman, 2017).

The InceptionV3 models perform rather admirably considering that they are not using any temporal information. What we do see is that using the pretrained weights and fine-tuning further improves performance, with the fine-tuned weights displaying the best performance. The InceptionV3-Scratch model overfits due to there being too many parameters in the network for fitting the UCF-101 dataset.

Architecture	**Parameters**	**Top-1**	**Top-5**
Temporal segment networks	70M	-	94.2%
I3D network	44M	-	98.0%
InceptionV3-Scratch	**23M**	**23.4%**	**38.9%**
InceptionV3-Pretrained	**23M**	**47.0%**	**73.2%**
InceptionV3-Finetuned	**23M**	**68.2%**	**87.2%**

Table 4.3: UCF-101 test results of our InceptionV3 models. Dashes are where values were not reported in the papers.

4.4.1 Interpreting the results

To better understand where the InceptionV3-Finetuned model is not performing well we look at the top-1 and top-5 accuracy per class (Figure 4.6), as well as the confusion matrix (Figure 4.7) from all the per-frame classifications over the test set. A general pattern that we can see is the classes for which the model does not perform well have spatial features very similar to other classes, and classes for which the model does perform rather well have very distinctive spatial features.

To explore this further we look at the class activation maps (CAMs). These maps show which part of an input image a layer considered important for the classification. We use the Grad-CAM method (Selvaraju *et al.*, 2017) to visualise these activation maps. This method uses gradients from the class label flowing into the final convolutional layer to produce a localisation map that highlights the important regions the network used for predicting that class label.

In Figure 4.8 we visualise the Grad-CAMs of samples from four classes; two that have a poor top-1 classification accuracy and have similar spatial features

(HandStandPushups and HandStandWalking, with accuracies less than 50%), and two that have distinctive spatial features and a good top-1 classification accuracy (PlayingCello and BrushingTeeth, with accuracies above 50%). The activation maps generally agree with our assumption that the model is relying on distinctive features for classifying the videos. HandStandPushups and HandStandWalking both activate similarly on the bare back of the person, possibly explaining the misclassifications, while for PlayingCello and BrushingTeeth the Grad-CAMs highlight the regions with distinctive objects.

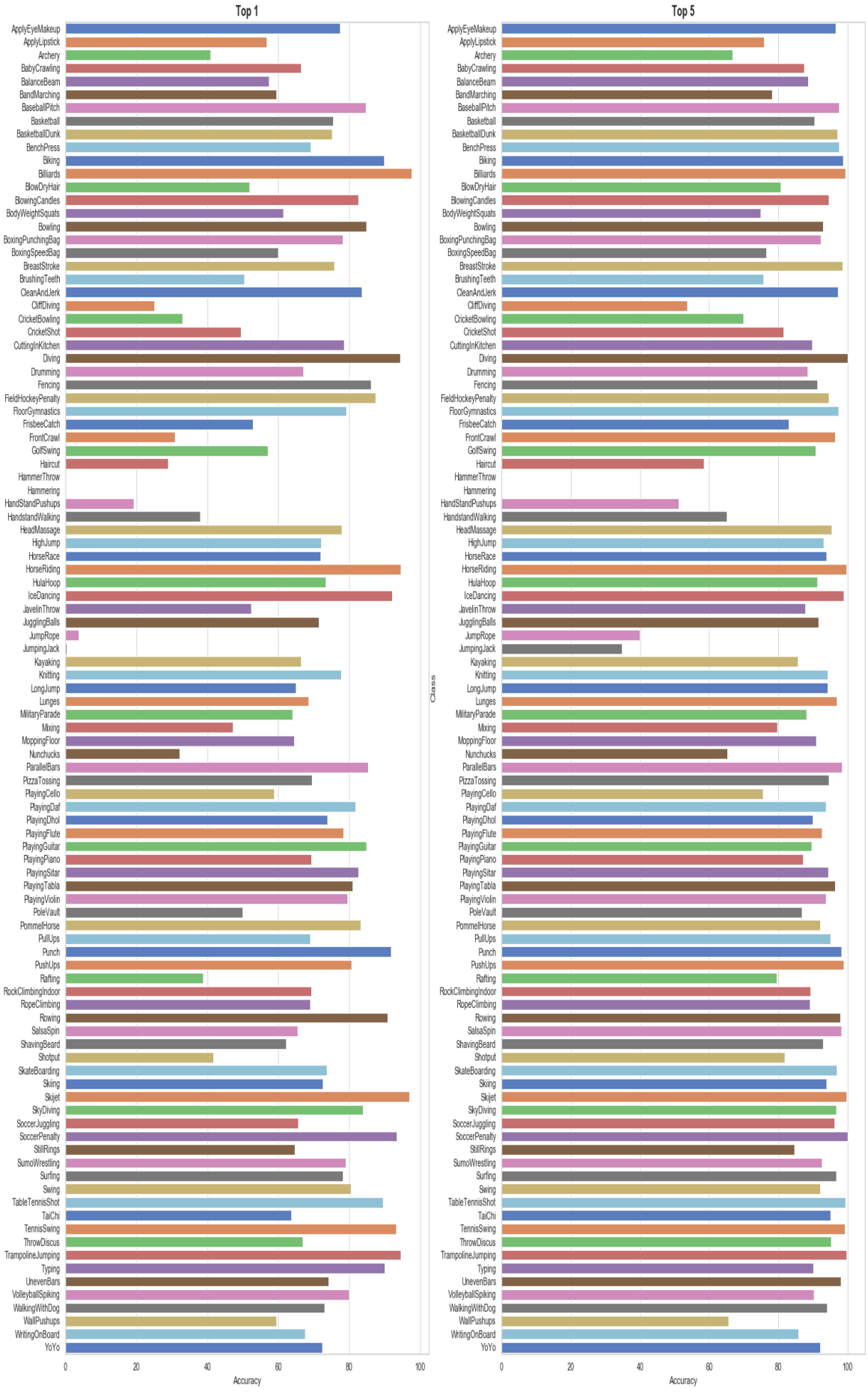


Figure 4.6: The top-1 and top-5 classification accuracy per class obtained, per frame on the UCF-101 test set using the InceptionV3-Finetuned model.

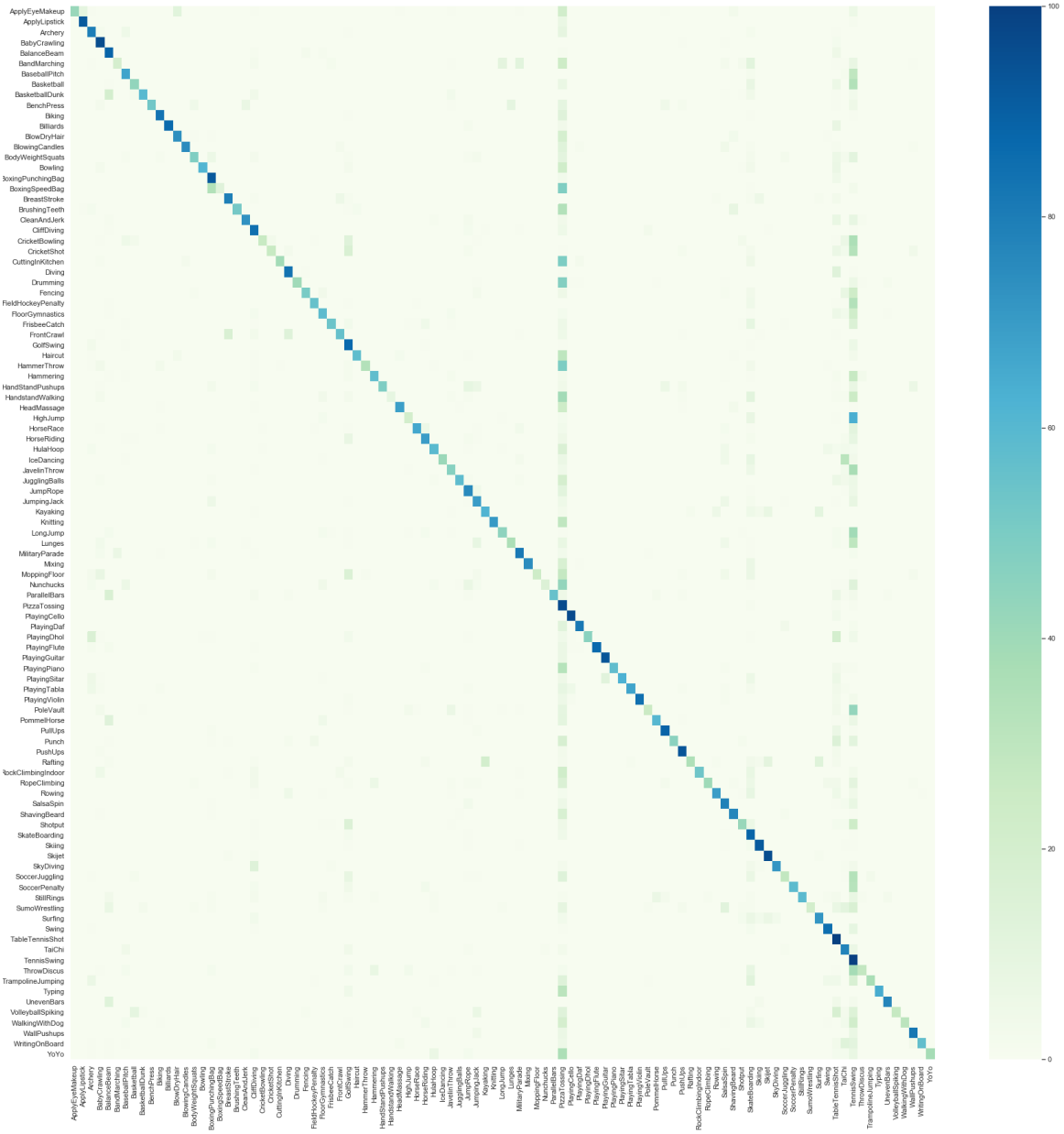


Figure 4.7: The confusion matrix obtained from the per-frame classifications of the UCF-101 test set using the InceptionV3-Finetuned model.

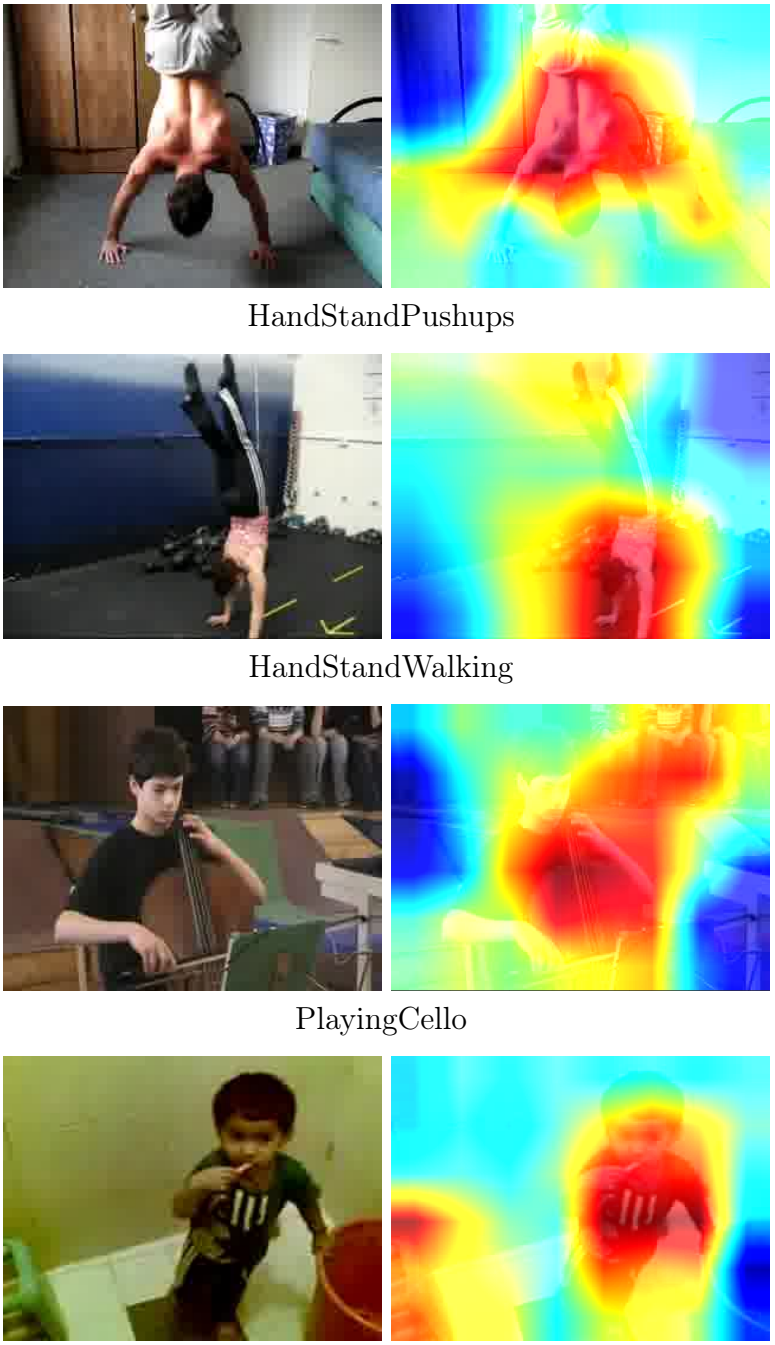


Figure 4.8: The Grad-CAMs (Selvaraju *et al.*, 2017) of samples from various classes, highlighting what regions of the input images were most important for the classification.

Chapter 5

Learning temporal features

To learn the temporal features required we need a model that scales well to large datasets, and learns complex patterns with long-term dependencies. There are two popular approaches for modelling time series data such as video: Markov models such as the hidden Markov model (HMM), and flavours of recurrent neural networks (RNNs). RNNs are a natural fit for video classification since they have the capacity to learn complicated long-term dependencies between actions, that can better help distinguish between classes, and are able to work with variable length sequences.

5.1 Recurrent networks

A recurrent structure enables neural networks to keep track of context or state, through internal loops that persist information as time moves forward. These loops are created by each step in the network, passing information from one step to the next. Each of these loops is typically referred to as a cell, and the process of chaining them together can be clarified by unrolling the network. In Figure 5.1, x_t is the input data at time step t, A is the cell containing the layer weights, and h_t is the output of the cell at time step t.

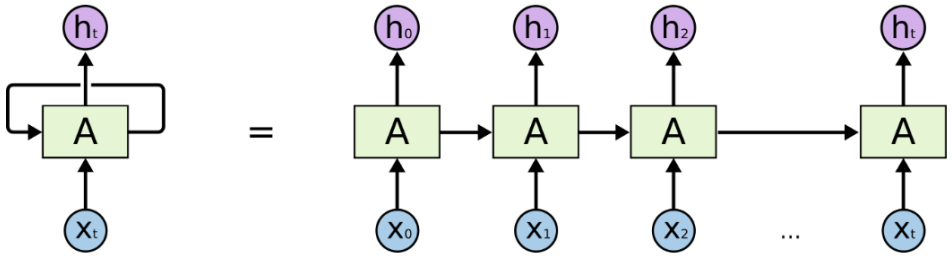


Figure 5.1: An example of an RNN and its unrolled version, taken from Olah (2015). The input x_t is passed to the cell A which outputs h_t and passes information to the next state.

A problem arises when there are too many steps in this sequence of message passing, in that the RNN has a difficult time of keeping track of relevant information, as shown by Bengio *et al.* (1994). To address this, researchers have proposed new cell structures that provide the network with mechanisms for more fine grained control over the information passed between the cells. These mechanisms are various fully-connected layers with activations that operate on the information passed, and are commonly referred to as gates. The long short-term memory (LSTM) cells (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRU) (Chung *et al.*, 2014) are the most well-known.

LSTM cells typically have three gates: an input gate, an output gate, and a forget gate. The first gate in an LSTM cell is the forget gate, which is a layer responsible for deciding which information from the previous state is important to keep. The next gate is the input gate deciding which of the information from the current input is going to be stored. Lastly the output gate is responsible for deciding what to output to the next state.

GRU cells only have two gates: the update gate and the reset gate. These can be thought of as the input and forget gates of the LSTM cell. Even without the additional gate, Chung *et al.* (2015) showed that GRU and LSTM cells yield comparable performance on many tasks. Not having an additional gate means that there are fewer parameters to learn in the network, making training easier when there is a constraint on memory or compute.

5.2 Training methodology

5.2.1 Data preparation

Each input video sample is structured such that every frame corresponds to one time step, downsampled to $50 \times 50 \times 3$ and then flattened to $7,500 \times 1$ before being fed into the recurrent models. The downsampling is necessary to constrain the number of parameters required by the network: flattening the original $320 \times 240 \times 3$ frame size would result in a 230,400-dimensional input vector. A single hidden layer with 512 units, or neurons, and a fully-connected layer with 101 hidden units for the classification would require about 350 million parameters, which makes training infeasible on most systems.

Although RNN models can handle variable length sequences they can be problematic for the forming of mini-batches during training. RNNs struggle to train properly on sequences that are very long, due to vanishing gradients (Bengio *et al.*, 1994). To get around this we use a sequence length of 100 while training. From a practical perspective batches made from shorter sequences are also computationally cheaper. To get the desired number of frames for every batch, sequences that are shorter than the desired number of frames are re-

Architecture	Epochs	Learning rate	Batch size
LSTM	30	Adaptive (Adam)	32
GRU	30	Adaptive (Adam)	32

Table 5.1: The hyperparameters used to train the LSTM and GRU models.

peated, and for sequences that are longer, a starting point that can lead to the correct sequence length is selected at random (see Figure 5.2).

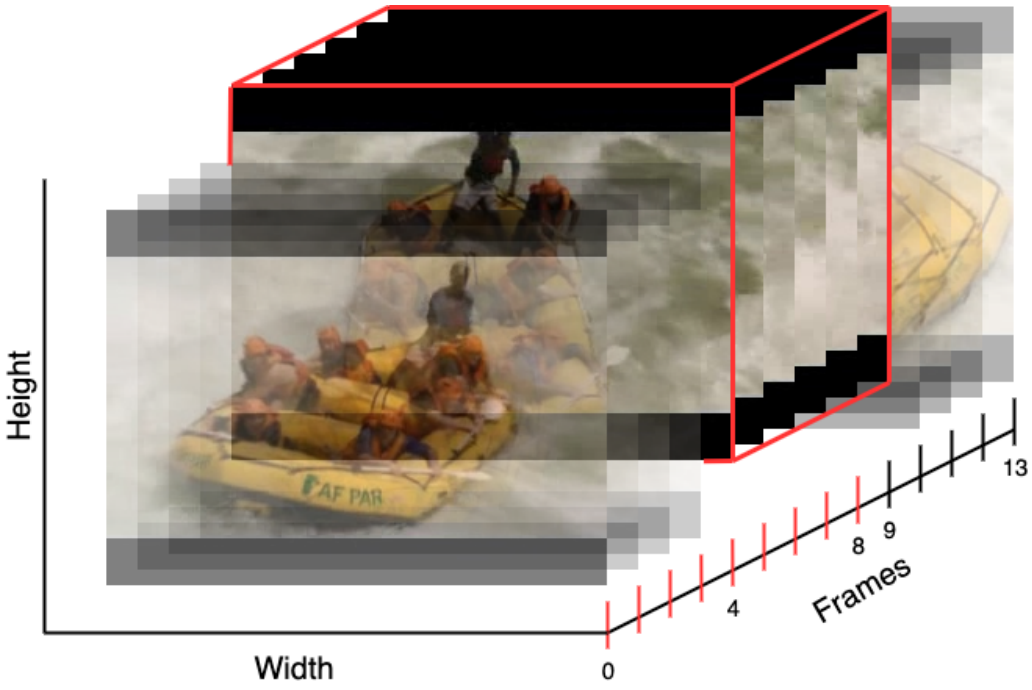


Figure 5.2: An example of the frame sequence selection process. Assuming the sequence length is 6, the valid starting points for selection are between 0 and 8, since that would allow for the desired sequence length. In this example index 4 is selected, making frames 4 to 9 the selected sequence.

Data augmentation was also applied to each of the videos, before downsampling the images. We followed the same approach as we did with the training data for the convolutional networks in Section 4.3.1. However, for consistency between frames, the same augmentation parameters applied to the first frame are applied to all the frames for a particular sample in a particular batch.

5.2.2 Hyperparameters

We experiment with LSTM and GRU cells in two separate models. Their architectures are shown in Figure 5.3. Since we do not use any transfer learning for these models, we make use of the Adam optimiser during training to speed up the process. The hyperparameters used are displayed in Table 5.1.

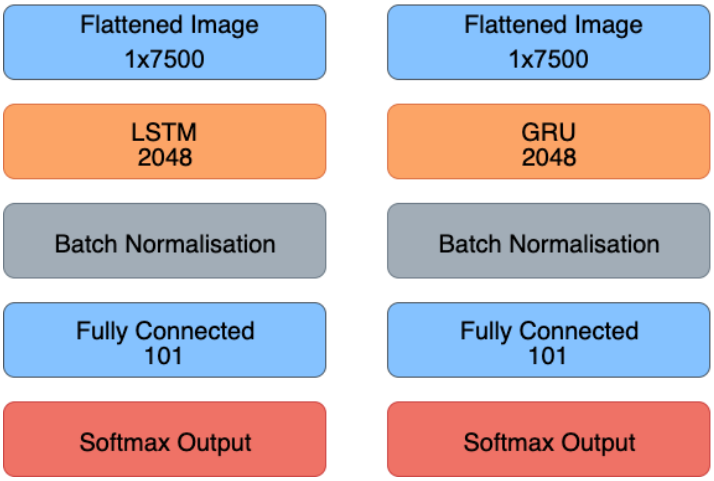


Figure 5.3: The architecture of the LSTM and GRU models.

5.3 Results

Results on the UCF-101 test set are shown in Table 5.2, with per-class accuracies and the confusion matrix in Figure 5.4 and Figure 5.5, respectively. We can see that the recurrent models perform poorly compared to the spatial models used in Chapter 4. There are several potential reasons for this. The convolutional layers of the InceptionV3 network are extremely good at summarising only the most important information in an image, where the basic downsampling mentioned in Section 5.2 compresses the information in a lossy way. There is insufficient data and compute to scale the model to make use of the input without the downsampling. The convolutional networks can also rely on strong visual cues, such as an instrument or piece of sporting equipment.

We opt to use the GRU model going forward. Our results show comparable results between the GRU and LSTM models, while the GRU model has far fewer parameters.

A better approach would be to make use of the effective data summarising capabilities of the fine-tuned model in Chapter 4 to better assist the recurrent models. We consider this in Chapter 6.

Architecture	**Parameters**	**Top-1**	**Top-5**
Temporal segment networks	70M	-	94.2%
I3D network	44M	-	98.0%
InceptionV3-Scratch	23M	23.4%	38.9%
InceptionV3-Pretrained	23M	47.0%	73.2%
InceptionV3-Finetuned	23M	68.2%	87.2%
LSTM	**33M**	**44.1%**	**72.8%**
GRU	**25M**	**45.8%**	**74.2%**

Table 5.2: UCF-101 test results of our LSTM and GRU models compared to the benchmarks and convolutional models from Chapter 4.

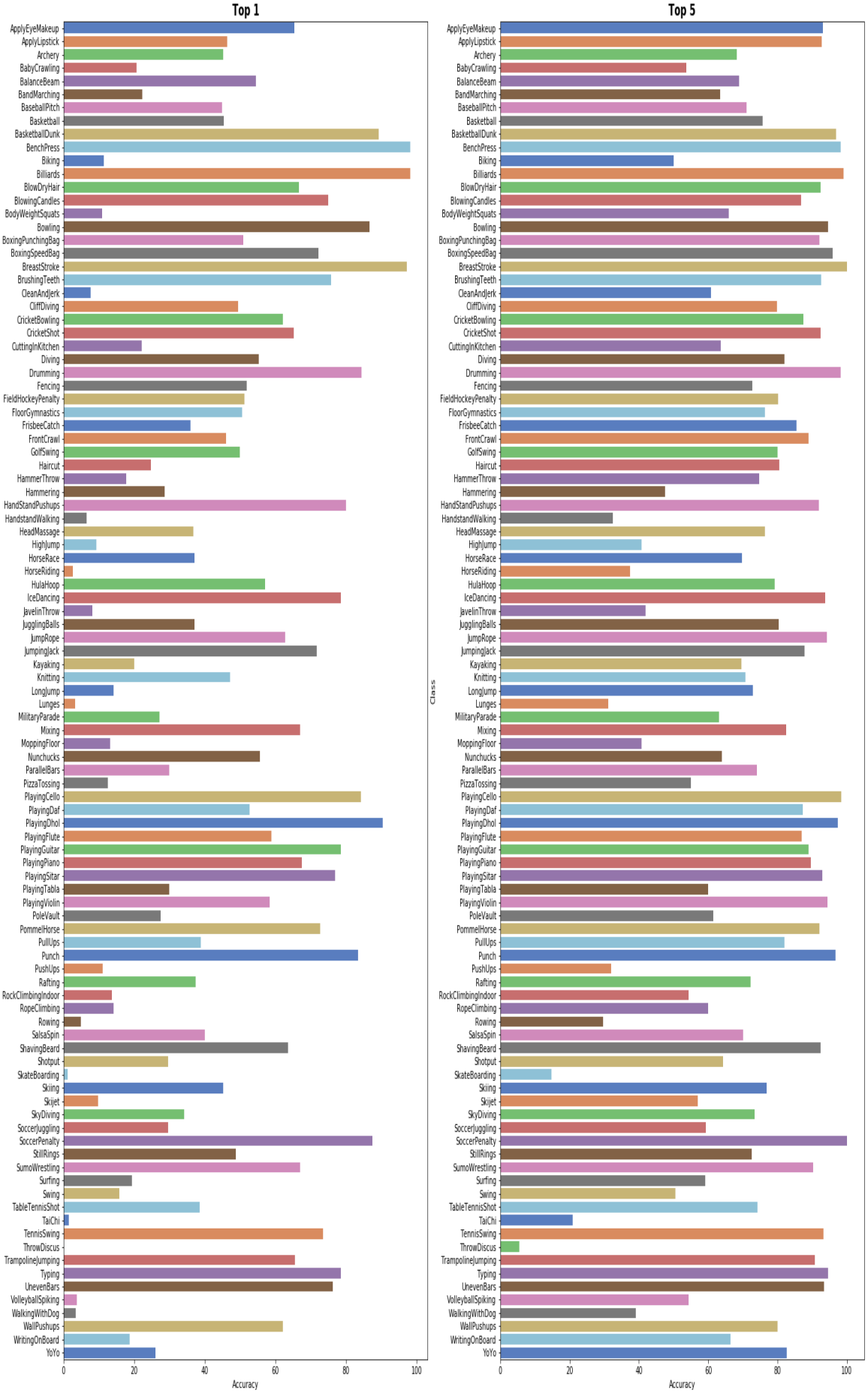


Figure 5.4: The top-1 and top-5 classification accuracy per class of the UCF-101 test set using the GRU model.

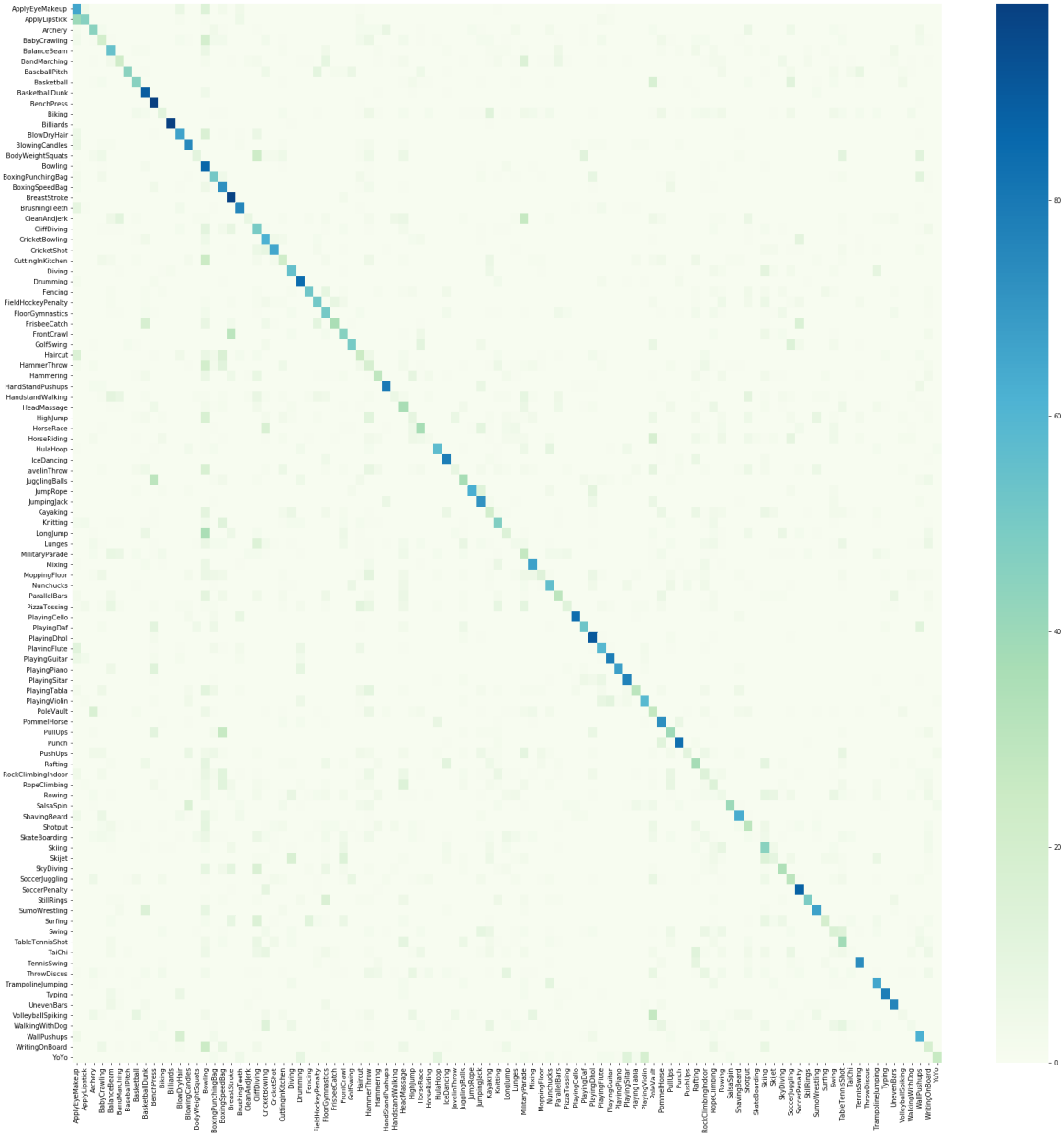


Figure 5.5: The confusion matrix obtained from the per-frame classifications of the UCF-101 test set using the GRU model.

Chapter 6

Combining spatial and temporal features

In Chapter 4 we demonstrated how to learn spatial features, and how they alone are insufficient for video classification. In Chapter 5 we did the same for temporal features. In this chapter we propose an architecture that combines the two into an end-to-end deep learning model that is both flexible and scalable.

6.1 Convolutional GRU

In this approach we look at an end-to-end deep learning model that combines the models developed in Chapters 4 and 5. The InceptionV3 modules, or indeed any convolutional network, learns to summarise the spatial information from each frame into a $1 \times 2,048$ representation before passing it to a fully-connected layer with softmax activation for classification. This representation can be thought of as a vector that summarises the spatial information of a frame, and can replace the simple downsampling used in Chapter 5. Instead of blindly compressing all the information we extract the spatial information relevant to making classifications.

We feed these frame representation vectors into a 512-unit GRU layer. We do not need as many parameters as we did in Chapter 5 since the input is smaller (2,048-dimensional instead of 7,500-dimensional). The GRU layer then keeps track of how the representations change from frame to frame. At every time step the output of the GRU is fed into a batch normalisation layer. Cooijmans *et al.* (2016) demonstrated that batch normalisation in recurrent models can lead to faster training times and improved generalisation. We place a fully-connected layer on the output of the GRU, with softmax activation for classification. A diagram of this architecture can be seen in Figure 6.1.

Our resulting model has 26 million parameters, which is almost half the num-

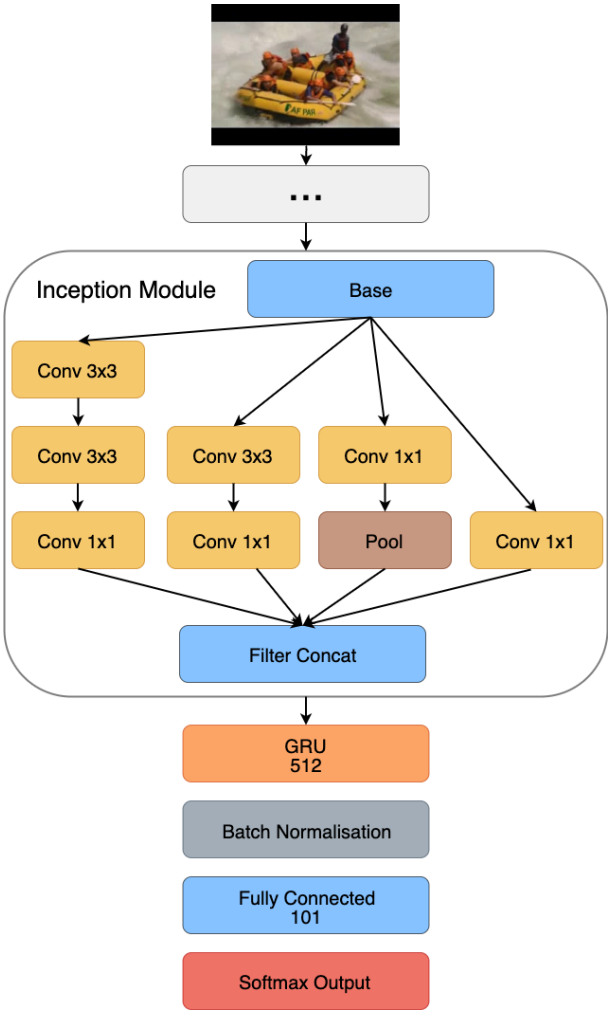


Figure 6.1: Our proposed architecture that combines the InceptionV3 network and the GRU to leverage both spatial and temporal features. The image shows several stacked InceptionV3 modules feeding into the GRU.

ber of parameters of the I3D network, and almost three times smaller than the TSN. This level of reduction in network size means significantly shorter training and inference times.

Our approach allows for some flexibility between video classification use cases. Along with being able to handle variable length videos, the network can assign a class to either the entire video (by only classifying the GRU output at the last time step) or to each frame, as illustrated in Figure 6.2. For videos where the topic or class may change through time, the model can output the class per frame and recognise the new class as it occurs. Models with fixed sized input have a more difficult time with this, since they would have to slide across the video and sections where the temporal footprint of the class is less than half of the input window might be missed. This sliding operation can be more computationally expensive too, depending on the required resolution of the

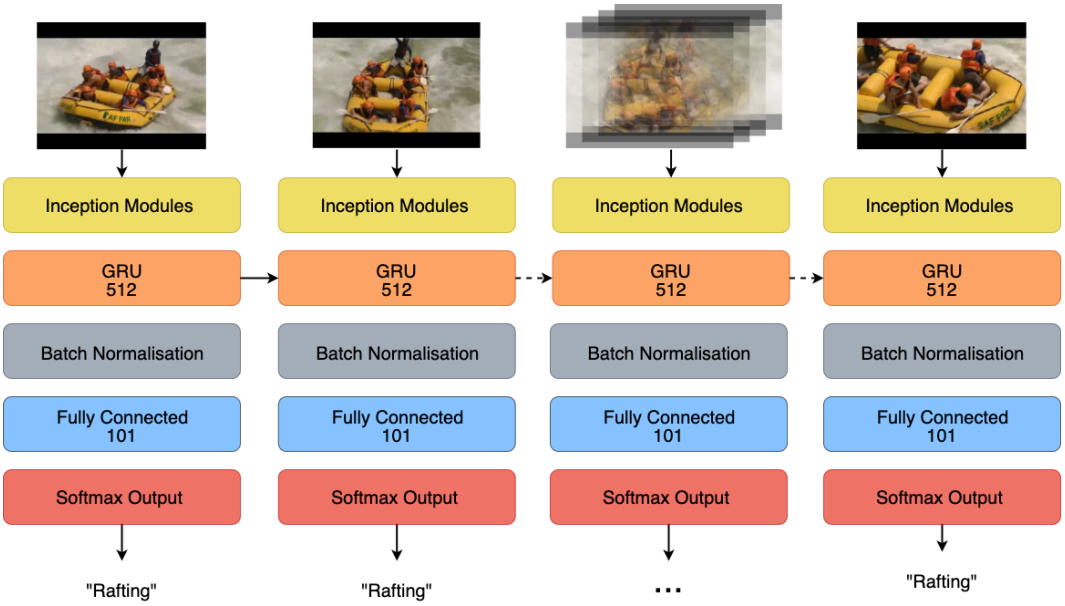


Figure 6.2: Our proposed architecture unrolled, outputting a label per time step.

classifications. If high resolution is required, a model with fixed input length n would require each frame to be processed $n-1$ times (assuming values cannot be cached), compared to our model that processes each frame only once.

Having a learned representation of each frame makes it possible to calculate the representations per frame and classify them later, by detaching the InceptionV3 component and storing its output. For a video that is 100 frames long, we could store 100 2,048-dimensional representations, instead of the 100 frames in their original resolution. This is useful for cases where memory is sensitive and timely responses are not.

6.2 Alternative considerations

6.2.1 3D convolutions

3D convolutional layers are similar to 2D convolutional layers that we mentioned in Chapter 4, except that they operate in three dimensions instead of two. The input data, the kernel and the output are now all 3-dimensional. See Figure 6.3 for a diagram.

Instead of convolving over the frames individually with 2-dimensional filters, we convolve with 3-dimensional filters over stacked frames, extending the receptive fields temporally to learn spatio-temporal features. An important benefit here is being able to transfer learned spatio-temporal feature maps from larger datasets.

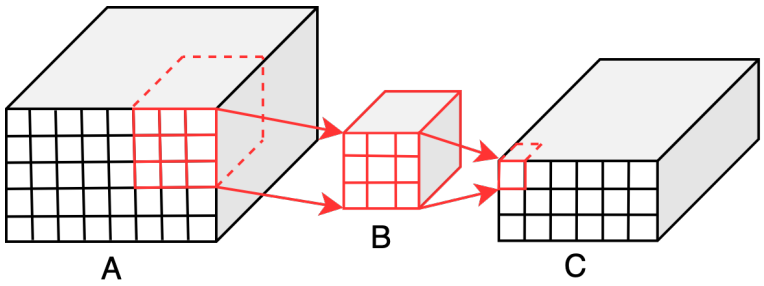


Figure 6.3: Example of how a 3D kernel summarises the input data within a 3D space.

These 3D convolutions are, however, not without their drawbacks. 3D convolutional networks cannot handle variable length sequences, and are limited to evaluating frames over a fixed window in the temporal dimension. 3D convolutions are also computationally expensive. The same design principles developed by Szegedy *et al.* (2015) that help drive very deep 2D convolutional networks, are not yet as well developed for their 3D counterparts. Although there has been some progress in the area (Xie *et al.*, 2018), 3D convolutions are still not nearly as efficient as 2D convolutions.

6.2.2 Optical flow

Optical flow can be defined as the extraction of the distribution of apparent velocities or movement of the brightness pattern in an image (Horn and Schunck, 1980). Figure 6.4 gives an example. Mathematically, optical flow is the estimation of motion between frames from time t to time $t + \Delta t$. The calculation itself is computationally inexpensive, and helps video classification models achieve considerable performance gains (Carreira and Zisserman, 2017) since it isolates the motion away from the rest of the information in the frames. The model can then have two input streams, one for spatial features and one for temporal features.

Having two input streams becomes computationally expensive, since it roughly doubles the number of parameters within a network. For this reason we decided not to use optical flow in our model and rely only on the raw RGB frames.

6.3 Training methodology

6.3.1 Hyperparameters

We train three versions of the model described in Section 6.1. ConvGru-Fixed is a version where the parameters in the pretrained InceptionV3 modules are not updated during training, only the GRU layer, the batch normalisation layer and the fully-connected layer. ConvGru-Finetuned is a version where all trainable parameters are updated. Finally, ConvGru-Scratch is a version

Figure 6.4: An example of optical flow using the TV-1 algorithm (Drulea and Nedevschi, 2011). The top row contains temporally ordered frames, each 0.25 seconds apart, with the calculated optical flow visualised in the bottom row.

Architecture	Epochs	Learning rate	Batch size
ConvGru-Scratch	100	0.001 Adaptive (Adam)	32
ConvGru-Fixed	30	0.0001 (SGD)	32
ConvGru-Finetuned	30	0.0001 (SGD)	32

Table 6.1: The hyperparameters used to train each of our ConvGru models.

where we train the full network from scratch, without any pretraining from the ImageNet dataset. The hyperparameters used during training for each of these models are listed in Table 6.1.

6.3.2 Dataset preparation

The dataset preparation for the models in this chapter is similar to that described in Section 5.2.1. However there are some technical considerations in the training of these networks. They can be unrolled, as opposed to using a symbolic loop, which duplicates the network weights for every time step and can improve training time at the expense of memory. Unlike in Chapter 5, this network is much smaller making it possible to use the additional memory.

Although it is possible to precalculate the features from the InceptionV3 module to train the GRU component separately with a reduced memory footprint, as done in Wang *et al.* (2017), we opt not to do this. Instead we apply data augmentation and feature extraction for every batch in the training iterations by randomly selecting a start point in the sequence, which can also be thought of as a sort of data augmentation to create more training sequences.

6.4 Results

6.4.1 Interpreting the results

The results in Table 6.2 show that combining the spatial and temporal approaches outperforms the models from Chapter 4 and Chapter 5, with the exception of ConvGru-Scratch. Without pretrained weights the model takes excessively long to train with a small learning rate, and overfits quickly if the learning rate is increased. This is likely due to there being too many parameters for training on the UCF-101 dataset alone without resorting to aggressive regularisation, as we saw with InceptionV3-Scratch in Chapter 4.

The fine-tuned model performed the best, but only when updated for a a few epochs with a very low learning rate. In Figure 6.5 we see the differences between some filters in the convolutions from the trained ConvGru-Fixed and ConvGru-Finetuned models.

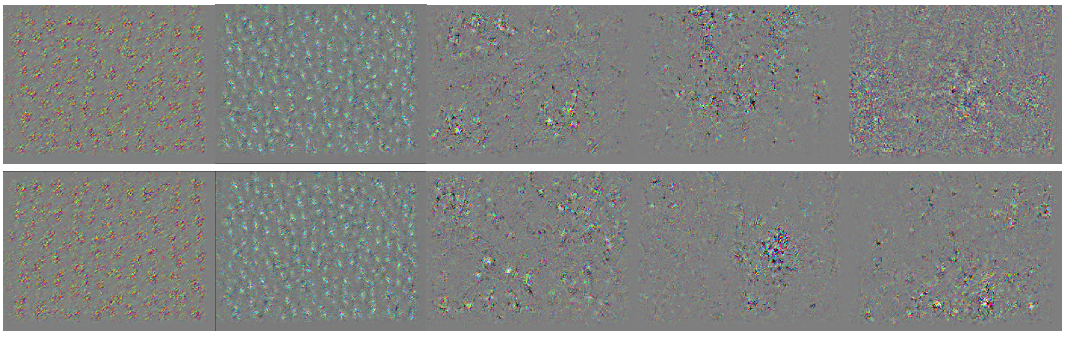


Figure 6.5: A visualisation of various filters after training, increasing in depth from the left. The top row is from ConvGru-Fixed and the bottom row is from ConvGru-Finetuned.

The changes in the convolutional filters between the two different training approaches indicate that perhaps incorporating the temporal component in the training can aid in finding features relevant to the classification, such as motion or blur features, or it could simply be that the network finds a different local minimum where the learned filters are different.

Looking at the per-class accuracy (Figure 6.6) and per-frame confusion matrix (Figure 6.7), as we did for the models in Chapter 4 and Chapter 5, we see better performance.

6.4.2 Comparison to state-of-the-art

6.4.2.1 I3D network

Since the start of this study there has been notable progress in general approaches to video classification. Most notably, Carreira and Zisserman (2017) showed that using a combination of transfer learning from the Kinetics dataset

Architecture	Parameters	Top-1	Top-5
Temporal Segment Networks	70M	-	94.2%
I3D Network	44M	-	98.0%
InceptionV3-Scratch	23M	23.4%	38.9%
InceptionV3-Pretrained	23M	47.0%	73.2%
InceptionV3-Finetuned	23M	68.2%	87.2%
LSTM	33M	44.1%	72.8%
GRU	25M	45.8%	74.2%
ConvGRU-Scratch	**26M**	**14.2%**	**21.3%**
ConvGRU-Fixed	**26M**	**68.7%**	**89.2%**
ConvGRU-Finetuned	**26M**	**71.1%**	**91.8%**

Table 6.2: UCF-101 test results of our ConvGRU models compared to the benchmarks, the convolutional models from Chapter 4 and the recurrent models from Chapter 5.

(Carreira *et al.*, 2019), optical flow and 3D convolutional layers can achieve state-of-the-art results. Their model achieves a top-5 accuracy of 98.0% on the UCF-101 test set. This approach has two drawbacks: 3D convolutional layers are computationally expensive (Xie *et al.*, 2018), and the model is limited to a fixed sequence length for classification, typically 50 to 100 frames.

Our model is computationally cheaper during training and inference, with half the number of parameters and only a 6.2% reduction in top-5 classification accuracy, and has the advantage of being able to handle variable length input.

6.4.2.2 Temporal segment networks

Wang *et al.* (2017) developed temporal segment networks to solve the problem of having a fixed length input. The trade-off is computational complexity. The network relies on a sampling strategy for frame selection and three input streams to extend the convolutional layers into the temporal dimension, resulting in over 70 million parameters.

Our model is almost three times smaller with only a 2.5% reduction in top-5 classification accuracy. We also have the advantage of not requiring the additional optical flow calculations, which makes our model easier to adapt to new datasets and deploy since no additional features need to be calculated.

6.5 Discussion

We demonstrated that we are able to develop a more scalable architecture than existing state-of-the-art approaches at the cost of model performance. The usefulness of our approach would depend on the application. In the field of medicine, for example, having a model that requires a fixed length sequence or is computationally expensive could likely be a worthwhile trade-off for an

additional boost in model performance. Our model might be better suited to less critical applications, or deployments to low-resource environments like edge computing platforms.

Our model is also flexible in terms of video length. Being a recurrent model it can unroll to the length of the video and output classifications per frame, making the option of varying classes possible.

Looking for new architectures to solve the problem of video classification might not result in performance gains as much as better ways of leveraging transfer learning. Our results indicate that the majority of our performance gains come from using the network weights pretrained on ImageNet and then fine-tuning them for our application, rather than the architectural decisions. The results of Carreira and Zisserman (2017) indicate the same; without pretraining on the Kinetics dataset their model achieved a 93.4% top-5 accuracy compared to 98% with pretraining. Similar architectures (Tran *et al.*, 2015) have also required transfer learning for optimal results.

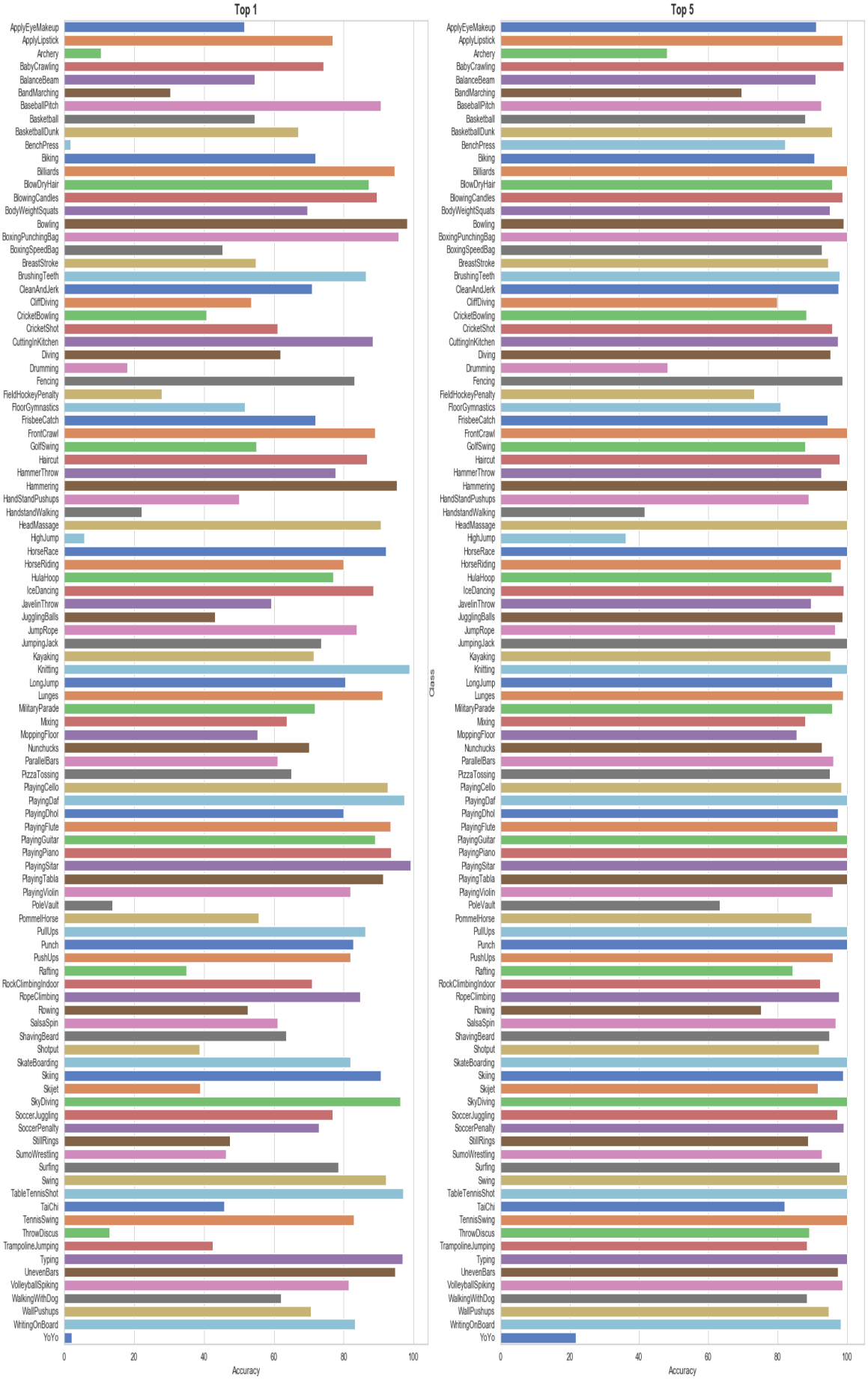


Figure 6.6: The top-1 and top-5 classification accuracy per class of the UCF-101 test set using the ConvGru-Finetuned model.

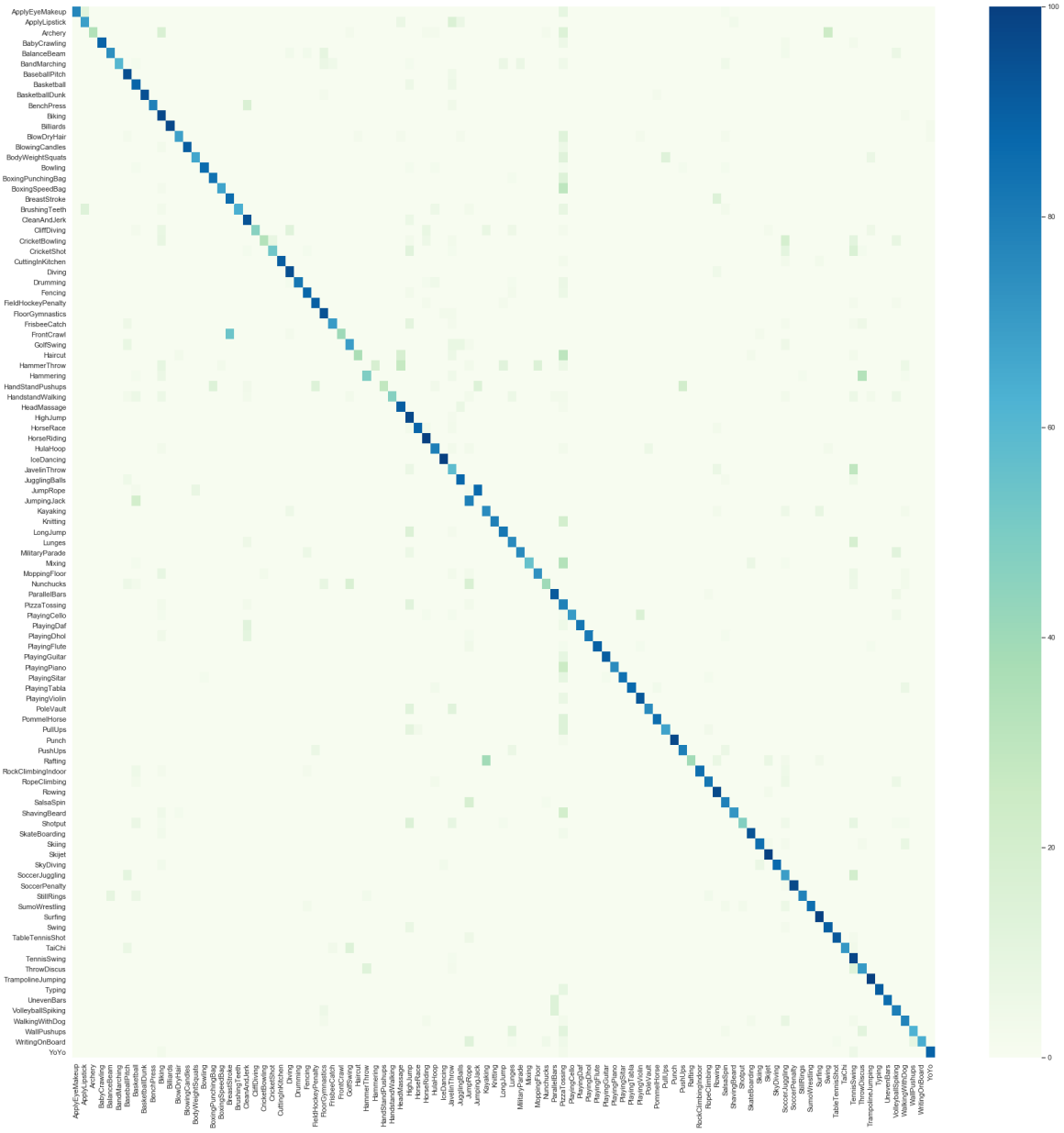


Figure 6.7: The confusion matrix obtained from the per-frame classifications of the UCF-101 test set using the ConvGru-Finetuned model.

Chapter 7

Conclusion

Our goal was to develop a scalable and flexible video classification model that could be used for real-world use cases. Scalable automated video classification would help to effectively analyse and manage the ever-growing video data that are being generated every day. State-of-the-art approaches, while performant in terms of classification accuracy, are computationally expensive and often require a fixed length video sequence as input.

7.1 Summary

In Chapter 1 we highlighted that the information in video sequences is high-dimensional, and is both spatially structured within each frame and temporally ordered between successive frames. Chapter 2 introduced the problem and looked at previous work. An overview of the potential datasets was given in Chapter 3. In Chapter 4 we explored using convolutional neural networks to learn the spatial features relevant to each frame. We considered a few existing convolutional neural network architectures to use as a starting point. We settled on the InceptionV3 architecture. It has less than half the parameters of the state-of-the-art model in image classification, while performing only marginally worse than the state-of-the-art.

We experimented with different transfer learning approaches, inspired by the work done by Yosinski *et al.* (2014). We showed that training the InceptionV3 model on the frames in the UCF-101 dataset from scratch achieves very poor classification performance. Thanks to the number of parameters the model overfits aggressively even with our data augmentation techniques. We settled on a two-phased training approach where we first fix the pretrained InceptionV3 layers and train the newly attached layers for an epoch, and then train only the top two InceptionV3 modules with a very low learning rate for a few epochs. We showed that using the spatial features alone causes the model to rely heavily on spatial features that are unique to classes, leading to poor

performance on classes that share similar spatial characteristics.

We compared LSTMs and GRUs in Chapter 5 for the learning of temporal features. Without downsampling the frames the parameter space of the recurrent networks explodes as the input size increases, quickly becoming computationally infeasible for training on our available hardware. The traditional downsampling techniques result in a lossy compression of the video information, resulting in poor model performance. We showed that the GRU and LSTM models achieve comparable performance, and opted to use the GRU for our recurrent component going forward given that it has far fewer parameters to train.

We proposed an architecture in Chapter 6 that leverages the InceptionV3 architecture to learn representations for each frame that can then be used by the recurrent component. We again experimented with different transfer learning approaches, and again we found that a two-phase approach works best. Our final model achieves a 71.1% top-1 and 91.8% top-5 accuracy, compared to the 98.0% top-5 accuracy from the state-of-the-art I3D model of Carreira and Zisserman (2017), while having half the number of parameters and an architecture that is capable of handling variable length sequences.

We went on to reason that our results, as well as the results of Carreira and Zisserman (2017), seem to indicate that the majority of the performance gains come from transfer learning on larger datasets rather than from the architectural design. This suggests that perhaps investigating various transfer learning approaches might be a good starting point for future work.

7.2 Future work

There are a few possible avenues for extending this work. As mentioned above, our results indicate that the performance gains in the top-1 and top-5 classification accuracy were due more to making use of pretrained weights, rather than the architectural decisions. Investigating transfer learning strategies to better utilise other datasets or by pretraining our entire architecture on larger video datasets like Kinetics or YouTube8M could likely further improve our results.

While 3D convolutional layers are restrictive in their input requirements (fixed length sequences) they can be highly effective at learning spatio-temporal features. It could be worthwhile exploring ways of making these layers more flexible to variable length inputs, perhaps by using attention mechanisms (Vaswani *et al.*, 2017).

Recurrent neural networks are not the only way to manage temporal context in deep learning. In fact, these networks have some hardware performance drawbacks. Recent work has been done on using pervasive attention and 2D

convolutional neural networks for sequence prediction (Elbayad *et al.*, 2018). Using more efficient operations like this could further improve the performance of video classification on limited hardware.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B. and Vijayanarasimhan, S. (2016). Youtube-8M: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*.

Agostinelli, F., Hoffman, M., Sadowski, P. and Baldi, P. (2014). Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359.

Bengio, Y., Simard, P. and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166.

Bishop, C.M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1st edn. Springer.

Bridle, J.S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In: *Neurocomputing*, pp. 227–236.

Carapetis, J.R. (2007). Rheumatic heart disease in developing countries. *New England Journal of Medicine*, vol. 357, no. 5, pp. 439–441.

Carreira, J., Noland, E., Hillier, C. and Zisserman, A. (2019). A short note on the Kinetics-700 human action dataset. *arXiv preprint arXiv:19.06987*.

Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? A new model and the Kinetics dataset. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4724–4733.

Chung, J., Gülçehre, Ç., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, vol. abs/1412.3555.

Chung, J., Gülçehre, Ç., Cho, K. and Bengio, Y. (2015). Gated feedback recurrent neural networks. *CoRR*, vol. abs/1502.02367.

Cooijmans, T., Ballas, N., Laurent, C., Gulcehre, C. and Courville, A. (2016). Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*.

Csurka, G., Dance, C., Fan, L., Willamowski, J. and Bray, C. (2004). Visual categorization with bags of keypoints. In: *ECCV Workshop on Statistical Learning in Computer Vision*, vol. 1, pp. 1–2.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In: *IEEE Conference on Computer Vision and Pattern Recognition*.

Drulea, M. and Nedevschi, S. (2011). Total variation regularization of local-global optical flow. In: *IEEE Conference on Intelligent Transportation Systems*, pp. 318–323.

Duong, D., Dinh, T.B., Dinh, T. and Duong, D. (2012). Sports video classification using bag of words model. In: Pan, J.-S., Chen, S.-M. and Nguyen, N.T. (eds.), *Intelligent Information and Database Systems*, pp. 316–325. Springer Berlin Heidelberg, Berlin, Heidelberg.

Elbayad, M., Besacier, L. and Verbeek, J. (2018). Pervasive attention: 2D convolutional neural networks for sequence-to-sequence prediction. *arXiv preprint arXiv:1808.03867*.

Erhan, D., Bengio, Y., Courville, A. and Vincent, P. (2009). Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*.

Fei-Fei, L. and Perona, P. (2005). A Bayesian hierarchical model for learning natural scene categories. In: *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 524–531.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

He, K., Zhang, X., Ren, S. and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, vol. 9, no. 8, pp. 1735–1780.

Horn, B.K. and Schunck, B.G. (1980). Determining optical flow. *Technical Report, Massachusetts Institute of Technology*.

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

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012). Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, pp. 1097–1105.

Kuehne, H., Jhuang, H., Garrote, E., Poggio, T. and Serre, T. (2011). HMDB: a large video database for human motion recognition. In: *International Conference on Computer Vision*.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature*, vol. 521, no. 7553, pp. 436–444.

LeCun, Y., Bottou, L., Orr, G.B. and Müller, K.-R. (1998). Efficient backprop. In: *Neural Networks: Tricks of the Trade*, pp. 9–50.

Lian, X., Huang, Y., Li, Y. and Liu, J. (2015). Asynchronous parallel stochastic gradient for nonconvex optimization. In: *Advances in Neural Information Processing Systems*, pp. 2737–2745.

Martin, J. (2017). Can facebook solve its violent video problem? *CNET*.
Available at: `www.cnet.com/news/facebook-live-violence-cleveland-thailand/`

Nair, V. and Hinton, G.E. (2010). Rectified linear units improve restricted Boltzmann machines. In: *International Conference on Machine Learning*, pp. 807–814.

Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Olah, C. (2015). Understanding LSTM networks.
Available at: `olah.github.io/posts/2015-08-Understanding-LSTMs`

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. and Batra, D. (2017). Grad-CAM: Visual explanations from deep networks via gradient-based localization. In: *International Conference on Computer Vision*, pp. 618–626.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Soomro, K., Zamir, A.R. and Shah, M. (2012). UCF-101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.03385*.

Tran, D., Bourdev, L., Fergus, R., Torresani, L. and Paluri, M. (2015). Learning spatiotemporal features with 3D convolutional networks. In: *International Conference on Computer Vision*, pp. 4489–4497.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u. and Polosukhin, I. (2017). Attention is all you need. In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Wang, J.R. and Nandan Parameswaran (2004). Detecting tactics patterns for archiving tennis video clips. In: *International Symposium on Multimedia Software Engineering*, pp. 186–192.

Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X. and Van Gool, L. (2017). Temporal segment networks for action recognition in videos. *arXiv preprint arXiv:1705.02953*.

Xie, S., Sun, C., Huang, J., Tu, Z. and Murphy, K. (2018). Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In: *European Conference on Computer Vision*, pp. 318–335.

Yosinski, J., Clune, J., Bengio, Y. and Lipson, H. (2014). How transferable are features in deep neural networks? In: *Advances in Neural Information Processing Systems*, pp. 3320–3328.