DEEP TEMPORAL ARCHITECTURES FOR ACTIVITY RECOGNITION

by

Todani Luvhengo

Submitted in partial fulfillment of the requirements for the degree
Master of Engineering (Computer Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

October 2017

SUMMARY

DEEP TEMPORAL ARCHITECTURES FOR ACTIVITY RECOGNITION

by

Todani Luvhengo

Supervisor(s):	Mr. H. Grobler
Department:	Electrical, Electronic and Computer Engineering
University:	University of Pretoria
Degree:	Master of Engineering (Computer Engineering)
Keywords:	Activity recognition, deep learning, recurrent neural networks, convolutional recurrent neural network, RNN, GRU, LSTM, SCGRU, LRCN

The amount of video content generated increases daily, three hundred hours of video content is uploaded to YouTube every 60 seconds[1]. There exists a need to sort, summarise, describe, categorise and retrieve video data based on the content (i.e. the activities occurring in the video). Activity recognition (i.e. automatically naming activities) is an important area for video analysis. Activity recognition has applications in robotics, video surveillance, multimedia retrieval, behaviour analysis, disaster warning systems and content-based browsing.

Automatically categorising activities given a video clip poses two main challenges, namely object detection and motion learning. An activity recognition system must detect and localise the agent as well as learn to categorise the action the agent is performing. This research hypothesises that learning models incorporating spatial and temporal aspects from video data should outperform models that learn only spatial or temporal features on activity recognition learning tasks. The above hypothesis is investigated by developing two deep learning architectures for activity recognition that learn temporally independent and dependent features respectively.minima do not exist.

[1] `https://fortunelords.com/youtube-statistics/`

A recurrent network (structurally constrained gated recurrent unit (SCGRU)) that adds contextual feature learning to gated recurrent units (GRUs) is proposed. Adding contextual features stabilises the hidden state of a GRU layer.

The approach taken to investigate activity recognition architectures in this research involved examining the architectures on four benchmark datasets and analysing the results to 1) find the best model for activity recognition, 2) examine the model's ability to learn salient temporal features, and 3) examine the model's computational complexity. SCGRU based models outperform GRU based models on the majority of the investigated activity recognition models and datasets.

LIST OF ABBREVIATIONS

3D-CNN	Three-dimensional convolutional neural network
Adam	Adaptive moment estimation
ADADELTA	Adaptive learning rate
AdaGrad	Adaptive gradient
AUC	Area under the curve
AP	Average precision
BoW	Bag-of-words
BPTT	Back propagation through time
CNN	Convolutional neural network
Conv-RNN	Convolutional recurrent neural network
Conv-GRU	Convolutional gated recurrent unit
Conv-SCGRU	Structurally constrained gated recurrent unit
CPU	Central processing unit
CRBM	Convolutional restricted Boltzmann machine
DBN	Deep belief network
DNN	Deep neural network
ELU	Exponential linear unit
FNN	Feedforward neural network
FPR	False positive rate
fps	Frames per second
GD	Gradient decent
GRU	Gated recurrent unit
GPU	Graphics processing unit
HDF5	Hierarchical data format 5
HF	Hessian-free
LRN	Local response normalisation
LSTM	Long short-term memory
LRCN	Long-term recurrent convolutional neural network
mAP	Mean average precision
MLE	Maximum likelihood estimation

MLP	Multilayer perceptron
NFL	No free lunch
NLL	Negative log likelihood
NP-hard	Non-deterministic polynomial-time hard
PR	Precision-recall
RAM	Random-access memory
RBM	Restricted Bolzmann machine
ReLU	Rectified linear unit
RNN	Recurrent neural network
ROC	Receiver operating characteristic
RPROP	Resilient propagation
RMSProp	Root mean square propagation
SCRN	Structurally constrained recurrent network
SCGRU	Structurally constrained gated recurrent unit
SGD	Stochastic gradient decent
SIFT	Scale-invariant feature transform
ST-DBN	Space-time deep belief network
TPR	True positive rate
VGG	Visual Geometry Group

TABLE OF CONTENTS

CHAPTER 1 RESEARCH OVERVIEW

1.1 INTRODUCTION

The amount of video content generated increases daily, three hundred hours of video content is uploaded to YouTube every sixty seconds[1]. There exists a need to sort, summarise, describe and retrieve video data based on the content (i.e. the activities occurring in the video). Activity recognition is the process of naming activities [1]. An activity, in this context, is a sequence of movements generated by an agent during the performance of a task. The agent can be a person, traffic, waves or landslides. The challenge is to label activities given a sequence of visual observations. Activity recognition has applications in robotics, video surveillance, multimedia retrieval, behaviour analysis, disaster warning systems and content-based browsing.

The activity recognition task requires both spatial and temporal aspects in order to contextualise the activity. Take as an example a video of a cricket bowler at the beginning of his delivery stride. A snapshot shows a man holding a ball. A sequence of snapshots reveals a man running with a ball in his hands. Increasing the number of snapshots further, shows a man running with a ball and throwing it on to the pitch (i.e. bowling). Increasing the number of snapshots further, shows a man running with a ball and bowling it to another, who strikes the ball with a bat. Cricket is the label given to the repetition of the last sequence of snapshots.

The illustration in the preceding paragraph shows that the current state of the agent, in the context of its previous actions, defines the activity category. To perform activity recognition, a system needs to recognise and locate the agent; and it also needs to track the agent spatially and temporally.

[1] https://fortunelords.com/youtube-statistics/

Functionally, activity recognition involves mapping sequential inputs $(x_1, x_2, \ldots, x_T)$ to fixed outputs (y), i.e. $(x_1, x_2, \ldots, x_T) \rightarrow y$. Environmental factors, temporal variations and dataset creation add difficulty to activity recognition [2]. In literature, activity recognition approaches are either classically orientated [1, 2, 3, 4] or deep learning orientated [5, 6, 7, 8].

Classical approaches to activity recognition generally follow a three-step framework, namely – feature extraction, activity representation and activity categorisation [3, 4]. This framework uses handcrafted features to summarise the content of a video clip and a classifier to categorise the activity of the clip. Classical approaches have achieved competitive results in activity recognition [2, 3, 4, 9, 10, 11].

The deep learning approach to activity recognition uses deep learning architectures to extract features from video data. Hinton, Osindero and Teh [12], Bengio *et al.* [13], and Ranzato *et al.* [14] demonstrated the tractability of learning deep architectures in 2006 when they achieved state-of-the-art performance across different machine learning tasks. Deep architectures learn multiple levels of representations from input data. This approach to learning has led to state-of-the-art results in learning tasks such as object recognition [15, 16, 17, 18], language modelling and generation [19, 20, 21, 22, 23], and image captioning [24, 25]. Companies such as Google, Kaggle, Facebook, Microsoft, IBM, Apple, and Baidu have invested in deep representation research. The investment has produced products such as Google Goggles, Google Voice, Watson analytic, Google Image Search and AlphaGo.

The success of deep learning architectures has led to research in deep neural network (DNN) architectures for activity recognition [5, 6, 7, 8, 26, 27, 28, 29, 30]. Deep architectures such as three-dimensional convolutional neural network (3D-CNN) [29], Space-Time Deep Belief Network (ST-DBN) [26], multi-resolution convolutional neural network (CNN) [28], two-stream CNNs [5, 7], and long-term recurrent convolutional neural network (LRCN) [6] have furthered activity recognition research and improved on the performances achieved by classical approaches.

1.2 RESEARCH OPPORTUNITY

A robust activity recognition architecture must recognise activities recorded in different settings such as cluttered scenes, changing backgrounds, scale changes, variations in agent appearance, variation in

lighting and changes in viewpoint. To achieve robustness, the architecture must learn both spatial and sequential aspects of video data.

Activity recognition research using deep learning architectures has focused on adapting spatial architectures to the task [5, 7]. Donahue *et al.* [6] combined sequential and spatial learning for activity recognition. However, this approach learns spatial (convolutional) features, which are independent in time, i.e. the spatial feature extraction at time t depends on only the current inputs (x_t). There is a research gap to investigate architectures that combine spatial and temporal learning to capture the time-step to time-step features, i.e. motion features.

1.3 RESEARCH OBJECTIVES AND HYPOTHESIS

The research hypothesis is,‘*“Deep learning architectures, which learn spatial and temporal features from video data, outperform architectures that assume temporal independence between successive frames on the activity recognition task”*.

Based on the preceding hypothesis, this research aims to develop and evaluate deep learning architectures for activity recognition; architectures that utilise both spatial and temporal properties inherent in video data. This research is empirical, in that it involves statistically analysing and evaluating the performance of different deep architectures on the activity recognition task. The empirical evaluation of the deep learning architectures aims to answer the following research questions:

1. How important is the temporal dimension to learning architectures for activity recognition?
2. Using similar deep learning architectures, what is the best architecture for activity recognition?
3. How do different deep learning architectures for activity recognition scale computationally?
4. How do different deep learning architectures for activity recognition compare in terms of optimisation?

1.4 SCOPE

Activity recognition is a complex problem that deals with naming an activity instance, even when performed by different agents under differing viewpoints, location and at varying speeds [1]. This

research focuses on activities captured from a single viewpoint assuming a static camera, and is limited to investigating deep learning approaches to activity recognition.

This research utilises CNNs, which are adept at learning tasks with spatial data [15, 16, 17, 18, 31], as the base architecture to capture the spatial aspects of the video data. Recurrent neural networks (RNNs), which are adept at learning sequential tasks [19, 20, 21, 22, 23, 24, 25], are used as the base architecture to learn the temporal aspects of the video data.

1.5 CONTRIBUTION

Previous deep learning research for activity recognition has focused on adapting spatial architectures to the task, e.g. two-stream architectures [5, 7]. This research builds upon this by extending the single-frame architecture to the temporal domain in the form of the Temporal-CNN. The Temporal-CNN architecture is optimised on sequential data in place of single frames. Donahue *et al.* [6] combined spatial and sequential architectures (i.e. LRCN) for activity recognition. The proposed convolutional recurrent neural network (Conv-RNN) architecture builds on this idea by combining sequential learning and spatial learning to capture motion information.

The gated recurrent unit (GRU) exponentially embeds previous hidden states in the context of the current input. This research proposes that exponentially embedding the current input in the context of previous inputs, will improve the performance of a GRU on sequential learning tasks. The proposed structurally constrained gated recurrent unit (SCGRU) embeds the current input in the context of previous inputs as well as previous hidden states in the context of the current input.

Constructing and optimising deep learning models is a non-trivial task. In this research, a UML framework was designed and implemented. The UML framework was developed with the aim of learning to implement DNN models and optimisation strategies. This research also introduced a framework to evaluate deep temporal activity recognition architectures.

1.6 RESEARCH OUTPUTS

The following article was submitted for publication:

- T. Luvhengo, H. Grobler, "Stabilising candidate activation in gated recurrent units (GRUs) using input context," International Journal of Automation and Computing, Submitted, October 2017.

1.7 OVERVIEW OF STUDY

A literature study was conducted to investigate the current state of the research in activity recognition, with special attention given to deep learning architectures for activity recognition. Chapter 2 provides a summary of the preceding literature study. Chapter 3 provides a detailed study of the deep learning approach to machine learning, with a particular focus on architecture composition, layer transformations, activation functions, model optimisation, optimisation problems and model regularisation.

Chapter 4 describes the proposed architectures for activity recognition (Conv-RNN and Temporal-CNN) as well as the introduced SCGRU recurrent neural network. The discussion on the computational complexity of the layer transformations is carried out in Chapter 4. The development, design and implementation of the UML framework is described in Chapter 4. Chapter 5 describes the activity recognition evaluation framework as well as activity recognit model construction and optimisation.

Chapter 6 contains a summary and discussion of the experimental results. Chapter 7 provides a detailed discussion of the results in the context of the research questions (see Section 1.3). Chapter 8 provides a summary of the work, a critical evaluation of the work and recommendations for future research.

CHAPTER 2 LITERATURE STUDY

2.1 CHAPTER OVERVIEW

This chapter provides an overview of the activity recognition learning challenge. Firstly, challenges to activity recognition are discussed. After that, four benchmark activity recognition datasets, namely UCF101 [32], HMDB51 [33], UCF11 [34] and Dynamic Scenes (Maryland) [35] are described and discussed. Then classical approaches to activity recognition are reviewed. Section 2.4 reviews deep learning architectures for activity recognition. Finally, the research gaps in activity recognition are identified and discussed.

2.2 ACTIVITY RECOGNITION

The activity recognition challenge, in the context of this research, is to automatically recognise activities from visual observations, even when performed by different agents under different viewpoints, different locations and at differing speeds. The automatic classification system uses training data to learn a model for activity recognition. This section provides a literature review on the activity recognition challenge.

2.2.1 Activity recognition challenges

2.2.1.1 Environmental factors

The environment wherein the activity occurs adds variation (e.g. occlusions, illumination and viewpoints) to the video recording. Dynamic and cluttered environments make agent localisation and

activity recognition difficult [1, 2, 3]. Such environments might occlude the agent performing the activity, and differing light conditions can change the agent's appearance [2].

A dynamic background, one caused by a moving camera, increases the complexity of localising the agent within the frame and characterising the agent's motion. Poppe [2] proposes combining multiple views, from multiple cameras, into a consistent representation as a way to approach viewpoint and occlusion issues. However, multiple views of the same activity are not readily available and combining these multiple views, when available, is a difficult problem.

This research is restricted to the recognition of activities captured from a single viewpoint. The single viewpoint must fully capture the activity, without occluding the agent. Performing data pre-processing (specifically contrast normalisation, which removes the average brightness of the images) addresses the illumination issues.

2.2.1.2 Temporal variation

The temporal aspect of an activity offers useful information for activity recognition. The temporal dimension requires activities segmented in time, which is not always the case in practice [2]. Creating annotated datasets addresses this problem, but creating them is expensive [2, 3].

The rate at which agents perform the same activity varies across different recordings. For algorithms that use motion features for activity recognition, the temporal extent of an activity is important. The temporal extent depends on the frame rate of the activity recording and the rate at which the agent performs the activity. Segmenting the video recordings at a constant time interval limit the effects of recording frames per second (fps) on algorithm performance.

2.2.1.3 Intra-class and inter-class variation

How different agents perform an activity varies from agent to agent, e.g. running movements can differ in speed and stride. An activity recognition algorithm must generalise over intra-class variations and distinguish between inter-class variations. Balancing between intra-class and inter-class variation is

subject to the amount of data available and the complexity of the algorithm, i.e. the bias-variance trade-off (see Section 3.5.1.1).

2.2.2 Activity recognition datasets

Benchmark datasets allow for the comparison of different activity recognition approaches, while also providing insights into their capabilities and limitations. This section discusses the most commonly used activity recognition benchmarking datasets, namely UCF101 [32], HMDB51 [33], UCF11 [34] and Dynamic Scenes (Maryland) [35].

2.2.2.1 UCF101 (Human actions)

UCF101 [32] is a human action dataset with 101 classes, each of which has at least 100 video clips. The dataset consists of YouTube videos recorded in unconstrained environments. The clips contain variation in camera motion, light conditions, and occlusions. The 101 classes fall into one of five types of human actions: human-object interaction, body-motion, human-human interaction, playing musical instruments, and sports.

The dataset contains 13 320 clips divided into 25 groups. Each clip group share common features such as background or actors. The clips have a frame rate of either 25 or 29.97 fps, a resolution of 320×240 pixels, and an average clip length of 7.21 seconds. Figure 2.1 shows randomly sampled frames from the dataset.

2.2.2.2 HMDB51 (Human motion database) dataset

HMDB51 [33] is a human action recognition dataset with 51 categories. The dataset contains videos sourced from films and public databases, e.g. the Prelinger archive, YouTube and Google videos. The HMDB51 dataset provides an extensive video dataset, which captures the richness and complexity of human actions [33]. Figure 2.2 shows randomly sampled frames from the dataset. The HMDB51 dataset contains 6 849 clips divided into 51 action categories, each of which contains at least 101 clips. To maintain consistency across the dataset, the video frames have a height of 240 pixels and a width

Figure 2.1. Eight randomly sampled images from the UCF101 dataset, corresponding to the different types of actions.

Figure 2.2. Eight randomly sampled images from the HMDB51 dataset, corresponding to the different types of actions.

scaled to maintain the aspect ratio of the original video clip. The action categories of the dataset fall into one of five groups:

- **General facial actions:** smile, laugh, chew, talk
- **Facial actions with object manipulation:** smoke, eat, drink
- **General body movements:** cartwheel, clap hands, climb, climb stairs, dive, fall on the floor, backhand flip, handstand, jump, pull up, push up, run, sit down, sit up, somersault, stand up,

turn, walk, wave

- **Body movements with object interaction:** brush hair, catch, draw sword, dribble, golf, hit something, kick ball, pick, pour, push something, ride bike, ride horse, shoot ball, shoot bow, shoot gun, swing baseball bat, sword exercise, throw
- **Body movements for human interaction:** fencing, hug, kick someone, kiss, punch, shake hands, sword fight

2.2.2.3 UCF11 (YouTube action)

The UCF11 [34] dataset has 1 600 videos in eleven activity categories sourced from YouTube and personal video collections. The lack of control during the capturing process resulted in the dataset having a mix of steady and shaky camera movements, cluttered backgrounds, variations in object scale, varied viewpoints, varied illumination, and low resolution. The video clips have a frame rate of 29.97 fps, and each video has one activity associated with it.

The actions in the dataset are: *basketball shooting, biking, diving, golf swing, horse riding, soccer juggling, swing, tennis swinging, trampoline jumping, volleyball spiking* and *walking*. Figure 2.3 shows randomly sampled frames from the dataset. The name of class activity lies above the image.

Figure 2.3. Eight randomly sampled images from the UCF11 (YouTube action) dataset corresponding to the different types of actions.

2.2.2.4 Dynamic scenes (Maryland)

The Dynamics Scenes (Maryland) [35] dataset contains thirteen classes collected from different video hosting websites such as YouTube. Each class contains ten video clips. The dataset has variations in illumination, background, frame rate, viewpoint, scale and camera dynamics. The variations are a result of having no control over the video capturing process. The dataset has a high intra-class variation.

The classes in this dataset are: *avalanche, boiling water, chaotic traffic, forest fire, fountain, iceberg collapse, landslide, smooth traffic, tornado, volcanic eruption, waterfall, waves* and *whirlpool*. Figure 2.4 shows eight randomly sampled frames from the dataset.

Figure 2.4. Eight randomly sampled images from the Dynamic Scene (Maryland) dataset.

The Maryland dataset is complex and realistic. Trying to categorise the scenes through static based classification methods lead to confusion of the {chaotic traffic and smooth traffic}, {avalanche and iceberg} and {waterfall and fountain} categories [35]. Similarly, categorising the scenes using only dynamic information would cause confusion among the {avalanche, landslide and volcanic eruptions}, and {tornado and whirlpool} categories [35].

2.3 CLASSICAL APPROACHES TO ACTIVITY RECOGNITION

Activity recognition algorithms fall into either classical or deep learning approaches. Classical approaches use handcrafted features, while deep learning approaches automatically learn features.

Although classical approaches have been successful when applied to the activity recognition challenge, deep learning approaches consistently out-perform classical approaches [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 36, 37].

The classical approach to activity recognition follows a three-step framework (introduced by Laptev *et al.* [3] and extended by Wang *et al.* [4]): feature extraction, activity representation and activity categorisation. The first step (feature extraction) involves extracting local visual features using dense or sparse interest points [3, 9, 10]. The local features aim to track moving objects in the video. The second step (action representation) encodes the extracted features to form a bag-of-words (BoW) histogram, which represents the entire video in a vector space. Thirdly, the vector space samples (along with their category labels) are used to construct a classifier (e.g. support vector machine (SVM)) to categorise the activities.

Local spatio-temporal video features are one of the most successful handcrafted features [3, 9, 10]. Local spatio-temporal features capture characteristic shape and motion from video clips to provide an independent representation [4], i.e. a representation that is independent of background clutter, spatio-temporal shifts, and multiple motions in the scene. Local spatio-temporal features capture shape and motion (around selected points) by measuring spatial, spatio-temporal image gradients, and optical flow. Spatio-temporal features set the early benchmarks in activity recognition [3, 4, 9, 10, 11].

2.4 DEEP LEARNING ARCHITECTURES FOR ACTIVITY RECOGNITION

Activity recognition involves mapping sequential inputs $(x_1, x_2, \ldots, x_T)$ to fixed outputs (y), i.e. $(x_1, x_2, \ldots, x_T) \rightarrow y$. An activity recognition architecture must localise the agent performing the activity, as well the activity the agent is performing. This means that the learning architecture must accept inputs that characterise the agent and activity (e.g. local spatio-temporal features) or it must infer both the location of the agent and categorise the activity from sequential inputs.

2.4.1 Overview of deep learning models for activity recognition

Deep learning approaches to activity recognition use DNN models capable of automatically learning features from raw data. DNN models have achieved state-of-the-art results in image-based recognition

tasks [15, 18, 38]. Success in image-based recognition has resulted in studies into DNN architectures for activity recognition [5, 6, 7, 8, 26, 27, 28, 29, 30].

Baccouche *et al.* [39] summarised video clips using a frame-wise BoW representation based on scale-invariant feature transform (SIFT) features and then trained a long short-term memory (LSTM) model on those features. Their model improved on the baseline of the small MICC-Soccer-Actions-4 [40] dataset.

Baccouche *et al.* [41] built on their earlier work [39] and used a 3D-ConvNet to encode the video into frame-wise feature vectors before training a LSTM model. This approach achieved state-of-the-art results on the KTH actions [35] dataset. Baccouche *et al.* [39, 41] trained two models independently, which meant separate optimisation of the feature extraction and classification steps.

Ji *et al.* [29] introduced a 3D-CNN architecture for human action recognition. Three-dimensional convolution is a result of convolving a 3D kernel with a cube formed by stacking adjacent frames together [29]. The activity recognition architecture introduced in [29], consists of a hard-wired layer, 3D-CNN layers and spatial pooling layers. The hard-wired layer encodes prior information in the form of grayscale, gradient, and dense optical flow frames. The 3D-CNN layers learn motion features. This approach did not achieve state-of-the-art results comparable to classical approaches.

Chen *et al.* [26] introduced the ST-DBN architecture, which stacks spatio-temporal convolutional restricted Boltzmann machine (CRBM) layers to extract spatio-temporal features. The ST-DBN architecture is a generative model, meaning it can be sampled to generate action sequences or provide missing frames in a video sequence. The ST-DBN is impartial to occlusions, i.e. its generative nature allows it to perform reconstructions (e.g. reconstruct the torso and legs of an agent given a view of the head) of the data and generate spatio-temporal predictions [26]. This approach did not achieve state-of-the-art results on the KTH actions [35] dataset when compared to classical methods.

Karpathy *et al.* [28] introduced a multi-resolution CNN architecture, which combined two separate streams of processing: a context stream, which uses a low-resolution image, and a fovea stream, which uses a high-resolution centre crop of the image. The architecture was investigated using a single-frame model, an early-fusion model, a late-fusion model, and a slow-fusion model of information across the temporal domain.

The single frame model accepts a single-frame as input to a CNN. The early-fusion model combines information across T frames, at a pixel level, by extending the first filters of the CNN model to have a temporal component. The late-fusion model combines outputs from two single-frame models (with shared parameters) processed using two frames that are 15 frames apart [28]. The slow-fusion model slowly fuses temporal information throughout the network to achieve global information in both spatial and temporal dimensions [28]. These architectures achieved state-of-the-art results on the UCF101 [32] and Sports-1M [28] datasets. The slow-fusion models had the best results across all datasets used.

Simonyan *et al.* [5] introduced a two-stream architecture that learns information from individual frames, as well as motion information between consecutive frames. The single-frame stream uses a pre-trained CNN (pre-trained on the ImageNet [42] dataset) to learn frame-wise features. The temporal stream uses a stack of T dense optical flow [43] frames as input to a CNN to capture motion features. The temporal stack is folded on to the channel axis. Wang *et al.* [7] build on Simonyan *et al.*'s [5] research to investigate the strategies for training two-stream architectures for activity recognition. Wang *et al.* [7] achieved state-of-the-art results on the UCF101 [32] dataset. The two-stream architecture is a combination of Karpathy *et al.*'s [28] single-frame and early-fusion models, with dense optical flow [43] inputs for the early-fusion model.

Donahue *et al.* [6] combined CNN and recurrent neural network (RNN) (specifically a LSTM) architectures to create a long-term recurrent convolutional neural network (LRCN). The LRCN architecture uses the CNN model to extract a sequential vector from a sequence of images, before passing the information to a LSTM layer to encode the sequences. The LRCN architecture achieved results on the UCF101 [32] dataset comparable to those in Simonyan *et al.* [5]. The LRCN architecture allows one to jointly optimise the CNN and LSTM models.

Based on the results and research trends in deep learning for activity recognition, this research focuses on two-stream [5, 7] and LRCN [6] architectures. The following sections provide detailed reviews of these architectures to discover the gaps in the research.

2.4.2 Two-stream convolutional architecture

To achieve activity recognition, Simonyan *et al.* [5] introduced a two-stream architecture. The two-stream architecture combines two deep hierarchical feature extraction models, one for object recognition and the other for motion recognition.

The object recognition stream uses single-frame RGB images sampled from each video clip in the dataset. The temporal/motion recognition stream uses a stacking of T-frame dense optical flow [43] images (x and y flow channels) sampled from each video in the dataset. The deep hierarchical feature extraction model used is a CNN, since it is adept at learning a hierarchy of increasingly complex and interpretable image features from images [31, 38, 44, 45, 46, 47].

The single-frame and temporal models are independently optimised. Fusion of the single-frame and temporal streams is achieved by using a two-to-one weighted linear combination of the model predictions (in favour of the optical flow model) [5]. The dense optical flow model has a higher weight because its inputs encode motion information. The scores of the two models were also combined using a SVM, which resulted in the best model performace [5].

2.4.2.1 Training and evaluating two-stream models

Simonyan *et al.* [5] and Wang *et al.* [7] sampled N frames (RGB and dense optical flow) from each video clip at 30 fps. The frame samples were linearly resized to have a width and height of 340×256. During training, the samples are cropped to have a width and height of 224×224 (Wang *et al.* [7] used multi-scale cropping). The cropping region is randomly sampled (spatially) [5], while Wang *et al.* [7] randomly chose one of the following cropping methods {centre, random, top-left, top-right, bottom-left, bottom-right} to crop each sample. The cropped samples are then horizontally flipped with a random probability of 0.5. Application of the cropping and random flipping pre-processing steps is consistent across all frames that are part of the same video clip. The samples are zero-centred using a mean image. The data augmentation strategy used by Wang *et al.* [7] resulted in better input variations and led to better results as compared to those of Simonyan *et al.* [5]. It should be noted that Wang *et al.* [7] and Simonyan *et al.* [5] used two different pre-trained models.

The single-frame model parameters (θ_v) are learned by jointly maximising the likelihood of the model's predictions ($\hat{y}$) subject to the video ground truth labels (y). Parameter optimisation results from minimising the negative log likelihood (NLL) of a sequence sampled from the training set X such that,

$$L(\hat{y}, y; \theta_v, X) = -\frac{1}{|X|} \sum_{(x,y)\in X} \log p\,(y|x, \theta_v) \tag{2.1}$$

where $|X|$ is the number of samples in the training set X. Equation (2.1) minimises the sample-wise log-likelihood of the model's prediction, which ensures that the model generalises to the training data.

At test time 25 RGB frames and T optical flow images are sampled (from each video) to test the single-frame and temporal streams respectively [5]. For each sampled frame, a further ten samples are generated through cropping and flipping the four corners and the centre of the frame. Model predictions are obtained by averaging across the sampled frames and their cropped regions for each model.

2.4.3 Long-term recurrent convolutional networks

The LRCN [6] architecture combines CNN and RNN architectures to achieve variable length sequence processing for action recognition. The LRCN architecture:

- directly maps variable-length inputs (e.g. video frames) to variable or fixed length outputs (e.g. natural language text);
- learns compositional representations in space and time, by using visual encoding (CNN) and sequential (RNN) models; and
- jointly optimises the visual encoding (CNN) and sequential (RNN) models by maximising the likelihood of the ground truth outputs (y) at each time-step t, conditioned on the input data up to time t

The LRCN architecture (see Figure 2.5) is temporally flexible and expressive, which means that the probability of overfitting when using small video datasets is high. Using a pre-trained deep hierarchical visual feature extraction model (e.g. AlexNet [15]) addresses overfitting and facilitates quick learning.

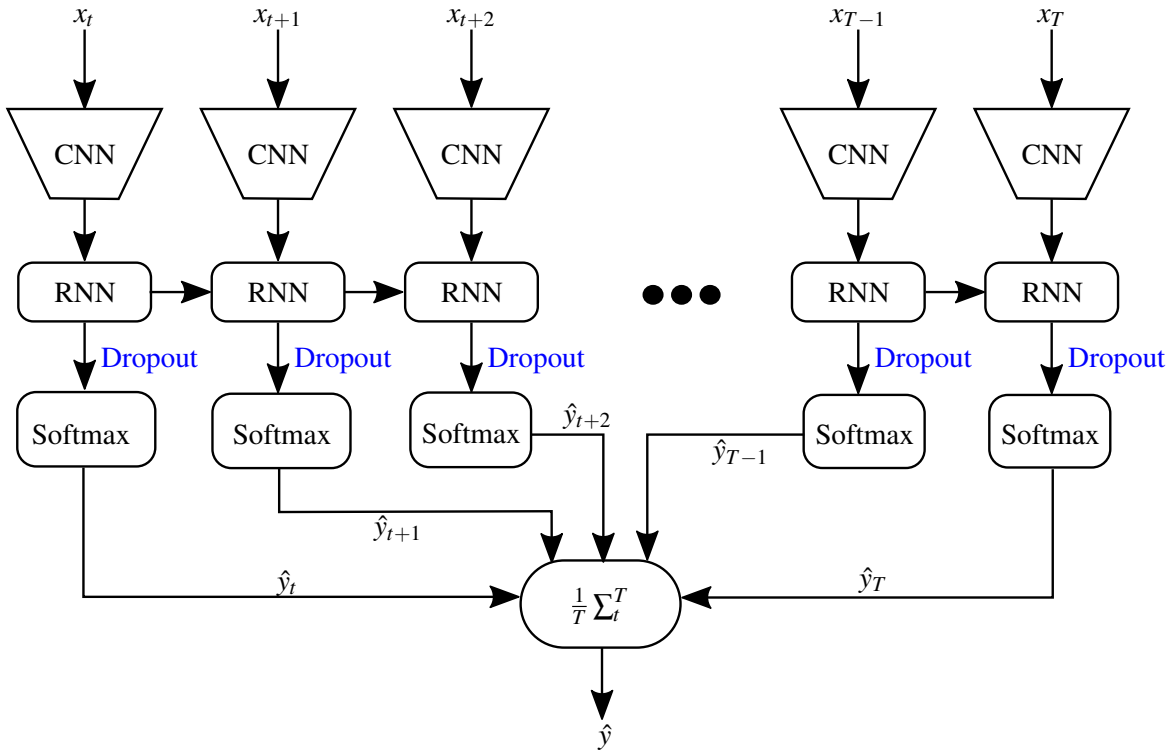


Figure 2.5. The LRCN architecture combines the strengths of CNNs for visual recognition and LSTM to process sequential images.

The LRCN architecture (see Figure 2.5) combines a deep hierarchical visual feature extraction model (CNN) with a model that learns temporal dynamics from sequential data (LSTM). Each incoming visual input (x_t) passes through a visual feature transformation $\upsilon_{\theta_v}(x_t)$, parametrised by θ_v, to produce a fixed-length vector representation $\upsilon_{\theta_v} \in \mathbb{R}^{D_h}$ at each time-step to create T feature vectors $\langle \upsilon_1, \upsilon_2, ..., \upsilon_T \rangle$ [6]. The recurrent sequential model υ_{θ_w}, parametrised by θ_w, maps the resulting visual encodings at each time-step t and hidden state h_{t-1} to an output h_t. The recurrent sequential model ensures sequentially ordered inference (on the encoded visual feature vectors) up to time T.

The LRCN model predicts a distribution $p(y_t|x_t, \theta_v, \theta_w)$ at time-step t using a softmax regression. These outputs are posterior probabilities of the activities at a given time-step, given all the inputs up to the current one, i.e. $p(y_t|x_{1:t}, y_{1:t-1}, \theta_v, \theta_w)$. The final classification for the input sample $x_{1:T}$, is a time-wise average of the predictions.

2.4.3.1 Training and evaluating LRCN models

Donahue *et al.* [6] used 16 frames (sequentially sampled from each video clip at 30 fps) as input to the LRCN models. During training, the videos were rescaled to 240×320, randomly cropped to 227×227 and then randomly flipped for data augmentation purposes. Using shorter video frames (16 frames), as opposed to using all the frames in the video clips, is a form of data augmentation [6]. At test time, 16 frame clips with a stride of 8 frames were extracted from each video. Prediction of a video clip is an average of the predictions across all 16 frame batches.

The LRCN architecture is end-to-end optimised, meaning that the parameters of the visual (θ_v) and sequential (θ_w) models are jointly optimised, unlike the architectures proposed in [39, 41]. The end-to-end optimisation strategy ensures that the visual feature extractor learns to pick out the aspects of the visual input that are relevant to the sequential classification task, activity recognition in this case.

The LRCN parameters (θ_v and θ_w) are learned by jointly maximising the likelihood of the output predictions ($\hat{y}_t$) at time t with respect to the ground truth labels (y_t) of the input data up to that point ($x_{1:t-1}, y_{1:t-1}$). To optimise the parameters (θ_v and θ_w) one minimises the NLL of a sequence sampled from the training set X such that,

$$L(\hat{y}, y; \theta_v, \theta_w, X) = -\frac{1}{|X|} \sum_{(x_t, y_t)_{t=1}^{T}} \sum_{t=1}^{T} \log p\left(y_t | x_{1:t-1}, y_{1:t-1}, \theta_v, \theta_w\right) \tag{2.2}$$

where $(x_t, y_t)_{t=1}^{T} \in X$. Equation (2.2) minimises both the temporal and sample-wise log-likelihood for the model's prediction, which ensures that the model generalises to the temporal dynamics of the samples, as well as the data generating distribution. Similar to [5, 7], Donahue *et al.* [6] train the LRCN models on both RGB and dense optical-flow inputs. The difference is that flow inputs in [6] have three channels, namely the x and y flow channels as well as a channel formed by calculating the x and y flow magnitude.

2.4.4 Data augmentation in activity recognition

Deep learning models are prone to overfitting, i.e. the model learns the peculiarities of the training data. One way to prevent overfitting is to get more training data [48, 49, 50, 51]. However, getting more data is expensive, so methods such as data augmentation are commonly employed.

Data augmentation involves applying transformations and distortion techniques to extend the data-set while preserving the label information. Augmenting datasets through transformations reduces overfitting while ensuring transformational invariance [5, 7, 8, 15, 52].

Data augmentation techniques such as cropping, scale and axis flipping reduce model overfitting in image-based classification tasks [5, 7, 8, 15]. In practice, the commonly used cropping methods are *random cropping, corner cropping* and *centre cropping*. Random cropping involves randomly choosing a position for the image-cropping window. Centre and corner cropping, involves either placing the cropping window at the centre of the image or one of the four image corners. At test time, model performance is an average of its performance on the centre and corner crops, to account for the cropping that happened during training.

Horizontal flipping involves randomly reversing the order of the horizontally aligned pixels [5, 7, 15]. Scale-augmentation involves randomly sampling the cropping window sizes from an array of different sizes [7, 8]. The cropped region is resized to the desired window size. Scale-augmentation introduces multi-scale, as well as aspect ratio augmentation. Scale augmentation builds model robustness to different object sizes. At test time, model performance is an average of its performance on the flipped and unflipped samples.

Data augmentation on large datasets results in increased memory usage. To avoid this, Krizhevsky *et al.* [15] introduced the concept of real-time data augmentation. Real-time data augmentation involves augmenting the data batch-wise (in a separate thread) while training is happening on the previous mini-batch. In this way, augmentation does not slow down training.

Through extensive experiments, Wang *et al.* [7] proposed the following augmentations to improve on the augmentation strategy used by Simonyan *et al.* [5]:

- Use centre, corner and random cropping during training. Centre and corner cropping results in better input variations to the model during training when compared to random cropping [7].
- Use multi-scale cropping by randomly sampling the cropping width and height from the set $\{256, 224, 192, 168\}$ [7, 8, 53], before rescaling the cropped samples to 224×224.

Random cropping results in selected regions with a high centre bias [7], which causes overfitting problems. Multi-scale cropping is an effective method for improving the performance of a model on object recognition [15, 54] and activity recognition tasks [5, 6, 7, 8].

In activity recognition, cropping can lead to most of the activity (or the agent performing the activity) being cropped out when the activity spans the complete frame or lies on the margin of the frame. A video mask (created through annotation or optical flow) masking out the region where the activity occurs can avoid this problem. Annotated datasets are expensive to generate, and optical flow is not possible for the datasets in this research because of camera movements.

2.4.5 Pre-trained deep learning models in activity recognition

The LRCN architecture uses a pre-trained model for frame-wise feature extraction, while single-frame architectures [5, 6, 7] fine-tune a pre-trained model on an activity recognition task. Using pre-trained models reduces overfitting and shortens training time.

Even though the input modalities for the two-stream architectures are different (RGB and dense optical flow), RGB pre-trained models are still used. The dense optical flow stream uses the same pre-trained model with the following modifications [7]:

- extract dense optical flow frames for each video and discretise them into the interval $[0, 255]$ by a linear transformation; and
- average the first layer's filter of the pre-trained model across the channel dimension, and then copy the averaged results to the model's channels.

Deep learning models for activity recognition use high dropout [55] rates (applied to the fully connected layers) [5, 6, 7, 8] to prevent hidden units from co-adapting. The use of pre-trained models has led to state-of-the-art results in deep learning models for activity recognition [6, 5, 7, 8]. Wang *et al.* [7] improved on the results achieved by Simonyan *et al.* [5] by using a better pre-trained model for RGB and dense optical flow inputs, as well as introducing better training heuristics for the models.

2.5 LITERATURE SUMMARY

Environmental factors (e.g. occlusions), temporal variation, intra-class and inter-class variations make automatic activity recognition difficult (see Section 2.2.1). Data pre-processing and using a single view point address environmental factor problems. The activity recognition learning model must differentiate between inter-class variation and generalise over intra-class. Creating annotated activity recognition datasets is difficult because different agents perform activities at different rates.

Benchmarking activity recognition learning models on complex datasets provides insight into model capabilities and limitations. Section 2.2.2 discussed four commonly used datasets, namely UCF101 [32], HMDB51 [33], UCF11 [34] and Maryland [35] – to benchmark the performances of the proposed learning models.

Section 2.4 provided a literature review on deep learning architectures for activity recognition. The literature review revealed two-stream [5, 7] and LRCN [6] architectures to be the most successful deep architectures in activity recognition. Both architectures use pre-trained object recognition models (to reduce overfitting), and are end-to-end optimised using RGB and optical flow inputs.

Data augmentation helps DNN models avoid overfitting by increasing the number of training data through transformation and distortion techniques. Section 2.4.4 discussed the different data augmentation strategies to improve the performance of DNN models for activity recognition.

2.6 RESEARCH GAP IN ACTIVITY RECOGNITION

The two-stream [5, 7] and LRCN [6] architectures achieve different results on the UCF101 [32] dataset. The two-stream models achieve accuracies of 72.8% on RGB inputs and 81.2% on flow inputs, whereas the LRCN models achieved accuracies of 68.19% and 77.46% respectively.

Possible causes of the disparity in performance between the two architectures include using two different pre-trained models; the visual encoding model (LRCN case) not extracting temporally coherent encodings; optimisation problems, specifically problems regarding the vanishing and exploding gradients [56] often associated with RNN architectures. This research aims to investigate:

- whether changing the temporal sampling method from sequential to equally spaced samples will improve on the results in [6];
- whether one will be able to improve performance on image base sequential learning tasks (specifically activity recognition) by combing the convolutional and sequential tasks into one layer to form a Conv-RNN architecture (see Chapter 4); and
- when using equivalent models what is the best deep learning architecture for activity recognition?

CHAPTER 3 DEEP LEARNING

3.1 CHAPTER OVERVIEW

This chapter provides a detailed study of the deep learning approach to machine learning. Firstly, an overview of deep learning, with an emphasis on motivations and optimisation, is presented. After that, a discussion on of the main transformation layers and activation functions is carried out. Section 3.4 discusses deep learning model optimisation, focussing on gradient descent algorithms and optimisation challenges. Finally, Section 3.5 presents regularisation strategies to avoid model overfitting.

3.2 DEEP LEARNING: AN OVERVIEW

The central idea of representation learning is, "deep learning models are exponentially more efficient at representing functions than shallow models" [57]. Deep learning architectures are capable of learning complicated functions that model high-level abstractions [57, 58]. The seminal publications [12, 13, 14] introduced and validated the greedy layer-wise algorithm to train deep models. The greedy layer-wise training strategy, which involves training each layer (of a model) separately, followed by fine-tuning the learned model, made it possible to train a deep belief network (DBN) efficiently.

Bengio *et al.* [13] furthered the research of Hinton, Osindero and Teh [12] by extending the restricted Boltzmann machine (RBM) to handle continuous-valued inputs. Their research provided further experiments, which demonstrated the effectiveness of the greedy layer-wise algorithm in optimising deep learning networks (specifically autoencoders and RBMs) [13]. The greedy layer-wise strategy works better when applied to unsupervised learning tasks than when applied to fully supervised learning

tasks [13]. Ranzato *et al.* [14] used sparse over-complete representations to initialise the first layer of a CNN, to achieve the then best error rate on the MNIST [59] dataset.

Simple computational layers are the compositional elements of deep architectures. The layer-to-layer mapping of a deep architecture is,

$$\upsilon(x) = \upsilon^{(L)}\left(\upsilon^{(L-1)}\left(\cdots \upsilon^{(1)}(x)\right)\right) \tag{3.1}$$

where, L is the number of layers and $\upsilon^{(i)}(\cdot)$ is the i-th layer mapping function. Bengio [57] showed that deep learning models extract abstract data representations, from low-level data representation. Representation learning algorithms transform input data into a form that makes learning supervised tasks easier.

3.2.1 Data representation in deep architectures

The performance of a learning algorithm depends on the choice of data representation, as such most of the effort in applying machine learning algorithms goes into extracting or generating features (data representations) to simplify the learning task. Deep architectures are efficient at extracting and disentangling gradually higher-level factors of variation that characterise the distribution of the input data [57].

The No Free Lunch (NFL) theorem [60] states that no learning algorithm is inherently superior to another. If an algorithm performs better than another on a particular problem, it is because the algorithm exploited a set of assumptions suitable for the problem at hand. Such an algorithm will not perform as well on problems where the assumptions do not hold. The NFL theorem suggests that for an algorithm to learn effectively, it has to discover the significant set of assumptions to solve a particular machine learning task. Based on the NFL theorem, a universal theory of learning is a theory of assumptions [61].

In deep learning, the NFL problem is partially addressed through transfer learning [5, 6, 7, 14, 62, 63, 64, 65]. Transfer learning involves the use of knowledge gained while solving one problem, to solve a different one, which is in the same domain. In deep learning, this involves transplanting the feature extracting layers from one model to another and fine-tuning the resultant model for the task at hand.

3.2.2 Maximum likelihood estimation

The deep learning models relevant to this research use maximum likelihood estimation (MLE) to learn from data. The MLE approach to machine learning requires that one define a probabilistic model, characterised by a set of learnable parameters θ. The learning model defines a probability distribution $p(X;\theta)$ over some training dataset X. Using statistical estimation over the distribution $p(X;\theta)$, an estimate of the optimal values of θ (values that best model the data X) result from of maximising the distribution $p(X;\theta)$,

$$\begin{aligned} \hat{\theta} &= \underset{\theta}{\operatorname{argmax}} \prod_i p(x^i;\theta) \\ &= \underset{\theta}{\operatorname{argmax}} \sum_i \log p(x^i;\theta) \end{aligned} \tag{3.2}$$

The MLE approach to machine learning aims to maximise the probability that the learning model $p(X;\theta)$ will generate the training data. The *log-likelihood* is convenient to work with because of its monotonicity and because the product $p(x^i;\theta) \in [0,1]$ can result in numerical underflow.

Given limited data, MLE is not always the best possible optimisation strategy. Bayesian inference is a better choice over MLE when presented with finite data [48, 49, 66], because the *prior* $p(X)$ is not discarded during optimisation. In Bayesian inference, predictions result from integrating over all possible values of θ, which is an intractable problem in certain applications. MLE and Bayesian inference are equivalent when the *prior* distribution $p(X)$ is constant. Adding a *regulariser* to MLE estimation (3.2), introduces a set of learnable parameters to act as a *prior* during MLE. However, this approach assumes that optimal penalty is a known parameter. Bayesian inference assumes the penalty is an unknown parameter.

Much of machine learning has to do with optimisation. Under the MLE strategy, one can define a cost/objective function,

$$J(X;\theta) = -\sum_i \log p(x^i;\theta) \tag{3.3}$$

where the objective is to minimize $J(X;\theta)$ subject to θ. Analytically solving $\dfrac{\partial}{\partial\theta} J(X;\theta) = 0$ for θ, is sometimes tractable. However, in the majority of cases, a solution is not possible by analytic means. In cases like these model optimisation uses iterative methods such as gradient descent (GD).

3.3 DEEP LEARNING ARCHITECTURES

Deep learning has led to state-of-the-art results in learning tasks such as object recognition [15, 16, 17, 18], language modelling and generation [19, 20, 21, 23, 22], image captioning [24, 25], and action recognition [6, 5, 7, 8, 26, 27, 28]. Multilayer perceptron (MLP), CNN and RNN are the three main architectures responsible for the majority of progress in this area.

3.3.1 Convolutional neural networks

Digital images display the spatial relationships that exist between objects in a scene through pixels. To learn these spatial relationships, LeCun and Bengio [59] introduced the CNN architecture. CNNs are adept at learning a hierarchy of increasingly complex features from images [15, 31, 44, 45]. This capability has led to state-of-the-art results in image classification [15, 18, 38, 54], localisation [18], and activity recognition [5, 6, 7, 27]. CNN layers extract features, which are invariant to spatial shifts.

3.3.1.1 Spatial convolution

A convolution operation, the central operator in a CNN, is an integral that expresses the amount of overlap one function g has when shifted over another function f,

$$y = f * g \tag{3.4}$$

Given an $n_1 \times n_2$, single channel image I and kernel/filter $k \in \mathbb{R}^{2h+1 \times 2w+1}$, the discrete convolution between the two is given by:

$$(I * k)_{s,r} = \sum_{u=-h}^{h} \sum_{u=-w}^{w} k_{u,v} \cdot I_{r+u,s+v} \tag{3.5}$$

where s, r represents a pixel in image I, which is the centre pixel of the kernel k.

A rule specifying the convolution at the border, such as *valid*, *same* and *full*, helps avoid overlap of the kernel with non-existent pixels. Full convolution pads the image to ensure that the kernel also operates on the edge pixels. Valid convolution ignores the edge pixels so that the entire kernel falls within the image. Same convolution performs a *valid* convolution on the image padded with zeros, such that the resulting image has the same dimension as the input.

The convolutional layer, the core building block of a CNN, performs convolution on the input image using multiple filters. The layer outputs D_h *feature maps*, which result from convolving D_h kernels with the input feature maps. Layer ℓ convolves the current layer's learnable filters with the previous layer's feature maps before applying a nonlinearity $\phi(\cdot)$ to the feature maps.

Let ℓ be the current convolutional layer. The input into layer ℓ consists of $m^{\ell-1}$ feature maps of size $m_1^{\ell-1} \times m_2^{\ell-1}$. The current layer ℓ outputs m^ℓ feature maps of size $m_1^\ell \times m_2^\ell$. The i-th feature map in layer ℓ is then given by:

$$\begin{aligned} \hat{h}_i^\ell &= b_i^\ell + \sum_{j=1}^{m^{\ell-1}} x_j^{\ell-1} * k_{i,j}^\ell \\ h_i^\ell &= \phi\left(h_i^\ell\right) \end{aligned} \tag{3.6}$$

where b_i^ℓ is the bias matrix and $k_{i,j}^\ell$ is filter kernel of size $2h_1+1 \times 2h_2+1$ that connects the j-th feature map in layer $\ell-1$ to the i-th feature map in layer ℓ [31, 45]. The output maps have size $m_1^\ell \times m_2^\ell$ calculated as follows:

$$\begin{aligned} m_1^\ell &= m_1^{\ell-1} - 2h_1^\ell \\ m_2^\ell &= m_2^{\ell-1} - 2h_2^\ell. \end{aligned} \tag{3.7}$$

CNNs implicitly extract the relevant features, by restricting the parameters to a local *receptive field.* A local *receptive field* is a sub-region of the input, which has the same size as the kernel/filter. Receptive fields allow one to apply the same elementary feature detectors across the entire image. This form of parameter sharing is useful when dealing with high dimensional inputs such as images. A receptive field size 3×3 is optimal when designing deep models for object detection [18, 38, 54, 67].

3.3.1.2 Spatial pooling

Spatial pooling progressively reduces the spatial size of feature maps, to build robustness to small distortions, control overfitting, and reduce the amount of computation in the network [31].

Let ℓ be a pooling layer. The layer outputs $m^\ell = m^{\ell-1}$ feature maps of reduced size. Spatial pooling requires two parameters, the spatial extent (f) and stride (s). The layer accepts feature maps ($x_i^{\ell-1}$) of size $m^\ell \times w_1 \times h_1$, and performs a downsampling operation,

$$x_i^\ell = \texttt{down}(x_i^{\ell-1})\big|_{s,f} \tag{3.8}$$

to produces feature maps of size where $m^{\ell} \times w_2 \times h_2$, where $w_2 = (w_1 - f)/s + 1$ and $h_2 = (h_1 - f)/s + 1$. The subsampling function $(\texttt{down}(\cdot))$ is either a max or averaging operation. The max pooling operation selects the maximum value that falls within the spatial window defined by $f \times f$; average pooling sums over each distinct $f \times f$ receptive block in the input feature map to produce the output feature map. Max pooling outperforms averaging in visual tasks [31, 45]; spatial extent and stride configurations of $\{f = 3,\ s = 2\}$ and $\{f = 2,\ s = 2\}$ result in the best performance [31, 38, 45, 54, 67].

3.3.2 Recurrent neural networks

Recurrent neural networks (RNNs) are the model of choice when learning sequential tasks, be it machine translation [68, 69], handwriting recognition [20], speech recognition [70], or language analysis [22, 71]. RNNs are scalable deep learning models capable of capturing the dynamics of sequential data through recurrent cycles in the network. Essentially, RNNs are MLP networks with recurrent connections. The parameters (θ) of a RNN are temporally tied, which allow the network to form generalisations from inputs of different sequential lengths.

Figure 3.1 illustrates an RNN architecture unrolled in time. The output at each time-step, depends on the inputs (x_t) and previous state (h_{t-1}). The activation function $(\phi(\cdot))$ maps the linear layer transformation to a nonlinear space.

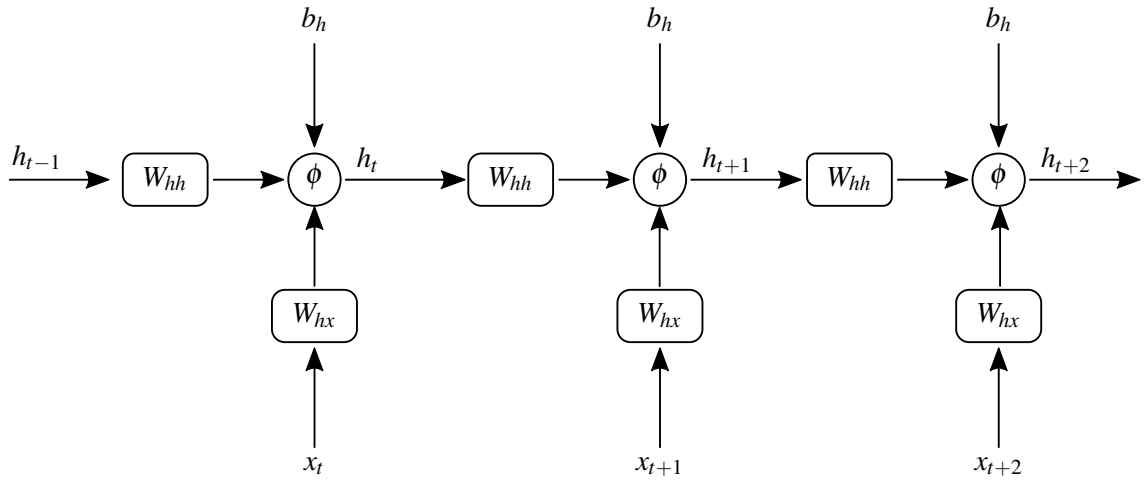


Figure 3.1. A recurrent neural network (RNN) unfolded across time-steps.

RNNs (see Figure 3.1) model the temporal dynamics of input sequences to form the present hidden state as follows:

$$\begin{aligned} \tilde{h}_t &= w^{(hx)} x_t + w^{(hh)} h_{t-1} + b^{(h)} \\ h_t &= \phi\left(\tilde{h}_t\right) \end{aligned} \tag{3.9}$$

where $x_t \in \mathbb{R}^{D_d}$ is the input vector at time t; $w^{(hx)} \in \mathbb{R}^{D_h \times D_d}$ is the weight matrix that conditions the input vector, x_t; $w^{(hh)} \in \mathbb{R}^{D_h \times D_h}$ is the weight matrix used to condition the previous hidden state, h_{t-1}; $b^{(h)} \in \mathbb{R}^{D_h}$ is the bias offset; and $\phi(\cdot)$ is the layer activation function. D_h is the number of hidden units in the layer and D_d is the number of input features.

Learning long-term dynamics using RNNs is a difficult task because of the vanishing and exploding gradient problem [56] (see Section 3.3.5), which results when propagating the gradients through many time-steps. The memory of a RNN is exposed, i.e. the memory is written to at each time-step (3.9), making it difficult to capture long-term dependencies.

A way to restrict when the network's memory gets updated is to add gates to the state update equations. Gating controls how the current input and previous state influence the current state. Instead of rewriting the memory at each time-step, gated recurrent networks keep the content and add the new content on top of that, which allows specific features from the input stream to persist without overwriting. The gating parameters, which depend on the current input and previous state, are learned during model optimisation.

Chung *et al.*'s [22] results, based on experiments using polyphonic music and raw speech datasets, demonstrated the superiority of gated units, i.e. LSTM [72, 73, 74] and GRU [21, 22, 71]. The performance of GRUs and LSTMs on polyphonic music and raw speech datasets is similar [22]. Based on these results, this research focuses on gated recurrent architectures for activity recognition.

3.3.2.1 Long short-term memory recurrent networks

The LSTM architecture, introduced in 1996 by Hochreiter and Schmidhuber [72], is a gated recurrent neural network. The LSTM architecture (see Figure 3.2) consists of three recurrent gates (input gate, forget gate and output gate) and an internal *cell state*.

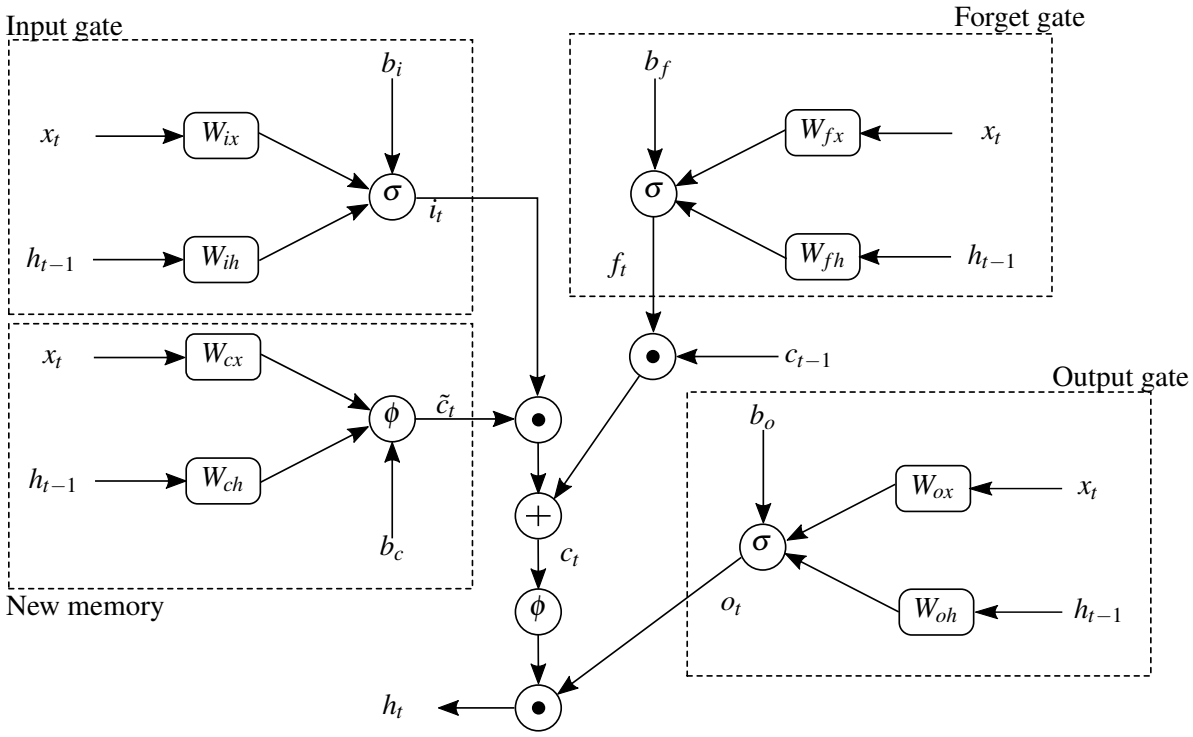


Figure 3.2. Internal state transition of a long short-term memory (LSTM) recurrent unit. Adapted from [75].

Recurrent gates allow the LSTM's memory cell to store and access information over an extended period. The input gate determines whether new memory ($\tilde{c}_t$) is worth preserving. The forget gate determines whether the previous cell state (c_{t-1}) should be retained or forgotten. The output gate determines what part of the current memory cell (c_t) is present in the current hidden state (h_t). In contrast to RNNs (3.9), a LSTM unit, through the gates, can control whether to keep the existing memory.

The LSTM state transition equations are given by:

$$i_t = \sigma\left(w^{(ix)}x_t + w^{(ih)}h_{t-1} + b^{(i)}\right) \tag{3.10}$$

$$f_t = \sigma\left(w^{(fx)}x_t + w^{(fh)}h_{t-1} + b^{(f)}\right) \tag{3.11}$$

$$o_t = \sigma\left(w^{(ox)}x_t + w^{(oh)}h_{t-1} + b^{(o)}\right) \tag{3.12}$$

$$\tilde{c}_t = \phi\left(w^{(cx)}x_t + w^{(ch)}h_{t-1} + b^{(c)}\right) \tag{3.13}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{3.14}$$

$$h_t = o_t \odot \phi(c_t) \tag{3.15}$$

where $\phi(\cdot)$ is the activation function; $\sigma(\cdot)$ is the gate/switching activation function (eg. *sigmoid*); $x_t \in \mathbb{R}^{D_d}$ is the input vector at time t; $w^{(:x)} \in \mathbb{R}^{D_h \times D_d}$ are the rectangular input weight matrices; $w^{(:h)} \in \mathbb{R}^{D_h \times D_h}$ are the square recurrent weight matrices; and $b \in \mathbb{R}^{D_h}$ represents the bias vector.

Equations (3.10) – (3.15) are the LSTM state transition equations, with the following operational functions:

- **Input gate** (3.10): The input gate uses the current input (x_t) and previous hidden state (h_{t-1}) to determine how much of the new memory ($\tilde{c}_t$) is worth preserving.
- **Forget gate** (3.11): The forget gate provides a way for the memory cells to reset themselves [73], by determining whether the previous cell state (c_{t-1}) is useful in the computation of the memory cell (c_t). Initialising the forget gate bias parameter $\left(b^{(f)}\right)$ with values close to 1, helps to learn long-term dependencies [76].
- **Output gate** (3.12): The output gate separates the memory cell from the hidden state. The memory cell state (c_t) holds much information, some of which is unnecessary for the hidden state update. The output gate determines which parts of the memory (c_t) are present in the hidden state (h_t).
- **New memory** (3.13): The current input (x_t) and previous hidden state (h_{t-1}) combine to generate the new cell memory ($\tilde{c}_t$).
- **Memory cell** (3.14): The internal state of the LSTM memory cell gets updated based on the output of a recurrent edge (controlled by the forget gate) and block input (controlled by the input gate). The sum of these two results produces the current cell state (c_t).

The LSTM architecture discussed in previous paragraphs does not include peep-hole connections [73], i.e. memory cell to gate connections. Peep-hole connections enable the LSTM to learn tasks that require precise timings and internal states, by controling information flow from the previous cell state (c_{t-1}) and the recurrent gates [70, 73, 77]. The temporal distance between events of an activity, activities investigated in this research, does not convey essential categorical information, and as such do not require precise timing.

3.3.2.2 Gated recurrent units

The gated recurrent unit (GRU), introduced in 2014 by Cho *et al.* [21], is a gated recurrent neural network. The GRU architecture (see Figure 3.3) consists of an update gate (z_t), a reset gate (r_t) and two interface ports (the current activation (h_t) and candidate activation $(\tilde{h}_t)$).

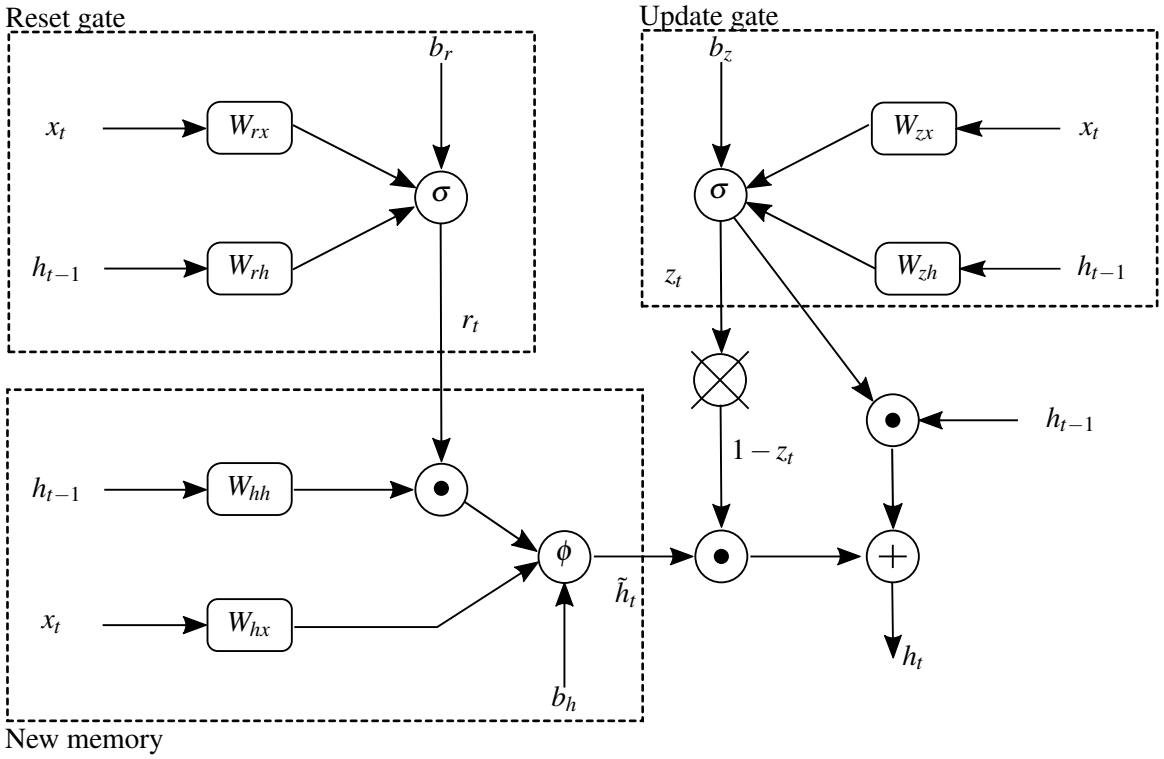


Figure 3.3. Internal state transition of a gated recurrent unit (GRU). Adapted from [75].

The reset gate uses the current input (x_t) and the previous hidden state (h_{t-1}), to control whether the previous hidden state (h_{t-1}) is important when computing the new memory. The update gate determines the relative importance of the previous hidden state (h_{t-1}) and new memory $(\tilde{h}_t)$ to the current state (h_t) [71].

If the reset gate's values were all set to ones and the update gate's values were all set to zeros, the architecture becomes a RNN. If the update gate's values are close to one, the previous state (h_{t-1}) gets copied to the current state (h_t), which makes it possible for the GRU to remember information for an extended period. The update gate (z_t) and reset gate (r_t) perform similar functions as the LSTM input gate and forget gate.

The GRU state transition equations are given by:

$$z_t = \sigma\left(w^{(zx)}x_t + w^{(zh)}h_{t-1} + b^{(z)}\right) \tag{3.16}$$

$$r_t = \sigma\left(w^{(rx)}x_t + w^{(rh)}h_{t-1} + b^{(r)}\right) \tag{3.17}$$

$$\tilde{h}_t = \phi\left(w^{(hx)}x_t + w^{(hh)}\left(r_t \odot h_{t-1}\right) + b^{(h)}\right) \tag{3.18}$$

$$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1} \tag{3.19}$$

where $\phi(\cdot)$ is the activation function; $\sigma(\cdot)$ is the gate/switching activation function; $x_t \in \mathbb{R}^{D_d}$ is the input vector at time t; $w^{(:x)} \in \mathbb{R}^{D_h \times D_d}$ are the rectangular input weight matrices; $w^{(:h)} \in \mathbb{R}^{D_h \times D_h}$ are the square recurrent weight matrices; and $b \in \mathbb{R}^{D_h}$ are the bias vectors.

Equations (3.16) – (3.19) are the GRU state transition equations, with the following operational functions:

- **New memory** (3.18): The new memory ($\tilde{h}_t$) is a consolidation of the current input (x_t) with the previous hidden state (h_{t-1}).
- **Reset gate** (3.17): The reset signal (r_t) determines how important the previous state (h_{t-1}) is when computing the new memory, making it possible to discount previous memories.
- **Update gate** (3.16): The update signal (z_t) determines the relative importance of previous state (h_{t-1}) and new memory ($\tilde{h}_t$) to the current state (h_t). This makes it possible for the GRU to discount or remember long sequences.
- **Hidden state** (3.19): The current state (h_t) is a result of exponentially averaging the past hidden state (h_{t-1}) and the new memory ($\tilde{h}_t$), using the update signal (z_t) as the smoothing factor.

The GRU architecture is simpler, faster (has fewer parameters to update) and optimises quicker than the LSTM architecture [76]. Jozefowicz *et al.* [76] found that the GRU architecture outperforms the LSTM on most tasks except language modelling. To this end, the GRU architecture forms the basis of the experiments of this research.

3.3.3 Stacking recurrent neural networks

The intuitive way to extend RNNs spatially is through stacking multiple layers on top of each other. Stacking RNNs in this manner leads to a model that is deep in both space and time. The stacked RNN

has multiple levels of transitions defined by:

$$\begin{aligned} h_t^\ell &= \upsilon_h^\ell \left(h_t^{\ell-1}, h_{t-1}^\ell \right) \\ &= \phi \left(w^{(h^\ell h^{\ell-1})} h_t^{\ell-1} + w^{(h^\ell h^\ell)} h_{t-1}^\ell + b^{(h^\ell)} \right) \end{aligned} \tag{3.20}$$

where h_t^ℓ is the hidden state of the ℓ-th layer at time t. The hidden states of all the levels are recursively computed from the bottom level.

Stacking RNN layers using (3.20) makes training difficult, because of the increased number of processing steps (between the bottom of the model and the top) and the vanishing gradient problem [56]. To decrease the number of processing steps between the top and bottom layers in stacked RNNs, Graves [20] introduced the concept of *skip connections*. Skip connections connect the inputs to all hidden layers as well as all hidden layers to the outputs. Stacking RNN layers with skip connections is defined by:

$$\begin{aligned} h_t^\ell &= \upsilon_h^\ell \left(x_t, h_t^{\ell-1}, h_{t-1}^\ell \right) \\ &= \phi \left(w_t^{(h^\ell x)} x_t + w_t^{(h^\ell h^{\ell-1})} h_t^{\ell-1} + w^{(h^\ell h^\ell)} h_{t-1}^\ell + b^{(h^\ell)} \right) \end{aligned} \tag{3.21}$$

where the matrix $\left(w_t^{(h^\ell x)} \right)$ connects the input x_t to layer ℓ. Skip connections increases the number of parameters to optimise, which is a problem when processing resources are a constraint.

Skip connections make training deep recurrent networks more accessible, by reducing the number of processing steps from the bottom to the top layer, thereby addressing the vanishing gradient problem [20]. Gated recurrent networks avoid the vanishing gradient problem by using additive instead of multiplicative recurrent dynamics (the memory cell (c_t) in LSTMs and hidden state (h_t) in GRUs). This research is restricted to gated recurrent neural networks, which do not experience the vanishing gradient problem as much as simple RNNs (see Section 3.3.2), and as such RNNs are stacked using the formulation in (3.20).

3.3.4 Activation functions

Activation functions transform the hidden states of a layer to highly nonlinear space. Using nonlinear transformations MLP models can in principle approximate any smooth function, with a degree of accuracy that depends on the number of hidden units used [78]. The choice of activation function affects the performance of a deep learning model.

3.3.4.1 Sigmoid activation

The sigmoid activation transforms the input (x) to the range $(0,1)$. Large negative numbers approach zero, while large positive numbers approach one. The sigmoid activation is given by:

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{3.22}$$

The sigmoid nonlinearity has fallen out of favour because of the following drawbacks:

- If the local gradient is small, no signal flows through the unit, which effectively shrinks the gradient, i.e. the parameters receive small to no updates (vanishing gradient problem).
- The narrow range of the activation means that one must be cautious when initialising the parameters, to prevent the units from saturating.
- The sigmoid output is not *zero-centred*, which is undesirable since units in the later layers of a model receive data that is not zero-centred.
- When used in deep networks, the activation of the last hidden layers saturate, while the activation at the lower layers remain constant [79].

In DNN models, the sigmoid function is used as a switch [72, 77, 80] (in gated recurrent networks) or as the output of a classification architecture [15]. Ioffe and Szegedy [17] showed that if one used batch normalisation at every layer the sigmoid activation can be used in DNN models.

3.3.4.2 Hyperbolic tangent activation

The hyperbolic tangent activation function scales the input to the range $(-1,1)$, and is given by:

$$\phi(x) = \tanh(x) \tag{3.23}$$

Like the sigmoid activation, the hyperbolic tangent activations saturate. However, hyperbolic tangent networks do not suffer from the same saturation behaviour observed with sigmoid networks, because of the function's symmetry at zero [79]. The hyperbolic tangent activation outperforms sigmoid activation function [31, 69, 76].

3.3.4.3 Rectified linear unit activation

The rectified linear unit (ReLU) activation function, first proposed by Nair and Hinton [81], is given by:

$$\phi(x) = \max(0, x) \tag{3.24}$$

The ReLU introduces a level of sparsity to the units, and it converges faster than the sigmoid and tanh functions [15]. ReLUs alleviate the vanishing gradient problem [56] because the derivative is one for activations greater than zero. The ReLU activation propagates the full gradient to the previous layer if the unit was active in the forward pass. A large gradient flowing through a ReLU unit can cause the parameters to update in such a way that the unit will never activate again, the so called "dying ReLU problem" [82, 83].

3.3.4.4 Exponential linear unit activation

The exponential linear unit (ELU) [84] activation function is an attempt to fix the "dying ReLU problem" [84]. Instead of the activation function being zero when the input $x \leq 0$, ELU activation allows saturation to negative values [84]. The ELU activation is given by:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right), & \text{if } x \leq 0 \end{cases} \tag{3.25}$$

where $\alpha > 0$ controls the saturation values for negative inputs. By allowing negative values, ELUs pushes the mean activations towards zero, which allows faster learning by moving the gradient closer to the natural gradient [17, 84].

3.3.4.5 Softmax activation function

The *softmax* activation function converts inputs into a posterior probability, which provides a measure of certainty [48]; this property is useful when dealing with categorisation problems where one knows the complete set of categories. The *softmax* function has the form:

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j \in C} \exp(x_j)} \text{ such that } \sum_i \phi_i(x) = 1 \text{ and } \phi_i(x) > 0 \tag{3.26}$$

where C are number of elements of x_i. The activation takes a vector of arbitrary real-valued scores (x) and squashes it to a vector of values between zero and one.

3.3.4.6 Choosing an activation function

The use of rectified linear activation units, such as ReLUs units has led to state-of-the-art results in different machine learning tasks [15, 18, 38]. Jarrett *et al.* [31] singled out the ReLU activation as the single most important factor in improving the performance on their recognition architectures.

For small datasets, using the ReLU activation can be even more important than learning the parameters of the hidden layers [31]. The major drawback of the ReLU activation is the "dying ReLU problem" [82, 83]. To fix the "dying ReLU problem", leaky-ReLUs [82] and ELUs [84] allow small activations for negative inputs to ensure that the gradients pass all the time. Clevert *et al.*'s [84] experiments on image-based learning tasks, show ELUs outperform ReLUs and leaky-ReLUs. Models in this research use ELU activation, unless stated otherwise.

3.3.5 The vanishing and exploding gradient problem

Deep learning models propagate inputs from one layer to the next (from one time-step to the next in case of RNNs). Gradients propagated through multiple stages tend to either vanish or explode [56, 85]. The vanishing and exploding gradient problem [56] is more prevalent in recurrent models than in deep models.

Consider a RNN network in (3.9) and a cost $J = \sum_{t=1}^{T} J_t$ that measures the model's performance on a learning task. The RNN error on the task is an accumulation of the temporal errors, $\partial J/\partial \theta$:

$$\frac{\partial J}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial J_t}{\partial \theta} \tag{3.27}$$

The error at each time step results from applying the chain rule differentiation up to time-step t:

$$\frac{\partial J_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial J_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \theta} \tag{3.28}$$

where $\partial h_t/\partial h_k$ is the partial derivative of the current state (h_t) subject to all the previous k hidden states. The expression $\partial h_t/\partial h_k$ can be expanded through chain rule differentiation over all hidden layers within the $[k,t]$ interval as follows:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W_{hh}^{\top} \times diag\left[g'(h_{j-1})\right] \tag{3.29}$$

The $\partial h_t/\partial h_k$ term links the errors in time from step t back to step k [85]. Long-term error contributions refer to components for which $k \ll t$ and short-term contributions refer to everything else.

The norm of the partial gradient $\partial h_t/\partial h_k$ is bounded by the product of the two norms $\left(\gamma_w = \|w_{hh}^{\top}\|\right)$ and $(\gamma_h = \|diag\left[g'(h_{k-1})\right]\|)$ through the relationship,

$$\left\|\frac{\partial h_k}{\partial h_{k-1}}\right\| = \|w_{hh}^{\top}\|\|diag\left[g'(h_{k-1})\right]\| = \gamma_w\gamma_h, \ \ \forall k \tag{3.30}$$

The norm of $g'(h_{k-1})$ can be as large as the max value of the activation function's gradient. The temporal component error is bounded by:

$$\left\|\frac{\partial h_t}{\partial h_k}\right\| = \left\|\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\right\| = (\gamma_w\gamma_h)^{t-k} \tag{3.31}$$

Small values of $(\gamma_w\gamma_h)^{t-k}$ cause the 'vanishing gradient problem', while large values cause the 'exploding gradient problem'.

3.3.5.1 Vanishing gradient problem solution

The vanishing gradient problem refers to the behaviour where long-term components approach a norm of zero. To address the vanishing gradient problem, Le *et al.* [86] initialise the hidden-to-hidden parameters using an identity matrix. Rectified linear units (e.g. ReLUs and ELUs) allow the gradient to backpropagate without getting attenuated. As stated in Section 3.3.3, gated recurrent networks avoid the vanishing gradient problem by using additive instead of multiplicative recurrent dynamics (the memory cell (c_t) in LSTMs and hidden state (h_t) in GRUs).

3.3.5.2 Exploding gradient problem solution

A large increase in the $\|diag\left[g'(h_{j-1})\right]\|$ term during training causes the exploding gradient problem. Norm gradient clipping is a heuristic to clip the gradients whenever they explode [85]. Algorithm 1 shows the pseudocode of the norm-clipping algorithm. The norm-clipping algorithm requires that one choose the hyperparameter $threshold$, which is task dependent. In practice, the heuristic $threshold < 15$ works sufficiently.

Algorithm 1 Pseudocode for norm gradient clipping

Require: parameter gradient $\frac{\partial J}{\partial \theta}$, norm $threshold$.

1: $\hat{g} \leftarrow \frac{\partial J}{\partial \theta}$
2: **if** $\|\hat{g}\| \geq threshold$ **then**
3: $\quad \hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|}\hat{g}$
4: **end if**

3.4 OPTIMISING DEEP LEARNING ARCHITECTURES

Optimising deep learning architectures is a non-trivial task, with many challenges, most notably the vanishing and exploding gradient problems [56]. The vanishing and exploding gradient problems are a side effect of using gradient-based learning techniques. Backpropagation [87] and backpropagation through time (BPTT) [88] use the gradient of the cost function (J) with respect to model parameters (θ), to update the parameters and minimise the cost.

In statistical machine learning, the goal is to improve some performance measure P, which can not be influenced directly [89]. Instead of minimising P directly, one minimises the cost J with the aim of improving P. Optimisation stops whenever overfitting begins to occur or after a predetermined number of iterations.

3.4.1 Empirical risk minimisation

Ideally minimising the cost function involves computing the expectation from the data generating distribution p_{data} as follows:

$$J(\theta) = \mathbb{E}_{(x,y)\sim p_{data}}[L(p(y|x,\theta),y)] \tag{3.32}$$

where, $L(\cdot)$ is the per-example loss function; $p(y|x,\theta)$ is the model's probabilistic output given sample x, and y is the model's target output associated with the given sample x. In practice, the data generating distribution p_{data} is unknown. Given a training set $(\{x,y\})$, one minimises the empirical distribution $\hat{p}_{data}$ such that:

$$J(\theta) = \mathbb{E}_{(x,y)\sim \hat{p}_{data}}[L(p(y|x,\theta),y)] = \frac{1}{m}\sum_{i=1}^{m} L(p(y_i|x_i,\theta),y_i) \tag{3.33}$$

where m is the number of training examples in the set [89]. Empirical risk minimisation involves minimising the average training error, which leads to overfitting.

3.4.1.1 Loss functions

The *loss* of a prediction ($\hat{y} = p(y|x,\theta)$) is a measure of the model's relative fit given the ground truth value (y) [90]. GD optimisation has a drawback in that useful loss functions, such as the zero-one loss (desirable in classification problems), have no useful derivatives (the derivative is either zero or undefined everywhere) [89]. In practice, one minimises a surrogate loss, which acts as a proxy to the real loss.

Minimising the zero-one loss is computationally intractable. The zero-one the loss is given by:

$$L(\hat{y}_i, y_i) = \mathbb{I}(\hat{y}_i, y_i) = \begin{cases} 1, & \hat{y}_i = y_i \\ 0, & \hat{y}_i \neq y_i \end{cases} \tag{3.34}$$

Optimisation using (3.34) is non-deterministic polynomial-time hard (NP-hard), because of the non-convex nature of the equation, as well as the discontinuity at zero. To make optimisation tractable for classification problems, one uses a surrogate loss function such as the *log-likelihood*.

The log-likelihood loss allows one to estimate the conditional probability of the model prediction ($\hat{y}$) given the given the ground truth value (y), which in a sense allows the model to choose the output that yields less error in expectation. The log-likelihood has the form:

$$L(\hat{y}_i, y_i) = \sum_{c \in C} y_{i,c} \log \hat{y}_{i,c} \tag{3.35}$$

where C is the number of categories. When using a surrogate loss, the test set's zero-one loss often continues to decrease even if the training set's zero-one loss reaches zero [89]. This effect arises because even when the expected zero-one loss is zero, the classifier's robustness can be improved by increasing the margin between the classes.

3.4.2 Gradient decent optimisation

Gradient decent (GD) optimisation algorithm moves the parameters (θ) of a model by some small amount ηg in the opposite direction of the gradient of the cost ($J(x;\theta)$) subject to the parameters (θ). Basic GD and stochastic gradient decent (SGD) are two variants of the GD algorithm. Basic GD updates the parameters of the learning model after iterating through the complete training set

(X^{train}):

$$\theta_j \leftarrow \theta_j - \eta \frac{1}{|X^{train}|} \sum_i \frac{\partial}{\partial \theta_j} J(x_i^{train}; \theta_j), \ \ \forall_j \tag{3.36}$$

Basic GD will achieve convergence [91, 92] given a regular cost function, initial values of θ sufficiently close to the optimal values, and a sufficiently small value of η. In practice, basic GD encounters the following problems:

- Basic GD scans through the entire dataset before a single update, which results in long training times if the training set is large.
- Large datasets cannot fit into random-access memory (RAM) of a central processing unit (CPU) or graphics processing unit (GPU).
- Basic GD does not always guarantee convergence to the optimal minima.

Instead of computing the update after iterating through the entire dataset, SGD estimates the gradient using a single randomly chosen example $x_i \subset X^{train}$, such that:

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial}{\partial \theta_j} J(x_i; \theta_j), \ \ \forall_j. \tag{3.37}$$

The SGD optimisation algorithm directly minimises the expected risk, because it uses examples randomly drawn from the ground truth distribution [91].

To ensure convergence of the SGD algorithm, the learning rate η_i must satisfy the Robbins-Monro conditions [49, 91]:

$$\begin{aligned} \sum_{i=1}^{\infty} \eta_i^2 &< \infty \\ \sum_{i=1}^{\infty} \eta_i &= \infty. \end{aligned} \tag{3.38}$$

where η_i is the learning rate schedule at time i. The convergence speed of the SGD algorithm depends on how noisy the approximation of the actual gradient is. Slowly decreasing learning rates cause the variance of the parameter (θ_i) estimate to decreases slowly as well [91]. Quickly decreasing learning rates, cause the expectation of the parameter (θ_i) estimate to be slow in reaching optimal values [91].

To reduce the noise/variance introduced by computing the update for each sample in the training dataset, one can get good estimates of the gradient by using a fraction of the training dataset $X^{mini} \subset X^{train}$

(*mini-batch*) in place of a single sample. Algorithm 2 shows the pseudocode for the mini-batch SGD algorithm.

Algorithm 2 Pseudocode for the mini-batch SGD algorithm

Require: learning parameters θ, learning rate η, training set X^{train}.

```
epoch ← 0
while Termination condition not reached do
    for X^mini ⊂ X^train do
        g_j ← 0
        for all (x_i) ∈ X^mini do
            g_j ← g_j + ∂/∂θ_j J(x_i; θ),  ∀_j
        end for
        θ_j ← θ_j − η/|X^mini| g_j,  ∀_j
    end for
    epoch ← epoch +1
end while
```

3.4.3 Stochastic gradient decent based optimisation algorithms

Stochastic gradient decent (SGD) converges slowly. To speed up training and online parameter optimisation algorithms such as SGD with momentum [92], root mean square propagation (RMSProp) [93], resilient propagation (RPROP) [94], adaptive learning rate (ADADELTA) [95], adaptive gradient (AdaGrad) [96], Hessian-free (HF) optimisation [97] and Adam [98] accelerate parameter learning. This section examines SGD with momentum [92], RMSProp [93], and adaptive moment estimation (Adam) [98] optimisation algorithms, because these three algorithms are more stable as compared the other optimisation algorithms [89].

3.4.3.1 Stochastic gradient decent with momentum

The SGD algorithm discussed in Section 3.4.2 is slow and susceptible to getting stuck at a local minimum. *Momentum* provides a way for the SGD algorithm to escape critical points, by accumulating a velocity vector of the directions of persistent reduction of the cost across iterations.

Adding a momentum term to SGD accelerates convergence, by directing the parameter search in the downhill direction. Given $J(\theta)$ as the cost function, adding momentum alters the learning rule in (3.37) (step 8 of Algorithm 2) to be:

$$\begin{aligned} v_t &= \mu v_{t-1} - \eta \frac{\partial}{\partial \theta_t} J(\theta_t) \\ \theta_t &= \theta_{t-1} + v_t \end{aligned} \tag{3.39}$$

where, η is the learning rate; $\mu \in [0,1)$ is the momentum coefficient; and v_t is initialised to zero. When μ is large relative to η, previous gradients affect the current direction more than the current gradient. Momentum addresses two problems: poor conditioning of the Hessian matrix and variance in SGD [89]. The momentum method relies on the strong assumption that the estimates of the gradient are noiseless and if the assumption does not hold, the estimates tend to veer off.

3.4.3.2 Root mean square propagation

The magnitude of the gradients, for the different parameters (θ), changes during optimisation, which makes choosing a global learning rate difficult. To address this issue, GD methods with adaptive learning rates such as RMSProp, Adam and RPROP exist. RMSProp [93] is a SGD optimiser that exploits the magnitude of recent parameter gradients to normalise the current gradient.

RPROP [94] uses the sign of the gradient as well as the idea of adapting the step-size separately for each parameter to speed up learning and escape the error surface critical points. The RPROP algorithm optimises the parameters using batch GD, which is impractical when dealing with large datasets, as it requires division by a different number for each mini-batch.

RMSProp [93], a mini-batch version of the RPROP algorithm, accumulates the running average of the recent gradient magnitude and divides the current gradient by this average to normalise the parameter gradients. Given $J(\theta)$ as the cost function, RMSProp alters the update rule in (3.37) (step 8 of Algorithm 2) to be:

$$\begin{aligned} r_t &= (1-\rho) \left(\frac{\partial}{\partial \theta_t} J(\theta_t) \right)^2 + \rho r_{t-1} \\ v_t &= \frac{\eta}{\sqrt{r_t}} \frac{\partial}{\partial \theta_t} J(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t. \end{aligned} \tag{3.40}$$

where $\rho \in [0,1)$ is the accumulation decay term. The r and v parameters are initialised to zero.

RMSProp anneals the parameter updates, which smooths out the variances for the gradient estimates and leads to stability during parameter search. RMSProp is an efficient and practical optimisation algorithm for deep neural networks [89].

3.4.3.3 Adaptive moment estimation

Adam [98] is a SGD algorithm based on an adaptive estimation of the first-order moment (gradient average) and the second-order moment (uncentred variance of the gradient). The Adam optimisation method suits problems with large datasets and parameters, as well as problems with noisy and sparse gradients [98].

Given $J(\theta)$, a possibly noisy and differentiable cost function, Adam optimisation aims to minimise the expected value of the cost subject to the parameters (θ). The minimisation happens by continually updating the exponential moving averages of the gradient (m_t) and the squared gradient (v_t). The hyperparameters $\beta_1, \beta_2 \in [0,1)$ control the exponential decay rates of the moving averages.

The moving averages are zero biased when initialised as vectors of zero's, which leads to moment estimates that have a zero bias during the initial iterations or when the decay rates ($\beta_{1,2}$) are small, i.e. when close to 1 [98]. To correct this bias, the first and second-order moment estimators are both divided by a term based on the decay rate [98].

The effective step taken in the parameter space at iteration t is given by:

$$\Delta_t = -\eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \tag{3.41}$$

The effective step-size Δ_t is invariant to the scale of the gradients. Rescaling the gradients g with factor c rescales m_t with a factor of c and v_t with a factor c^2, which cancel out: $c \cdot \hat{m}_t / \sqrt{c^2 \cdot \hat{v}_t} = \hat{m}_t / \sqrt{\hat{v}_t}$. The effective step has an upper and lower bound given by:

$$\|\Delta_t\|_\infty \leq \begin{cases} \eta(1-\beta_1)/(\sqrt{1-\beta_2}), & (1-\beta_1) > \left(\sqrt{1-\beta_2}\right) \\ \eta, & \text{otherwise} \end{cases} \tag{3.42}$$

The first bound occurs in the most severe case of sparsity, that is when a gradient has been zero at all time-steps except at the current time-step. The second bound η is the maximum magnitude of the step that can be taken in the parameter space at each time-step. Algorithm 3 illustrates the pseudocode for the Adam optimisation algorithm.

Algorithm 3 Pseudocode for the Adam algorithm

Require: Learning rate η, Initial parameters θ, Decay rates β_1 and β_2 and constant ε.

$m_j \leftarrow 0, \ \forall j$

$v_j \leftarrow 0, \ \forall j$

$t \leftarrow 0$

while Stopping criterion not met **do**

  $X^{mini} \subset X^{train}$

  $g_j \leftarrow 0, \ \forall j$

  for $(x_i) \in X^{mini}$ **do**

    $g_j \leftarrow g_j + \frac{\partial}{\partial \theta_j} J(x_i; \theta_j), \ \forall j$

  end for

  $t \leftarrow t + 1$

  $m_j \leftarrow \beta_1 m_j + (1-\beta_1) g_j, \ \forall j$ # Biased first moment

  $v_j \leftarrow \beta_2 v_j + (1-\beta_2) g_j^2, \ \forall j$ # Biased second moment

  $\hat{m}_j \leftarrow \frac{m_j}{1-(\beta_1)^t}, \ \forall j$ # Bias-corrected first moment

  $\hat{v}_j \leftarrow \frac{v_j}{1-(\beta_2)^t}, \ \forall j$ # Bias-corrected second moment

  $\Delta\theta_j = -\eta \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon}, \ \forall j$ # Parameter updates

  $\theta_j = \theta_j + \Delta\theta_j, \ \forall j$

end while

3.4.4 Choosing an optimisation algorithm

Choosing an optimisation algorithm is a non-trivial task. Adam combines the following advantages of AdaGrad [96] and RMSProp [93]:

- AdaGrad's ability to deal with sparse gradients.
- RMSProp's ability to deal with non-stationary objectives.
- RMSProp's ability to deal with AdaGrad's radically diminishing learning rates.

RMSProp with momentum [89] and Adam optimisation algorithms differ in the following ways:

- RMSProp with momentum generates its parameter updates using a momentum based on the

rescaled gradient, whereas Adam updates are directly estimated using a running average of the first and second moment of the gradient.

- RMSProp does not have a bias correction term. Kingma *et al.* [98] empirically showed the importance of bias-correction. Bias-correction led to Adam outperforming RMSProp with momentum across different hyperparameter settings on various datasets [98].

Adam is an effective optimisation method for image captioning models [24], language models [71], image generation [99] and activity recognition models [8] and as such, this research uses it to optimise all models trained.

3.4.5 Optimisation challenges for deep learning models

Optimising deep learning models is a difficult task. This difficulty arises mainly because direct generalisation error minimisation is not possible, because of model identifiability, because of the existence of local minimums, because of the vanishing and exploding gradient problems (see Section 3.3.5), because of an ill-conditioned Hessian matrix, and because of finite datasets. The presence of local minimums makes non-convex optimisation a tough task. Model identifiability can cause optimisation to get stuck in one of the local minimums, while a global minimum exists.

3.4.5.1 Online gradient decent and generalisation error

Ideally, one would like to minimise the generalisation error in place of the per-example loss across the training set [89]. During the first stages of training, optimisation algorithms (those that use mini-batch estimates) directly minimise the generalisation error. After the first epoch, the algorithm starts to minimise the per-example loss instead of the generalisation error [89]. A possible way to directly optimise the generalisation error is by using large datasets, as this would make online learning a possibility. This problem of not being able to minimise the generalisation error directly is addressed through the use of data augmentation methods (see Section 2.4.4).

3.4.5.2 Local minima

Models with equivalently parametrised latent variables (θ) encounter local minimums because of the so-called *model identifiability* problem [89]. An identifiable model is one wherein a sufficiently large training set can rule out all but one setting of the model parameters. Model identifiability means that there exists a large number of local minima in a model's cost function. Local minima that arise from non-identifiability are equal to each other, meaning that model identifiability is not a result of non-convexity, but a result of the model architecture [89].

The existence of many local minima is the most prominent problem in neural network optimisation. However, for sufficiently large neural networks, most local minima have a low-cost function value [100]. The important thing (in model optimisation) is to find a local minimum that has a low value as opposed to finding the true global minimum [100, 101, 102]. To verify whether the optimisation problem arises from local minima, one can track the norm of the gradients across iterations. If the norm does not shrink to insignificant size, then the problem is not a result of local minima.

3.4.5.3 Gradient decent and ill-conditioning:

Ill-conditioning of the Hessian matrix can result in gradient descent moving uphill [89]. To illustrate this effect consider the GD update of the parameters (θ), $\theta = \theta - \eta g$, where η is a learning rate and g is the gradient of the cost function $J(\theta)$ subject to parameters (θ). A second-order Taylor series expansion estimates the value the cost function at the new point to be:

$$J(\theta') \approx J(\theta) - \eta g^\top g + \frac{1}{2}\eta^2 g^\top H g \tag{3.43}$$

where H is the Hessian of J. The $-\eta g^\top g$ term is always negative, meaning that the cost function of the parameters will always move downhill if the cost function is linear. The second-order term $\frac{1}{2}\eta^2 g^\top H g$ can be negative or positive depending on the eigenvalues of H and the alignment of the corresponding eigenvectors with g. The alignment means that on steps where g aligns with large positive eigenvalues of H, the learning rate must be sufficiently small, or the second-order term will result in gradient descent moving uphill [89].

This research uses first-order GD optimisation algorithms (specifically Adam) to optimise the models, to avoid ill-conditioning the Hessian matrix.

3.4.5.4 Saddle points and plateaus

In convex optimisation, any point with zero gradient is a global minimum. In non-convex optimisation, a point with a zero gradient might be a global minimum or a local minimum. Local minima are common in low dimensional spaces and rare in high dimensional spaces. The eigenvalues of the Hessian matrix at a local minimum are all positive [100].

A saddle point has a local minimum along one cross-section of the cost function and a local maximum across the other cross-section [89], which means that the Hessian matrix at that point has both positive and negative eigenvalues. The negative and positive Hessian eigenvalues are more likely in high dimensional spaces as compared to low dimensional spaces [100]. Eigenvalues are more likely to be positive as we approach regions of lower cost, meaning that local minima are more likely to have a lower cost than high cost; points with high cost are more likely to be saddle points.

The implications of exponentially multiple saddle points remain unclear [89]. However, first order training methods such a SGD are not hindered by saddle points because SGD moves downhill, instead of trying to find critical points like second-order methods. The proliferation of saddle points in high dimensional spaces explains why second-order methods have not succeeded in replacing first-order methods. Second-order methods are susceptible to jump into a critical point [89, 100].

3.5 REGULARISING DEEP LEARNING ARCHITECTURES

Statistical machine learning aims to learn models that generalise well to both seen (training) and unseen (test) data. The desired outcome of training a learning model is a model that exhibits the best generalisation performance. In practice though, most learning models are prone to overfitting, because of their expressive capabilities [57]. Regularisation is the main method used in statistical machine learning to avoid overfitting.

Regularisation is any component of the model training process or prediction procedure, which accounts for the limitations of the training data [103]. This element is necessary because a learning model with multiple degrees of freedom can learn to approximate the data. The two most popular regularisation

strategies involve placing constraints on the learning model parameters or adding terms to the cost function (J).

The constraints and penalties placed on the learning models improve its performance when chosen [48, 49, 55, 104] properly. The constraints and penalties encode specific prior knowledge to promote generalisation. In practice, most of the regularisation strategies are estimators that trade increased bias for reduced variance [103].

3.5.1 Overfitting in statistical learning

Overfitting occurs when the learning model generalises to the peculiarities of the training data, instead of generalising to the distribution that generated the data. One way to prevent overfitting is to get more training data [49, 48, 50, 51]. However, getting more data is expensive and is not always a practical option.

3.5.1.1 Bias-variance trade-off and overfitting

The bias of a learning model is a measure of the model's error based on the algorithm's assumptions. Variance is a measure of the model's error sensitivity to small changes in the training set. The ideal situation is where the model has low bias and variance, which ensures that the model does not *underfit* or *overfit* the data [50].

High variance means that the model is too complex for the data. Increasing the dataset size, decreasing the complexity of the model or using better regularisation decreases model variance [50, 51]. To manage high bias, one can increase the number of features, make the model more complex or use fewer training samples [50, 51].

The bias-variance trade-off is the process of managing both the bias and variance of a model. Managing bias ensures that the model does not underfit the data and managing variance ensures that the model does not overfit the data. Overfitting occurs when a learning model has high variance and low bias.

3.5.2 Parameter regularisation

Parameter regularisation has the form:

$$\Phi(\theta) < c \tag{3.44}$$

where Φ is some transform function, θ represents the model parameters, and c is some constant. Adding a penalty term to the cost function results in penalty regularisation:

$$\tilde{J}(\theta;X) = J(\theta;X) + \lambda\Omega(\theta) \tag{3.45}$$

where X is the training data $\{x_i, y_i\} \in X$; $\Omega(\theta)$ is the regularisation term, and $\lambda \in \mathbb{R}^+$ is the hyperparameter that reflects the relative importance of the regularisation penalty to the standard cost $J(\theta;X)$. Adding a regularisation term to the loss function encourages smooth mappings by penalising large values of the parameters, which decreases the amount of nonlinearity that the network models. Minimising the sum of $J(\theta;X)$ and $\Omega(\theta)$ corresponds to finding the right trade-off between overfitting on the training data and modelling the data generating process.

3.5.2.1 L^p regularisation

The most popular forms of regularisations have the form L^P-norm:

$$\Omega(\theta) = \|\theta\|_p^p = \left(\sum_{j=0}^{|\theta|} |\theta_j|^p\right)^{\frac{1}{p}} \tag{3.46}$$

where θ are the model's learnable parameters. The L^p norm maps vectors to non-negative values. In statistical machine learning L^1 and L^2 norms are the most popular L^P-norms.

L^1 regularisation penalises the non-zero components of the learning parameters, which causes parameter sparsity [104]. The L^1 norm computes the sum of absolute values of the individual parameters:

$$\Omega(\theta) = \|\theta\|_1 = \sum_j |\theta_j| \tag{3.47}$$

L^1 regularization contributes a constant factor, $\text{sign}(\theta)$, to the gradient:

$$\begin{aligned} \frac{\partial \tilde{J}(\theta;X)}{\partial\theta} &= \frac{\partial}{\partial\theta}\left(J(\theta;X) + \lambda\|\theta\|_1\right) \\ &= \frac{\partial J(\theta;X)}{\partial\theta} + \lambda\,\text{sign}(\theta) \end{aligned} \tag{3.48}$$

The L^2 (*weight decay*) regularisation moves the parameters closer to the origin:

$$\Omega(\theta) = \frac{1}{2}\|\theta\|_2^2 \tag{3.49}$$

The effect of L^2 regularisation can be analysed by looking at its gradient subject to the parameters:

$$\begin{aligned}\frac{\partial \tilde{J}(\theta;X)}{\partial \theta} &= \frac{\partial}{\partial \theta}\left(J(\theta;X)+\frac{\lambda}{2}\|\theta\|_2^2\right) \\ &= \frac{\partial J(\theta;X)}{\partial \theta}+\lambda\theta \end{aligned} \tag{3.50}$$

Taking a single gradient update means (3.37):

$$\theta_j \leftarrow (1-\eta\lambda)\,\theta_j - \eta\frac{\partial J(\theta_i;X)}{\partial \theta_i},\ \ \forall_j \tag{3.51}$$

Equation (3.51) shows that the L^2 regularisation encourages the parameters to be small with every single parameter update. L^2 regularisation has the effect of penalising large weight vectors, which diffuses the vectors [104].

3.5.2.2 Max-norm constraint regularisation

Max-norm regularisation enforces an absolute upper bound on the magnitude of the parameters (θ):

$$\Phi(\theta) \leftarrow \|\theta\|_2 < C \tag{3.52}$$

where C is the clamping constant (typical values of C are 4 or 5). Max-norm regularisation improves performance when combined with dropout [55].

3.5.3 Dropout regularisation

Dropout is a method that reduces the propensity of a neural network to overfit on the data [55]. Dropout aims to prevent hidden units from co-adapting by temporarily and stochastically dropping units of a hidden layer at a probability rate p during training and averaging the weights at test time. Dropout allows one to combine many different learning models efficiently [55].

L^p-norm regularisation requires that one to modify the cost function, with the aim of forcing the parameters to be smaller. Dropout modifies the network itself [55]. Its effect on the parameters is to pull them towards the average of the other models, without forcing a constraint on the final value of the parameters.

Let $x^{\ell-1}$ denote the input to the model layer ℓ and h^{ℓ} denote its output. Dropout, randomly (and temporarily) drops the hidden neurons in the layer at probability p, as follows:

$$\begin{aligned} r^{\ell} &\sim \text{Bernoilli}(p) \\ \tilde{h}^{\ell} &= r^{\ell} \odot x^{\ell-1} \\ h^{\ell} &= \upsilon\left(\tilde{h}^{\ell}\right) \end{aligned} \tag{3.53}$$

where υ is the layer transformation and r^{ℓ} represents a vector of independent Bernoulli random variables, with each variable having a probability p of being inactive. Sampling occurs with each forward propagation. During backpropagation, only the parameters connected to the active units (during forward propagation) receive updates.

Consider a neural net with D_h hidden units. The network is a collection of 2^{D_h} possible thinned neural networks, and applying dropout to the network amounts to sampling "thinned" networks. A thinned network consists of all the units that survived dropout. Applying dropout has the effect of averaging a large number of networks, which overfit in various ways; averaging these networks has the effect of reducing overfitting [55].

A dropout network should have at least n/p units [55], because pn units are present after dropout. The network should have at least n/p units. To counteract the reduced capacity of a neural network due to dropout. Due to the amount of noise introduced by dropout in the gradients, many gradients tend to cancel each other out. A dropout net typically uses $10-100$ times the learning rate that was optimal for a standard neural net [55]. Dropout introduces an extra hyperparameter p, the probability of retaining a unit.

3.6 CHAPTER SUMMARY

This chapter reviewed and discussed deep learning concepts. Deep learning is concerned with learning high-level abstractions from low-level data through the use of stacked layer transformations. This research is concerned with deep learning models that are optimised using maximum likelihood estimation (MLE). The MLE approach to statistical learning assumes a constant *prior*. Regularisation acts as a prior in MLE estimation.

Regularisation (see Section 3.5) helps reduced model variance by placing constraints on the model parameters (e.g. L^p regularisation) and the model itself (e.g. dropout). L^2 regularisation, the most popular parameter regularisation method, penalises large model parameters. Max-norm regularisation places an upper bound on the parameter norm. Dropout averages 2^n thinned networks, which overfit the data in unique ways, to reduce overfitting. Dropping units of a network at random prevents coadaptation [55].

Deep learning models are composed of multiple layer mappings, mappings such as MLPs, CNNs and RNNs. MLPs learn dependencies between data features, CNNs extract spatial features from images, and RNNs capture sequential dependencies through recurrent connections (see Section 3.3). Activation functions transform layer mappings to a highly non-linear space, which makes it possible for MLPs to approximate any smooth function [78]. Rectilinear activation functions (e.g. ReLU and ELU) are the most successful activations in deep learning (see Section 3.3.4). Learning long-term dependencies using RNNs is difficult because the memory is written to at each time-step. Gate recurrent neural networks (GRU and LSTM) control when the memory is updated.

Recurrent neural networks (RNNs) experience the vanishing and exploding gradient problems (see Section 3.3.5). The vanishing and exploding gradient problems are a result of propagating the gradients through many time-steps. The vanishing gradient problem is addressed through initialisation, using linear activations and using additive instead of multiplicative recurrent dynamics in a RNN, e.g. LSTMs and GRUs. The exploding gradient problem is addressed through gradient norm-clipping [85].

Optimising deep learning architectures is a non-trivial task (see section 3.4). In practice the true distribution from which the data is sampled is unknown, so an empirical distribution is minimised instead. Useful loss functions such the zero-one loss have discontinuities, so a surrogate loss such as the log-likelihood is minimised instead. MLE optimisation involves finding the parameters that minimise the cost function. The cost is minimised using iterative methods such as gradient descent.

The gradient descent (GD) algorithm moves the parameters of a model in the opposite direction of the cost. Basic GD is guaranteed to achieve convergence (see Section 3.4.2). Gradient decent converges slowly, so in practice, stochastic gradient descent (SGD), specifically mini-batch SGD, and its variants (RMSProp, AdaGrad and Adam) are used. Adam, which combines RMSProp and AdaGrad advantages, is the best mini-batch SGD algorithm (see Section 3.4.4).

CHAPTER 4 APPROACH

4.1 CHAPTER OVERVIEW

There are three approaches one can take to improve the performance of a deep learning model on a learning task, namely:

- Improve model architecture – by introducing a new architecture or defining a new layer.
- Improve model initialisation – better initialisation of the parameters and transfer learning ensures that early stage gradients have certain beneficial properties.
- Improve model optimisation – through better regularisation, employing better optimisation algorithms or using a better loss function.

This research aims to improve the performance of deep learning models for activity recognition by introducing new learning layers, which incorporate both the spatial and temporal aspects of activity recognition data.

This chapter discusses the approaches taken in this reseach to perform activity recognition. Firstly, a new recurrent network, the SCGRU (see Section 4.2) is proposed. After that, a discussion on recurrent convolutional layers is presented. Then a discussion on the computational efficiency of the deep learning layer transformation is carried out. Two architectures for activity recognition architectures, namely Temporal-CNN (see Section 4.5.1) and Conv-RNN (see Section 4.5.2) are proposed. Finally, the UML framework for constructing and optimising deep learning models is presented.

4.2 STRUCTURALLY CONSTRAINED GATED RECURRENT UNITS

Mikolov *et al.* [105] added a context layer (s_t) to the RNN (3.9) to create a structurally constrained recurrent network (SCRN). The context layer (s_t) learns long-term dependencies by stabilising the hidden layer state changes. The context layer caches the inputs (x_t) at each time-step to learn topic information [105].

Adding a context layer (s_t) to the RNN transforms (3.9) to be:

$$\begin{aligned} s_t &= (I-Q)\, w^{(sx)} x_t + Q s_{t-1} \\ \tilde{h}_t &= w^{(hx)} x_t + w^{(hs)} s_t + w^{(hh)} h_{t-1} + b^{(h)} \\ h_t &= \phi\left(\tilde{h}_t\right) \end{aligned} \tag{4.1}$$

where $Q \in (0,1)$ is a diagonal caching matrix, the elements of which are a result of applying a sigmoid transformation to a vector β such that $\text{diag}(Q) = \sigma(\beta)$; $w^{(xs)} \in \mathbb{R}^{D_h \times D_d}$ is the context embedding matrix and $w^{(hs)} \in \mathbb{R}^{D_h \times D_h}$ is the context recurrent matrix. The context layer (s_t) exponentially embeds previous inputs.

The hidden layer of a RNN changes state at each time-step, and the context layer stabilises these changes. The hidden state (h_t) of a GRU (3.19) layer is an exponential average of the previous hidden state (h_{t-1}) and candidate activation $\left(\tilde{h}_t\right)$. In a sense, the hidden state (h_t) of a GRU embeds the previous hidden states (h_{t-1}) much like the context layer (s_t) of a SCRN embeds the previous inputs.

This research proposes adding a context layer (s_t) to the GRU to stabilise the candidate activation $\left(\tilde{h}_t\right)$ state changes. Figure 4.1 shows the proposed SCGRU. The context memory (s_t) consolidates the current input (x_t) with the previous input context memory (s_t). The reset gate determines whether the previous state (h_{t-1}) and context memory (s_t) are important when computing the new memory $\left(\tilde{h}_t\right)$. The update gate determines the relative importance of the previous state (h_{t-1}) and new memory $(\tilde{h}_t)$ to the current hidden state (h_t).

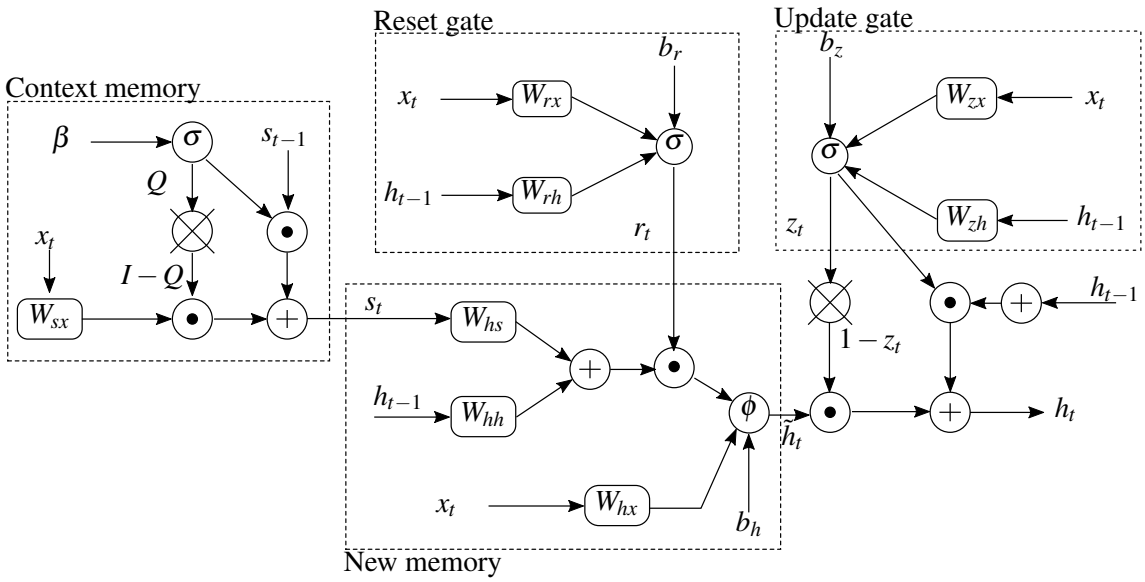


Figure 4.1. Internal state transition of a SCGRU architecture. The context memory (s_t) exponentially embeds past inputs, while the current hidden state (h_t) embeds previous states.

The SCGRU state transition equations are given by:

$$z_t = \sigma\left(w^{(zx)}x_t + w^{(zh)}h_{t-1} + b^{(z)}\right) \tag{4.2}$$

$$r_t = \sigma\left(w^{(rx)}x_t + w^{(rh)}h_{t-1} + b^{(r)}\right) \tag{4.3}$$

$$s_t = (I-Q)\,w^{(sx)}x_t + Qs_{t-1} \tag{4.4}$$

$$\tilde{h}_t = \phi\left(w^{(hx)}x_t + w^{(hh)}\left(r_t \odot h_{t-1}\right) + w^{(hs)}\left(r_t \odot s_t\right) + b^{(h)}\right) \tag{4.5}$$

$$h_t = (1-z_t)\odot\tilde{h}_t + z_t \odot h_{t-1} \tag{4.6}$$

where $\phi(\cdot)$ is the activation function; $\sigma(\cdot)$ is the gate/switching activation function; $x_t \in \mathbb{R}^{D_d}$ is the input vector at time t; $w^{(:x)} \in \mathbb{R}^{D_h \times D_d}$ are the rectangular input weight matrices; $w^{(:h)} \in \mathbb{R}^{D_h \times D_h}$ are the square recurrent weight matrices; and $b \in \mathbb{R}^{D_h}$ are the bias vectors.

The context memory (s_t) allows the SCGRU to exponentially accumulate contextual information about the inputs the network has seen, which allows it to encode long-term dependencies. Equation (4.6) allows the SCGRU to accumulate contextual information about the network's hidden states. The SCGRU updates its state based on the current input (x_t), previous hidden state (h_{t-1}) and context memory (s_t).

Le *et al.* [86] demonstrated that embedding past inputs (through identity initialisation) allowed a RNN

model to learn long-term dependencies. Identity initialisation of the hidden-to-hidden parameters, means that the previous hidden state (h_{t-1}) gets copied and then added to current input context [86]. Identity initialisation ensures that the hidden unit error derivatives remain constant through time, which addresses the vanishing and exploding problem (see Section 3.3.5). The SCGRU (4.2) formulation builds upon this idea by embedding previous inputs as well as previous states.

4.3 CONVOLUTIONAL RECURRENT LAYERS

Videos are temporally smooth, i.e. any activity associated with a given patch is restricted to a local spatial neighbourhood between successive frames. Fully connected layers, when applied to 2D inputs, disregard the spatial structure inherent in image data. However, convolutional layers take into account the spatial structure of image data (see Section 3.3.1).

The visual encoding part of the LRCN [6] architecture assumes successive frames to be temporally independent, i.e. the model encodes the spatial information independently for each successive frame. The RNN part of the LRCN [6] architecture encodes sequential visual encodings to learn temporal dependencies for activity recognition. This research argues, "to build an effective architecture for activity recognition, requires an architecture that learns sequential features using both convolutional and fully connected layers".

4.3.1 Convolutional recurrent neural network

The proposed Conv-RNN combines convolution and recurrent nodes, by replacing the multiplication operation of a RNN with a convolutional one. Figure 4.2 shows a Conv-RNN unrolled across time-steps. The outputs of a Conv-RNN are feature maps at each time-step t. The kernels/filters of a Conv-RNN are spatially and temporally tied. Spatially tying the filters ensures filter sharing across different spatial locations. Temporally tying the filters ensure that the filters learn contextual motion features. The outputs of the previous step along with the current input feature maps, combine to produce the output feature maps (h_t). The input into the layer at time t consists of D_d input feature maps of size $m_1 \times m_2$ and D_h recurrent feature maps of size $n_1 \times n_2$. The hidden-to-hidden state transitions use *same* convolution. Same convolution simplifies computation and improves performance in deep architectures for object recognition [18, 38, 54].

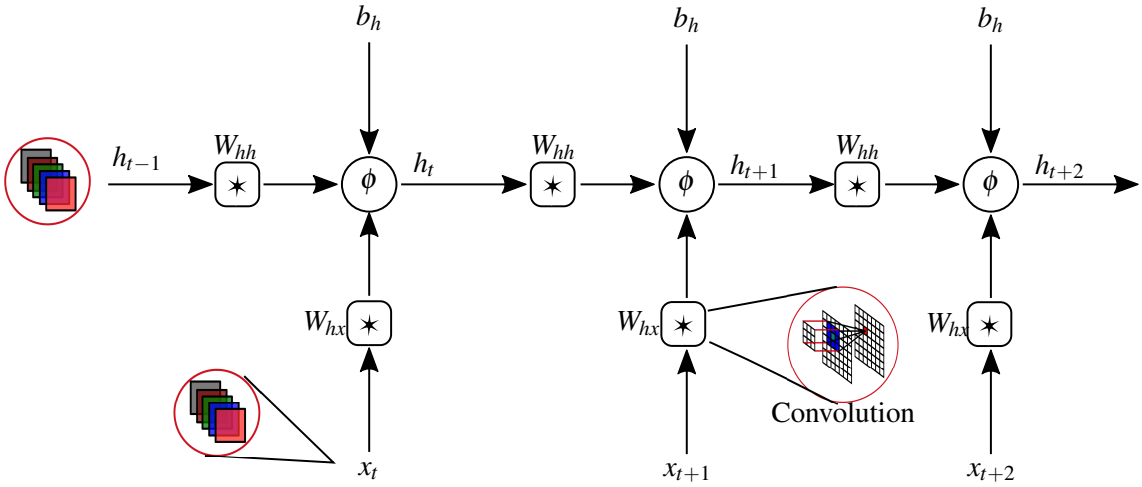


Figure 4.2. A convolutional recurrent neural network Conv-RNN unfolded across time-steps

The output at time t is given by:

$$h_t = \phi\left(w^{(hx)} * x_t + w^{(hh)} * h_{t-1} + b^{(h)}\right) \tag{4.7}$$

where $\phi(\cdot)$ is the activation function; $x_t \in \mathbb{R}^{D_d \times m_1 \times m_2}$ is a 3D stack of input feature maps at time t; $w^{(hx)} \in \mathbb{R}^{D_h \times D_d \times h_1^x \times h_2^x}$ is a 4D stack of input filters, $w^{(hh)} \in \mathbb{R}^{D_h \times D_h \times h_1^h \times h_2^h}$ is a 4D stack of recurrent filters; and $b^{(h)} \in \mathbb{R}^{D_h}$ is the bias offset. D_h and D_d are the number of output and input feature maps.

The Conv-RNN formulation preserves the spatial and temporal topology of the inputs, through convolutional and recurrent operations. The Conv-RNN (4.7) fully exposes the recurrent filters at each time step, which makes learning long-term motions difficult. To learn long-term motion, the Conv-RNN formulation is extended to gated recurrent networks.

4.3.2 Convolutional gated recurrent unit

The convolutional gated recurrent unit (Conv-GRU) extends the ideas in Section 4.3.1 to gated recurrent networks, specifically the GRU (see Section 3.3.2.2). Adding recurrent gates to the state update in (4.7), allows one to learn long-term motions and avoid the vanishing/exploding gradient problem (see Section 3.3.5). The Conv-GRU formulation substitutes the multiplication operation (3.16) with

convolutions as follows:

$$z_t = \sigma\left(w^{(zx)} * x_t + w^{(zh)} * h_{t-1} + b^{(z)}\right) \tag{4.8}$$

$$r_t = \sigma\left(w^{(rx)} * x_t + w^{(rh)} * h_{t-1} + b^{(r)}\right) \tag{4.9}$$

$$\tilde{h}_t = \phi\left(w^{(hx)} * x_t + w^{(hh)} * (r_t \odot h_{t-1}) + b^{(h)}\right) \tag{4.10}$$

$$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1} \tag{4.11}$$

where $\phi(\cdot)$ is the activation function; $\sigma(\cdot)$ is the gate activation; $x_t \in \mathbb{R}^{D_d \times m_1 \times m_2}$ is a 3D stack of input feature maps at time t; $w^{(:x)} \in \mathbb{R}^{D_h \times D_d \times h_1^x \times h_2^x}$ are 4D stacks of input filters; $w^{(:h)} \in \mathbb{R}^{D_h \times D_d \times h_1^h \times h_2^h}$ are 4D stacks of recurrent filters; and $b^{(:)} \in \mathbb{R}^{D_h}$ are the bias offsets. The Conv-GRU state transition equations (4.8)–(4.11) are defined over a local spatial neighbourhood of size $(h_1^c \times h_2^c),\ \ c \in \{x,h\}$, at pixel location (i,j) of the current input (x_t) and previous state (h_{t-1}). The Conv-GRU model implicitly assumes spatially and temporally smooth motion between frames.

Ballas *et al.* [8] arrived at the same formulation and achieved state-of-the-art results on the UCF101 [32] dataset (79.9% on the RGB inputs). The last time-step output of the recurrent convolutional network (RCN) layer, of the GRU-RCN architecture, is pooled and passed to a classifier [8]. The GRU-RCN architecture classifies pixel, which is undesirable when dealing with high dimensional inputs, such as frames. This research proposes a different architecture based on the Conv-GRU layers.

4.3.3 Convolutional structurally constrained gated recurrent unit

The convolutional structurally constrained gated recurrent unit (Conv-SCGRU) extends the ideas in Section 4.3.1 to the SCGRU (see Section 4.2). The Conv-GRU formulation substitutes multiplication with convolution in a SCGRU as follows,

$$z_t = \sigma\left(w^{(zx)} * x_t + w^{(zh)} * h_{t-1} + b^{(z)}\right) \tag{4.12}$$

$$r_t = \sigma\left(w^{(rx)} * x_t + w^{(rh)} * h_{t-1} + b^{(r)}\right) \tag{4.13}$$

$$s_t = (I - Q) \odot \left(w^{(sx)} * x_t\right) + Q s_{t-1} \tag{4.14}$$

$$\tilde{h}_t = \phi\left(w^{(hx)} * x_t + w^{(hh)} * (r_t \odot h_{t-1}) + w^{(hs)} * (r_t \odot s_t) + b^{(h)}\right) \tag{4.15}$$

$$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1} \tag{4.16}$$

The new hidden state is a result of adding (element-wise) the previous hidden state (h_t) and new memory $(\tilde{h}_t)$ with scaling using the update signal z_t.

4.4 ARCHITECTURE COMPUTATIONAL EFFICIENCY

Computational complexity measures how an algorithm's runtime and resource usage scale, as the size of the data (inputs) increases. Computational complexity, for machine learning architectures, measures the number of operations an architecture performs per input sample and the number of parameters loaded into memory.

The computational complexity of a feedforward neural network (FNN), $h = \phi\left(w^{(hx)}x + b^{(h)}\right)$, is given by $O(D_h D_d)$. D_d and D_h are the number of input and output features. The complexity scales proportionally to the dimensionality of the input and output. The computational complexity of the convolutional layer (3.6) is given by $O\left(D_h D_d h_1 h_2 m_1^{\ell-1} m_2^{\ell-1}\right)$, where D_d and D_h are the number of input and output feature maps.

Given a layer that preserves the dimensionality of the input, convolutional layers (i.e. Conv-RNNs and CNNs) are computationally more efficient than fully connected layers (i.e. FNNs and RNNs). Convolutional layers require $O\left(D_d^2 h_1 h_2 m_1^{\ell-1} m_2^{\ell-1}\right)$ multiplications, while FNN require $O\left(\left[D_d m_1^{\ell-1} m_2^{\ell-1}\right]^2\right)$. Convolutional layers require $m_1 m_2 / h_1 h_2$ less multiplications than FNNs on images.

The computational complexity of the spatial pooling layer (3.8), with spatial extent f and stride s, is $O\left(D_d(m_1^{\ell-1} - f + s)(m_2^{\ell-1} - f + s)\right)$. The complexity scales linearly subject to the dimensionality of the input $\left(m_1^{\ell-1} \times m_2^{\ell-1}\right)$ and the number of input feature maps (D_d).

The computational complexity of the GRU (3.16) is given by $O\left(3TD_h(D_d + D_h)\right)$, where T is the number sequential time-steps. The computational complexity of the SCGRU layer (4.2) is given by $O\left(4TD_h(D_d + D_h)\right)$. The GRU architecture is 25% more computationally efficient than the SCGRU architecture.

The computational complexity of the Conv-GRU (4.8) is $O\left(3TD_h(D_d h_1^x h_2^x m_1^{\ell-1} m_2^{\ell-1} + D_h h_1^h h_2^h m_1^\ell m_2^\ell)\right)$, where T is the number sequential time-steps. The computational complexity of the Conv-SCGRU (4.12) is $O\left(4TD_h(D_d h_1^x h_2^x m_1^{\ell-1} m_2^{\ell-1} + D_h h_1^h h_2^h m_1^\ell m_2^\ell)\right)$.

4.5 ARCHITECTURES FOR ACTIVITY RECOGNITION

This research proposes two architectures for activity recognition, namely Temporal-CNN and Conv-RNN. This section describes the proposed activity recognition architectures.

4.5.1 Temporal-CNN architecture

Single-frame architectures (see Section 2.4.2) treat activity recognition as an object recognition task. This research proposes an extension to this architecture by optimising the architecture on sequential frames instead of a single frame. Figure 4.3 shows the proposed Temporal-CNN architecture. A CNN model is applied at each time-step, and a softmax layer makes predictions. The Temporal-CNN architecture learns features that are temporally independent.

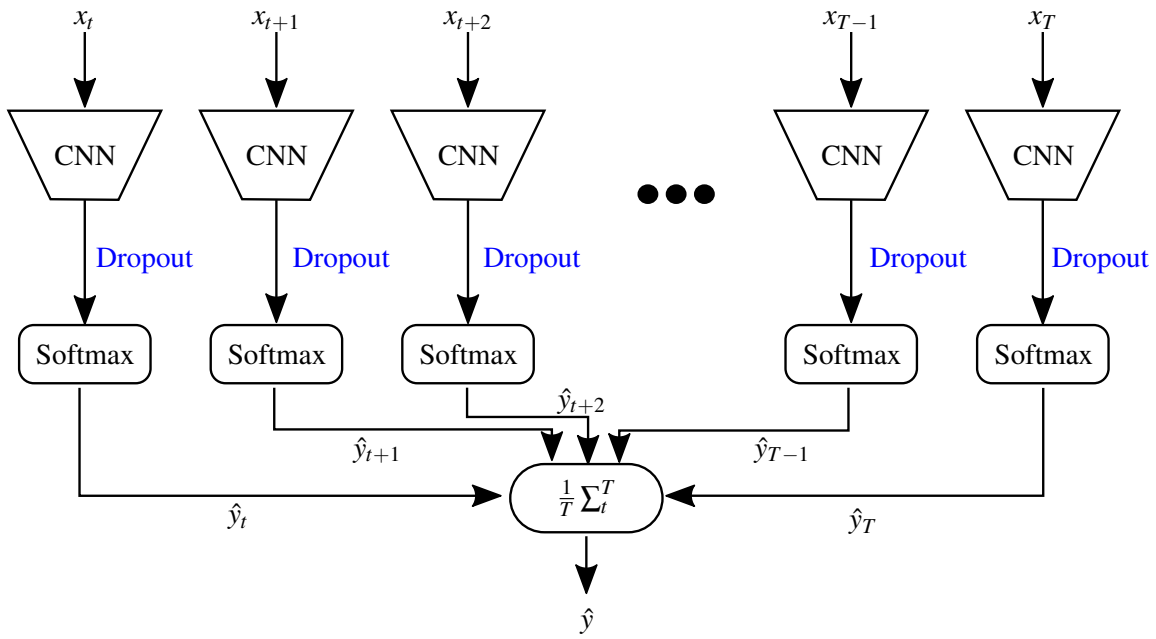


Figure 4.3. Temporal-CNN architecture for activity recognition. The output of the architecture is an average of the predictions at each time-step.

Temporal architectures, e.g. LRCN, aim to learn temporal dependencies given sequential feature vectors. This research proposed the Temporal-CNN architecture as a control architecture to compare the performance of architectures that learn temporal dependencies and those that do not, on the activity recognition task.

4.5.2 Conv-RNN architecture

Convolutional recurrent neural network (Conv-RNN) architectures for activity recognition combine convolutional layers (see Section 3.3.1) with Conv-RNN layers (see Section 4.3) as shown in Figure 4.4. The convolutional layers filter the images to remove noise and extract objects at each time-step. The Conv-RNN layer extract motion features from the CNN feature maps.

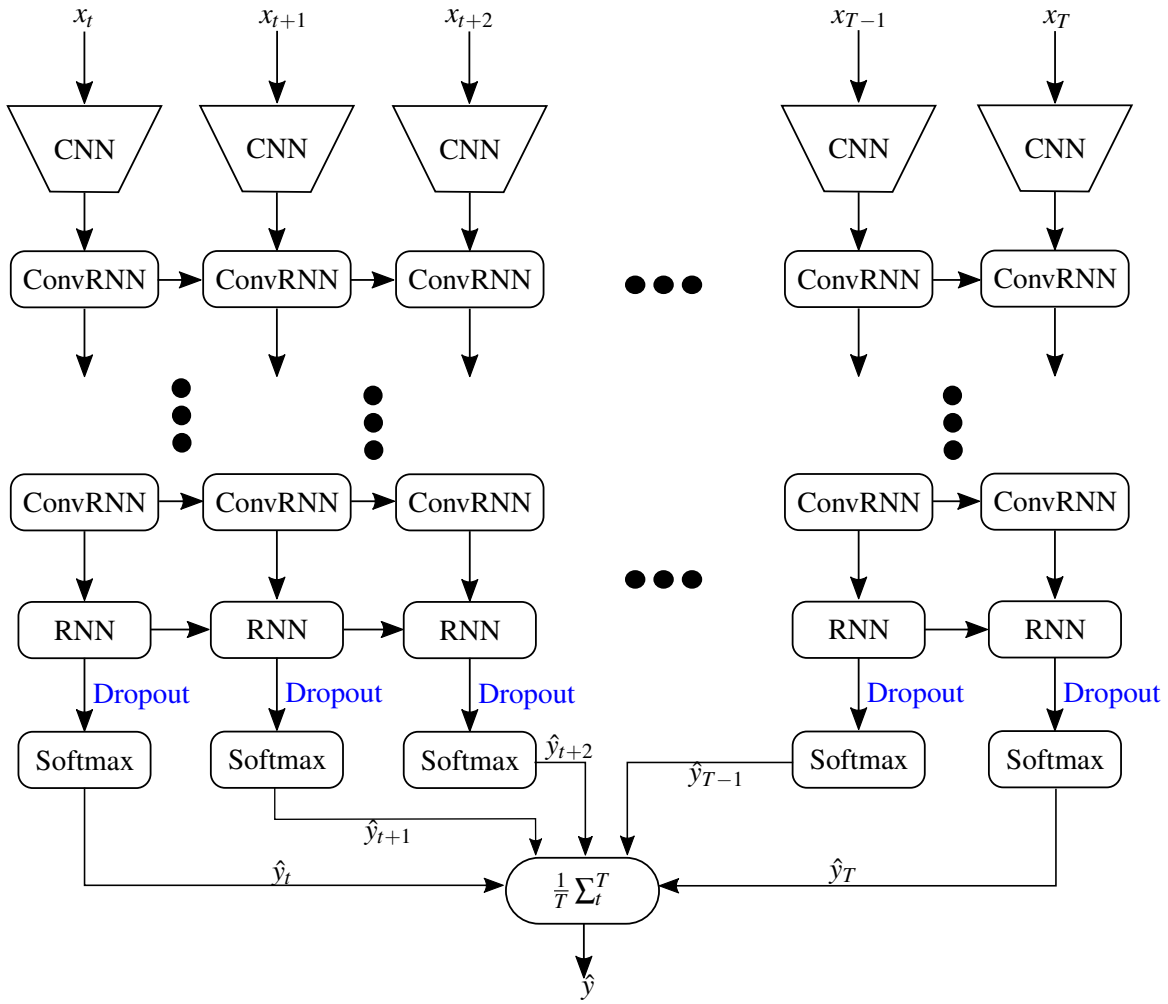


Figure 4.4. Conv-RNN architecture for activity recognition. The convolutional layers filter the images to remove noise and extract objects at each time-step.

The proposed Conv-RNN architecture is comparable to the GRU-RCN [8] architecture. The two architectures differ in that 1) the Conv-RNN architecture does not feed the outputs of each layer to a classifier, 2) the Conv-RNN architecture has a RNN layer, and 3) the Conv-RNN architecture performs classification at each time-step instead of classifying the last time-step.

4.6 FRAMEWORK DESIGN AND IMPLEMENTATION

Deep learning models contain many layers and millions of parameters, which make constructing and optimising these models difficult. The UML framework was designed and implemented to simplify model construction and training. Keras[1], which is currently the most popular deep learning framework, was not used because 1) it was necessary to learn how to implement the deep learning ideas in research, and 2) Keras was discovered six months into the research. At the start of the research Lasagne[2] and Pylearn2[3] where the most popular deep learning libraries.

4.6.1 Framework overview

The UML framework 1) *provides quick model prototyping*, 2) *can be easily extended*, and 3) *is easy to use*. The UML framework was developed using the Python[4] programming language and it incorporates Theano [106], SciPy [107], Numpy [108], Scikit-learn [109], Scikit-image [110] and Fuel [111] libraries.

Theano [106] is a popular library for implementing deep learning models [83, 85]. Theano provides easy access to GPU and simplifies model construction through symbolic representations for mathematical expressions. Symbolic expressions provide access to automatic differentiation of complex expressions, which is vital for deep learning optimisation. Reverse mode automatic differentiation is particularly useful in deep learning because DNN model optimisation involves optimising millions of parameters against a single cost function. All models in this research are implemented and optimised using Theano.

The Fuel [111] library provides a simplified pipeline for performing data augmentation (see Section 2.4.4). Scikit-image [110] library provided functions to process video frames and Scikit-learn [109] provided metrics to benchmark the architectures.

[1] `https://keras.io/`
[2] `https://lasagne.readthedocs.io/en/latest/`
[3] `http://deeplearning.net/software/pylearn2/`
[4] `https://www.python.org/`

4.6.2 Framework structure

The UML framework design (see Figure 4.5) is designed to provide modularity in constructing and optimising DNN models, much like Keras[5]. Each UML learning layer provides standard transformations given an input symbolic tensor. The output of a layer is a symbolic expression, which forms part of the computational graph up to that layer. The UML framework (see Figure 4.5) contains seven core modules, namely data, layers, optimisation, models, pre-processing, stats and misc.

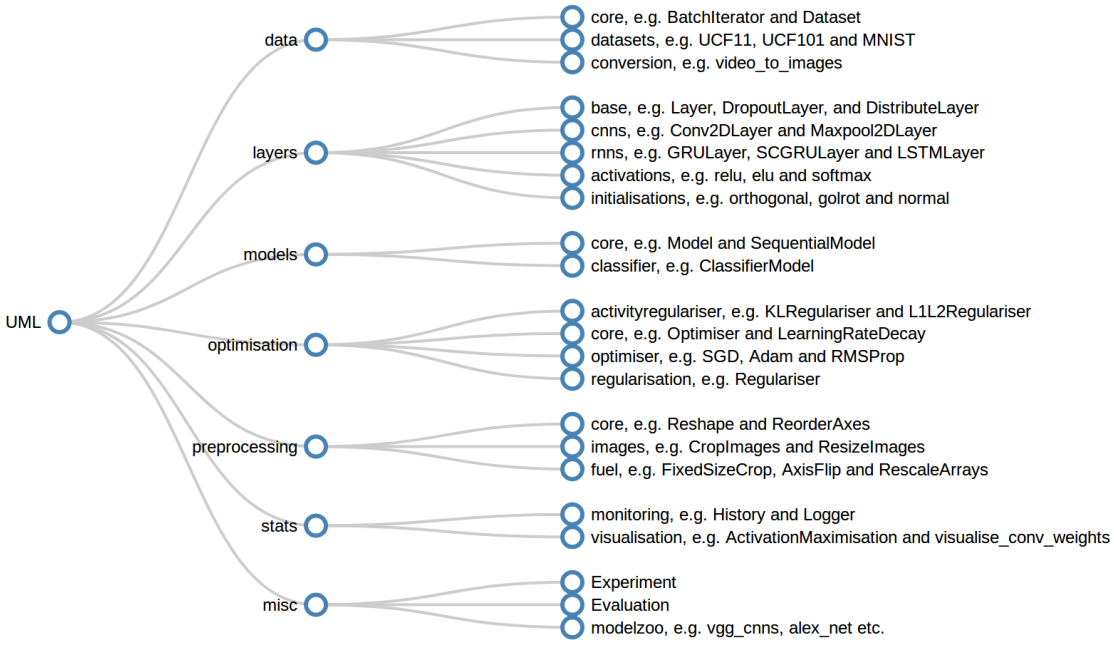


Figure 4.5. UML framework core modules

data – The data module provides a common interface to transform, load and iterate through datasets such as UCF101, HMDB51, UCF11 and MNIST. The datasets are stored in Hierarchical Data Format 5 (HDF5) files, an efficient format to store large datasets.

layers – The layers module contains implementations of layer transformations such as FNNs, CNNs and RNNs using a common interface. All layer implementations inherit from the `Layer` class which handles the parameters and the layer-to-layer symbolic graph connection.

models – The models module implements an interface to stack layers and optimises the learning model. The base class `Model` also inherits from `Layer`, which allows one to treat a model as a layer

[5] `https://keras.io/`

transformation. This abstraction is useful in cases where one would like to apply a model along an axis of the data, e.g. a CNN applied at each time-step of the video data.

optimisation – The optimisation module implements the model optimisation (see Section 3.4) and regularisation (see Section 3.5) algorithms. The optimisation algorithms take the symbolic graph for the learning model and compute the per iteration update of the model parameters given a cost function. Parameter updates can be constrained using a regularisation scheme.

pre-processing – The pre-processing module implements data pre-processing and augmentation functions such as contrast normalisation, axis flipping, and image cropping. The augmentation classes are implemented using the Fuel [111] library.

stats – The stats module implements classes to monitor model training and to visualise layer activations, parameters and learned features.

misc – The misc module implements classes to log a training experiment (`Experiment`), evaluate a model (`Evaluate`) and to load Caffe [112] pre-trained models used in this research.

4.6.3 Framework examples

The MNIST [59] softmax regression problem is implemented to illustrate the basic functionality of the UML library. Figure 4.6 shows the UML framework code listing to train a softmax regression model using the the MNIST [59] dataset. A softmax regression model is parameterised by a weight matrix $\left(w^{(hx)}\right)$ and bias vector $\left(b^{(h)}\right)$. The `HiddenLayer` class implements the FNN transformation on the inputs before applying a softmax activation function.

The `InputLayer` class defines the root Theano [106] symbolic tensor for input data. The `compile` method builds Theano functions to train and make predictions using the model's computational graph. The `fit` method optimises the model on data X and targets y.

Figure 4.7 shows the UML code listing for the LRCN6 [6] model for activity recognition. The LRCN6 model use a pre-trained CNN (CNN-S [67] model in this research) model as the visual encoding model.

```
from uml.layers import InputLayer, HiddenLayer
from uml.models import SequentialModel

# Construct regression model
model = SequentialModel(name='mnist_regression')
model.add(InputLayer(size=784, shape=(None, 784), name='data'))
model.add(HiddenLayer(size=10, activation='softmax', name='fc1'))

# Build computational graphs and train model
model.compile(loss='cce', optimiser='Adam')
model.fit(X=X, y=y)  # {X, y} are the mnist data and ground truth values
```

Figure 4.6. UML code listing for a MNIST dataset softmax regression model.

The `DistributeLayer` class applies a given layer along the given axis, the temporal axis in this case. The `DropoutLayer` class implements dropout (see Section 3.5.3). Dropout is applied at each time-step during training.

```
from uml.layers import InputLayer, HiddenLayer, DropoutLayer, FlattenLayer,
    DistributeLayer, GRULayer
from uml.optimisation import Adam, Regulariser
from uml.models import SequentialModel

# VGG_CNN_S model: input to drop6 layers
dmodel= SequentialModel(layers=cnns.layers[:-3], name='cnns-drop6')
dp7 = DropoutLayer(p=0.9, name='drop7')
fc8 = HiddenLayer(size=101, activation='softmax', name='fc8')

model = SequentialModel(name='LRCN-6')
model.add(InputLayer(size=3, shape=(None, 10, 3, 224, 224), name='data'))
model.add(DistributeLayer(layer=dmodel, axis=1, name='cnns-drop6'))
model.add(GRULayer(size=1024, name='rnn7', activation='elu', seq_output=True,
     seq_length=10))
model.add(DistributeLayer(layer=dp7, axis=1, use_scan=False, name='drop7'))
model.add(DistributeLayer(layer=fc8, axis=1, use_scan=False, name='fc8'))

# Build computaional graphs and train model
regulariser = Regulariser(max_norm=5.0, l2=1e-6, norm_thresh=10.)
params = {'loss': 'tcce',  # CCE loss averaged across time
          'metrics': ('tacc', 'tnll'),  # Monitoring metrics
          'optimiser': Adam(regulariser=regulariser, learn_rate=1e-4)}
model.compile(**params)
model.fit(X=X, y=y)  # {X, y} are the video sequences and ground truth values
```

Figure 4.7. Code implementation of the LRCN6 model using the UML framework.

The `GRULayer` class implements the GRU (see Section 3.3.2.2). The `tnll` function implements the NLL loss in (2.2). The `tacc` and `tcce` functions average the accuracies and categorical cross-entropy at each time-step.

4.7 CHAPTER SUMMARY

This chapter introduced the approaches taken to improve the performance of deep learning architectures for activity recognition. Section 4.2 proposed a new recurrent network, the SCGRU. The SCGRU adds a context memory (s_t) to the GRU to stabilise the candidate activation. The SCGRU embeds previous inputs (s_t) as well as previous states (h_t).

The Conv-RNN layers, discussed in Section 4.3.1, combine convolution with recurrent nodes. The Conv-RNN kernels are spatially and temporally tied to ensure that they capture motion information. Conv-GRU (see Section 4.3.2) and Conv-SCGRU (see Section 4.3.3) extend the Conv-RNN formulation to gated recurrent units to avoid the exploding and vanishing gradient problems.

Section 4.4 discussed the computational complexity of the different computational layers used in deep learning. Convolutional layers are more efficient on high dimensional inputs than FNN layers. Similarly, Conv-RNN layers are more computationally efficient than RNN layers. The GRU is 25% more computationally efficient than the SCGRU.

Section 4.5 proposes two architectures for activity recognition, namely Temporal-CNN and Conv-RNN. The Temporal-CNN architecture adapts the single-frame architecture to the temporal domain. The Conv-RNN architecture combines convolutional layers and Conv-RNN layers for activity recognition.

Section 4.6 discusses the UML framework design, structure and implementation. The UML framework was implemented using the Python[6] programming language to simplify model construction and optimisation.

[6] `hhtps://www.python.org`

CHAPTER 5 EXPERIMENTATION AND EVALUATION

5.1 CHAPTER OVERVIEW

To determine the usefulness of a statistical learning architecture, a quantitative evaluation of its performance on a learning task is performed. An architecture is evaluated on:

- Its ability to learn good representations from data.
- Its computational complexity.
- Its learning time, i.e. how long it takes to train the architecture.
- Its processing time, i.e. how long it takes to process an input (after training).
- Its performance on a machine learning task.

This chapter describes an evaluation framework for temporal and spatio-temporal deep learning architectures for activity recognition. Firstly the metrics to measure the performance of activity recognition models are discussed. After that, a temporal sampling strategy for the video data is discussed. Then the models investigated in this research are described. Finally, a discussion on model training and evaluation is presented.

5.2 ACTIVITY RECOGNITION MODEL EVALUATION

Activity recognition is a classification learning task, which requires classification metrics to benchmark performance. Classification metrics arise from data in a *confusion matrix*, a table containing informa-

tion on actual (ground truth) and predicted (model outputs) labels. The performance of a classification model is a measure of how well the model can predict the ground truth labels. The activity recognition models used in this research are predictive models learned using datasets with ground truth labels (see Section 2.2.2).

5.2.1 Activity recognition performance metrics

Accuracy, precision, recall, receiver operating characteristic (ROC) curve, area under the curve (AUC), mean average precision (mAP) are the metrics of choice when evaluating classification models. This section discusses the merits of each metric for activity recognition.

5.2.1.1 Accuracy score

Accuracy is highly dependent on the distribution of positive and negative samples in the evaluation set. Given an evaluation set with 1 000 samples, 990 of which have a negative label, if the model classifies all samples as negative, the accuracy is 99%, even though the classifier missed all the positive cases. Accuracy as a performance metric is useful given uniformly distributed categories in the evaluation set. The datasets used in this research have an approximately equal number of samples in each category, which makes accuracy an apt metric to evaluate model performance.

5.2.1.2 AUC score

Given ground truth and predicted labels, precision measures the fraction of retrieved instances that are relevant, recall measures the probability of retrieving a relevant sample, and false positive rate (FPR) measures the likelihood of retrieving a non-relevant sample. An ideal classification system is one with high precision and high recall. Precision and recall calculations ignore true negative predictions (non-relevant samples), which make them useful metrics when the category distribution of the evaluation set is not uniform.

The ROC curve is a plot of the true positive rate (TPR) against the FPR at different thresholds. The ROC curve is a useful metric in binary classification problems. In multi-class classification, the ROC curve is summarised into a single score (AUC) for each label, in a one-vs-all classification scheme.

The AUC score is a single number, which represents the area under the ROC curve. Given a random sample, AUC is the probability that a ranking system assigns a high score to a positive sample. The metric is sensitive to unequal distributions of the classes because of the equal emphasis it places on false positive and negative errors.

5.2.1.3 Mean average precision score

The precision-recall (PR) curve is a plot of the precision against recall at different thresholds. The area under the PR curve (AUC-PR) or average precision (AP) is single value metric that represents the area under the precision-recall curve. The AP score places more emphasis on the positive samples, meaning it is not as sensitive to class imbalances as the AUC-ROC score. The AP score is given by:

$$\mathrm{AP} = \frac{\sum_{k=1}^{N} (P(k) \times \mathrm{rel}(k))}{\sum_{k=1}^{N} \mathrm{rel}(k)} \tag{5.1}$$

where N is the total number of samples, $P(k)$ is the precision at cut-off k, and $\mathrm{rel}(k)$ is an indicator function, which equals one if the sample ranked k is a true positive and zero otherwise.

Mean average precision (mAP) extends AUC-PR to multi-classification problems. he mAP metric assumes that the classes are equally distributed. mAP is given by:

$$mAP = \frac{1}{C} \sum_{i}^{C} \mathrm{AP}(i) \tag{5.2}$$

where C is the number of classes.

5.2.2 Activity recognition dataset evaluation protocols

The evaluation protocols for UCF101 [32] and HMDB51 [33] are the same. The datasets have three different splits, each one of which has training and testing data. Each UCF101 split contains 9 537 training videos, while each HMDB51 split contains $\sim$3 551 training videos. Model training uses split one, for both UCF101 and HMDB51 datasets. Average classification accuracy and mAP scores on the test set of split one are reported. Split one is the most commonly used split in literature [5, 7, 6, 8].

The evaluation protocol for the Dynamic Scenes (Maryland) [35] dataset is a leave-one-out cross-validation strategy [35, 113, 114]. Performance is reported by measuring the mean classification accuracy, mAP and AUC scores. The standard evaluation protocol for the UCF11 [34] varies. Sharma *et*

al. [115] split the dataset into 975 videos for training and 625 for testing; Liu *et al* [34] used a leave-one-out cross-validation strategy; Ravanbakhsh *et al.* [116] used 25-fold leave-one-out cross-validation strategy. This research uses a 10-fold cross-validation strategy, due to model training times (model training took 3 to 5 days when using 10-fold cross-validation). Performance results are reported by measuring the average classification accuracy, AUC and mAP scores.

5.3 ACTIVITY RECOGNITION DATASET CONFIGURATION

5.3.1 Temporal sampling for activity recognition

Actions occupy both spatial and temporal dimensions. The spatial dimension contains the pose and location of an agent performing an activity in the frame. To perform activity recognition, one has to identify the agent and track its motion across a sequence of frames. Spatially, the agent must fall within the frame, which is the case for the datasets used in the research (see Section 2.2.2).

The video clips for the datasets used in this research have an inconsistent number of frames, which does not matter for single-frame architectures [5, 28]. The LRCN and Conv-RNN architectures require sequential frames, and for simplicity, only inputs with constant spatial and temporal dimensions are considered as inputs to these architectures.

When sampling the videos temporally, it is important to ensure that the entire activity is fully captured in the sampled sequence of frames, to ensure proper sequence learning. The datasets used in the research have no frame-wise annotations of where the action starts and ends. Donahue *et al.* [6] used 16 successive frames, sampled from each video clip at 30 fps, as input to their model. This approach assumes that the entire activity falls within the 16 frames, which is not always the case (see Figure 5.1(a)). Figure 5.1(a) shows the first 16 frames sampled (at 30 fps) from a video clip of the *Soccer Penalty* category of the UCF101 [32] dataset. The samples contain the first third of a penalty kick.

Sampling T successive frames from a video clip does not account for activities that span greater than T frames (see Figure 5.1(a)). This research proposes that one should sample T equally spaced frames (sampled at the video clip's frame rate), to avoid sampling an incomplete sequence of an activity.

(a) The first sixteen (top-left to bottom-right) sequential samples.

(b) Sixteen (top-left to bottom-right) equally spaced sequential samples.

Figure 5.1. Sequential (Figure 5.1(a)) vs. equally spaced sampling (Figure 5.1(b)) on the *Soccer Penalty* category of the UCF101 [32] dataset.

Figure 5.1(b) shows 16 equally sampled frames from a video clip of the *Soccer Penalty* category of the UCF101 [32] dataset. This sampling procedure is more desirable than the incomplete sequence generated in Figure 5.1(a). If an agent performs an activity multiple times, e.g. when a person performs many *Push-ups*, equally spaced sampling results in a sequence containing multiple activity occurrences. However, this scenario is more acceptable than not capturing the full sequence.

5.3.2 Dataset organisation

Activity recognition datasets used in this research are a collection of video samples. Before any pre-processing, the video samples are converted to images and stored as a 5D dataset in a HDF5 file. Storing the data in a HDF5 file allows for faster batch loading from hard drive to RAM. Loading the entire dataset to RAM was impossible due to resource limitations.

The converted datasets are organised based on the number of frames T, dataset split and frame type (RGB). The datasets are created by sampling T equally spaced frames from each video clip. The dataset image frames have a width and height of 320×240. This research is restricted to RGB inputs

5.3.3 Data pre-processing

Choosing a suitable representation for the input data is a vital part of any machine learning task. Appropriate data representation speeds up convergence and leads to better generalisation in statistical learning algorithms [66, 77]. Highly correlated data creates an unnecessarily high-dimensional input space, which leads to an excessive number of learning parameters and overfitting. Overfitting is addressed through data augmentation (see Section 2.4.4) and model architecture (i.e. dropout (see Section 3.5.3)).

If the dataset is stationary, which is the case for images, zero-centring the data speeds up training. Zero-centring removes the average intensity of the data points (pixels). The stationary property does not apply across the different image channels. Individually, channels exhibit the stationary property, and as such, contrast normalisation is applied channel-wise. Channel-wise feature extraction allows the learning algorithm to focus on extracting structure in the images. The RGB data inputs are contrast

normalised using the VGG_mean[1] mean image. The VGG_mean[1] image is used because the CNN-S [67] model is used as the base for all models investigated in this research (see Section 5.4.1).

5.4 ACTIVITY RECOGNITION MODELS

5.4.1 Base model

This research aims to investigate the best architecture for activity recognition. To this end, all models follow the same structure to ensure comparability. This research uses an eight-layer DNN model, because of limited hardware resources and for compatibility with results in [5, 6]. The eight-layer structure is a staple in deep hierarchical visual learning, e.g. AlexNet [15], ZFNet [47] and Visual Geometry Group (VGG)-CNNs [54, 67].

The CNN-S model [67], which achieved a 13.1% top-5 error score on the validation set of the ImageNet [42] dataset, is the base model used for all models evaluated in this research. The CNN-S model (see Table 5.1) contains five convolutional layers (Conv 1–5), two fully-connected layers (fc6 and fc7) and a softmax layer (fc8). The design for the CNN-S [67] model is based on the ZFNet [47] architecture. The model accepts a three channel image with a width and height of 224×224.

Table 5.1 shows the layer configuration for the CNN-S [67] model. The three rows provide information associated with each layer. For convolutional layers, the first row specifies the number of convolution filters and their receptive field size as '$num \times size \times size$'. The second row indicates the convolution stride ('st.') and spatial padding ('pad'). The third row indicates whether a local response normalisation (LRN) [31, 45] operation takes place and whether a max-pooling down-sampling operation happens. The fc6 and fc7 layers have 4 096 units. Dropout is used to regularise the fc layers. All layers use the ReLU activation function.

The CNN-S[2] Caffe [112] model was converted to the UML framework (see Section 4.6). The conversion was validated by visualising the Conv1 weight parameters to confirm that they resemble Gabor-like filters (see Figure 5.2).

[1] `http://www.robots.ox.ac.uk/~vgg/software/deep_eval/releases/bvlc/VGG_mean.mat`

[2] `http://www.robots.ox.ac.uk/~vgg/software/deep_eval/releases/bvlc/VGG_CNN_S.caffemodel`

Table 5.1. Layer configuration for the CNN-S [67] model.

Conv1	Conv2	Conv3	Conv4	Conv5	fc6	fc7
$96 \times 7 \times 7$	$256 \times 5 \times 5$	$512 \times 3 \times 3$	$512 \times 3 \times 3$	$512 \times 3 \times 3$	4096	4096
st. 2, pad 0	st. 2, pad 1	st. 1, pad 1	st. 1, pad 1	st. 1, pad 1	dropout	dropout
LRN, 3× pool	LRN, 2× pool	–	–	3× pool	–	–

Figure 5.2 shows the Gabor-like filter learned by the first layer of the CNN-S model.

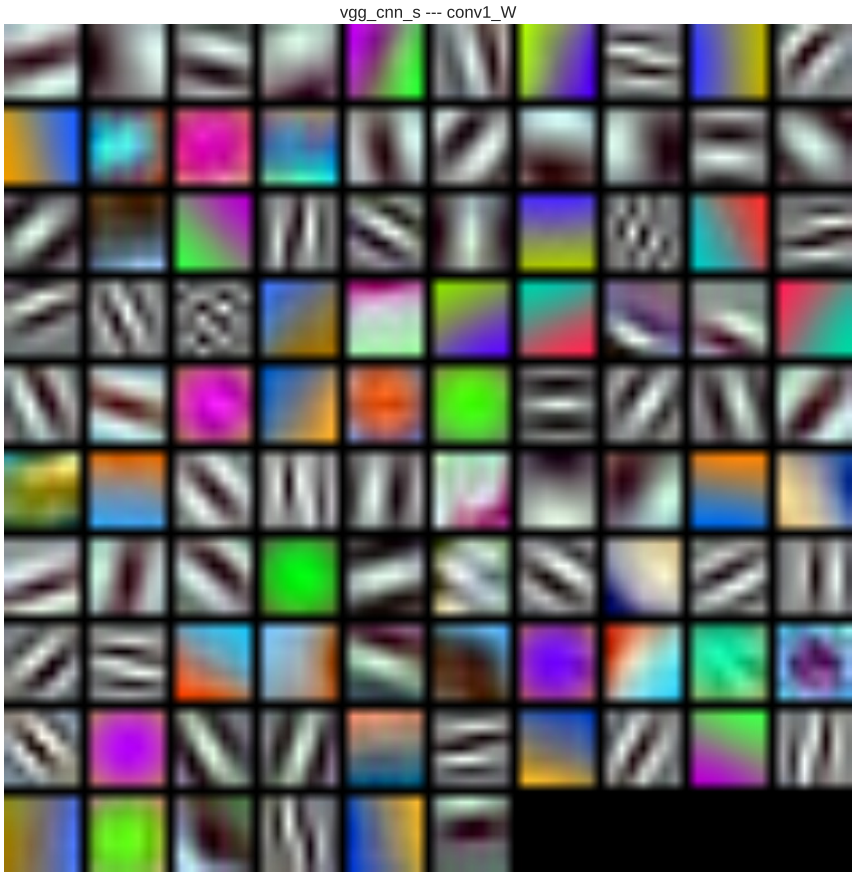


Figure 5.2. The 96 convolutional kernels of size $96 \times 7 \times 7$ learned by the Conv1 layer of the CNN-S model.

Activation maximisation [16, 117, 118, 119, 120] was also used to validate model conversion (see Figure 5.3). Activation maximisation provides insight into DNNs models, by identifying the types of inputs that maximally activate the units of a layer. Figure 5.3 shows activation maximisation visualisation of 20 randomly selected fc8 units of the CNN-S model.

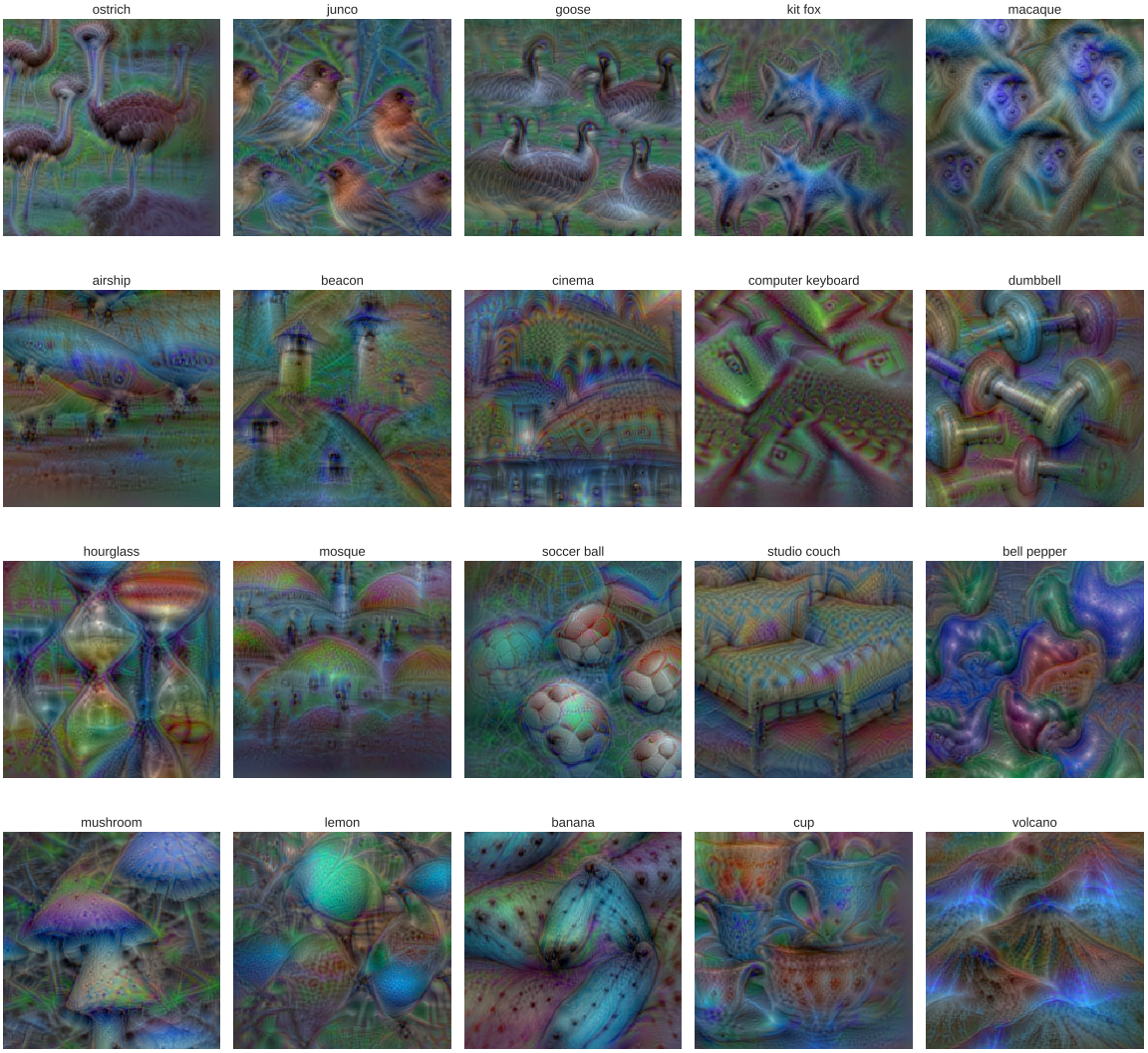


Figure 5.3. Activation maximisation visualisation of 20 randomly selected fc8 units of the CNN-S model.

5.4.2 Temporal-CNN models

The Temporal-CNN architecture for activity recognition (see Section 4.5.1) is the base architecture to compare models that learn features independently at each time-step and those that learn temporally dependent features (LRCN [6] and Conv-RNN). The Temporal-CNN architecture applies a CNN model (CNN-S [67] in this case) at each time-sep (t) for T time-steps. Temporal-CNN architectures are optimised in the same manner as temporal models.

Temporal-CNN models are end-to-end trained using the pre-trained CNN-S [67] model. The fc6 and fc7 layers use dropout rates of $p = 0.8$ and $p = 0.9$ respectively. The dropout rates are based previous literature [5, 7]. The softmax layer (fc8) is modified based on the number of categories in each dataset.

5.4.3 Conv-RNN models

The Conv-RNN models replace the Conv3–Conv5 layers of the CNN-S [67] model with Conv-RNN layers (see Section 4.3). The Conv1–Conv2 layers of the CNN-S [67] model are distributed along the temporal dimension of the input video frames. The fc6 layer is replaced with a RNN layer that applies Cooijmans *et al.*'s [121] recurrent batch normalisation. The RNN layer, which has 1 024 learning units (as suggested in [6]), is followed by a temporally distributed dropout layer with a dropout rate of $p = 0.9$. The softmax layer (applied at each time-step) is modified based on the number of categories in each dataset.

This research investigated two different Conv-RNN models:

- **Conv-RNN01**: Replaces the Conv5 layer of the CNN-S [67] model with an equivalent (same configuration and number of units, i.e. 512) Conv-RNN layer followed by RNN, dropout and softmax layers.
- **Conv-RNN03**: Replaces the Conv3–Conv5 layers of the CNN-S [67] model with equivalent Conv-RNN layers followed by RNN, dropout and softmax layers.

GRU and SCGRU models (based on the Conv-RNN01 and Conv-RNN03 models) are trained for each dataset. The Conv-RNN and RNN layers use the ELU activation (see Section 3.3.4.4).

5.5 MODEL TRAINING AND EVALUATION PROTOCOL

This section describes the approach taken when training and evaluating models for activity recognition. The protocol, based on previous research [5, 7, 8] (see Section 2.4), is the same for all modelss; differences lie in the number of time-step samples and batch-size used during training and testing.

The data pre-processing steps are the same for all models trained and evaluated. The pre-processing steps are applied consistently across all frames of a sample. The N, T-frame, samples are linearly resized to have a width and height of 340×256. The samples are then multi-scale cropped using square windows randomly sampled from $\{256, 224, 192, 168\}$ sizes; the cropping region is randomly sampled from the {centre, random, top-left, top-right, bottom-left, bottom-right} positions of the image. The cropped samples are rescaled to have a width and height of 224×224. The rescaled samples are fliped with a random probability of 0.5, before contrast normalisation, using the VGG_mean[3] image, is applied. The above data pre-processing steps are based on the discussion in Section 2.4.4.

The models are optimised using Adam [98] (using the suggested hyperparameters and a learning rate of $\eta = 10^{-4}$) using NLL cost in (2.2) on page 18. The learning rate was chosen from $\eta \in \{10^{-3}, 5 \times 10^{-4}, 2 \times 10^{-4}, 10^{-4}, 10^{-5}\}$ by examining the 1st-fold performance of the models using the Maryland [35] dataset. Max-norm and L^2 regularisations (*max-norm*= 5 and $L^2 = 10^{-6}$) are used to regularise the models. A gradient clipping $threshold = 10$ is applied to avoid the exploding gradient problem.

At test time, the multi-scale and random cropping augmentation steps are removed. T equally spaced frames are extracted and cropped using {centre, top-left, top-right, bottom-left, bottom-right} image positions to create five samples from each video clip [5, 6, 7, 8]. The five samples are flipped and combined with the unflipped samples to create ten T frame samples [5, 6, 7, 8]. The ten sample predictions are averaged to create T predictions, which are then averaged to create a vector of softmax predictions ($\hat{y}$). Model performance is a measure of how close these predictions are to the ground truth labels (y).

5.6 CHAPTER SUMMARY

This chapter describes an evaluation framework for deep temporal architectures for activity recognition. Section 5.2 discussed accuracy, AUC and mAP as metrics to evaluate the performance activity recognition models. The discussion showed that accuracy and mAP are apt metrics for the datasets used in this research, due to the test sets having uniformly distributed number of examples per-category. The

[3] http://www.robots.ox.ac.uk/~vgg/software/deep_eval/releases/bvlc/VGG_mean.mat

UCF101 [32] and HMDB51 [33] datasets have training and testing sets. UCF11 [34] and Maryland [35] datasets are evaluated using cross-validation.

Section 5.3.1 proposes sampling equally spaced frames from a video clip. This sampling strategy avoids sampling an incomplete sequence of an activity (see Figure 5.1). The sampled frames are stored as a 5D dataset in an HDF5 file, for fast batch loading during training. The sequential frames are contrast-normalised using the VGG_mean[4] image.

Section 5.4 discusses the activity recognition models evaluated in this research. The model structure is based on the pre-trained CNN-S [67] model. The CNN-S [67] is converted from Caffe [112] to the UML framework and validated using filter visualisation (see Figure 5.2) and activity maximisation (see Figure 5.3).

Section 5.5 discussed the model training and evaluation steps. Wang *et al.*'s [7] data augmentation strategy is used to train the models. The models are optimised using Adam [98] and the NLL cost defined in (2.2). The models are regularised using L^2 and *max-norm* regularisation with gradient norm-clipping. The models are evaluated using the same strategies in [5, 6, 7, 8] (see Section 5.5).

[4] `http://www.robots.ox.ac.uk/~vgg/software/deep_eval/releases/bvlc/VGG_mean.mat`

CHAPTER 6 EXPERIMENTAL RESULTS

6.1 CHAPTER OVERVIEW

This research compared three different architectures for activity recognition, namely LRCN, Temporal-CNN and Conv-RNN. Temporal-CNN architectures (see Section 4.5.1) assume temporal independence between sequential frames. LRCN architectures (see Section 2.4.3) use deep visual models (CNNs) to encode sequential frames into sequential feature vectors and a RNN learns the temporal dynamics from the sequential feature vectors. The Conv-RNN architectures (see Section 4.5.2) combine CNNs and RNNs to learn spatio-temporal features for activity recognition.

This chapter examines models based on the precceding architectures on the UCF101 [32], HMDB51 [33], UCF11 [34] and Maryland [35] activity recognition datasets (see Section 2.2.2). The experiments are carried out to determine the best architecture and model for activity recognition based on the average and the per-category performances. Comparison of results with state-of-the-art results is carried out in Section 7.3.

6.2 UCF101 ACTIVITY RECOGNITION TASK

The UCF101 [32] activity recognition task involves automatically classifying 101 human activities. This section presents an overview of model performance on the UCF101 dataset.

6.2.1 UCF101 model performance summary

Table 6.1 summarises the performance of different models on split one of the UCF101 dataset. The models are trained using $T = 10$ RGB frames as inputs, and model performance is measured using accuracy and mAP on the test set of split one. Notably, LRCN based models (specifically LRCN6-SCGRU model) achieve the best performance on the dataset. The performance of Conv-RNN based models is inversely proportional to the number of Conv-RNN layers. All models experience overfitting, with ConvSCGRU03 model overfitting the most (an accuracy difference of 44.74% between the training and testing sets).

Table 6.1. Model performance comparison on split one of the UCF101 [32] the dataset.

Model	Testing set		Training set	
	Accuracy (%)	mAP (%)	Accuracy (%)	mAP (%)
ConvGRU01	62.569	65.950	99.696	99.701
ConvGRU03	58.736	63.244	99.717	99.719
ConvSCGRU01	63.230	65.822	99.769	99.773
ConvSCGRU03	54.851	59.463	99.591	99.597
LRCN6-GRU	67.301	70.066	99.769	99.773
LRCN6-SCGRU	**68.226**	**70.752**	99.769	99.773
Temporal-CNNS	67.275	69.837	99.822	99.824

The LRCN6-GRU model outperforms the Temporal-CNNS model by a small margin, a performance gap of 0.026%. The LRCN6-SCGRU and LRCN6-GRU model performance differ by 0.925%. The LRCN6-SCGRU, with an accuracy score of 68.226%, is comparable to Donahue *et al.*'s [6] LSTM based LRCN model which achieved an accuracy of 68.19% averaged across the three UCF101 dataset splits. SCGRU based models outperform GRU based models (an accuracy difference of 0.925% for LRCN models and 0.661% for Conv-RNN01 models), except in the case of Conv-RNN03 models (an accuracy difference of -3.885%).

6.2.2 UCF101 model categorical analysis

A summary of the model performance (see Table 6.1) on a dataset does not reveal the complete picture of the model performance. Analysing the per-category classification accuracy of the models provides further insight on model performance. Figure 6.1 shows a boxplot of the per-category classification accuracy of the different models evaluated on split one of the UCF101 [32] dataset. The LRCN6-SCGRU model (see Figure 6.1) is the one model to not miss a category during classification (LRCN6-GRU and Temporal-CNNS models miss one category each). The Conv-RNN models miss multiple categories and have the 25% of the categories achieving less than 47% accuracy.

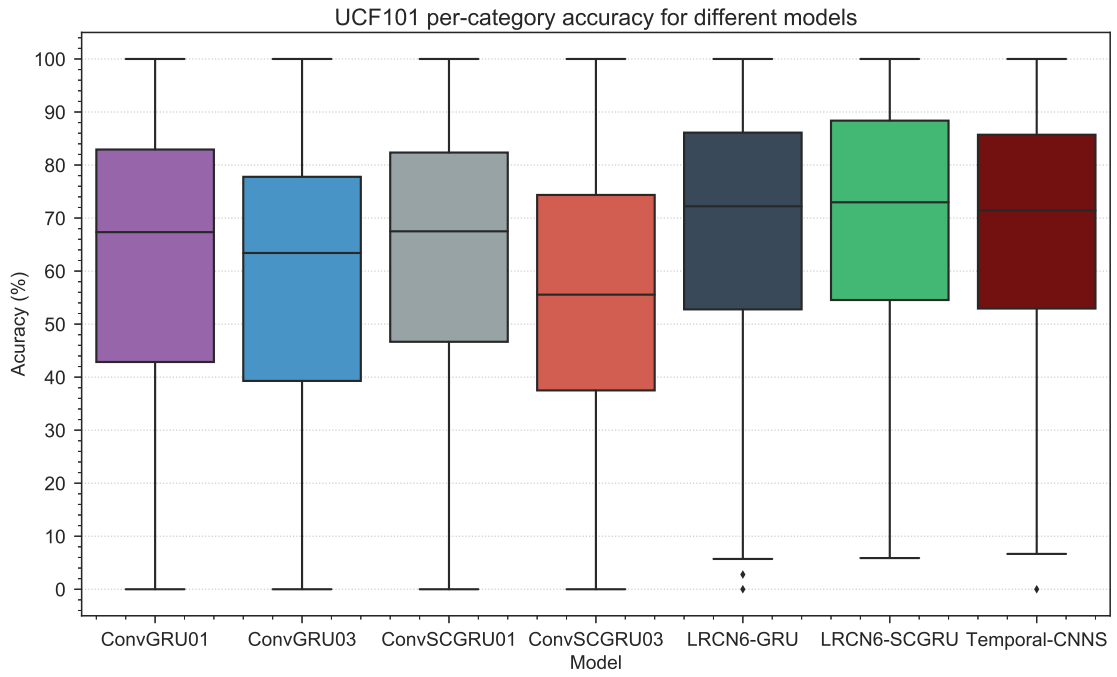


Figure 6.1. The per-category classification accuracy of different models on split one of the UCF101 dataset.

The best performing model (LRCN6-SCGRU) classifies 75% of the categories with an accuracy greater than 54%, and 25% of the categories have an accuracy greater than 88%. The ConvSCGRU01 model classifies 75% of the categories at a higher accuracy than the ConvGRU01 model (47% compared to 44%), which explains the small percentage gain in accuracy (0.661%) between the models.

Figure 6.1 showed that the investigated models struggle to categorise certain activities. Table 6.2 shows categories for each model with an accuracy less than 20%, accuracies are in parentheses. The LRCN6-

SCGRU and Temporal-CNNS models have the fewest number of categories with an accuracy less than 20%, four and five categories respectively (see Table 6.2). However, all models struggle to classify *Handstand Walking , Nunchucks, Jump Rope* and *Push Ups* activities. These activities are repeated multiple times within a video clip, so the $T = 10$ equally spaced sampled frames might not contain the full repeated sequential action (see Section 5.3.1). However, the fact that the Temporal-CNNS model also struggles with this categories suggests the cause of the miss classifications might be something else. The SCGRU based Conv-RNN model miss the most categories, three each.

Table 6.2. Model accuracy for the worst performing categories (accuracy less than 20%) on split one of the UCF101 [32] dataset.

ConvGRU01	ConvGRU03	ConvSCGRU01	ConvSCGRU03	LRCN6-GRU	LRCN6-SCGRU	Temporal-CNNS
Juggling Balls (0)	Handstand Walking (0)	Handstand Walking (0)	Nunchucks (0)	Jump Rope (0)	Handstand Walking (5.882)	Handstand Walking (0)
Jump Rope (0)	Nunchucks (2.857)	Jump Rope (0)	Playing Daf (0)	YoYo (2.778)	Jump Rope (10.526)	Push Ups (6.67)
Handstand Walking (2.941)	Push Ups (10)	Nunchucks (0)	Push Ups (0)	Nunchucks (5.714)	Push Ups (13.33)	Nunchucks (8.571)
Boxing Punching Bag (8.163)	Jump Rope (10.526)	Pizza Tossing (6.061)	Body Weight Squats (3.33)	Boxing Punching Bag (6.122)	Nunchucks (14.286)	JumpRope (10.526)
Archery (12.195)	Cricket Shot (12.245)	Shaving Beard (11.628)	Jump Rope (5.263)	Push Ups (6.67)		YoYo (16.67)
YoYo (13.889)	Pizza Tossing (15.152)	Hammering (15.152)	Boxing Punching Bag (6.122)	Body Weight Squats (13.33)		
Nunchucks (14.286)	Body Weight Squats (16.67)	Cricket Shot (16.327)	YoYo (8.33)	Handstand Walking (14.706)		
Pull Ups (17.857)	Pull Ups (17.857)		Handstand Walking (8.824)	Pizza Tossing (15.152)		
High-Jump (18.919)	YoYo (19.444)		Hammering (12.121)	High-Jump (18.919)		
Jumping Jack (18.919)			Archery (12.195)			
			Brushing Teeth (16.67)			
			Wall Push Ups (17.143)			
			Pull Ups (17.857)			

6.3 HMDB51 ACTIVITY RECOGNITION TASK

The HMDB51 [33] activity recognition task involves automatically classifying 51 human action activities. This section presents an overview of model performance on the HMDB51 dataset.

6.3.1 HMDB51 model performance summary

Table 6.3 summarises the performance of different architectures on split one of the HMDB51 [33] activity recognition dataset. The models are trained using $T = 10$ RGB frames as inputs, and model performance is measured using accuracy and mAP on the testing, validation and training sets. LRCN based models (specifically LRCN6-SCGRU model) achieve the best performance on the dataset (an accuracy of 38.583%, which is 2.034% better than the LRCN6-GRU model). The performance of Conv-RNN based models decreases as the number of Conv-RNN layers increases. All models experience overfitting, with ConvGRU03 and ConvSCGRU01 models overfitting the most with accuracy differences of 69.718% and 69.356% between the training and testing sets respectively.

Table 6.3. Model performance comparison on split one of the HMDB51 [33] dataset.

Model	Testing set		Validation set		Training set	
	Accuracy (%)	mAP (%)	Accuray (%)	mAP (%)	Accuracy (%)	mAP (%)
ConvGRU01	31.955	35.515	36.855	53.582	97.832	97.959
ConvGRU03	24.672	28.237	36.675	51.546	94.114	95.240
ConvSCGRU01	29.396	34.448	37.995	52.247	98.761	98.817
ConvSCGRU03	24.409	27.376	34.814	49.132	85.103	87.927
LRCN6-GRU	36.549	37.037	**45.138**	51.602	93.495	94.409
LRCN6-SCGRU	**38.583**	**37.229**	44.358	**54.618**	92.453	93.784
Temporal-CNNS	30.512	33.651	41.537	53.156	98.761	98.812

The SCGRU based LRCN model (LRCN6-SCGRU) achieves the best accuracy (38.583%) on the testing set. However, the LRCN6-GRU model has the best performance on the validation set, albeit with a lower mAP score (51.602%) than the LRCN6-SCGRU's 54.618%. Notably, the performance of GRU based Conv-RNN models differs by 0.18% and 7.323% on the validation and testing sets

respectively. The ConvSCGRU03 model experienced high bias, an accuracy of 85.103% on the training set compared to ConvGRU03's 94.114% accuracy, which suggest that the model was not optimised for long enough and the model complexity is low for this type of architecture.

6.3.2 HMDB51 categorical analysis

Figure 6.2 shows a boxplot of the per-category classification accuracy of the different models evaluated on split one of the HMDB51 [33] dataset.

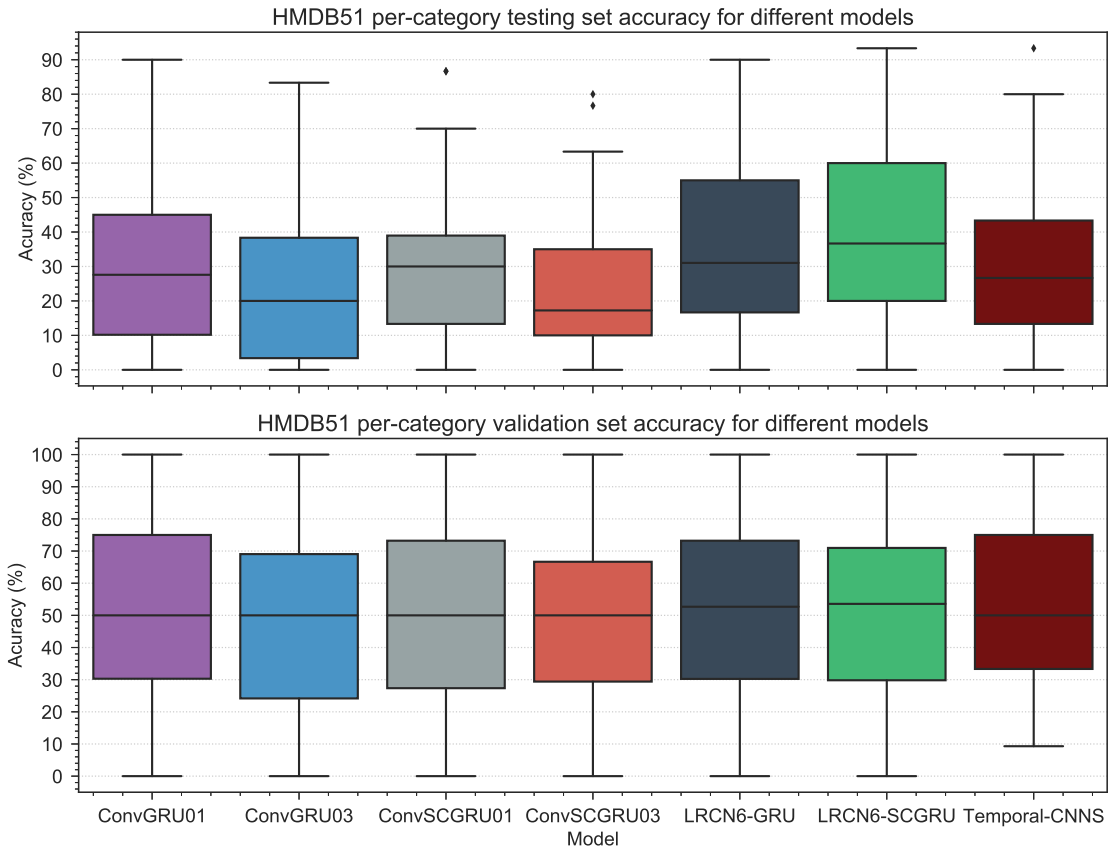


Figure 6.2. The per-categorical model performance of different models on split one of the HMDB51 dataset.

All models (see Figure 6.2) fail to classify some categories on the test set. Temporal-CNNS is the one model to not miss a category on the validation set. Notably, all Conv-RNN based models have an accuracy greater than 50% for 50% of the categories on the validation set. On the testing set, the Conv-RNN03 models overfit more than the Conv-RNN01 models. The LRCN-SCGRU model has

the best performance because it achieves accuracies greater than 20% on 75% of the categories. The ConvSCGRU03 model struggles because 75% of the categories have accuracies less than 35%.

Table 6.4, shows categories for each model with an accuracy less than or equal to 15% (accuracies are in parentheses). The investigated models struggled to classify some HMDB51 [33] activities (see Table 6.4). The ConvGRU03 model completely misclassify $9/51$ categories and has $24/51$ categories with accuracies less than or equal to 15%. The ConvSCGRU03 model classify $23/51$ categories with accuracies less than or equal to 15%, and completely miss $3/51$ categories. The best performing model (LRCN-SCGRU) misses $5/51$ categories.

Table 6.4. Model accuracy for the worst performing categories (categories with an accuracy less than or equal to 15%) on split one of the HMDB51 dataset.

ConvGRU01	ConvGRU03	ConvSCGRU01	ConvSCGRU03	LRCN6-GRU	LRCN6-SCGRU	Temporal-CNNS
clap (0)	wave (0)	fall floor (0)	cartwheel (0)	throw (0)	cartwheel (0)	cartwheel (0)
kick (0)	pick (0)	jump (0)	throw (0)	cartwheel (3.33)	pick (0)	pick (0)
pick (0)	kick (0)	kick (0)	wave (0)	pick (3.33)	punch (0)	punch (0)
throw (0)	jump (0)	pick (0)	smile (3.33)	stand (3.33)	throw (0)	throw (0)
wave (3.33)	flic flac (0)	cartwheel (3.33)	kick (3.33)	wave (3.33)	wave (0)	wave (3.33)
cartwheel (6.67)	stand (0)	hit (3.33)	turn (3.33)	punch (3.448)	sit (3.33)	flic flac (6.67)
climb (6.67)	eat (0)	situp (3.33)	punch (3.448)	fall floor (6.67)	swing baseball (3.33)	stand (6.67)
fall floor (6.67)	clap (0)	wave (3.33)	laugh (6.67)	smoke (6.67)	shoot gun (3.448)	swing baseball (6.67)
swing baseball (6.67)	sword exercise (0)	throw (3.448)	drink (6.67)	kick (10)	hit (6.67)	sit (10)
punch (6.897)	handstand (3.33)	sit (6.67)	clap (6.67)	sit (10)	sword exercise (6.67)	handstand (13.33)
hit (10)	turn (3.33)	run (10)	stand (6.67)	swing baseball (10)	kick (10)	hit (13.33)
run (10)	laugh (3.33)	smoke (13.33)	pick (6.67)	handstand (13.33)		kick (13.33)
stand (10)	hit (3.33)	swing baseball (13.33)	swing baseball (10)			smoke (13.33)
sword (10.345)	throw (3.448)	sword exercise (13.33)	kick ball (10)			sword exercise (13.33)
handstand (13.33)	fall floor (6.67)	turn (13.33)	jump (10)			somersault (13.793)
	push (6.67)		fencing (10)			
	swing baseball (6.67)		eat (10)			
	shoot gun (6.897)		climb stairs (10)			
	cartwheel (10)		shoot ball (13.33)			
	climb stairs (10)		hit (13.33)			
	kick ball (10)		fall floor (13.33)			
	fencing (13.33)		sword exercise (13.33)			
	sword (13.793)		draw sword (13.33)			
	shoot bow (13.793)					

6.4 UCF11 ACTIVITY RECOGNITION TASK

The UCF11 [34] activity recognition task involves automatically classifying 11 human actions. This section presents an overview of the 10-fold cross-validation model performance on the UCF11 dataset.

6.4.1 UCF11 model performance summary

Table 6.5 summarises the 10-fold cross-validation performance of different models on the UCF11 [34] activity recognition dataset. The models are trained using $T = 10$ RGB frames as inputs, and model performance is measured using accuracy, mAP and AUC scores. The Temporal-CNNS model achieved the best performance (an accuracy of 97.438%) on the dataset. All models achieved accuracies greater than 92%.

Table 6.5. The 10-fold cross-validation model performance on the UCF11 [34] dataset.

Model	Accuracy (%)	mAP (%)	AUC (%)
ConvGRU01	96.125	96.114	99.886
ConvGRU03	92.938	93.008	99.623
ConvSCGRU01	95.125	95.193	99.850
ConvSCGRU03	93.938	93.884	99.642
LRCN6-GRU	97	97.029	99.906
LRCN6-SCGRU	96.562	96.557	99.871
Temporal-CNNS	**97.438**	**97.469**	**99.937**

6.4.2 UCF11 categorical analysis

Figure 6.3 shows a boxplot of the 10-fold cross-validation per-category classification accuracy for different models on the UCF11 [34] dataset. Temporal-CNNS, ConvGRU01 and LRCN6-GRU models perfectly classify five, four and four categories respectively.

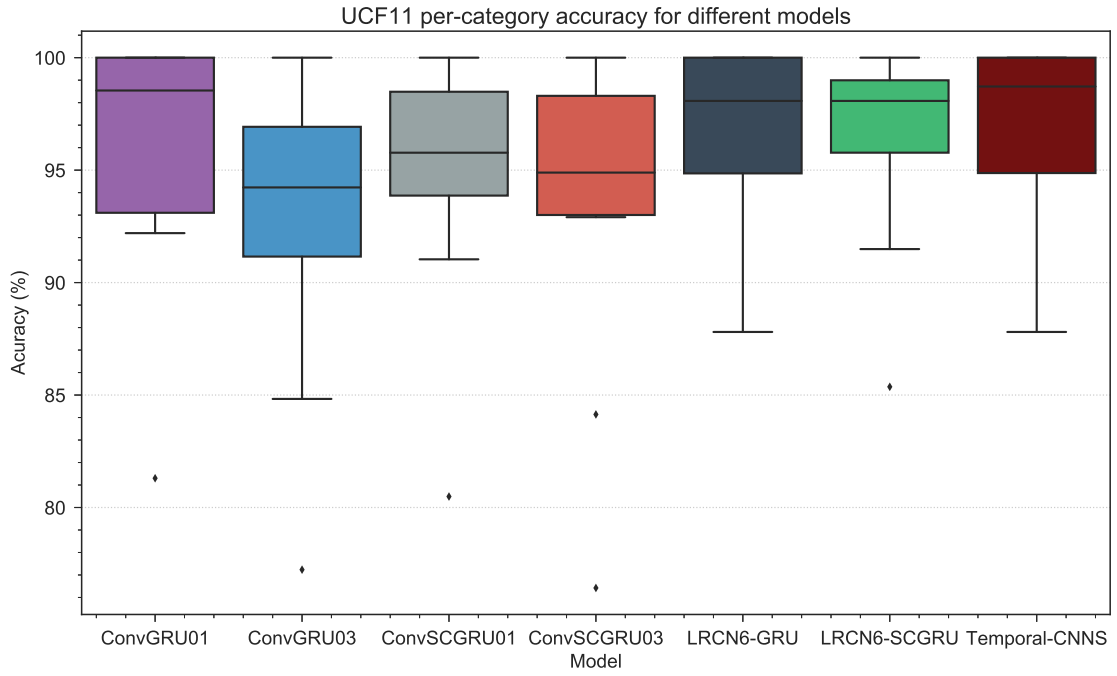


Figure 6.3. The 10-fold cross-validation per-category classification accuracy of different models on the UCF11 [34] dataset.

All models struggle to classify the *walking* activity category. The ConvGRU03 and ConvSCGRU03 achieve accuracies of 77.236% and 76.423% respectively on the walking activity, which explains the performance dip in these models.

6.5 DYNAMIC SCENES (MARYLAND) ACTIVITY RECOGNITION TASK

The Dynamic Scenes (Maryland) [35] activity recognition task involves automatically classifying 13 motion based activities. This section presents an overview of the 26-fold cross-validation model performance on the Maryland [35] dataset.

6.5.1 Maryland performance summary

Table 6.6 summarises the 26-fold cross-validation performance of different models on the Maryland [35] activity recognition dataset. The models are trained using $T = 16$ RGB frame samples as inputs (see Section 5.3.1), and model performance is measured using accuracy, mAP and AUC scores.

The LRCN6-SCGRU model achieved the best performance (85.385% average accuracy) on the dataset. The Conv-RNN03 based models have the worst performance (accuracies of 70.760% and 68.462% for ConvSCGRU03 and ConvGRU03 models respectively).

Table 6.6. The 26-fold cross-validation model performance on the Maryland [35] dataset.

Model	Accuracy (%)	mAP (%)	AUC (%)
ConvGRU01	82.308	83.666	98.173
ConvGRU03	68.462	68.814	95.032
ConvSCGRU01	79.231	77.967	98.365
ConvSCGRU03	70.769	70.622	95.788
LRCN6-GRU	81.538	85.016	97.506
LRCN6-SCGRU	**85.385**	**88.511**	**98.955**
Temporal-CNNS	79.231	82.176	98.109

6.5.2 Maryland categorical analysis

Table 6.7 shows categories for each model with an average accuracy less than 60% (accuracies are in brackets). All models struggle to classify *landslide* and *volcano eruption* motions, which have a high inter-class correlation (see Section 2.2.2). The high inter-class correlation might explain the low performance on these categories and suggests that the models failed to find a decision boundary between the classes. The best performing model (LRCN6-SCGRU) has one category with an accuracy less than or equal to 60%.

Table 6.7. Model accuracy for the worst performing categories (categories with an accuracy less than or equal to 60%) on the Maryland [35] dataset.

ConvGRU01	ConvGRU03	ConvSCGRU01	ConvSCGRU03	LRCN6-GRU	LRCN6-SCGRU	Temporal-CNNS
landslide (60)	landslide (30)	volcano eruption (40)	landslide (40)	landslide (40)	landslide (60)	landslide (60)
volcano eruption (60)	whirlpool (50)	landslide (60)	avalanche (60)	volcano eruption (60)		waterfall (60)
whirlpool (60)	avalanche (60)	whirlpool (60)	volcano eruption (60)			whirlpool (60)
	waterfall (60)		waterfall (60)			

Figure 6.4 shows a boxplot of the per-category classification accuracy for different models on the Maryland [35] dataset. LRCN6-SCGRU model perfectly classifies 30% (four) of the categories. The ConvGRU03, ConvSCGRU03, ConvSCGRU01 and LRCN6-GRU models achieved accuracies of 30%, 30%, 40% and 40% respectively on the landslide activity. This outlier category explains the performance dip of these models.

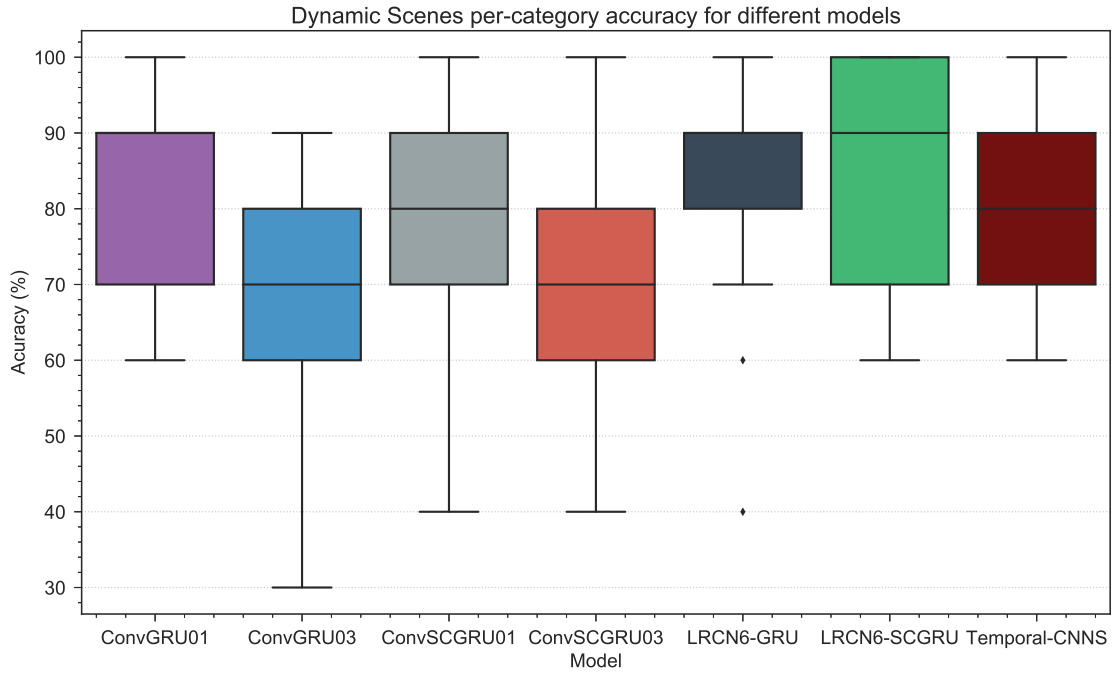


Figure 6.4. The 26-fold cross-validation per-category classification accuracy of different models on the Maryland [35] dataset.

6.6 CHAPTER SUMMARY

This chapter examined the performance of LRCN, Temporal-CNN and Conv-RNN based models on the UCF101 [32], HMDB51 [33], UCF11 [34] and Maryland [35] activity recognition datasets. The performance of Conv-RNN base models decreased as the number of Conv-RNN layers (see Section 4.3.1) increased. Conv-RNN03 models had the worst performance and overfit models across all datasets. ConvGRU03 and ConvSCGRU01 models experienced the worst overfiting on the HMDB51 [33] dataset (see Section 6.3). ConvSCGRU03 model experienced high bias on the HMDB51 [33] dataset (see Sction 6.3).

Of the investigated models, the LRCN6-SCGRU (with an accuracy of 68.226%) had the best performance on the UCF101 [32] dataset. All models overfit the UCF101 [32] data, and the ConvSCGRU03 model overfits the most with an accuracy difference of 44.74% between the training and testing sets. All models struggle to classify *Handstand Walking , Nunchucks, Jump Rope* and *Push Ups* activities.

All investigated models struggled on the HMDB51 [33] dataset (see Section 6.3). The best performing model (LRCN6-SCGRU, with a testing set accuracy of 38.583%) on the dataset missed 5/51 categories. The HMDB51 [33] is a complex dataset, which has testing and training sets sampled from different distributions.

The Temporal-CNNS model with an accuracy of 97.438% (see Table 6.5) outperformed all other investigated model on the UCF11 [34] dataset. A categorical analysis of the models on the UCF11 [34] showed the *walking* activity as the most difficult to classify. The LRCN6-SCGRU model, with an accuracy of 85.386% accuracy, outperformed all other investigated models on the Maryland [35] dataset (see Section 6.5). Landslide and volcano eruption activities were the most difficult to classify for all models. The high inter-class correlation between landslide and volcano eruption motions might explain the low model performance on these categories.

The proposed SCGRU based models consistently outperformed GRU based models for LRCN and Conv-RNN architectures, except in a few exceptional cases – the Conv-RNN03 models on the UCF101 [32] and Conv-RNN01 on the HMDB51 [33] and UCF11 [34].

CHAPTER 7 DISCUSSION

7.1 CHAPTER OVERVIEW

This chapter answers the research questions raised in Section 1.3. Firstly, the importance of the temporal dimension is examined and discussed. After that, the results in Chapter 6 are discussed in the context of existing activity recognition literature. Finally, an analysis of the LRCN6-SCGRU and ConvSCGRU03 model training, on the UCF101 [32] dataset, is performed to examine what happens during model optimisation.

7.2 TEMPORAL DIMENSION IN ACTIVITY RECOGNITION

This section answers the question, *how important is the temporal dimension to deep learning architectures for activity recognition?* (see Section 1.3). Activities have a spatial and temporal component and categorising activities without taking into account the temporal dimension amounts to object recognition. To this effect, three temporal architectures for activity recognition were investigated. The investigated architectures:

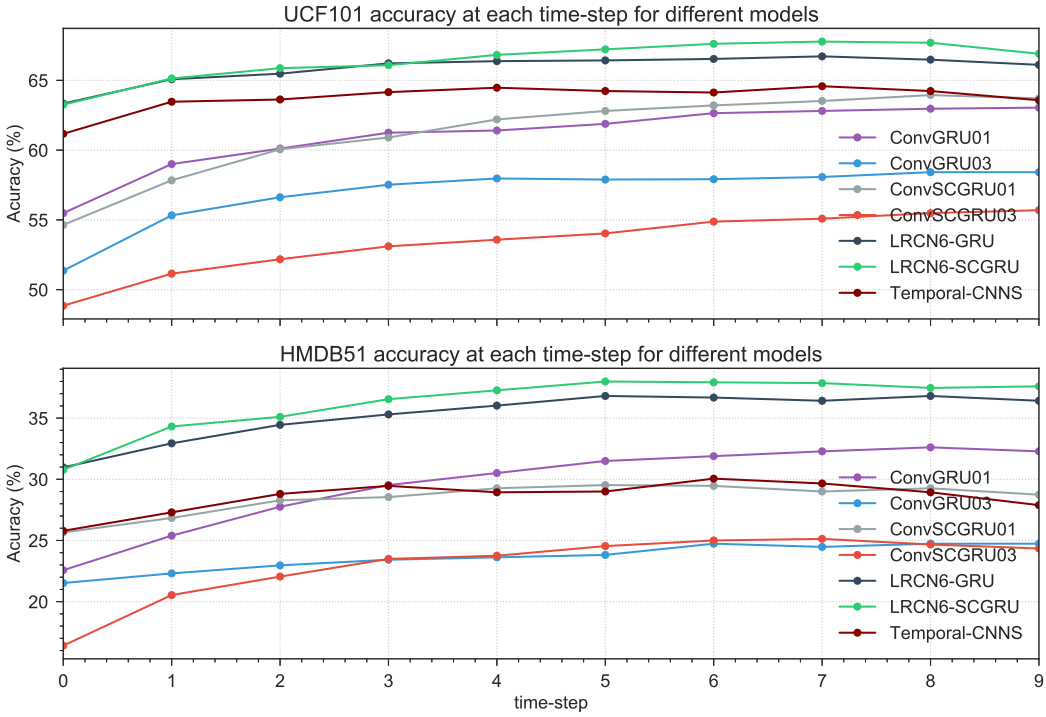
- learn features independently at each time-step (Temporal-CNN (see Section 4.5.1));
- extract feature vectors using a visual model and then learn temporally dependent features from the extracted feature vectors (LRCN (see Section 2.4.3)); and
- extract spatio-temporal features using Conv-RNN and RNN layers (Conv-RNN (see Section 4.5.2)).

Intuitively a temporal categorisation model performs object recognition for the first few frames and then starts to classify salient motion. Examining the model's average accuracy at each time-step, i.e. at each successive frame, should reveal a monotonically increasing accuracy with each time-step (for a temporally dependent model). A temporally independent model should have an average accuracy that is more or less constant at each time-step, with small perturbations from time-step to time-step. Figure 7.1(a) shows the accuracy at each time-step for different models on split one of the UCF101 [32] and HMDB51 [33] datasets, while Figure 7.1(b) shows the same information for the UCF11 [34] and Maryland [35] datasets.

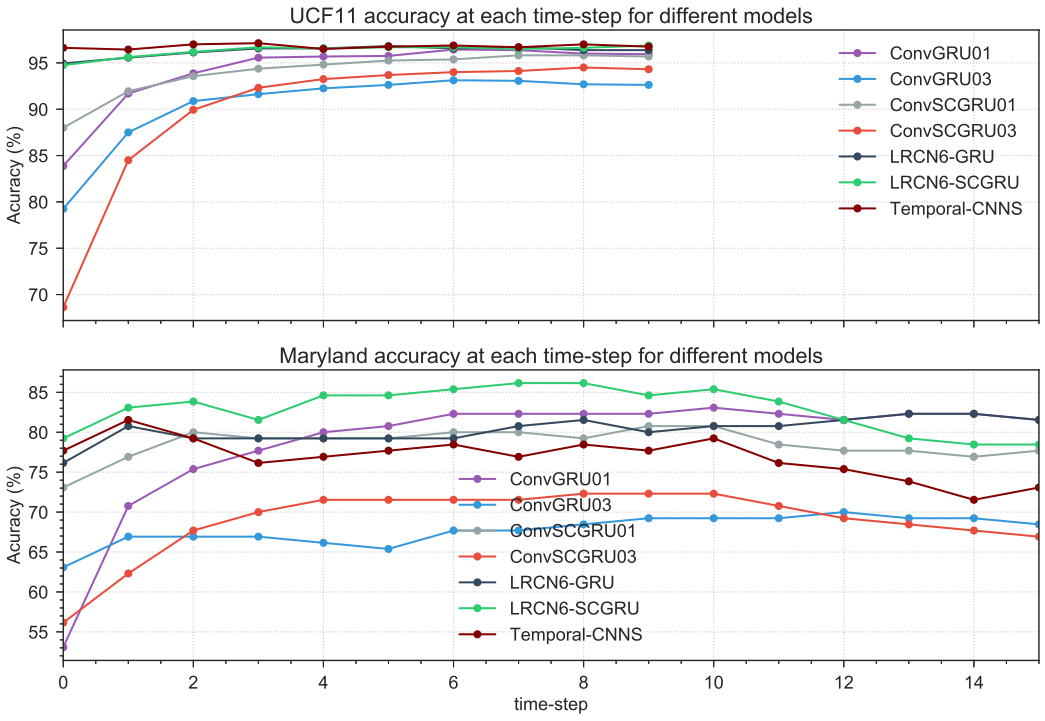
The average accuracy of LRCN based models steadily increases for the UCF101 [32] and HMDB51 [33] datasets. The LRCN6-GRU model accuracy increases slowly compared to the LRCN6-SCGRU model. The increase in average accuracy as the number of time steps increases indicates that LRCN based models do learn salient motion information. For the UCF11 [34] dataset, the average accuracies of the LRCN models stay constant after the third time-step, which shows that these models perform object recognition on the UCF11 [34] activity recognition task. The average accuracy at each time-step for the LRCN based models is erratic for the Maryland [35] dataset, which suggests that the models may experience a reset after some time-steps have occurred.

The accuracy of the Temporal-CNNS model remains relatively constant with each time-step for all datasets except the Maryland [35] dataset. The Conv-RNN based models show the best accuracy increase from the first to the last time-step on the UCF101 [32], HMDB51 [33] and UCF11 [34] datasets. On the Maryland [35] dataset, the Conv-RNN average accuracy increases and then plateaus, with a decrease in the accuracy after the eleventh time-step. Notably, the accuracy of the best performing model (LRCN6-SCGRU) also decreases after the eleventh time-step on the Maryland [35] dataset, which suggests a hidden state reset.

The average accuracy at zeroth time-step coincides with the best performing models (on all the datasets), except for the ConvGRU01 model on the Maryland [35] dataset. The ConvGRU01 model has the worst accuracy (53.077%) at time-step zero ($t = 0$), but achieves the second best accuracy (82.308%) on the Maryland [35] dataset (see Figure 7.1(b)). The Conv-RNN based models have the best performance gain from $t = 0$ to $t = T$, which suggests that these architectures are better at learning salient motion information than LRCN and Temporal-CNN architectures.



(a) Model accuracy at each time-step for the UCF101 (top) and HMDB51 (bottom) datasets.



(b) Model accuracy at each time-step on UCF11 (top) and Maryland (bottom) datasets.

Figure 7.1. Model accuracy at each time-step for different models on the UCF101 (a), HMDB51 (a), UCF11 (b) and Maryland (b) datasets.

7.3 THE BEST ARCHITECTURE FOR ACTIVITY RECOGNITION

This section aims to answer the research question, *using similar deep learning architectures, what is the best learning architecture for activity recognition?* (see Section 1.3). This research evaluated three architectures (seven models) on four different activity recognition datasets, to answer the preceding question.

Of the investigated architectures, the LRCN is the best architecture for activity recognition, followed by the Temporal-CNN and Conv-RNN architectures (see Chapter 6). LRCN based models achieved the best accuracy and mAP scores on the UCF101 [32], HMDB51 [33] and Maryland [35] datasets. On the UCF11 [34] dataset, the Temporal-CNN based model achieved the best performance, followed by LRCN based models (see Table 6.5).

Of the investigated models, the LRCN6-SCGRU is the best model for activity recognition. It achieved the best accuracy and mAP scores on the UCF101 [32], HMDB51 [33] and Maryland [35] datasets. The LRCN6-SCGRU model's 68.226% accuracy is 2.876% below Donahue *et al.*'s [6] 71.1% accuracy achieved on split one of the UCF101 [32] dataset. This research uses $T = 10$ frame samples compared to Donahue *et al.*'s [6] $T = 16$ frame samples to train and evaluate the models. The Temporal-CNNS model achieves an accuracy of 67.275% compared to Donahue *et al.*'s [6] single-frame model accuracy of 69.00% on split one of the UCF101 [32] dataset. Ballas *et al.* [8] achieved an accuracy of 79.9% using RGB inputs on split one of the UCF101 [32] dataset, which is the state-of-the-art result for DNN models trained using RGB inputs of the UCF101 [32] dataset.

Simonyan *et al.* [5] achieved an average accuracy of 40.5% over the three splits of the HMDB51 [33] dataset using a single-frame model trained on RGB inputs. Sharma *et al.* [115] achieved accuracies of 33.46% and 40.98% on the HMDB51 [33] dataset using softmax regression and LSTM attention models fine-tuned on GoogLeNet [38] features. The Temporal-CNNS and LRCN6-SCGRU models achieved accuracies of 30.512% and 38.583% on split one of the HMDB51 [33] dataset. The CNN-S [67] based Temporal-CNN model's average accuracy is 2.948% below Sharma *et al.*'s [115] GoogleLeNet [38] single-frame model. Simonyan *et al.* [5] used multi-task learning, i.e. the models were trained on training data from all splits of the HMDB51 [33] dataset, which explains the difference between their results and those of the Temporal-CNNS model and Sharma *et al.*'s [115] softmax regression model. Sharma *et al.* [115] sampled each video clip at 30 fps and split it into multiple blocks of 30 frames.

The models in this research achieved results on the HMDB51 [33] in line with DNN models [6, 115] that use RGB frames as inputs.

Sharma *et al.* [115] evaluated their LSTM attention models on the UCF11 [34] dataset and achieved the best accuracy of 84.96%. Sharma *et al.* [115] split the UCF11 [34] dataset into 975 training and 625 testing videos, which makes a direct comparison with results in this research difficult. Ravanbakhsh *et al.* [116] achieved the best accuracy of 89.5% on the UCF11 [34]. Ravanbakhsh *et al.* [116] used 25-fold cross-validation and selected the video frames containing the activity using snippet selection (key-frame coding). This research achieved the state-of-the-art accuracy of 97.438% (Temporal-CNNS model) using 10-fold cross-validation.

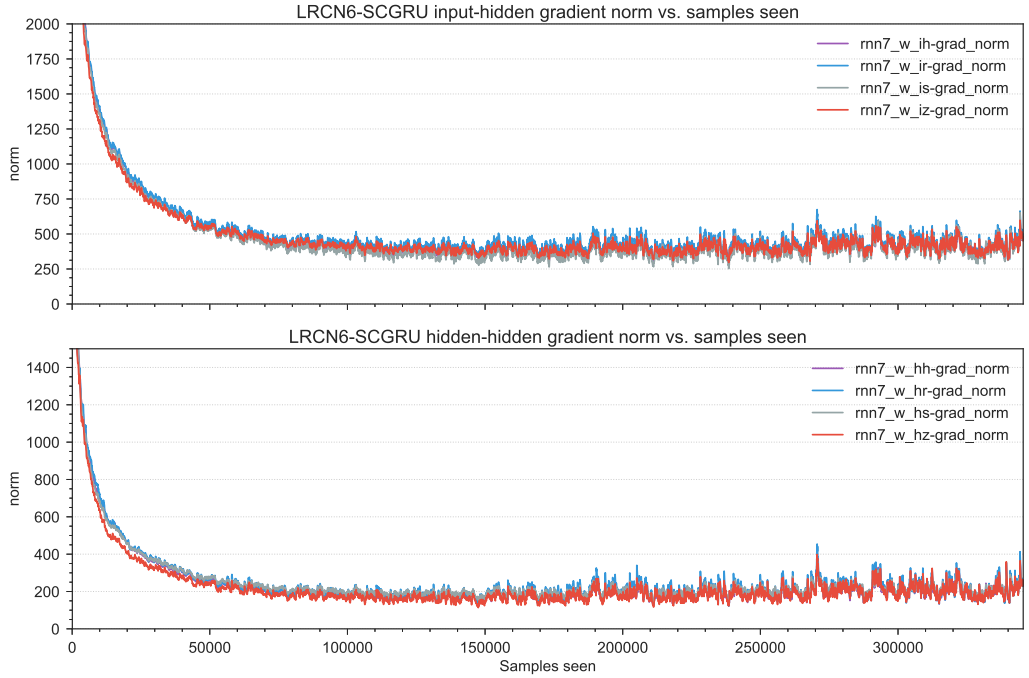
Feichtenhofer *et al.* [114] and Jia *et al.* [113] achieved accuracies of 77.69% and 87.7% using a leave-one-video out cross-validation strategy on the Maryland [35] dataset. The LRCN6-SCGRU model achieved an accuracy of 85.385% using a 26-fold cross-validation strategy, i.e. five video samples, instead of one sample, are used for validation. The research achieved results comparable to the state-of-the-art on the Maryland [35] dataset.

7.4 MODEL TRAINING ANALYSIS

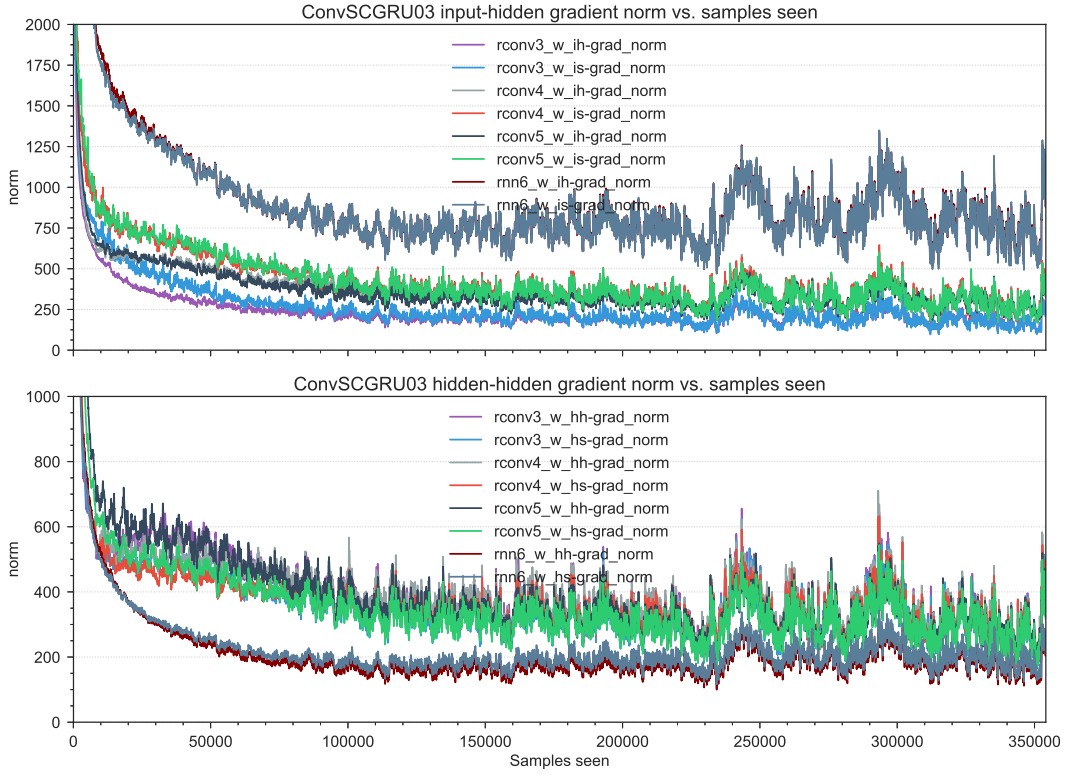
The results of Chapter 6 revealed that the investigated architectures are prone to overfitting. An examination of the gradient norm and learning curves on the best performing model (LRCN6-SCGRU) and worst performing model (ConvSCGRU03) was carried out, to investigate possible causes for this overfitting. This section examines the training these models on the UCF101 [32] dataset.

Figure 7.2 shows the input-to-hidden and hidden-to-hidden per sample seen gradient norms of the LRCN6-SCGRU (see Figure 7.2(a)) and ConvSCGRU01 (see Figure 7.2(b)) models during training. The gradient norms of both models never decrease to insignificant values, which indicates that the models do not experience the vanishing gradient problem (see Sections 3.3.5 & 3.4.5).

The ConvSCGRU03 model gradients are noisy (see Figure 7.2(b)), which suggest that optimisation gets stuck and fails to compute a local direction to minimise the objective loss. The noisy gradients might also be due to the batch size being too small to estimate the actual gradient. LRCN, Temporal-CNN



(a) LRCN6-SCGRU {input,hidden}-to-hidden gradient norm during training.



(b) ConvSCGRU03 {input,hidden}-to-hidden gradient norm during training.

Figure 7.2. Gradient norm per sample seen for the LRCN6-SCGRU and ConvSCGRU03 models, on the UCF101 dataset.

and Conv-RNN01 models use a batch size of 64 for the UCF101 [32] and HMDB51 [33] datasets. Conv-RNN03 models use a batch size of 48 due to GPU memory constraints. A batch size of 32 was used to train all models on the UCF11 [34] and Maryland [35] datasets.

The RNN layer gradient norms are stable compared to Conv-RNN layer gradient norms. The relative stability of the gradients between these two layers might be due to using Cooijmans *et al.*'s [121] recurrent batch normalisation on the RNN layer.

Figure 7.3 shows the learning curves for the LRCN6-SCGRU and ConvSCGRU03 models on the UCF101 dataset. The training log-loss continually decreases, which suggests that the objective function (2.2) does not have local minima or the models are parametrised to a sub-region of the loss function where local minima do not exist.

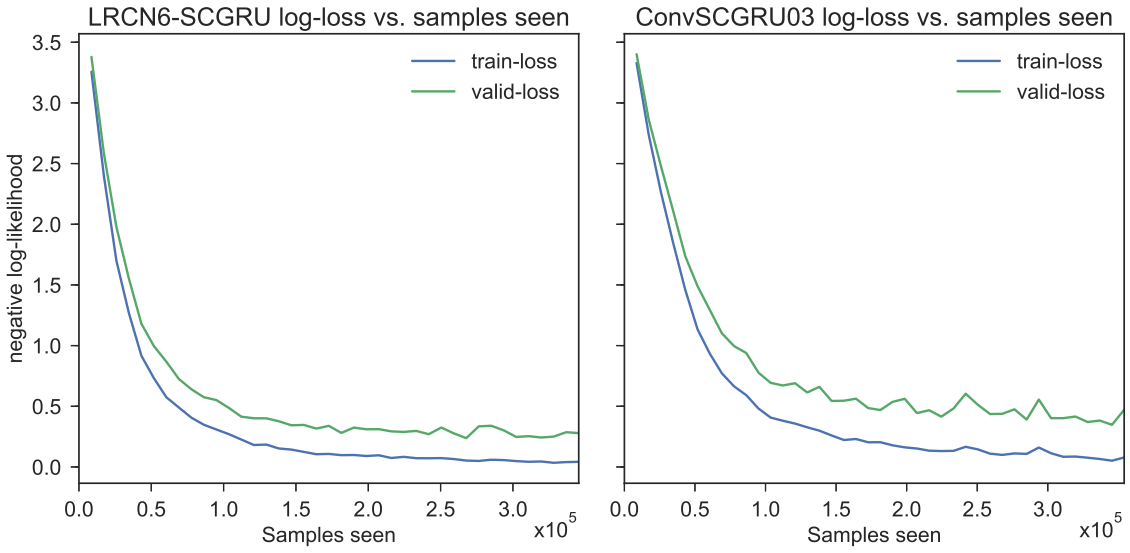


Figure 7.3. LRCN6-SCGRU and ConvSCGRU03 model log-loss on the validation and training sets of the UCF101 dataset during training. The negative log-likelihood is computed at the end of every epoch, i.e. after 8 637 samples, on the entire training (8 637) and validation (900) sets.

Analysis of Figure 7.3 indicates that model optimisation fails to compute good local updates after seeing a certain number of updates, i.e. training gets stuck in a sub-region of the loss function, which led to model overfitting. The layers of the GRU-RCN [8] model are constrained to learn features that can be used to classify the UCF101 categories. This constraint might have allowed the GRU-RCN [8] model to avoid getting stuck in a sub-region of the loss function as seen in the ConvSCGRU03 model.

7.5 CHAPTER SUMMARY

This chapter answered the objective questions raised in Section 1.3. *How important is the temporal dimension to deep learning architectures for activity recognition?* The temporal dimension is important to activity recognition, the best performing architecture (LRCN) demonstrated temporal learning on all data sets except the UCF11 [34], where the starting average accuracies are very high (see Figure 7.1). The average accuracy of the temporally independent architecture (Temporal-CNN) remains constant for all datasets except the Maryland [35].

Using similar deep learning architectures, what is the best learning architecture for activity recognition? Of the investigated architectures, the LRCN is the best architecture for activity recognition (see Section 7.3). LRCN based models achieved the best peformances on the UCF101 [32], HMDB51 [33] and Maryland [35] datasets. The SCGRU based models consistantly achived the best performances across all datasets, except the UCF11 [34] dataset.

How do different deep learning architectures for activity recognition compare in terms of optimisation? Section 7.4 compares the best performing model (LRCN6-SCGRU) and worst performing model (ConvSCGRU03) on the UCF101 [32] dataset, by examining the gradient norm and learning curves of these models. The gradient norms of the ConvSCGRU03 model are noisy, which suggest that optimisation gets stuck and fails to compute a local direction to minimise the cost. The learning curves (see Figure 7.3) indicates that both models are parametrised to a sub-region of the loss function where local minima do not exist.

CHAPTER 8 CONCLUSION

8.1 SUMMARY OF THE WORK

This research proposed and developed two architectures for activity recognition in the form of Temporal-CNN and Conv-RNN (see Sections 4.5.1 and 4.5.2). The research also introduced a new recurrent network in the form of SCGRU (see Section 4.2). To achieve this, literature studies on activity recognition (see Chapter 2) and deep learning (see Chapter 3) were conducted. The proposed architectures, layers and evaluation framework (see Chapter 5) were designed and implemented (see Section 4.6). The activity learning architectures were evaluated on the UCF101 [32], HMDB51 [33], UCF11 [34] and Maryland [35] activity recognition datasets.

Conv-RNN architectures combine CNN and Conv-RNN layers to perform activity recognition. Conv-RNN (see Section 4.3.1) layers replace the multiplication operation of RNNs with convolution. Using convolution in a RNN allows it to learn spatio-temporal features, by preserving the spatial and temporal topology of the inputs. Temporal-CNN architectures transform the single-frame architecture to a temporal one by applying a CNN model at each time-step.

The proposed SCGRU adds context memory (s_t) to the GRU. The context memory caches the inputs at each time-step to learn topic information [105]. The context memory (s_t) of the SCGRU exponentially embeds the current input in the context of previous inputs and stabilises the GRU's candidate activation $\left(\tilde{h}_t\right)$. The reset gate (r_t) of a SCGRU determines the importance of the previous state (h_{t-1}) and the context memory (s_t) when computing the new memory $\left(\tilde{h}_t\right)$.

Different architecture models were trained and evaluated (on four different datasets) using equally sampled frames from each video clip (see Section 5.3.1), using data augmentation, and using pre-processing steps described in Sections 5.3.3 and 5.5. Model performance was evaluated using accuracy, mAP and AUC metrics. The models were further evaluated on their categorical classification performance (see Chapter 6), temporal learning capability (see Section 7.2) and their optimisation (see Section 7.4). The computational complexity of the learning layers used by each activity recognition architecture was analysed (see Section 4.4).

8.2 CRITICAL EVALUATION OF THE WORK

The results achieved in this research are encouraging. The performance of LRCN based models on the UCF101 [32], HMDB51 [33] and Maryland [35] datasets are comparable to state-of-the-art results (see Section 7.3). The difference in performance between the models investigated in this research and those in literature might be due to data sampling, RNN layer differences and optimisation hyperparameter setting. Temporal-CNN, LRCN, and Conv-RNN models achieved state-of-the-art results on the UCF11 [34] dataset. However, the results were obtained using 10-fold cross-validation instead of the usual 25-fold cross-validation.

Conv-RNN based models experienced high levels of overfitting. The Conv-RNN model performance decreased as the number of Conv-RNN layers increased. Examining the gradient norm and learning curves for the ConvSCGRU03 model trained on the UCF101 [32] dataset revealed that these models get stuck during optimisation (see Section 7.4). Temporal-CNN based models performed in-line with single-stream models and achieved the best results on the UCF11 [34] dataset. However, the model struggled on the HMDB51 [33] dataset compared to Simonyan *et al.*'s [5] single-frame model. The single-frame model in [5] was trained by combining the training sets of the three splits of the HMDB51 [33] dataset, which might explain the performance difference (see Section 7.3).

LRCN based models demonstrated temporal learning on the UCF101 [32], HMDB51 [33] and Maryland [35] datasets. On the UCF11 [34] dataset, LRCN models behaved as object recognition models. This suggests that the UCF11 [34] dataset is not complex enough to be used as a benchmark for activity recognition. Conv-RNN models demonstrated temporal learning on all datasets (see Figure 7.1).

The proposed SCGRU based models consistently outperformed GRU based models for LRCN and Conv-RNN architectures, except in a few exceptional cases – the Conv-RNN03 models on the UCF101 [32] and Conv-RNN01 on the HMDB51 [33] and UCF11 [34]. These results are encouraging and suggest that *adding context memory to the GRU stabilises the candidate activation* $\left(\tilde{h}_t\right)$. Further experimentation on longer sequences and language tasks is required to confirm the preceding statement. SCGRU based models consistently demonstrated the better temporal gain in accuracy when compared to equivalent GRU based models (see Figure 7.1).

The hypothesis raised at the beginning of this research was, *"Deep learning architectures, which learn spatial and temporal features from video data, outperform architectures that assume temporal independence between successive frames on the activity recognition task"*.

The preceding hypothesis statement is both true and false. True, in that LRCN (temporally dependent) models outperform Temporal-CNN (temporally independent) models. False, in that Conv-RNN models fail to outperform Temporal-CNN models, on all datasets used in this research. Conv-RNN models overfit the data more than Temporal-CNN models. The most probable reason why Conv-RNN models fail to outperform Temporal-CNN models is that this research failed to optimise these models properly (see Section 7.4). Ballas *et al.*'s [8] results support the preceding statement.

8.3 FUTURE WORK

One area for future research is to evaluate SCGRU on tasks with a longer sequential lengths such as language, speech and music. Evaluating the SCGRU on these tasks should reveal how well this layer captures long-term dependencies when compared to GRU and LSTM.

An area for future research is to investigate possible ways to automatically annotate starting and end points of an activity in a video clip. Doing this would allow one to train models for activity recognition using the full activity, which would enable one to properly evaluate a learning model's ability to learn salient motion information.

Another area for future research is to find better optimisation methods that can escape flat areas of a loss function.

REFERENCES

[1] D. Weinland, R. Ronfard, and E. Boyer, "A survey of vision-based methods for action representation, segmentation and recognition," *Computer Vision and Image Understanding*, vol. 115, no. 2, pp. 224–241, 2011.

[2] R. Poppe, "A survey on vision-based human action recognition," *Image and Vision Computing*, vol. 28, no. 6, pp. 976–990, 2010.

[3] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, (Anchorage, AK), pp. 1–8, IEEE, 2008.

[4] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid, "Evaluation of local spatio-temporal features for action recognition," in *Proceedings of the British Machine Vision Conference 2009*, (London, UK), pp. 124.1–124.11, British Machine Vision Association, 2009.

[5] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems (NIPS)* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), vol. 27, pp. 568–576, Curran Associates, Inc., 2014.

[6] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Boston, MA), pp. 2625–2634, IEEE, 2015.

[7] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao, "Towards good practices for very deep two-stream ConvNets," *arXiv preprint arXiv:1507.02159*, pp. 1–5, 2015.

[8] N. Ballas, L. Yao, C. Pal, and A. Courville, "Delving deeper into convolutional networks for learning video representations," in *International Conference on Learning Representations, ICLR 2016*, (San Juan, Puerto Rico), pp. 1–11, 2016.

[9] A. H. Shabani, D. A. Clausi, and J. S. Zelek, "Evaluation of local spatio-temporal salient feature detectors for Human action recognition," in *2012 Ninth Conference on Computer and Robot Vision*, (Toronto, Ontario), pp. 468–475, IEEE, 2012.

[10] S. Sadanand and J. J. Corso, "Action bank: A high-level representation of activity in video," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, (Providence, RI), pp. 1234–1241, IEEE, 2012.

[11] I. Laptev, B. Caputo, C. Schüldt, and T. Lindeberg, "Local velocity-adapted motion events for spatio-temporal recognition," *Computer Vision and Image Understanding*, vol. 108, no. 3, pp. 207–229, 2007.

[12] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[13] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proceedings of the 19th International Conference on Neural Information Processing Systems* (P. B. Scholkopf, J. C. Platt, and T. Hoffman, eds.), pp. 153–160, MIT Press, 2006.

[14] M. A. Ranzato, Y. LeCun, C. Poultney, S. Chopra, and A. Ranzato, "Efficient learning of sparse representations with an energy-based model," in *Proceedings of the 19th International Conference on Neural Information Processing Systems* (P. B. Scholkopf, J. C. Platt, and T. Hoffman, eds.), (Vancouver, B.C, Canada), pp. 1137–1144, MIT Press, 2006.

[15] A. Krizhevsky, I. Sulskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information and Processing Systems (NIPS)* (F. Pereira,

C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, (Nevada, LA), pp. 1097–1105, Curran Associates, Inc., 2012.

[16] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *International Conference on Learning Representations, ICLR 2014*, (Banff, Canada), pp. 1–8, 2014.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of The 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37, (Lile, France), pp. 448–456, JMLR, 2015.

[18] 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)*, (Las Vegas, NV, USA), pp. 770–778, IEEE, 2016.

[19] J. Martens, "Generating text with recurrent neural networks," *Neural Networks*, vol. 131, no. 1, pp. 1017–1024, 2011.

[20] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, pp. 1–43, 2013.

[21] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Stroudsburg, PA, USA), pp. 1724–1734, Association for Computational Linguistics, 2014.

[22] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Deep Learning and Representation Learning Workshop*, (Montreal, Canada), pp. 1–9, 2014.

[23] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, pp. 1–13, 2015.

[24] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, (Lile, France), pp. 2048–2057, JMLR, 2015.

[25] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Boston, MA), pp. 3128–3137, IEEE, 2015.

[26] B. Chen, B. M. Marlin, and J.-a. Ting, "Deep learning of invariant spatio-temporal features from video," in *NIPS 2010 Deep Learning and Unsupervised Feature Learning Workshop*, (Whistler, BC, Canada), pp. 1–9, 2010.

[27] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng, "Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis," in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Providence, RI), pp. 3361–3368, IEEE, 2011.

[28] 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*, (Columbus, OH, USA), pp. 1725–1732, IEEE, 2014.

[29] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[30] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler, "Convolutional learning of spatio-temporal features," in *Proceedings of the 11th European Conference on Computer Vision: Part VI*, pp. 140–153, Springer Berlin Heidelberg, 2010.

[31] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *2009 IEEE 12th International Conference on Computer Vision*, (Kyoto, Japan), pp. 2146–2153, IEEE, 2009.

[32] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: a dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, pp. 1–7, 2012.

[33] 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*, (Barcelona, Spain), pp. 2556–2563, IEEE, 2011.

[34] Jingen Liu, Jiebo Luo, and M. Shah, "Recognizing realistic actions from videos "in the wild"," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, (Miami, FL), pp. 1996–2003, IEEE, 2009.

[35] N. Shroff, P. Turaga, and R. Chellappa, "Moving vistas: Exploiting motion for describing scenes," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (San Francisco, CA), pp. 1911–1918, IEEE, 2010.

[36] G. Willems, T. Tuytelaars, and L. Van Gool, "An Efficient Dense and Scale-Invariant Spatio-Temporal Interest Point Detector," in *Computer Vision – ECCV 2008: 10th European Conference on Computer Vision, Part II*, (Marseille, France), pp. 650–663, Springer Berlin Heidelberg, 2008.

[37] S. Vishwakarma and A. Agrawal, "A survey on activity recognition and behavior understanding in video surveillance," *Visual Computer*, vol. 29, no. 10, pp. 983–1009, 2013.

[38] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Boston, MA), pp. 1–9, IEEE, 2015.

[39] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Action classification in soccer videos with long short-term memory recurrent neural networks," in *Artificial Neural Networks – ICANN 2010: 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part II* (K. Diamantaras, W. Duch, and L. S. Iliadis, eds.), pp. 154–159, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[40] L. Ballan, M. Bertini, A. D. Bimbo, and G. Serra, "Action categorization in soccer videos using string kernels," in *2009 Seventh International Workshop on Content-Based Multimedia Indexing*, pp. 13–18, IEEE, 2009.

[41] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *Proc. Int. Conf. Human Behavior Understanding (HBU)* (A. A. Salah and B. Lepri, eds.), vol. 7065 of *Lecture Notes in Computer Science*, pp. 29–39, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[42] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, (Miami, FL, USA), pp. 248–255, IEEE, 2009.

[43] G. Farnebäck, "Two-Frame Motion Estimation Based on Polynomial Expansion," in *Lecture Notes in Computer Science* (J. Bigun and T. Gustavsson, eds.), pp. 363–370, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[44] Y. LeCun and Y. Bengio, "Convolution networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks* (M. A. Arbib, ed.), (Cambridge, MA, USA), pp. 255–258, MIT Press, 1998.

[45] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, (Paris, France), pp. 253–256, IEEE, 2010.

[46] A. Krizhevsky, "Learning multiple layers of features from tiny images," technical report, University of Toronto, Toronto, Canada, 2009.

[47] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision - ECCV 2014 - 13th European Conference*, vol. 8689, (Zurich, Switzerland), pp. 818–833, Springer, 2014.

[48] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1 ed., 2015.

[49] K. P. Murphy, *Machine learning: A probabilistic perspective*. Cambridge, Massachusetts: The MIT Press, 1 ed., 2012.

[50] Z. Ivezic, A. Connolly, J. VanderPlas, and A. Gray, *Statistics, data mining and machine learning in astronomy*. Prinston, NJ: Princeton University Press, 1 ed., 2014.

[51] D. Brain and G. I. Webb, "The need for low bias algorithms in classification learning from large data sets," in *Principles of Data Mining and Knowledge Discovery*, vol. 2431, pp. 62–73, Springer Berlin Heidelberg, 2002.

[52] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning - ICML '07*, vol. 24, (New York, NY, USA), pp. 473–480, ACM Press, 2007.

[53] P. Baldi and P. Sadowski, "The dropout learning algorithm," *Artificial Intelligence*, vol. 210, no. 1, pp. 78–122, 2014.

[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations, ICLR 2015*, (San Diego, CA), pp. 1–14, 2014.

[55] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, pp. 1929–1958, 2014.

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

[57] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[58] Y. Bengio and Y. Lecun, "Scaling Learning Algorithms towards AI," *Large Scale Kernel Machines*, no. 1, pp. 321–360, 2007.

[59] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[60] Y. C. Ho and D. L. Pepyne, "Simple explanation of the no-free-lunch theorem and its implications," *Journal of Optimization Theory and Applications*, vol. 115, no. 3, pp. 549–570, 2002.

[61] D. George, *How the brain might work: A hierarchical and temporal model for learning and recognition*. {Ph.D. dissertation}, Stanford University, 2008.

[62] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: Transfer learning from unlabeled data," in *Proceedings of the 24th International Conference on Machine Learning (2007)* (Z. Ghahramani, ed.), (Corvallis, OR, USA), pp. 759–766, ACM, 2007.

[63] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," in *JMLR: Workshop and Conference Proceedings 7* (G. Dror, M. Boullé, I. Guyon, V. Lemaire, and D. Vogel, eds.), vol. 7, (Paris, France), pp. 1–20, JMLR, 2011.

[64] G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent, A. Courville, and J. Bergstra, "Unsupervised and transfer learning challenge: A deep learning approach," in *Proceedings of Machine Learning Research* (I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, eds.), vol. 27, (Bellevue, Washington, USA), pp. 97–110, PMLR, 2012.

[65] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3-4, pp. 197–387, 2013.

[66] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. New York, NY, USA: Wiley-Interscience, 2 ed., 2004.

[67] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *Proceedings of the British Machine Vision Conference 2014* (M. Valstar, A. French, and T. Pridmore, eds.), pp. 6.1–6.12, British Machine Vision Association, 2014.

[68] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, vol. 27, pp. 3104–3112, Curran Associates, Inc., 2014.

[69] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, pp. 1–35, 2015.

[70] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–11, 2016.

[71] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated feedback recurrent neural networks," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, vol. 37, (Lile, France), pp. 2067–2075, JMLR, 2015.

[72] S. Hochreiter, S. Hochreiter, J. Schmidhuber, and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–80, 1997.

[73] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[74] F. a. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, no. 1, pp. 115–143, 2002.

[75] M. Mohammadi, R. Mundra, and R. Socher, "CS224D: Deep learning for NLP - Lecture notes part IV," 2015. [Online]. Available: `http://cs224d.stanford.edu/lecture_notes/notes4.pdf`. Accessed: 02 August 2017.

[76] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of The 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37, (Lile, France), pp. 2342–2350, JMLR, 2015.

[77] A. Graves, *Supervised sequence labelling with recurrent neural networks*, vol. 385 of *Studies in Computational Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[78] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[79] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)* (Y. W. Teh and M. Titterington, eds.), vol. 9, (Sardinia, Italy), pp. 249–256, JMLR, 2010.

[80] F. Gers, *Long short-term memory in recurrent neural networks*. {Ph.D. dissertation}, Universität Hannover, 2001.

[81] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning*, no. 3, (Haifa, Israel), pp. 807–814, JMLR, 2010.

[82] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *International Conference on Machine Learning (ICML) Workshop on Deep Learning for Audio, Speech, and Language Processing*, (Atlanta, Georgia), 2013.

[83] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, (Atlanta, Georgia, USA), pp. 1319–1327, JMLR, 2013.

[84] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *Proceedings of International Conference on Learning Representations, ICLR 2016*, (San Juan, Puerto Rico), pp. 1–13, 2016.

[85] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of The 30th International Conference on Machine Learning*, vol. 28, (Atlanta, Georgia), pp. 1310–1318, JMLR, 2013.

[86] Q. V. Le, N. Jaitly, and G. E. Hinton, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units," *arXiv preprint arXiv:1504.00941*, pp. 1–9, 2015.

[87] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Readings in Cognitive Science*, pp. 399–421, Elsevier, 1988.

[88] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[89] Y. Bengio, I. J. Goodfellow, and A. Courville, "Optimization for training deep models," in *Deep Learning*, ch. 8, pp. 238–290, Unpublished, 2015.

[90] C. Sammut and G. I. Webb, *Encyclopaedia of machine learning*. Boston, MA: Springer US, 2010.

[91] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, pp. 421–436, Berlin, Heidelberg: Springer Berlin Heidelberg, 2 ed., 2012.

[92] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nımes*, vol. 91, no. 8, 1991.

[93] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," lecture video, Coursera: Neural Networks for Machine Learning, 2012.

[94] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," in *IEEE International Conference on Neural Networks*, pp. 586–591, IEEE, 1993.

[95] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, p. 6, 2012.

[96] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 1–40, 2011.

[97] J. Martens, "Deep learning via Hessian-free optimization," *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, vol. 951, pp. 735–742, 2010.

[98] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *The International Conference on Learning Representations (ICLR)*, (San Diego, CA), pp. 1–15, 2015.

[99] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "DRAW: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, pp. 1–16, 2015.

[100] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), pp. 2933–2941, Curran Associates, Inc., 2014.

[101] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *International Conference on Learning Representations, ICLR 2014*, (Banff, Canada), pp. 1–9, 2014.

[102] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Learning hierarchical category structure in deep neural networks," in *Proceedings of the 35th annual meeting of the Cognitive Science Society* (M. Knauff, ed.), pp. 1271–1276, 2013.

[103] Y. Bengio, I. J. Goodfellow, and A. Courville, "Regularization of deep or distributed models," in *Deep Learning*, ch. 7, pp. 228–274, Unpublished, 2015.

[104] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," in *Twenty-first international conference on Machine learning - ICML '04*, (New York, New York, USA), p. 78, ACM Press, 2004.

[105] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato, "Learning longer memory in recurrent neural networks," *arXiv preprint arXiv:1412.7753*, pp. 1–9, 2014.

[106] J. Bergstra, O. Breuleux, F. F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A CPU and GPU math compiler in Python," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, (Austin, TX, USA), pp. 1–7, 2010.

[107] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[108] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[109] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2012.

[110] S. Van Der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "Scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 2014.

[111] B. van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio, "Blocks and Fuel: Frameworks for deep learning," *arXiv preprint arXiv:1506.00619*, pp. 1–5, 2015.

[112] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia - MM '14*, (New York, NY, USA), pp. 675–678, ACM Press, 2014.

[113] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, U. C. B. Eecs, A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 675–678, 2014.

[114] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Bags of spacetime energies for dynamic scene recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2681–2688, 2014.

[115] S. Sharma, R. Kiros, and R. Salakhutdinov, "Action recognition using visual attention," *arXiv preprint arXiv:1511.04119*, pp. 1–11, 2015.

[116] M. Ravanbakhsh, H. Mousavi, M. Rastegari, V. Murino, and L. S. Davis, "Action recognition with image based CNN features," *arXiv preprint arXiv:1512.03980*, pp. 1–10, 2015.

[117] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," Technical Report 1341, University of Montreal, Quebec, Canada, 2009.

[118] A. Mahendran and A. Vedaldi, "Visualizing deep convolutional neural networks using natural pre-images," *International Journal of Computer Vision*, vol. 120, no. 3, pp. 223–255, 2016.

[119] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding Neural Networks Through Deep Visualization," in *International Conference on Machine Learning - Deep Learning Workshop 2015*, (Lile, France), p. 12, 2015.

[120] A. Nguyen, J. Yosinski, and J. Clune, "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks," in *2016 Workshop*

on Visualization for Deep Learning, International Conference on Machine Learning (ICML), (New York, NY, USA), pp. 1–23, 2016.

[121] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville, "Recurrent batch normalization," *arXiv preprint arXiv:1;603.09025*, pp. 1–10, 2016.