Video Frame Prediction Using Convolutional LSTM Networks

by

Michael D. Feist

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

This thesis evaluated Convoultional LSTM (ConvLSTM) for frame prediction to help better understand motion in neural networks. Three different neural networks were implemented and trained. The three networks included, the original ConvLSTM paper by Shi *et al.* [35], the Spatio-Temporal network by Patraucean *et al.* [32], and the VPN by Kalchbrenner *et al.* [21].

Implementing and training the three networks allowed thorough testing and comparison of the accuracy and performance of the networks. The original ConvLSTM was reasonably easy to re-implement and train, and it performed as expected. The Spatio-Temporal network and VPN were unable to achieve the desired results, but still showed improvements over the ConvLSTM. The VPN outperformed both the ConvLSTM and spatio-temporal networks and performed better than the reported results in Patraucean *et al.* [32]. The VPN also showed that it over-fits the data.

Acknowledgements

I would like to thank my supervisor Dr. Pierre Boulanger and my supervisory committee Dr. Nilanjan Ray and Dr. Kumaradevan Punithakumar.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the last decade, the development of neural networks has revolutionized the field of computer vision. Some of these neural networks have solved numerous object classification [13], [24], [27] problems. Many of them can learn automatically features in the data that would be nearly impossible to recreate with hand-crafted solutions [24], [42]. These capabilities have motivated many researchers in other research fields, such as speech recognition [12], natural language processing [1], and artificial intelligence for games [29] to use a similar approach to solve their problems.

Although most of the work in this field deals with static image identifications and segmentation, more recent results have been suggested to deal with video and motion analysis. Understanding motion is essential for computer vision and AI, as most real-world applications must deal with dynamic environments. For instance, accurate motion predictions can help a robot navigate its environment and avoid collisions.

The main objective of this thesis is to implement and compare three state-of-the-art unsupervised neural network architectures capable of performing frame prediction. The three neural network architectures are Convolutional LSTM Network, Spatio-Temporal Autoencoder, and Video Pixel Network. The networks are trained and tested with two moving handwritten digits from the MNIST [26] database. The data consist of sequences of two moving digits from the MNIST training set. Each sequence of 20 frames is 64 x 64 pixels in size.

Chapter 2 review the current state-of-the-art of convolutional neural network as it applied to temporal data inputs. Chapter 3 describes in detail the three network architectures used in this thesis. All important parameters to these networks, such as the kernel sizes, number of filters, and hyperparameters, will be presented. Although all three networks use a version of the ConvLSTM architectures, each network has its unique structure and approach to frame prediction. By implementing all three networks, some insight into the strengths and weaknesses of each architecture will be gained. Chapter 4 first looks at replicating the results of each implementation to the results found in the original articles by computing the average error of ten predicted frames. As will become evident, averaging the error can hide subtle and important details of the network. We will also analyze the error distributions of each network as well as randomly picked test samples to help answer the following questions:

RQ1. Do the networks maintain an accurate motion for each digit, or do the digits drift from their correct position in the image?

RQ2. Do the networks maintain the spacial structure of the digits, or do the digits become blurry and unrecognizable?

Chapter 4 explore the robustness of each networks. These tests include increasing the number of digits, up to four digits, and using a different data set for the digits. In Chapter 5, we will then conclude and compare each architecture's pros and cons. We will also explore how one can improve their performance.

1.1 Contributions of the thesis

The main thesis contributions are:

- Implement and test three frame prediction Convolutional LSTMs based architectures found in the literature;

- Test and compare the performance of each architecture with a simple image sequence based on MNIST moving handwritten digits dataset;
- Analyze the pros and cons of each architecture and its potential improvement.

Chapter 2

Literature Review

2.1 Introduction to Neural Networks

Although there has been a lot of interest and progress in artificial neural networks (ANN) over the last decade, ANN dates back to the 1940s. Warren S. McCulloch and Walter Pitts [28] were two of the first researchers to propose the idea of simulating the neural activity of the brain. Their work was a long way from our current neural network models, mainly because they were unable to implement their ideas on any computer available at the time. However, their work laid the foundation for today's research. Even though the requirement of large computing power is still an issue, recent parallel computing capabilities such as the Graphic Processing Units (GPUs) and large supercomputers have allowed the research community to train large and complex networks in a reasonable amount of time. For example, the supercomputer developed for the OpenAI company is a single system with more than 285,000 CPU cores, 10,000 GPUs, and 400 gigabits per second of network connectivity for each GPU server [25].

In 2010, the AlexNet architecture [24] had been introduced and was able to outperform the best-handcrafted algorithms for image classification. This pioneering work has led many researchers to move away from the standard handcrafted approach towards more efficient approaches based on neural networks [12], [13], [24], [27], [29], [30]. The AlexNet results [24] also showed that it was possible to train larger networks on low-cost GPUs. Currently, many neural network approaches dominate the research fields, such as com-

puter vision [13], [24], [27], speech recognition [12], and speech synthesizers [30]. Researchers have also used neural networks to create a more general artificial intelligence (AI) by teaching the AI to play games [29].

Modern-day neural networks need two elements to be successful in addition to the network architecture, which needs to be designed to represent the problem accurately. First, as mentioned previously, neural networks require massive computing power to calculate the vast amount of connection weights (more than two million) between the neurons. The connection weights are updated using gradient descent and backpropagation algorithms, which are very costly, especially when the number of neurons increases. Second, to train neural networks, one needs to have a large amount of data to determine the network's parameters. Companies like Facebook and Google specialize in collecting data to be used to train those networks. Besides the private data that companies have gathered over the years, there are also large public data sets created by research institutes such as the MNIST [26] and ImageNet [3]. These data sets allow researchers to test new ideas and compare their results against other networks.

2.2 Convolutional Neural Networks

As applications move away from small, simple images such as the MNIST [26], fully connected neural networks start to struggle. Since a fully connected layer has a connection to each pixel, when the size of the image increases, then the number of parameters in the network must increase accordingly. The most common method to overcome this shortcoming is to use convolutional neural networks (CNN) [26]. Instead of learning the connections to every pixel, like in a fully connected network, a convolutional neural network learns convolution kernels to filter the input image at different scales. These convolution kernels also have the benefit of being differentiable, making it possible to train the network using backpropagation algorithms.

Today's top image classification neural networks are based on this convolutional approach [13], [24], [27]. The convolutional approach allows the

networks to be deeper due to their smaller overhead. They also outperform fully connected layers, even when a fully connected layer is feasible, such as with the MNIST dataset. One of the reasons the convolutional approach works well is that they can deal with the spatial locality of features. In some images, neighboring pixels are more relevant than those on the opposite side of the image. In convolutional layers, the kernel is only seeing a small part of the image at a time. Zeiler and Fergus have tried to understand what a convolutional neural network learns [42]. In their example, the first layers of the network learned simple structures like edges and corners. The following layers then learned the more complex structures like a person's nose or eye.

2.3 Deep Neural Networks

Increasing the number of layers has been shown to improve the accuracy of neural networks [5], [13] recognition. However, there are some issues with increasing neural network depth. The obvious problems are the increase in the computing requirements and the training time because increasing the number of layers also increases the number of parameters. As mentioned by He *et al.* [13] increasing the number of layers in a network can harm its accuracy. In their paper, they propose a new architecture called ResNets that can preserve and even increase the accuracy when the number of layers increases. The authors show that increasing the depth of a ResNet does not affect the accuracy allowing them to create networks with more than 152 layers. Their approach is based on adding "shortcut connections" where previous layers could pass the learned information through the network without going through the deeper layers of the network. These shortcut connections usually skip one or two layers. In the case of the ResNet architecture, "the shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers" [13].

2.4 Image Generation

There has been much research to make neural networks generate data. Some of the early approaches used autoencoder architectures [15]. The main idea is to reduce the dimensions of the data into a smaller vector using a network called "an encoder" and then recreate the higher dimensional data using another network called "a decoder." The major flaw with traditional autoencoder is that the encoded vector, or latent space, can be sparse. Therefore, if the latent space is randomly sampled, then the reconstructed data will likely be non-sensical. In many applications, the decoder produced by the traditional autoencoder is not a reliable generator.

Later works have attempted to create a better encoded latent space by introducing a variational autoencoder (VAE) [23], [33]. By putting extra constraints on the latent space, VAEs force the latent space to be continuous. Therefore, the decoder of a trained VAE can be randomly sampled and generate new meaningful data. Many results found in the literature show the successful generation of new images, such as handwritten numbers that closely resemble those found in the MNIST dataset [33].

An interesting property of VAEs [23], [33] is that the latent space created is continuous and allows interpolation. If one takes two randomly sampled points in the latent space, point A and point B, one can interpolate between the generated images. As the interpolated point moves from point A to B, the generated image slowly transforms between the two samples. For example, using the MNIST dataset, if point A generates the digit "1" and point B generates the digit "7", then moving from point A to point B in the latent space will create images that transition from "1" to "7". While interpolating, the network should not generate any random images that do not closely represent a digit from the MNIST database. This is not the case with traditional autoencoders as their latent space may not be continuous.

The main drawback to VAEs [23], [33] compared to other generative models, is that VAEs are trying to reduce both the reconstruction and the latent variable cost. This tends to produce lower quality results. The images gen-

erated by VAEs are usually missing the fine details and can often be blurry. This is especially the case with more complex images, such as the ones found in ImageNet [3]. On simpler images, like the MNIST numbers, VAEs can actually perform quite well as a generative model.

Another approach to generating data is through the use of Generative Adversarial Networks (GANs) [10]. GANs use two neural networks where one network is a generative model G that generates fake data, and the other is a discriminative model D that learns to detect fake data. During the training process, the generative model attempts to fool the discriminative model into predicting the generated data is from the training data. Simultaneously, the discriminative model attempts to detect data generated from G, versus data that came from the training data. After the training, network G can be used to create new data. Goodfellow *et al.* [10] trained the network to learn to generate images. One of the datasets used was the MNIST [26]. After training, the generative model could create new images close to the original images. Like VAEs, the latent space of GANs is continuous, which means one can randomly sample the latent space, and the generative model will produce meaningful data.

One disadvantage of GANs is that they are more difficult to train. Since the training process is a min-max problem between two networks, one network must not get overtrained before updating the other network. Goodfellow *et al.* [10] describes a situation where the generative model can collapse the latent space, z, of a generative model, G, too quickly. This means the latent space will not be able to represent the dataset accurately.

PixelCNNs [31] take a slightly different approach to image generation. Instead of generating an image from a latent vector, PixelCNNs predict the value of a pixel from the values of the previous pixels. Therefore, one method for generating images is to input an image with random pixel values and have the PixelCNN generate an image from the noise. This approach usually requires the network to be run multiple times before a valid image is produced. Unlike VAEs and GANs this approach is difficult to generate specific images, such as a specific digit from MNIST database.

Van Oord *et al.* [31] had a more useful use case, where the PixelCNN generated a missing part of an image. The authors showed examples of images with the bottom half of the image missing, and the PixelCNN was able to regenerate the missing part. This seems to be closer to the intended use case for PixelCNNs. They can be used to correct errors or missing information from existing images rather than generate new images from a latent vector.

2.5 Optical Flow Neural Networks

Many researchers [6], [17], [41] have expanded the application of neural networks to other areas of computer vision. One such area of research is the computation of optical flow in videos. Optical flow is a way to track the motion of objects, frame by frame, in a video. By tracking features of an object, the velocity of pixels can be calculated because the pixel intensity changes between frames. One optical flow method described in Fleet and Weiss [7] consists of using temporal gradients from successive video frames to compute how pixels move from one frame to the next. Many algorithms are based on intensity conservation assumption, where the intensity of a pixel is assumed to be constant between two frames. These handcrafted methods are easier to implement than neural network approaches but do not produce very accurate results as they are susceptible to occlusions and noise issues.

FlowNet [6] introduces a new neural network architecture that can compute optical flow. Instead of looking at single-pixel evolution between two consecutive frames, the FlowNet uses convolutional layers to calculate the flow in a hierarchy of larger neighborhoods. The output is an image in which each pixel represents the optical flow. FlowNet uses a supervised approach, meaning the network needs both the input and output during training. The algorithm uses computer-generated video frames from computer graphic 3D renderings to ensure an accurate ground truth flow map during training.

FlowNet and the follow-up paper [17] have shown to accurately generate an optical flow map that is much less sensitive than its handcrafted versions. However, one criticism of this approach is that one needs the flow map ground

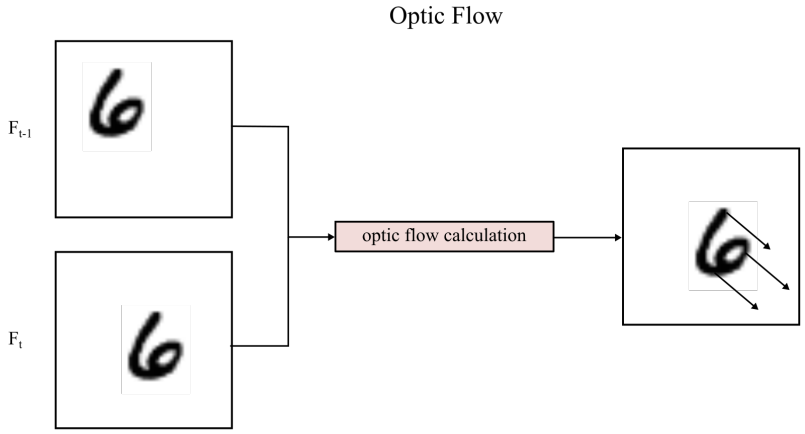

Figure 2.1: An example of optical flow. Given two images an output image is returned with vectors showing the motion between the two images.

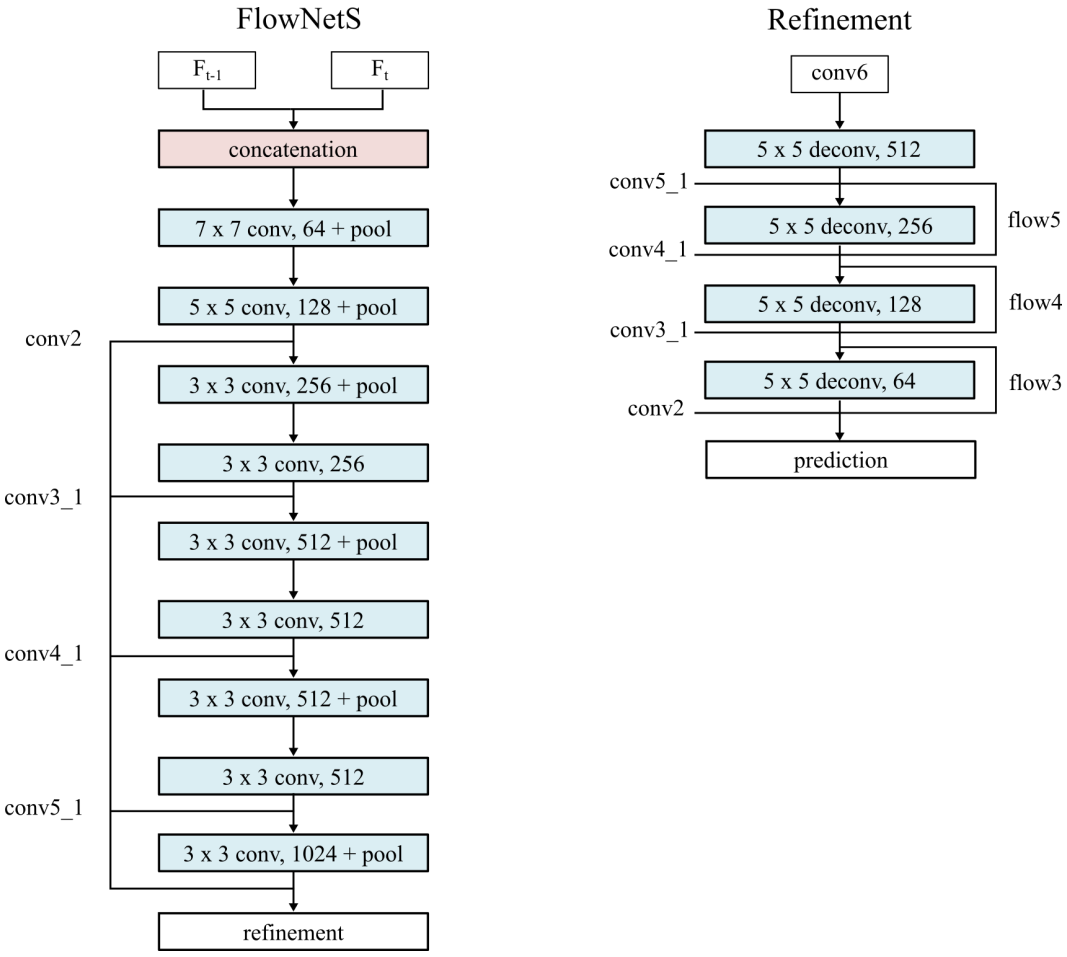

Figure 2.2: The FlowNet simple [6].

truth to train the network. For research, this is not a problem, but it can be challenging to generate an accurate reference optical flow map for real-world data.

Table 2.1: Comparison of Optical-Flow Methods

Approach	Chairs	KITTI			
		Avg. All		Avg. NOC	
	Test	Train	Test	Train	Test
FlowNetS [6]	2.7	7.5	9.1	5.3	5.0
Back to Basics [41]	5.3	11.3	9.9	4.3	4.6

Yu *et al.* [41] present an unsupervised approach for neural networks to generate optical flow, which is referred to as back to basics [41]. The authors do not require the ground truth flow map in their training. The algorithm tries to warp the first frame in the video to match the second frame to learn the optical flow. The warping consists of applying a translation to each pixel in the first frame and then attempt to match those pixels in the second frame. The way the pixels are moving, or warped, correspond to the optical flow that needs to be computed. This unsupervised approach introduces a smoothing function that forces neighboring pixels to move at similar velocities. Without this smoothing function, the network would struggle to learn any useful optical flow mapping. One limitation of this approach is that some edges get blurred. Even with these limitations, the back to basics [41] approach claims to produce similar results as FlowNet. However, when tested with reference test data, FlowNet still outperforms this approach.

2.6 Self-Supervised Learning

Given the data there are different approaches to training a neural network. When the data is labeled then the model can be trained in a supervised approach [13], [24], [27]. This usually provides the best results; however, it can be difficult to gather a large collection of labeled data. Another approach is unsupervised learning [10], [23], [33]. Unsupervised learning does not require the data to be labeled. Instead the model is structured in a way that allows

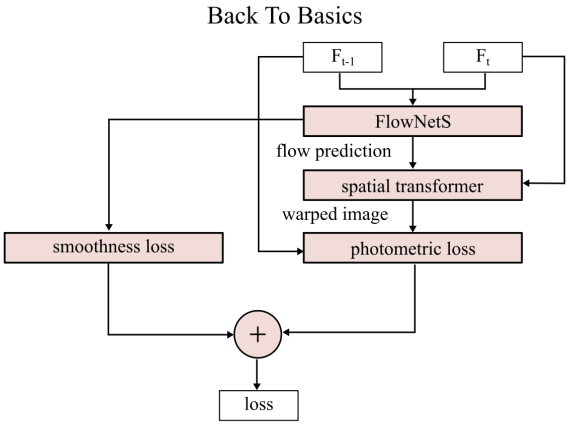

Figure 2.3: The network used in back to basics [41].

it to learn features of the data.

Self-supervised learning is when the data is not labeled, but the problem is structured like a supervised problem. Natural language processing (NLP) uses this approach when training transformers [4]. NLP transformers use a large corpus of text for their data. Unlike some of the image classification problems mentioned previously this data of text is not labeled. However, the data still has a structure to it. For example the order of the sentences and words matters. Therefore classification problems are created from the data. One training problem is to mask out a random word and have the model predict the missing word. Another is to take two sentences and predict if the second sentence should come after the first. Similar to the supervised approach the model is given an input and a label. However, the labels are not hand crafted before training begins and instead are created by an algorithm.

2.7 Frame Prediction

Other researchers have tried doing frame prediction with neural networks [21], [32], [35], [36]. These neural networks use multiple video frames to predict the next frame in a video. Many of these neural networks do not predict just one frame, but instead, they try to predict many frames. There are some algorithms that can also create a video frame from a single frame [39], [40].

Some of the more obvious use cases of these algorithms are to fix missing frames in a video or to add extra frames to simulate slow-motion video [20].

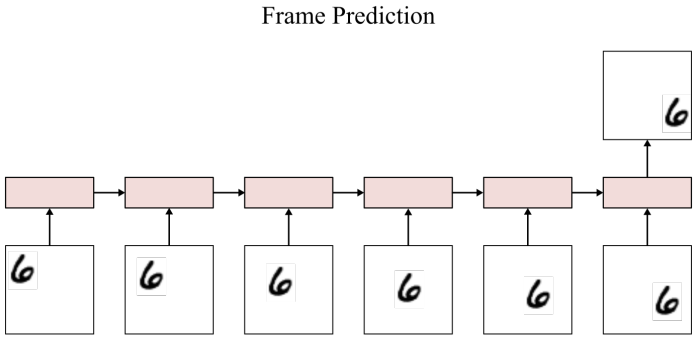

Figure 2.4: An example of frame prediction.

One area that frame prediction algorithms were used that may not be as obvious was in a maze solving AI [18]. A DeepMind paper by Jaderberg *et al.* [18] used an auxiliary task that predicted future rewards to help better navigate a three-dimensional maze. Given the last three frames that the bot saw, this additional task predicted the likelihood of receiving a reward in the next unforeseen frame. In this case, the reward predictor does not generate a new frame like the above frame predictors but instead outputs the probability of a reward.

When working with temporal data, the network can be crafted to handle a fixed number of time steps. This is the approach that FlowNet [6] took and can usually be the simplest. Using a fixed number of frames can cause issues when the data has a variable number of frames. There could also be information in previous frames that the network is ignoring and can help with future predictions. One approach is to use recurrent neural networks.

Recurrent neural networks (RNNs) [34] allow artificial neural networks to handle temporal and variable-length data. This network architecture has been used in many applications, including, speech recognition [12], translation [1], video [36], and artificial intelligence in games [29]. Similar to a hidden layer in a feed-forward neural network, a recurrent layer takes input from the previous layer and outputs to the next layer. Unlike feed-forward networks, RNNs

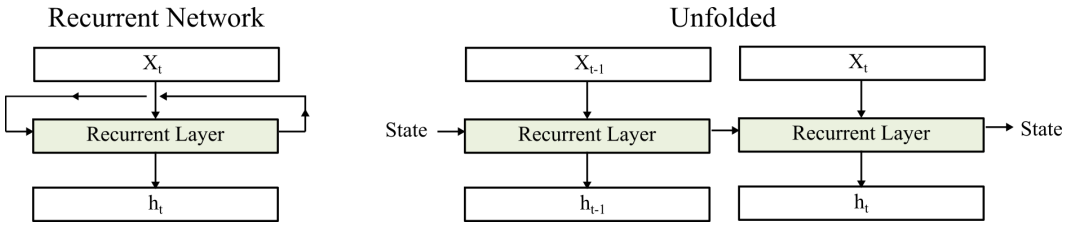

Figure 2.5: A simple recurrent neural network.

have a different internal state that allows the network to remember important information from previous time steps. Unfortunately, it can be difficult to train long RNNs as they can suffer from vanishing or exploding gradients [16]. The vanishing gradients can make the long-term dependencies seem far less important than the short-term dependencies.

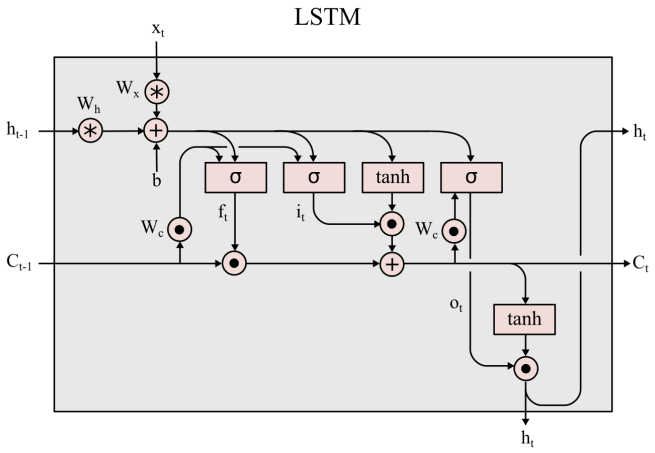

Figure 2.6: A long short-term memory (LSTM) module [16].

Long short-term memory (LSTMs) [16] were introduced to help solve the gradient problems with RNNs. There are variations on LSTMs; however, it is common for an LSTM unit to consist of a memory cell, an input gate, an output gate, and a forget gate. The gates are used to control the flow of information to and from the cell. The drawback to LSTMs is their increased memory usage and training time as the network must now store and learn weights for all the gates. Since the benefits of LSTMs usually outweigh the drawbacks, LSTMs are commonly used for recurrent neural networks.

Srivastava *et al*. [36] introduced the idea that LSTMs can be used for frame

prediction. They created a simple test set of two moving MNIST digits [33] and had decent results predicting the future frames. Later work introduced a method called ConvLSTMs, which combines LSTMs with convolutional networks [35]. In the next chapter, I will review and implement multiple versions of frame prediction networks.

Chapter 3

Predicting Frames Using Three Neural Networks

In this chapter, I will explore three different neural network architectures for frame prediction: Convolutional LSTM Network, Spatio-Temporal Autoencoder, and Video Pixel Network. First a description of the data used for training and testing using the MNIST database will be provided. Then I will describe the structure of the three network architectures and their training process.

3.1 Network Training Data

The networks are trained and tested with two moving handwritten digits database from the MNIST [26]. This database was first developed by Srivastava *et al*. [36]. The data consist of sequences of two moving digits from the MNIST training set. Each sequence of 20 frames is 64 x 64 pixels in size. All three networks are trained on two moving digits.

Using this test set will allow us to compare my research results to the results proposed by the original authors. However, to double-check, a research test set of two moving MNIST digits is also created. The digits are selected from the MNIST test set and moved using the same algorithm as the training data for generating the motion.

In addition, using the original dataset, all three networks are also tested on sequences ranging from one to four digits. The purpose is to test the versatility

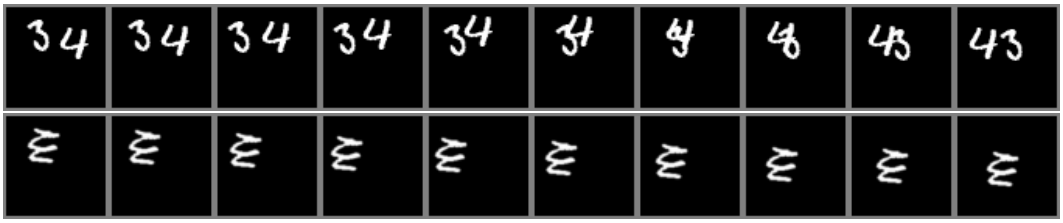

Figure 3.1: **Top:** Sample from the two moving MNIST digits. This is the type of data the models were trained on. **Bottom:** Sample from the one moving EMNIST digit. The models were not trained on this data and is only used for testing.

of the network's motion tracking. One of the question to be answered is, can the networks handle more than two digits? If the error of a network increases drastically after two digits, then the network has most likely learned to only track two objects. A random test sample with four digits will be used to analyze further insight into the behavior of the networks.

The networks are also all tested on sequences where the digits are taken from the EMNIST [2] dataset instead of the MNIST. The EMNIST dataset contains handwritten characters other than the digits zero to nine, like the digits found in MNIST. The other reason for using the EMNIST dataset is that it is formatted similarly to MNIST. The digits in the EMNIST dataset are the same size and have the same pixel values as the digits found in the MNIST dataset. The only difference is the shape of the digits. By using the EMNIST dataset, one can determine whether the networks rely heavily on the spatial structure of the digits.

3.2 Networks Architectures

This section will discuss the architecture of each network. Their performances will be described in Chapter 4.

3.2.1 Convolutional LSTM Network

Srivastava *et al.* [36] were one of the first authors to use long-term short term memory (LSTM) networks [8], [16] to predict future frames in videos. Their work showed promising results but required a large number of parameters

because the networks were fully connected, and therefore, each pixel in the image needed its own connection. This is a fundamental flaw in using traditional LSTM networks with images. Shi *et al.* [35] introduced the use of convolutional LSTM (ConvLSTM) networks to help solve this problem. By combining traditional LSTM architecture with convolution, the resulting network needed fewer parameters while being able to achieve better performance. The ConvLSTM network is expressed mathematically by:

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \\
C_t &= f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o) \\
H_t &= o_t \circ \tanh(C_t)
\end{aligned}
\tag{3.1}
$$

where '$*$' denotes the convolution operator and '$\circ$' denotes the Hadamard product. Like traditional LSTMs, the ConvLSTM still has an input, X_t, input gate, i_t, forget gate, f_t, output gate, o_t, hidden state, H_t, and cell state, C_t. The hidden state and cell state are passed on to the LSTM in the next time step. The hidden state is also passed onto the next layers in the model. All the remaining variables are the weights and biases, W and b, and these variables are learned during training. This equation is also shown graphically in Figure 3.2.

The equation used for the ConvLSTM networks is almost identical to the equation for the standard LSTM networks [9], [11], [36] except that the multiplication is replaced by a convolution. Therefore, instead of vectorising the image and using a traditional LSTM network with matrix-vector multiplication as shown in Srivastava *et al.* [36], the image structure is preserved, and a convolution operation is applied. With a convolutional layer, only the weights for the kernels are learned, and the kernel size is much smaller than the input images. The fully connected LSTM network [36] required $\sim$ 142 million parameters whereas the ConvLSTM network [35] only required $\sim$ 7 million parameters and is capable of outperforming the fully connected network.

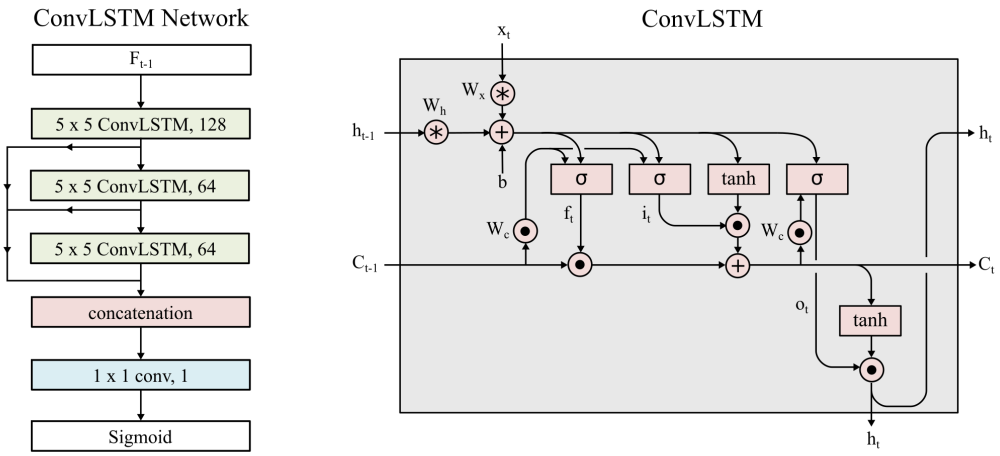

Figure 3.2: **Right:** The structure of the ConvLSTM network. More details of the network are described above. **Left:** The structure of a ConvLSTM layer is shown in equation 3.1.

The ConvLSTM network, I use is the same network described in Shi *et al.* [35], and its architecture can be seen in Figure 3.2. The network consists of three ConvLSTM layers and a final output convolutional layer. All the outputs of the ConvLSTM layers are concatenated together before being passed to the final layer. The ConvLSTM layers use a kernel size of 5 x 5 with 128, 64, and 64 hidden states. The final convolutional layer has a kernel size of 1 x 1 and is followed by a sigmoid activation layer. Due to the concatenation performed on all ConvLSTM layers, the final output layer has an input feature map with 256 channels. To train the network, a cross-entropy loss [1] function is used. This network is by far the simplest architecture of the three networks, making it the easiest to build and train.

3.2.2 Spatio-Temporal Video Auto-encoder with Differentiable Memory

The second network I implement was introduced by Patraucean *et al.* [32]. Its architecture can be seen in Figure 3.3. The core concept of this network is to use a ConvLSTM network [35] with a modified spatial transformer [19] to

[1] The cross-entropy loss of the predicted frame P and the ground-truth frame T is defined as $-\sum_{i,j,k} T_{i,j,k} \log P_{i,j,k} + (1 - T_{i,j,k}) \log(1 - P_{i,j,k})$

learn the optic flow. Then, the learned optic flow will transform the features of the current frame in the sequence to closely match the next frame in the sequence. The network is similar to a convolutional auto-encoder and adds a learned optic flow module. The optical flow module is referred to in the original paper [32] as a temporal auto-encoder. Due to this structure, the network can be broken down into three distinct structures: the convolutional encoder, the temporal auto-encoder, and the convolutional decoder.

Spatio-Temporal Network Architecture

The first layer of the encoder is composed of a convolutional layer with a 7 x 7 kernel and 16 filters. This layer is then followed by a tanh non-linearity activation function and a 2 x 2 max-pooling layer. The output of the convolutional encoder is then passed to both the temporal auto-encoder and the modified spatial transformer.

The temporal auto-encoder is a type of network capable of learning the motion/optical flow. It starts with a 7 x 7 ConvLSTM layer with 64 hidden states. The output of the ConvLSTM layer is then passed to three convolutional layers. The first two layers use a kernel size of 15 x 15. The last layer, with a kernel size of 1 x 1, outputs the optical flow. None of these layers have an activation layer.

Using ideas taken from Jaderberg *et al.* [19], the output from the convolutional encoder is warped using the learned optical flow. This warping function is a differentiable lookup table. By using optical flow, the pixel at each location of the current frame is moved to its new location on the next frame. This new warped image is then passed to a convolutional decoder where it is first resized, using a nearest-neighbor spatial up-sampling to match the original size of the input. This resized frame is then passed to a convolutional layer with a 7 x 7 kernel and is then followed by a sigmoid activation layer that outputs the predicted frame.

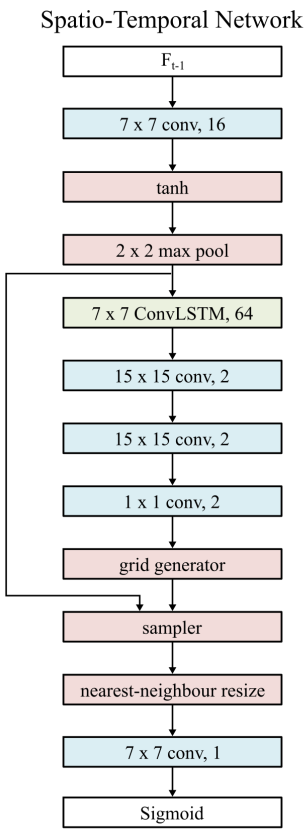

Figure 3.3: The structure of the Spatio-Temporal video auto-encoder network.

Spatial Transformer

The spatial transformer first proposed by Jaderberg *et al.* [19] applies a spatial transformation to the features during the forward pass of a network. This transformation is not handcrafted but instead is learned during the training of the network. The authors did this by creating a sub-network to learn the parameters of a transformation matrix. Note that these parameters can be different on every forward pass as the network is generating them. The transformation matrix is then applied to a grid that is used as a lookup table for a differentiable sampler.

A modified spatial transformer is used for the spatio-temporal network. This is the same transformer introduced in the paper by Patraucean *et al.* [32]. Unlike the original spatial transformer proposed by Jaderberg *et al.* [19], a transformation matrix is not applied to the entire grid, but instead,

the learned optical flow applies a translation for each pixel of the grid. The spatial transformer takes the learned optic flow and the feature map created by the encoder as input. Let $U \in \mathbb{R}^{HxWxC}$ be the feature map, with width W, height H, and C channels. A grid is generated, $G \in \mathbb{R}^{HxWx2}$, to have the same width and height as the feature map. The grid has two channels for the translation in the x and y directions. The original paper by [19] added a third channel filled with ones to the grid. This allows a transformation matrix to be used and the translation to be calculated in a single matrix operation. However, the spatio-temporal network replaces a matrix transformation with a simple addition because it is only working with the translation of each pixel. Therefore, the extra channel filled with ones is not needed. This network is expressed mathematically as:

$$\mathcal{T}_\theta(.,.) = \begin{pmatrix} 1 & 0 & \theta_x \\ 0 & 1 & \theta_y \end{pmatrix} \tag{3.2}$$

$$\mathcal{T}_\theta(G) = G + \theta \tag{3.3}$$

Each point on the grid are normalized to between $-1 \leq x_i, y_i \leq 1$. Therefore a grid for a 5 x 5 feature map would look as following:

$$G = \begin{bmatrix} -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\ -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\ -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\ -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\ -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \end{bmatrix} \begin{bmatrix} -1.0 & -1.0 & -1.0 & -1.0 & -1.0 \\ -0.5 & -0.5 & -0.5 & -0.5 & -0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} \tag{3.4}$$

The learned optic flow, θ, is added to the grid to give the transformation, $\mathcal{T}_\theta(G)$.

This transformed grid, $\mathcal{T}_\theta(G)$, is then submitted to a differentiable bi-linear sampler kernel define by:

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|) \tag{3.5}$$

where (x_i^s, y_i^s) are the source pixels in $\mathcal{T}_\theta(G)$ and V_i^c is the output for pixel i and channel c. Note that the same transformation is applied to each channel so $\mathcal{T}_\theta(G)$ does not have the same number of channels as the feature maps U and V.

Loss Function

The spatio-temporal video auto-encoder is the only network of the three that does not use a cross-entropy loss function to determine if the iteration has converged. Instead, it uses a L2 loss function with a smoothness penalty applied to the optic flow. The L2 loss function is defined as:

$$\|T - P\|_2^2 \tag{3.6}$$

where P is the predicted frame and T the ground-truth frame. The smoothness penalty is calculated by applying a convolution over the flow map and then applying a weighted Huber loss function [32], [38]. The Huber loss function is defined as the following:

$$\mathcal{H}_\delta(a_{i,j}) = \begin{cases} \frac{1}{2}a_{i,j}^2, & \text{for } \mid a_{i,j} \mid \leq \delta \\ \delta(\mid a_{i,j} \mid -\frac{1}{2}\delta), & \text{otherwise} \end{cases}. \tag{3.7}$$

Patraucean *et al.* defined the convolution filter used in the Huber loss function as a "fixed, non-trainable", 2 x 3 x 3 filter, corresponding to a 5-point stencil, and 0 bias [32]. This convolution filter calculates the local gradients of the flow map, $\Delta\mathcal{T}$, and is defined by:

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.8}$$

Using these equations the total loss function is defined as:

$$\mathcal{L}_t = \|T_t - P_t\|_2^2 + w_\mathcal{H}\mathcal{H}(\Delta\mathcal{T}) \tag{3.9}$$

where $w_\mathcal{H}$ is set to 10^{-2} and the delta, δ, in the Huber loss function is set to 10^{-3}.

3.2.3 Video Pixel Network

The third network is the Video Pixel Network (VPN). It was originally developed by Kalchbrenner *et al.* [21] and combined a convolutional LSTM network [35] with a PixelCNN [31]. The VPN comprises three parts: an encoder, a ConvLSTM unit, and a PixelCNN decoder. The encoder and decoder follow

a structure similar to the networks introduced in ResNets [13] and the PixelCNN [31]. The authors of the VPN tried two versions. The first one used convolutional layers with a pre-activation ReLu similar to the PixelCNN [31] and is described in [14]. The second network introduces units called residual multiplicative blocks (RMBs) that incorporate gates, similar to LSTMs, into the convolutional layer. On the MNIST dataset, the RMBs performed about the same as the ReLu networks, but the RMBs outperformed the ReLu networks on the real-world data the authors tested. Since the authors showed that there is no significant difference in performance between the two models on the MNIST data, I decided to use the ReLu model as it is a simpler model and will limit the number of variables that could complicate its implementation.

The authors of the VPN did not go into great detail describing the architecture of their network and its implementation. Many details were missing in the paper, and I had to look to previous works by the authors to fill in the gaps. Therefore, when I go over the implementation of the VPN network, I will be describing my implementation of the VPN described in the paper [21].

Background: PixelCNN

The VPN is heavily inspired by the PixelCNN described in the original paper by van den Oord *et al.* [31]. The PixelCNN is used to generate images on a pixel by pixel level. The network predicts the current pixel based on all the previous pixels in the image, row by row. Therefore the probability of an n x n image is:

$$p(x) = \prod_{i=1}^{n^2} p(x_i|x_1, ..., x_{i-1}) \tag{3.10}$$

To achieve this, the authors masked the pixels in a convolutional layer so the layer could only use as input the previous pixels. After the first layer, the following layers also get the prediction of the current pixel x_i from the previous layers as input. This led to two types of masks referred to as A-type and B-type, which are defined as follows:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3.12}$$

The A-type mask is only used in the first layer and limited the connections to previous pixels. All the following layers use the B-type mask and are also able to see the current pixel, x_i. This allowed layers to pass their prediction of the current pixel, x_i, onto the next layers.

The decoder in the VPN is a PixelCNN. It used the output of the ConvLSTM layer, $F_{<t}$, as context along with the current frame, F_t, to make a prediction. The prediction of the next frame is then computed using:

$$p(x) = \prod_{t=0}^{T} \prod_{i=0}^{n} \prod_{j=0}^{n} p(x_{t,i,j} | x_{<}) \tag{3.13}$$

where $x_{<} = x_{(t,<i,<j)} \cup x_{(<t,:,:)}$ and combines data from both the current frame, F_t, and all previous frames, $F_{<t}$. Equation 3.13 is similar to equation 3.10, but it adds a time variable and uses previous frames as part of the prediction. The current frame is only submitted into the network during training and is used to help teach the network the spatial structure of the image. When a VPN is tested, the current frame is not passed into the decoder, and instead, a blank image filled with zeros is used. This way, the network will only be able to use previous frames in its prediction for the current frame, F_t.

Structure of the Encoder and Decoder

The encoder and decoder networks I implement follow a similar architecture to the PixelCNN [31], but the encoder does not implement any pixel masking. The encoder started with a 7 x 7 convolutional layer with 256 channels. This layer has no activation layer that follows, and the output is passed directly to 16 residual blocks with a kernel size of 3 x 3, as shown in [31] and Figure 3.4. The residual blocks use a pre-activation ReLu inspired by the work of He *et*

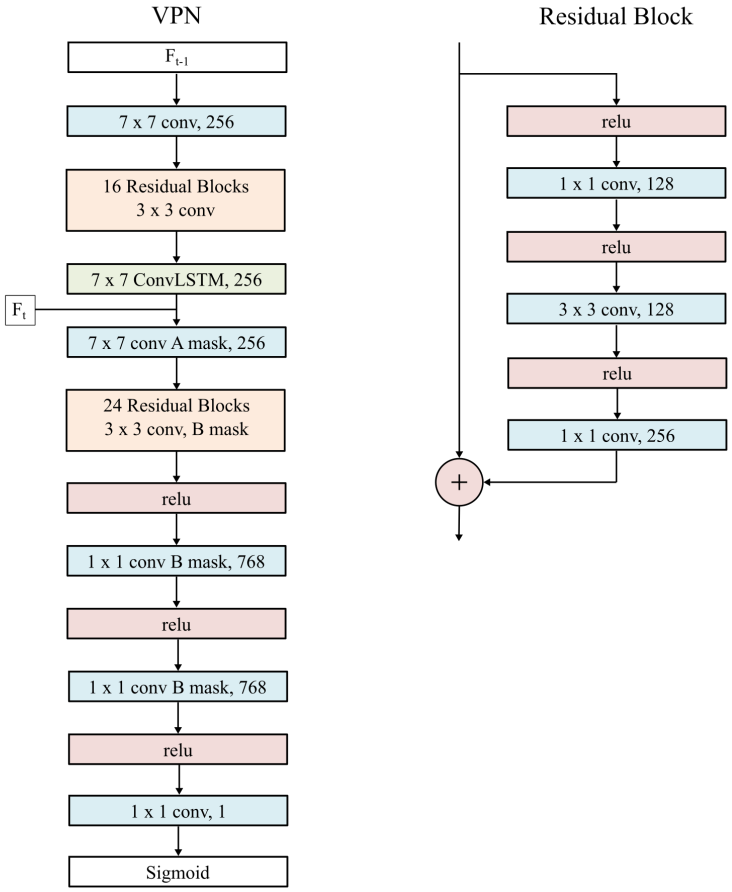

Figure 3.4: **Right:** The structure of the VPN network. **Left:** The structure of a Residual Block that was introduced in introduced in [13] and then modified to follow the structure above in [31].

al. [14]. The output of the encoder is passed to a single ConvLSTM layer with a kernel size of 7 x 7 and 256 hidden states.

The decoder starts with a 7 x 7 convolutional layer with 256 channels, and like the encoder, this layer has no activation layer. This first convolutional layer is masked with the A-type mask described above. This masking allows the layer to take the output from the ConvLSTM as well as the current frame, F_t, as input. Since the A-type mask is used, the current pixel, x_i, is masked out for F_t. This helps the decoder learn the correct structure, but does not allow the network to pass the current pixel of F_t all the way through the network to the output. Next, the output from the convolutional layer is passed to the 24 residual blocks with a kernel size of 3 x 3. Each 3 x 3 layer in the residual

block is masked with a B-type mask, which was described in PixelCNN [31]. After the residual blocks, there are two convolutional layers with a kernel size of 1 x 1 and a B-type mask where both layers have 768 channels. The final layer outputs the resulting image.

Depth of the VPN

The ConvLSTM and spatio-temporal networks described in the previous sections use only a few layers to capture the motion. The limited depth of the two networks requires the kernel sizes of the layers to be large enough that the receptive field of the network can accurately capture the motion. One can see this in the spatio-temporal network when it uses two 15 x 15 convolutional layers. If the kernels in the network are too small, the networks will fail to capture the larger motions.

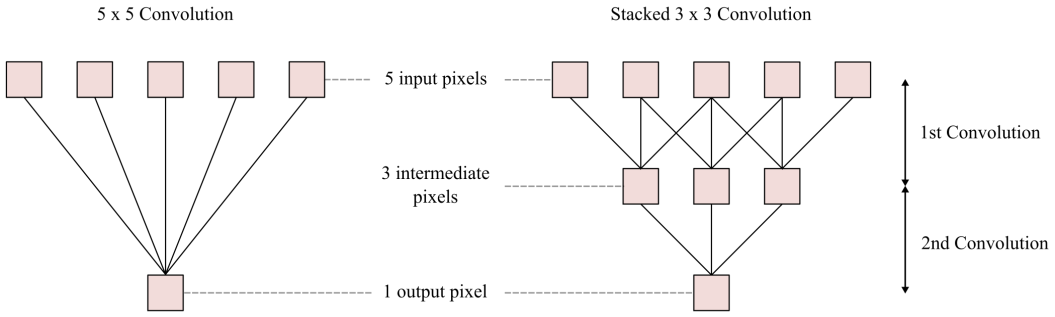

Figure 3.5: The figure shows that a single 5 x 5 convolution layer has the same receptive field as two stacked 3 x 3 convolution layers.

Instead of building a shallow network with a limited number of layers with a large kernel size, the VPN network uses more layers with smaller kernel size and a larger number of channels to help encode the complexities of the motion. Although the kernel size for most of the layers is small, only 3 x 3, the encoder has a respective large field due to the depth of the network. The 16 layers is equivalent to a single layer with a 33 x 33 kernel.

Going deeper also allows a network to capture a richer structure. This has been shown in work done by Eldan and Shamir [5]. They showed that a 3-layer network could learn structures a 2-layer network is unable to learn.

Also, using a deeper network has been shown to perform better in practice. If one looks at the VGGNet [27] it has 16 layers with $\sim$ 140 million parameters. While the ResNet [13] outperformed the VGGNet with 152 layers and only using $\sim$ 2 million parameters.

3.3 Setup

All the networks described in this section are simulated in Tensorflow version 1.8. Most of the networks can run on a single NVIDIA GeForce GTX 1080 ti; however, the VPN [21] requires too many resources to train in a timely manner, and therefore this network is trained on four NVIDIA P100 Pascal GPUs with 16 GB of HBM2 memory from Compute Canada. Even with the four P100 cards, the network takes multiple weeks to train.

3.4 Training the Networks

All three networks use RMSProp [37] for optimization and have a decaying learning rate. The ConvLSTM network, the Spatio-Temporal network, and the VPN are all started with a different learning rate of 10^{-3}, 10^{-4}, and $3 \cdot 10^{-4}$ receptively.

3.4.1 Training the VPN: Frame Masking

As mentioned previously, the VPN uses the current frame, F_t, as part of the input for the PixelCNN decoder during training. Part of the current frame, F_t, is masked out, so the decoder does not see the entire image. This masking allows the decoder to learn the structure of the image. By seeing what pixels should come first, the decoder can predict what pixels should be in the current position. If the network does not mask out the current pixel of F_t then the network will learn how to pass the current pixel through the network.

One downside to this approach is that the network will put too much weight on the input F_t even with the masking. To counter this effect, I weighted the cost function at F_t. This required me to run the decoder twice during each training step. Once with F_t passed as input and once without. Then I weighted

the two-loss functions and added them together. I found that having a weight of 0.2 worked well for the F_t loss.

Chapter 4

Evaluation and Comparison of the Three Networks

This chapter describes the results and analyzes my various implementations of the three networks discussed in Chapter 3. These networks are ConvLSTM, Statio-Temporal, and VPN. The chapter will begin by looking at what the trained networks can achieve on the same test data used by the original authors. In the next section, individual test samples will be analyzed. Finally, the three networks will be tested and analyzed for different data sources.

4.1 Checking the Implementation of the Three Architectures

The experiments started with the implementation of the ConvLSTM network by Shi *et al.* [35]. This was found to be a good starting point as the ConvLSTM network is the easiest to implement, and it would help confirm if my implementation of ConvLSTMs is correct. Validating the ConvLSTM implementation is vital because the other two networks use a ConvLSTM layer. As seen in Table 4.1, my results for the ConvLSTM network are comparable to the results published by Shi *et al.* [35]. One can safely assume that there are no significant flaws in my implementation of the ConvLSTM. If the other networks are unable to achieve similar results, then the possible errors occur elsewhere in the network.

The second network implemented was the Spatio-Temporal network for-

merly described by Patraucean *et al.* [32]. This network was initially implemented using only information found in the paper; however, results were not as expected. To double-check my implementation, the authors' source code, which they posted online, was reviewed. Although I referred to the source code to confirm my implementation, the original paper did provide a clear description of their network.

When comparing my implementation with the original paper and going through the entire code base, no issues with my code were found. The authors did use the Torch library instead of the Tensorflow deep learning library that I used. However, even though the libraries are different, the same functions are being used in both implementations. Also, both libraries are industry-proven standards; it is unlikely that a function implemented in one of the libraries has a bug.

During training, the same optimizer, *rmsprop*, was used in both implementations and with the same hyperparameters. Other hyperparameters and optimizers, such as Adam [22], were used to test whether better results could be achieved. All testing produced similar or worse results. No better results were achieved. Therefore, the same optimizer and hyperparameters described by the original authors [32] were used.

In the paper, the authors used a different cost function for training than the other two networks. The authors also formatted their test results differently using cross-entropy, which is the same cost function as the other networks. The authors might have been unclear about their results, and the expected results posted in Table 4.1 are not correct. Another possibility is that the authors' testing methodology might have been different from the other networks or that they might have used different test data. Either way, I was not able to achieve similar results.

The third network implemented was the VPN, which was created by Kalchbrenner *et al.* [21]. As stated in Chapter 3, the implementation in the original paper was unclear at times; therefore, I cannot be entirely sure that my network is the same as the network the authors used in their paper. The authors did not post their source code, and only the original paper was used as a ref-

erence. However, multiple variations were tested to eliminate possible errors. These variations include changing the kernel size of the ConvLSTM, increasing the depth of the network, using a 7 x 7 convolutional layer after the ConvLSTM to better match the original PixelCNN [31] algorithm, connecting the output of the ConvLSTM directly into the residual blocks of the decoder. Other minor tweaks were tested, such as changing some of the hyperparameters. Nothing seemed to make any drastic improvements.

The one variation that had the most significant effect on the performance was increasing the kernel size in the ConvLSTM. Due to the number of filters used by the ConvLSTM layer, increasing the kernel size caused the number of trainable parameters to increase drastically. The authors did not mention the kernel size of the ConvLSTM. However, they did mention a kernel size of 3 x 3 on other layers, and it was initially presumed they used a kernel size of 3 x 3 for the ConvLSTM layer. However, to give the best chances for the network to work well, I used a kernel size of 7 x 7 as it performed the best in my testing. I was unable to increase the kernel size beyond 7 x 7 due to memory limitations.

Even with the different tests and tweaks to the VPN network, results published in the original paper were not achieved. The network did outperform the other networks and is still one of the best performing networks for this task. Most versions of the VPN tested were able to achieve an average training error between 220 and 280. Networks with a kernel size of 3 x 3 for the ConvLSTM layer had a significantly higher error. Once the kernel size was increased to 5 x 5, the network performed much better. The change to a 7 x 7 kernel had diminishing returns compared to the jump to 5 x 5 from 3 x 3.

Table 4.1: Comparison of Results

Network	Expected	Results	1st-Frame	10th-Frame
ConvLSTM	367.2	361.6	219.7	457.0
Spatio-Temporal	179.8	354.7	302.3	395.2
VPN	87.6	170.3	119.8	216.1

Table 4.1 shows the results of the three networks as published in the original papers, labeled “Expected”, and the results obtained from my implementation.

The error values are the average error over the ten predicted frames. The last two columns correspond to the average error of the first and last frames, respectively. All errors were calculated using cross-entropy using the same test data provided by Srivastava *et al.* [36].

4.2 Training

Table 4.2: Number of Parameters in Networks

Network	Number of Parameters
ConvLSTM	$3.7 * 10^6$
Spatio-Temporal	$1.0 * 10^6$
VPN	$3.8 * 10^7$

Training of the ConvLSTM network was reasonably straightforward. The paper was detailed enough to allow the re-implementation and training of the model. Training took about two days on an NVIDIA GTX 1080 TI and the trained model performed as expected.

The spatio-temporal network also took about two days to train on a 1080 ti. It was not difficult to achieve an error of 354.7 which is shown in Table 4.1. However, the model could not achieve anything better than the reported 354.7. Different optimizers, learning weights, parameter initializations, and cost functions were used. All performed as well or worse than the reported results. Better results were not achieved.

If the spatio-temporal network was trained only to predict one frame into the future, then a significantly lower error could be achieved. During training, an error of approximately 210 was achieved, which is considerably better than the reported one frame prediction of 302.3 shown in Table 4.1. However, once this network was asked to predict more frames, the performance was significantly worse, and for ten frame predictions, the error was approximately 600. This approach was, therefore, scrapped as the purpose of the network was to predict ten frames.

This large discrepancy between learning to predict one frame versus ten frames could be caused by a limited number of parameters. The network might

be unable to maintain important information from previous frames. However, if the network is only tasked with predicting one frame, then it may not need to store as much information for the later frames. This would allow it to use more parameters for the generation of the first frame. Increasing the size of the layers could help; however, this would not be the original network described by Patraucean *et al.* [32].

The VPN took multiple weeks to train on four NVIDIA P100 Pascal GPUs with 16 GB of memory. With the VPN described in Chapter 3 it was not difficult to achieve the results is shown in Table 4.1. However, it does take a significant amount of training time to converge.

4.3 Over-fitting

To confirm the accuracy of the models, and the validity of the data, a second test set with two moving MNIST [26] digits was created. The motion of the digits was generated using the same algorithm as the training data. In order to test the spacial reconstruction of the digits, the digits were pulled from the test set of the MNIST database.

Table 4.3: Comparison of Test Set

Network	Original Test Set	Research Test Set
ConvLSTM	361.6	361.5
Spatio-Temporal	354.7	354.8
VPN	170.3	205.5

Table 4.3 shows the error of my research test set compared to the original test set provided by Srivastava *et al.* [36]. As expected, the error between the two test sets is comparable for the ConvLSTM and Spatio-Temporal networks. However, the error for the VPN is significantly lower for the original test set. Furthermore, the error of 170.3 is almost the same as the training error. Therefore, this discrepancy in error is likely caused by the VPN over-fitting the MNIST data.

This over-fitting is likely caused by the PixelCNN as it is designed to generate new images. It has probably learned to generate MNIST digits from

the training set of the MNIST database. When it is given a new digit from the MNIST test set, it will convert it to something that came from the MNIST training set. This would explain why the resulting testing error is higher than the training error of approximately 170. If the PixelCNN is the issue, then improving the performance of the PixelCNN will likely increase the over-fitting.

During training, the VPN struggled to achieve an error lower than 200. It was only once an error lower than 200 was achieved that the discrepancies between the two test sets appeared. Likely, if an error lower than 200 is achieved, then the model will be over-fitting the MNIST data. Therefore, it will not improve and might even start to perform worse on the test MNIST data.

As stated above, the kernel size of the ConvLSTM layer in the VPN was limited to 7 x 7. If a network with a larger kernel size is trained, then the model might perform better. It could also increase the over-fitting. It is important to test thoroughly. A validation set used while training would help to identify and limit any over-fitting.

This case of over-fitting shows a potential flaw in the original test data. Others who use the original test data should be aware and test their models on other test sets pulled from the test set in the MNIST database.

4.4 Comparing the Three Networks

This section describes a more in-depth comparison of the three networks. It can be challenging to see exactly how well the networks perform and where they fall apart if only the average error is considered. Unfortunately, papers often publish the average error in the results. By implementing all three networks, I was able to run more detailed tests and gain further insight into how they perform.

Because the VPN has discrepancies between the two test sets, as shown in Section 4.3, both test sets will be used where possible. If the test set is not specified, it came from my research test set and not the one provided by Srivastava *et al.* [36].

4.4.1 Distribution of the Networks Error

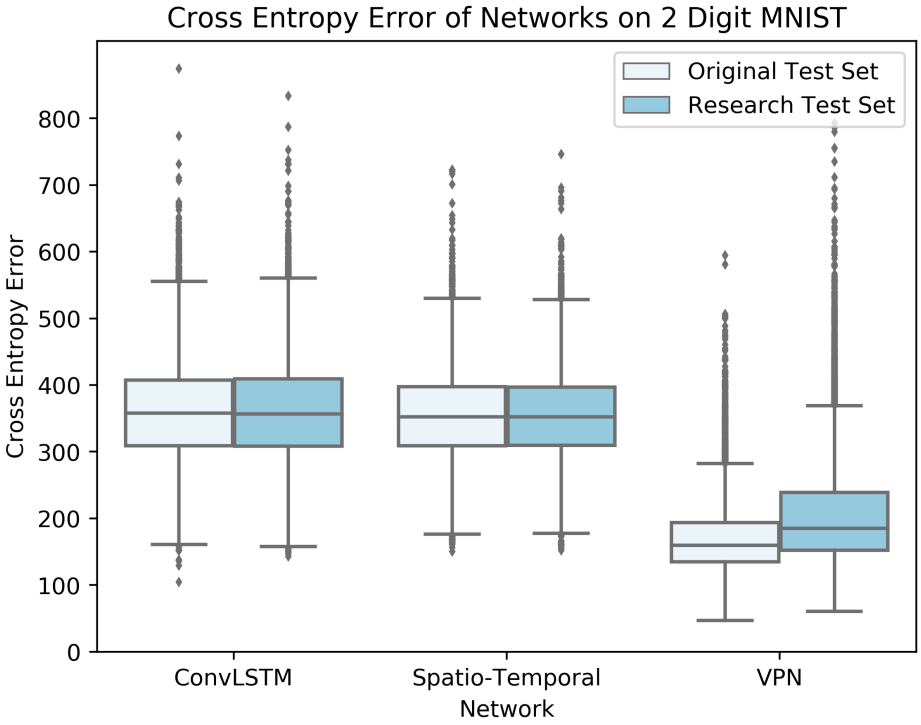

Figure 4.1: Box plot showing the average error after all ten frames. This is the same test data that all three papers used for their testing, the MNIST data with 2 digits.

I began by looking at the distribution of the error for each network. Box plots were used to compare the distributions of all three networks, as shown in Figure 4.1. Each sample in the distributions is still the average error over all ten frames.

The ConvLSTM and the spatio-temporal networks perform about the same, but the ConvLSTM has a slightly larger variance. Figure 4.1 shows that the ConvLSTM minimum is marginally lower than the spatio-temporal network. This suggests that there are some test samples where the ConvLSTM network will slightly outperform the spatio-temporal network. However, since the Spatio-Temporal network has a slightly lower average, the spatio-temporal network outperforms the ConvLSTM on average over the ten frames.

The VPN has a significantly lower error on average than both of the other

two networks. When comparing the models with the original test set, the VPN also has the smallest variance. A smaller variance implies a more consistent performance. When using my test set, the VPN still performs better on average; however, the distribution has a positive skew. This implies that the majority of the test samples will have a low error, but there are going to be test samples that show drastically worse performance. This can be seen in Figure 4.1 when looking at the two test sets for the VPN. My research test set has a significantly higher maximum and larger outliers.

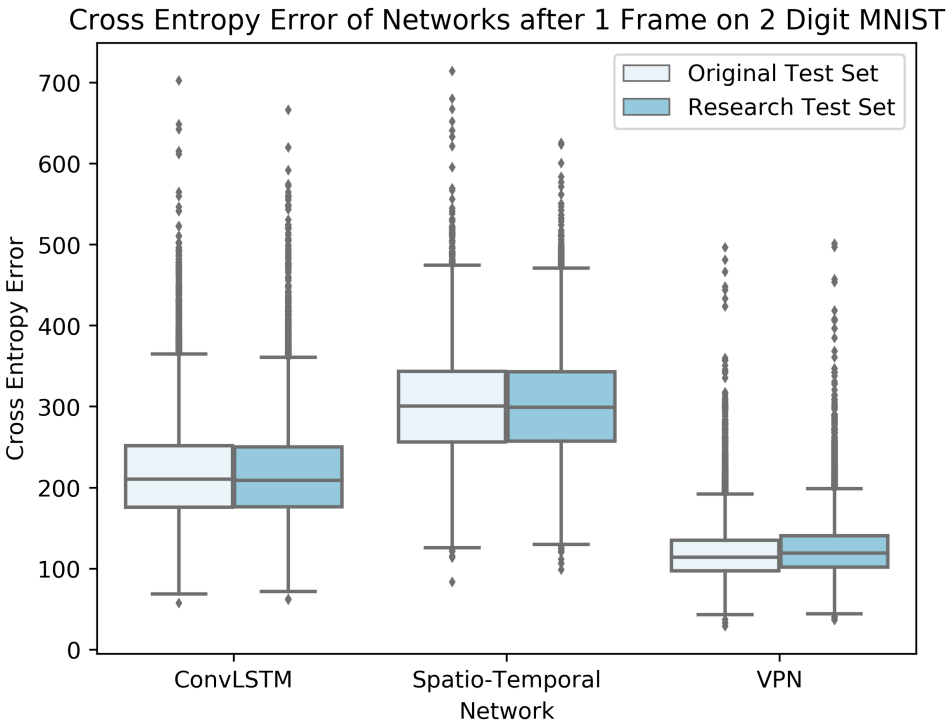

Figure 4.2: The same data as in figure 4.1, but only showing the error of the first frame instead of the average of all ten frames.

Next, I looked at the error of the first frame instead of averaging the error of all ten frames. When looking at only one frame, the errors of the other nine frames were dropped from the results. These results, as shown in Figure 4.2, reveal some interesting insight into the behavior of the three networks. The ConvLSTM now significantly outperforms the Spatio-Temporal network. The ConvLSTM would be the better choice if only predicting one frame into the

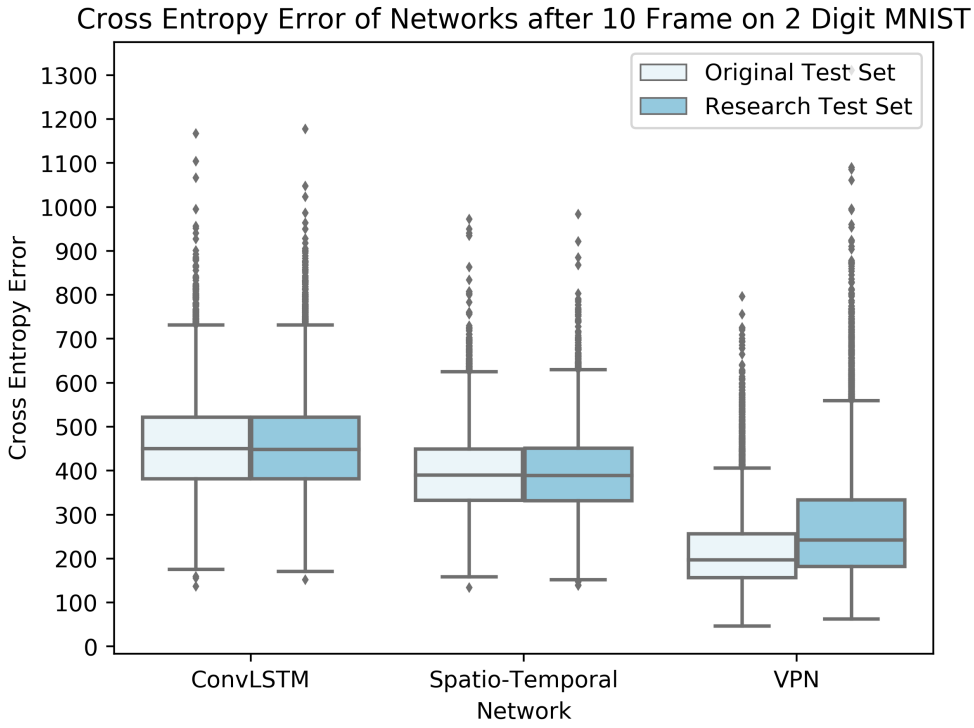

Figure 4.3: The same data as in figure 4.1, but only showing the error of the tenth frame instead of the average of all ten frames.

future as needed. This also implies that the ConvLSTM network was likely better at generating new frames, but was unable to maintain the required information over time. Therefore, the error was likely caused by the structure of the digits degrading over time.

Figure 4.3 shows a large increase in error of the last frame as compared to the first frame, as seen in Figure 4.2. This supports the idea that the ConvLSTM network was likely better at generating new frames, but could not maintain needed information over time. It still does not tell us exactly why the network performs worse in later frames. Section 4.4.2 will look at individual test samples to gain further insight.

When only predicting one frame, the VPN still outperforms the other two networks, as shown in Figure 4.2. Like the other two networks, the error of the VPN increases over time, as shown in Figure 4.3. This suggests that the VPN still struggles to maintain the needed information over time and the structure

of the digits degrades over time.

There was a surprising result when I looked only at the error of the first frame, as shown in Figure 4.2. The error distribution of the first frame in my research test set does not show the same positive skew, as shown in Figure 4.1 where all ten frames were used. However, the last frame of the VPN, as shown in Figure 4.3, shows an increase in error and a positive skew. This suggests that the over-fitting in my research test set is occurring in the later frames.

Because the only difference between the original test set and my test set is that my test set pulls the digits from the MNIST test set instead of the training set, the increase in error likely comes from the digits themselves. Also, since the error is similar after only one frame, then the error likely comes from the network being unable to maintain the spacial structure of the digits. Due to the generative nature of the VPN, it is likely the digits are being converted into digits that resemble digits found in the MNIST training set. Section 4.4.2 will look at individual test samples to gain further insight. Section 4.5 shows a specific test sample where the VPN converts a digit to resemble a digit from the MNIST training set.

The final test was to check whether my research test samples have a correlation of the error distribution in all three networks. For example, if I pick a random test sample, do the networks find that test to have the same relative error compared to all the other test samples? Does network A find a test sample easy (e.g., does it have one of the lowest relative errors?). And if network A finds this test sample to be easy, then does network B also find it an easy test sample? Or does network B find this test sample to be one of the more difficult test samples relative to the entire test set?

To test the correlation, each test sample was given an index number and was sorted based on the error. Next, a correlation test was run on the index numbers. If the tests are in the same order, the correlation would be one. If they are in the complete opposite order, then the correlation would be a negative one. If they are in a random order, the correlation would be close to zero.

The correlation coefficients from the Spearman's test, shown in Table 4.4,

Table 4.4: Results of the Spearman's Correlation Test

	Spearman's Correlation
ConvLSTM and Spatio-Temporal	0.798
ConvLSTM and VPN	0.574
Spatio-Temporal and VPN	0.553

indicate a strong correlation between the ConvLSTM and Spatio-Temporal networks. Then there is a moderate correlation between the VPN and the other two networks. This means that in general, if one test sample performs well on a network, then there is a high likelihood that it will perform well on another network. Based on these findings, I hypothesize that there is a good chance that some features in the test data cause problems for all these networks.

4.4.2 A Look at Individual Test Samples

This section will examine five randomly selected test samples. Due to the over-fitting in the VPN, the samples were taken from my research test data set. The digits in my research data set are taken from the MNIST [26] test set. The purpose was to see how well the networks handle specific test samples. Do the digits in the image become unreadable, or do they maintain their spacial structure? Does each network correctly move the digits, or does the digits' motion deviate from the ground truth?

Unless otherwise specified, the following figures show an individual test sample where: row one is the ground truth, row two is the ConvLSTM, row three is the Spatio-Temporal network, row four is the VPN, row five is the overlap of rows one and two, row six is the overlap of rows one and three, and row seven is the overlap of rows one and four. The last three rows help show the error between the ground truth and the three networks' results. In the last three rows, the white digit is the ground truth, and the red is the results. Since the white digit of the ground truth covers most of the results, it is difficult to tell when the structure has failed. However, it shows where the digits' motion deviates from the ground truth as another digit in red appears.

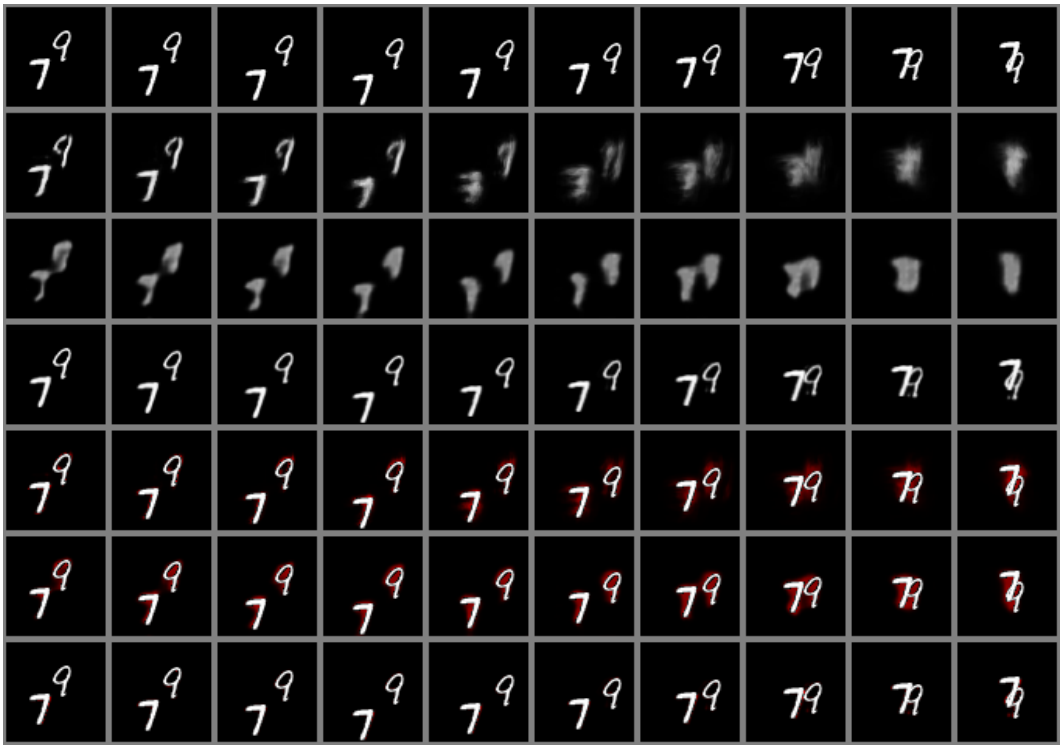

Figure 4.4: Sample 608

Figure 4.5: Sample 3713

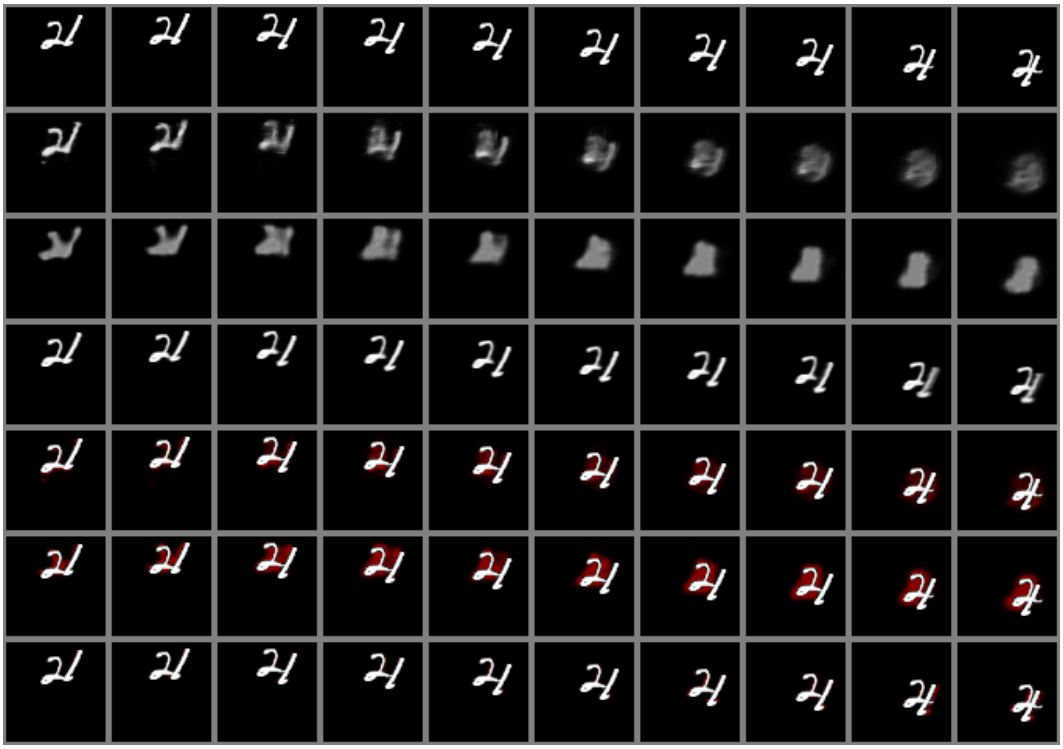

Figure 4.6: Sample 4543

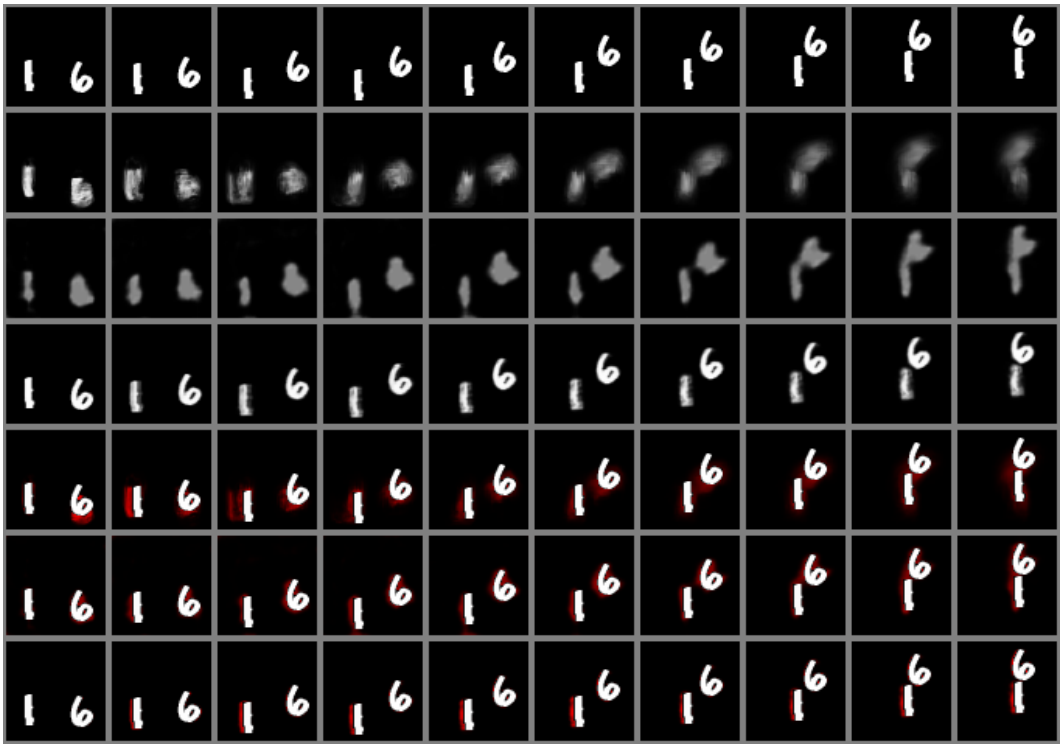

Figure 4.7: Sample 5214

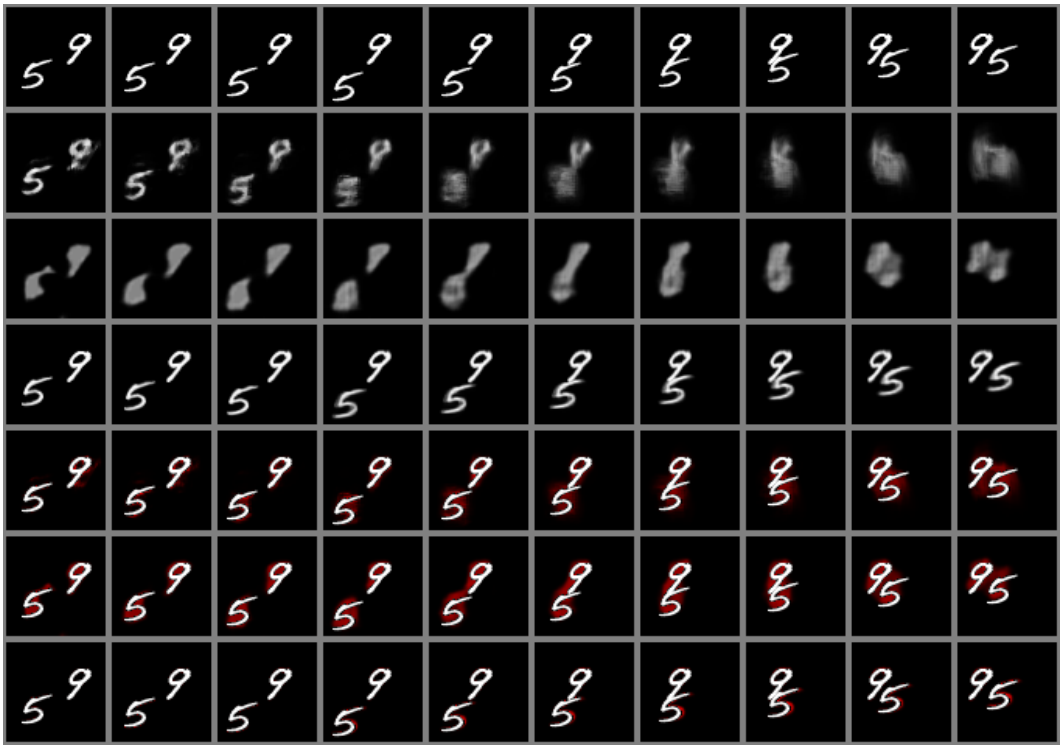

Figure 4.8: Sample 5721

Figures 4.4, 4.5, 4.6, 4.7, and 4.8, show that all three networks seem able to track the motion of the digits. The last frame of Figure 4.5 shows this especially well. Although the digits get blurred and the spacial structure is lost, the digits end in the image's relatively correct positions. This suggests that the networks are not struggling to learn the digits' motion but are struggling to maintain the digits' spacial structure.

If the networks are struggling to maintain the digits' spacial structure, this would explain why the VPN performs the best. The decoder of the VPN is a PixelCNN, and its purpose is to generate images. Therefore it can generate a new digit based on the previously encoded structural information. The VPN network also has the most parameters, allowing it to encode more information than the other networks.

In the test sample 3713 for the ConvLSTM, the second row in Figure 4.5, the first frame shows that the ConvLSTM is able to generate a fairly accurate prediction. However, as time goes on, the digits' structure is lost, and the digits end up being two blurry blobs. On the same test sample, the VPN,

shown in the fourth row, can maintain the digits' spacial structure through all ten frames.

Even when there is no overlap, the ConvLSTM and Spatio-Temporal networks struggle to maintain the structure. Figure 4.7 shows even in the first frame, the digits in the ConvLSTM and Spatio-Temporal networks have visible artifacts. By the fifth frame, most of the structural information is lost even though the digits do not overlap.

The decoder of the VPN could still generate blurry digits. Figure 4.7 shows that the tenth frame loses some of the structure, especially for the digit "1". Interestingly, the digits lose some of their spacial structure even when there are no overlapping digits. So even without overlap, the VPN can still struggle to maintain the structure of the digits.

4.4.3 Effects of Increasing the Number of Digits

This section looks at how increasing the number of digits affects the networks. A simple line plot, as shown in Figure 4.9, is used to illustrate the changing error. The average error of all ten frames is used in the plot.

As shown in Figure 4.9, the error rate increases almost linearly when the number of digits is increased. Based on this line plot and my previous conclusions, much of the error will likely come from a loss of structural information. Therefore the digits are becoming blurred in the later frames. Figure 4.10 shows an example with four digits and confirms this conclusion. Revisiting Section 4.4.2 it can be seen that the tracking of the two digits was maintained, but the structure was lost.

Figure 4.10 shows that even with four digits, as compared to two digits, the networks can maintain a somewhat correct motion. The motion of the digits can be observed, particularly with the VPN on the last row. However, even looking at the other two networks, there is some evidence of correct motion. With the blurry digits and loss of spatial structure, digits' motion can be difficult to observe. For the spatio-temporal network on the third row, the "1" digit moves across the image from left to right. This is clear when looking at frames four to eight. The ConvLSTM network is even harder to see any

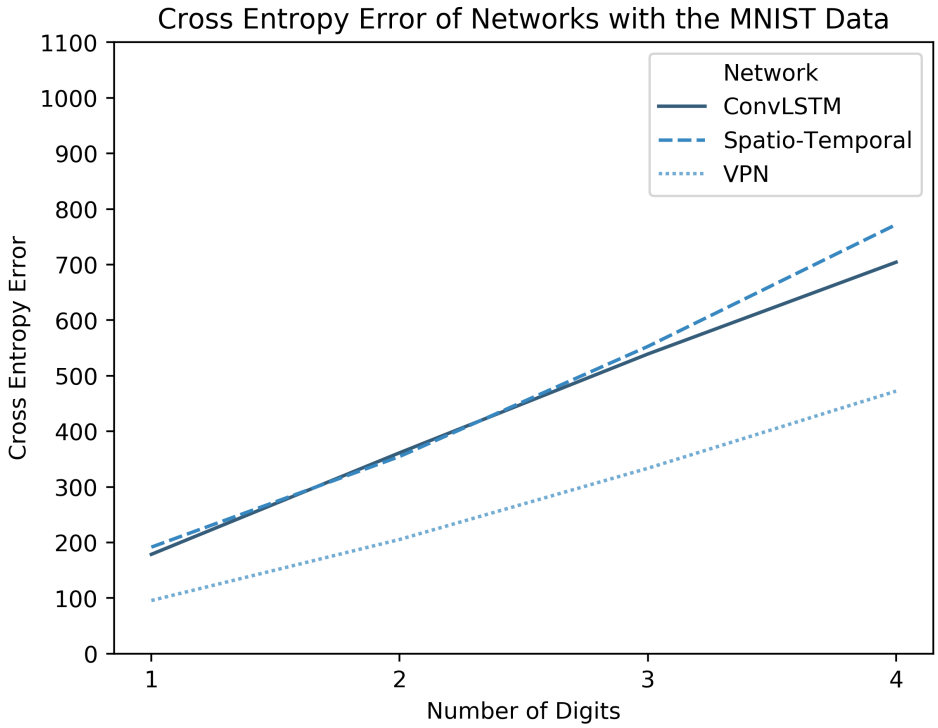

Figure 4.9: This shows how each network is effected by the changing of the number of digits in the images. All networks were still only trained once on data with two digits.

movement as the structure is completely degraded. However, the "7" moves down and then back up to the cluster of digits.

After looking at the sample, it appears that the networks are able to maintain the motion when increasing the number of digits. Just like when there were only two digits, the networks are still going to lose the digits' structure over time. The VPN appears to be able to maintain the structure of the digits for a longer time.

Although the structure is maintained longer with the VPN, artifacts begin to appear. Looking at frames four and on, the "1" starts to turn into a "6". This seems to be caused when one of the "7" digits and the "1" overlap in the third frame. During the overlap, a "6" appears. To a human observer, it is clear that these are just two overlapping digits. The VPN did not make this distinction.

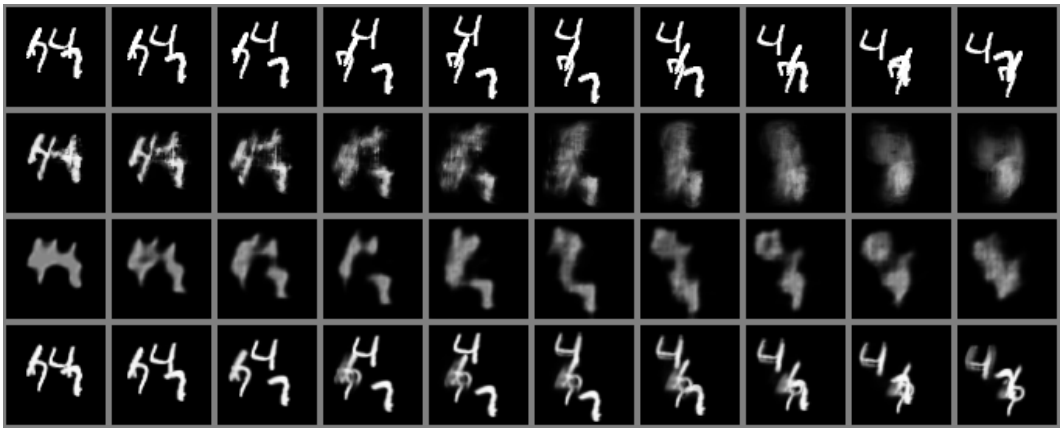

Figure 4.10: A random sample from the test set with four digits. The top row is the ground truth, followed by the ConvLSTM, Spatio-Temporal network, and finally the VPN.

4.5 Digits From a Different Dataset

Based on my previous findings, it appears that the biggest struggle for all these networks is maintaining the spatial structure of the digits. To test the network's ability to handle unique digits, the networks are tested on digits that are from a different dataset. Hopefully, this will not allow the networks to fall back on structures similar to the ones seen in training. The VPN is likely to struggle with this task as we already saw it struggle with the digits pulled from the MNIST test set. However, how well do these networks handle unique digits?

Figure 4.11 shows the error distribution of the MNIST and EMNIST test sets. Each sample is the average for all ten frames. The test samples only include one digit to isolate the network's ability to handle a new digit with a unique spacial structure. All networks perform worse on the new dataset, as shown in Figure 4.11. This is to be expected as the networks are not trained on this data. Both the ConvLSTM and Spatio-Temporal networks perform about the same on both the EMNIST data. The VPN has the most significant increase in error from the MNIST to the EMNIST dataset. There is also a substantial increase in the variance and a positive skew for the VPN. This implies that some test samples will give significantly worse results when running on the VPN. This is clear when looking at the maximum and outliers

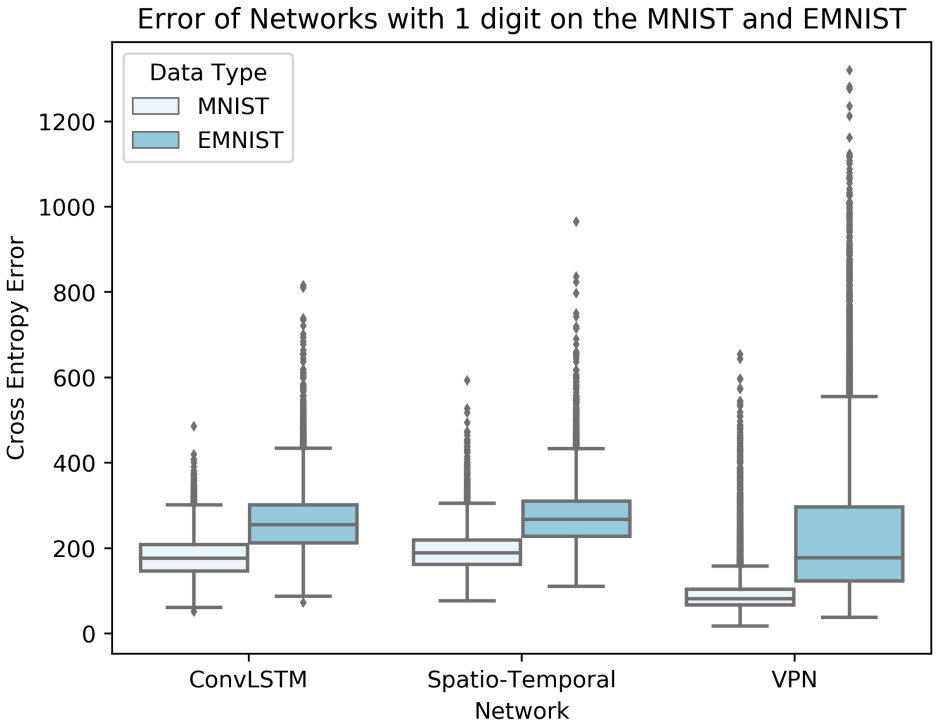

Figure 4.11: A box plot showing the difference in error between MNIST and EMNIST datasets.

of the VPN in Figure 4.11.

Figure 4.12: A sample showing how the spacial structure will degrade over time.

Figure 4.12 shows an individual test sample and gives a better insight into what is happening with the VPN network. The random test sample shows the VPN struggling to maintain the spacial structure of this new digit. In the last frames, the digit starts to become blurred as well as the VPN starts to add another edge, creating loops, to the right of the digit. It appears that the VPN tries to turn this digit into a "8" as to closer resemble something from

the MNIST training set.

This is one possible downside to the VPN and using a generative network like the PixelCNN as a decoder. The PixelCNN will learn to generate new images based on images it has previously seen during training. Therefore, if the PixelCNN is given a unique image, it will most likely generate an image closer to something it has already seen. This is expected, as this is what it is trained to do. Now, as seen in Figure 4.8, this can help generate cleaner images. But the generative process can also correct what it perceives to be an error when it is not an error and is instead a correct part of the image.

Chapter 5

Conclusion

5.1 Overview and Research Results

In the fields of computer vision and AI, many datasets contain motion. To improve our current systems, it is advantageous to have a strong understanding of our tools' strengths and weaknesses, such as LSTMs, ConvLSTMs, and PixelCNNs. This thesis evaluated ConvLSTM for frame prediction to help better understand motion in neural networks. Three different neural networks were implemented and trained. The three networks included, the original ConvLSTM paper by Shi *et al.* [35], the Spatio-Temporal network by Patraucean *et al.* [32], and the VPN by Kalchbrenner *et al.* [21].

Implementing and training the three networks allowed thorough testing and comparison of the accuracy and performance of the networks. This thesis started by testing these networks on the original data provided by Srivastava *et al.* [36]. This allowed me to compare my research results with those of the original authors. The original ConvLSTM was reasonably easy to re-implement and train, and it performed as expected. The Spatio-Temporal network and VPN were unable to achieve the desired results, but still showed improvements over the ConvLSTM. The VPN outperformed both the ConvLSTM and spatio-temporal networks and performed better than the reported results by Patraucean *et al.* [32].

This thesis tested the networks on alternative data. This alternative data was similar to the original data provided by Srivastava *et al.* [36], but varied in ways to stress different aspects of the networks. The original data contained

two MNIST [26] digits moving around in an image. The digits were pulled out from the MNIST training set.

The first alternative dataset proposed in this thesis was almost identical to the original data; however, the digits were pulled out from the MNIST testing set. The purpose of this data was to detect whether any networks were over-fitting the data. A significant flaw in the original experiment proposed by Srivastava *et al.* [36] was that the training and testing both used data from the MNIST training set. The networks can over-fit the data, and this can go undetected if the networks are tested on the same data as they were trained on. After testing on my research data, the ConvLSTM and spatio-temporal networks showed no signs of over-fitting. The VPN did show that it over-fit the data. The PixelCNN decoder most likely caused this.

Using my proposed research data, individual samples were analysed. The networks appeared to maintain accurate motion for the digits. However, all networks struggled to preserve the spatial structure of the digits. In the last frames, the digits became blurry blobs. The VPN was able to keep the spatial structure longer but still started to struggle in later frames.

The second alternative dataset proposed used digits from the MNIST testing set and varied the number of digits in the image. My research testing started with one digit and incrementally increased the number of digits until four digits. When the number of digits increased, the networks were able to track the motion of the digits. However, the networks struggled to maintain the spatial structure of the digits in later frames. This showed that the networks were not limited to only tracking two digits.

The final alternative dataset proposed used digits from the EMNIST [2] dataset. This test was designed to test the networks' reliance on the spacial structure of the digits. All networks performed worse on this data, suggesting they relied heavily on the shape of the digits. My case study on the VPN showed the VPN slowly turned the digit in the EMNIST dataset into something closer to a digit in the MNIST dataset. This highlights that these networks cannot be used on different data without retraining.

5.2 Implementation Challenges

The ConvLSTM and Spatio-Temporal papers were written and straightforward to implement. The VPN paper lacked essential details. For re-implementation and further testing, the authors must include a detailed description of every layer and its parameters. Also, the hyperparameters are important for training and need to be included. The VPN was vague on some of the layers and their kernel sizes. This made it difficult to re-implement.

The VPN drastically increased the number of parameters in the network. This seems to be a growing trend in machine learning. Instead of creating more efficient networks, researchers are only focused on reducing the error. This can make it difficult for researchers without access to super-computers to compete in the field. Increasing the number of parameters makes it difficult to know if the approach was better, or if the network performed better because it contained more parameters.

5.3 Future Work

In the thesis, the VPN was shown to over-fit the data. Other networks and future work should be cautious with over-fitting. To combat over-fitting, the networks should be tested on data that is taken from the MNIST test set. Also, a validation set should be used during training. This will help detect early over-fitting.

Current methods are still not perfect at predicting future frames on the simple MNIST data. There is still room for improvement using the MNIST data. Researchers should probably focus on the ConvLSTM as the networks struggle to maintain structural information into the later frames. It might be beneficial to test alternatives to LSTMs and recurrent networks. Once better solutions to this problem are discovered, more real-world test data will be needed. Like our current data, this new data should be shared and used between the varying implementations.

The current test data does not implement any form of changing velocity.

This was outside the scope of this research as it would require the networks to be trained on different data. However, acceleration is important as most real-world motion does not have a constant velocity.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv*, vol. 1409, Sep. 2014. 1, 13

[2] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *CoRR*, vol. abs/1702.05373, 2017. arXiv: 1702.05373. [Online]. Available: http://arxiv.org/abs/1702.05373. 17, 50

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009. 5, 8

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810 . 04805. [Online]. Available: http://arxiv.org/abs/1810.04805. 12

[5] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," *CoRR*, vol. abs/1512.03965, 2015. arXiv: 1512.03965. [Online]. Available: http://arxiv.org/abs/1512.03965. 6, 27

[6] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," *CoRR*, vol. abs/1504.06852, 2015. arXiv: 1504.06852. [Online]. Available: http://arxiv.org/abs/1504.06852. 9–11, 13

[7] D. Fleet and Y. Weiss, "Optical flow estimation," in *Handbook of mathematical models in computer vision*, Springer, 2006, pp. 237–257. 9

[8] F. A. Gers, J. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, pp. 2451–2471, 2000. 17

[9] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Mar. 2003, ISSN: 1532-4435. DOI: 10 . 1162 / 153244303768966139. [Online]. Available: https://doi.org/10.1162/153244303768966139. 18

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf. 8, 11

[11] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. arXiv: 1308.0850. [Online]. Available: http://arxiv.org/abs/1308.0850. 18

[12] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Ng, "Deepspeech: Scaling up end-to-end speech recognition," Dec. 2014. 1, 4, 5, 13

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv e-prints*, arXiv:1512.03385, arXiv:1512.03385, Dec. 2015. arXiv: 1512.03385 [cs.CV]. 1, 4–6, 11, 24, 26, 28

[14] ——, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016. arXiv: 1603.05027. [Online]. Available: http://arxiv.org/abs/1603.05027. 24, 26

[15] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006, ISSN: 0036-8075. DOI: 10.1126/science.1127647. eprint: https://science.sciencemag.org/content/313/5786/504.full.pdf. [Online]. Available: https://science.sciencemag.org/content/313/5786/504. 7

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735. 14, 17

[17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," *CoRR*, vol. abs/1612.01925, 2016. arXiv: 1612.01925. [Online]. Available: http://arxiv.org/abs/1612.01925. 9

[18] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *CoRR*, vol. abs/1611.05397, 2016. arXiv: 1611.05397. [Online]. Available: http://arxiv.org/abs/1611.05397. 13

[19] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu koray, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 2017–2025. [Online]. Available: http://papers.nips.cc/paper/5854-spatial-transformer-networks.pdf. 19–22

[20] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super slomo: High quality estimation of multiple intermediate frames for video interpolation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018. 13

[21] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," *CoRR*, vol. abs/1610.00527, 2016. arXiv: 1610 . 00527. [Online]. Available: http://arxiv.org/abs/1610.00527. ii, 12, 23, 24, 28, 31, 49

[22] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG]. 31

[23] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2014. 7, 11

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257. 1, 4, 5, 11

[25] J. Langston, *Microsoft announces new supercomputer, lays out vision for future ai work*, May 2020. [Online]. Available: https://blogs.microsoft.com/ai/openai-azure-supercomputer/. 4

[26] 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, Nov. 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791. 1, 5, 8, 16, 34, 40, 50

[27] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, Nov. 2015, pp. 730–734. DOI: 10.1109/ACPR.2015.7486599. 1, 4, 5, 11, 28

[28] W. S. McCulloch and W. Pitts, "Logical calculus of the ideas immanent in nervous activity.," in *Bulletin of Mathematical Biophysics 5*, 1943, pp. 115–133. DOI: 10.1007/BF02478259. 4

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602. 1, 4, 5, 13

[30] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016. arXiv: 1609.03499. [Online]. Available: http://arxiv.org/abs/1609.03499. 4, 5

[31] A. V. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, Jun. 2016, pp. 1747–1756. [Online]. Available: http://proceedings.mlr.press/v48/oord16.html. 8, 9, 23–27, 32

[32] V. Patraucean, A. Handa, and R. Cipolla, "Spatio-temporal video autoencoder with differentiable memory," *CoRR*, vol. abs/1511.06309, 2015. arXiv: 1511.06309. [Online]. Available: http://arxiv.org/abs/1511.06309. ii, 12, 19–21, 23, 31, 34,

[33] D. J. Rezende, S. Mohamed, and D. Wierstra, *Stochastic backpropagation and approximate inference in deep generative models*, 2014. arXiv: 1401.4082 [stat.ML]. 7, 11, 15

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986. 13

[35] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 802–810. [Online]. Available: http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf. ii, 12, 15, 18, 19, 23, 30,

[36] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 843–852. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045118.3045209. 12–14, 16–18, 33–35, 49,

[37] T. Tieleman and G. Hinton, *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning, 2012. 28

[38] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof, "Anisotropic huber-l1 optical flow," vol. 1, Jan. 2009. DOI: 10.5244/C.23.108. 23

[39] T. Xue, J. Wu, K. L. Bouman, and W. T. Freeman, "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks," in *Advances In Neural Information Processing Systems*, 2016. 12

[40] ——, "Visual dynamics: Stochastic future generation via layered cross convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 41, no. 9, pp. 2236–2250, 2019. 12

[41] J. J. Yu, A. W. Harley, and K. G. Derpanis, "Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness," *CoRR*, vol. abs/1608.05842, 2016. arXiv: 1608.05842. [Online]. Available: http://arxiv.org/abs/1608.05842. 9, 11, 12

[42] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. arXiv: 1311.2901. [Online]. Available: http://arxiv.org/abs/1311.2901. 1, 6