Neural Networks for Traffic Speed Predictions in Major City Road Networks

Charles Peters

August 21, 2020

Abstract

This paper presents a machine learning approach for forecasting traffic density in a large road network. Traffic density exhibits both spatial and temporal correlations throughout the network, which can be interpreted as an image or a matrix, where one axis corresponds to the time steps and the other corresponds to different locations in the network. The spatio-temporal matrix can be used as in input for a neural network, which aims to predict a vector containing the future speeds at each location, effectively the next row of this spatio-temporal matrix. Three main types of neural networks are proposed, and their traffic speed predictions are compared.

Contents

1 Introduction

Traffic congestion causes significant time delays to all drivers around the world. More importantly it also results in millions of injuries and deaths [1], takes a huge toll on the global economy, and the fumes released lead to air pollution that damages the environment [2]. While it is almost impossible to eliminate traffic congestion entirely, knowing the density and direction of the traffic flow can allow you to make smarter decisions that will save you time and money, and hopefully lead to overall safer conditions on the roads. Therefore accurate traffic prediction is one of the most important tasks successful transportation management systems aim to carry out. Early methods for traffic flow prediction mainly consist of statistical approaches, such as support vector machines (SVM) [3]. While these techniques have proven to perform effective time series analysis, they fail to capture both the spatial and temporal relationships exhibited in road networks, and therefore perform rather poorly when applied to major cities with the largest road networks.

In recent years, machine learning methods have become one of the most widely used techniques for prediction tasks. Neural networks in particular have proven to be especially promising, and is an area that continues to be extensively researched. Most machine learning models aim to find relationships between variables based on features provided. It is often difficult to know exactly what features to provide in order to obtain optimal prediction accuracy, and so neural networks solve this issue by extracting the features themselves. The basic building block of a neural network is a neuron, based off the biological neurons in the central nervous system. The neuron takes an input and computes an output. The 'axon' between the input and the output consists of hidden layers which apply a function to the weighted sums of the inputs, with the addition of bias terms, to obtain the outputs [4]. The weights and biases are learnt through training, usually by backpropagation [5].

There are several different variations of neural networks, which each have their own advantages and disadvantages. Convolutional Neural Networks (CNNs) have already proven successful when dealing with the spati-temporal features present in road networks [6]. CNNs in particular work very well when dealing with images [7], and so one approach to this traffic prediction task is to represent the traffic as an image, and attempt to use a CNN to extract features from the image. Traffic density can be visualised by 2D spatio-temporal images. One dimension corresponds to the timestamp, the other to each road segment, and at each coordinate is the traffic density represented by different colours. An example of a spatio-temporal image can be

seen in Figure 1.

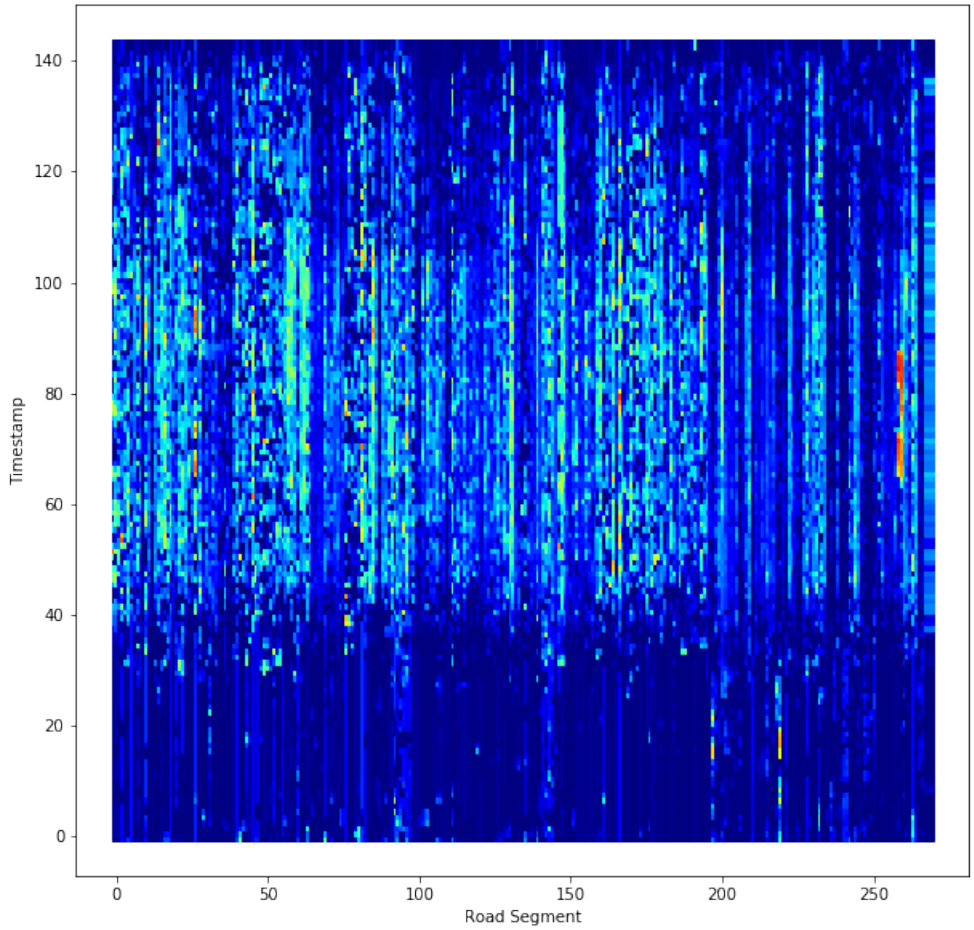


Figure 1: A spatio-temporal image representation of the traffic in Nottingham. The colour at each coordinate corresponds to traffic density, quantified as average speed at that time stamp divided by free flow speed. A darker colour represents lower values for traffic density, whereas the red and orange points are where traffic density is greatest. Note that the order of roads along the x-axis is random, and so adjacency in the image does not necessarily correspond to roads that are located near each other.

Recurrent Neural Networks (RNNs) are another type of neural network that work especially well when dealing with sequential data, making them suitable for time series analysis and similar tasks [4]. Since we are attempting to predict future traffic density based on historical readings, effectively predicting the next term in a sequence, RNNs would also appear to be a suitable choice for our task. Long short-term memory (LSTM) RNNs have already shown success with traffic prediction by considering the task as a time-series forecast [8].

Another option, more recently discovered, are Capsule Networks (CapsNet). One issue with CNNs is that they ignore positional invariance. For example, if we have a CNN that classifies an image as a face, but change the positions of the nose, eyes, ears etc., the CNN should still class

the image as a face. This is due to the pooling layers in CNNs which construct higher order features locally, and in doing so ignore some less significant spatial features that span across the entire image. By replacing the pooling layer in a CNN with dynamic routing, CapsNets are designed to capture the hierarchical spatial features present [22].

In this paper we discuss the effectiveness of different types of networks for traffic prediction. Some of the models used make use of the speed data alone, while others make use of additional information available. All of the models attempt to predict the speed of the cars at future time intervals. The previous speed information is first represented as an $n \times m$ matrix, where n is the number of timestamps we have historical speed data for, and m is the number of road segments in our road network. This matrix effectively contains the same information that is visualised in the spatio-temporal images, such as that in Figure 1. The road network we aim to make predictions for is Nottingham. A map of the road network is shown in Figure 2.

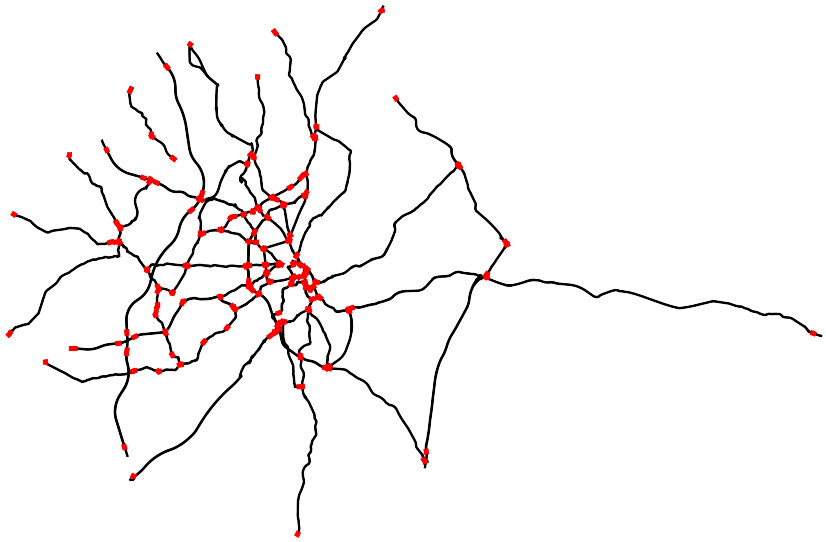


Figure 2: A map of the road network within a 10km radius of Nottingham's centre. The red dots mark the specific locations of each road segment where the speed readings are taken.

2 Data Collection

Real-time traffic speed data was collected from HERE [10]. A script was used to store XML files containing the relevant data every 10 minutes. The data collected contains information on the free flow speeds of each road, both the capped and uncapped speed readings (the capped speed readings are set to the speed limit for instances where the recorded average speed is higher than

the legal speed limit), jamming factor (an arbitrary value quantifying the quality of travel on the road segment at that particular time), and a confidence level of the speed reading. For our models, the only data we are interested in are the free flow speeds and uncapped speed readings. The uncapped speed readings are we what train our model on, and in turn attempt to predict, while the free flow speeds are used to determine traffic density – as traffic density in this case is quantified as uncapped speed divided by free flow speed. The confidence was assumed to be 1 (we can assume all speeds readings were recorded with 100% accuracy) when making our predictions.

In order to understand the spatio-temporal relationships within the road network, the data is converted into a matrix where the number of columns corresponds to the number of road segments, and the number of rows corresponds the number or time steps. A representation of the matrix is shown below:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1269} \\ x_{21} & x_{22} & \cdots & x_{2269} \\ \vdots & \vdots & \ddots & \vdots \\ x_{23041} & x_{23042} & \cdots & x_{2304269} \end{bmatrix} \tag{1}$$

The dimensions are 2304 $\times$ 269 because there are 269 roads in our road network and the data used in our training set is from 16 consecutive days – the data has 10 minute intervals meaning there are 2304 time steps within this period.

This matrix could be converted to the spatio-temporal images similar to the one shown in Figure 1, and a CNN can be used to extract features from the images. However, this is not necessary, instead the networks we used are trained on sequences in this matrix and aim to output a 1$\times$ 269 vector which contains the next speed reading for each road. Effectively, we are just aiming to predict the next term in a sequence. Therefore we must split this matrix into appropriate sequences. How we split our matrix depends on how much historical data we want to train our models on. For real applications it is often more beneficial to build predictions with minimal amount of data, however having more historical data to look at allows the network to map more features and will more than likely lead to more accurate predictions. Also as traffic congestion generally follows daily cycles, having access to data from at least the previous 24 hours allows our network to get a better understanding of what current time it is when making its prediction. In reality traffic follows much longer cycles than 24 hours – lower traffic density on holidays such as Christmas day for example – however, it is not viable to have sequences

these long for our network as the number of training parameters would be far too large. As a result two different sequence lengths were used in testing: 36 time steps (equal to 6 hours) and 288 time steps (equal to 2 days).

3 Traffic Speed Prediction

3.1 Convolutional Neural Network

CNNs have demonstrated on countless occasions their incredible ability to understand features in images, by applying 2D filters and using pooling layers to reduce the overall parameter size of data flowing into the network. However, for our predictions we have used 1D convolutional layers. While 2D CNNs have been proven to be useful in extracting features from spatio-temporal images, 1D convolutional layers specialise in dealing with time series analysis tasks and generally have lower computational complexity [11]. As a result, predicting the change in traffic density is a task well suited to the applications of 1D CNNs.

While all CNNs aim to apply convolutional filters and pool together the extracted features, there are an infinite number of combinations of layers and filters that can be integrated together to form a CNN. Finding the optimal architecture for your CNN can be a difficult task, and so it is often useful to research CNNs that have been used for similar tasks.

Initially we constructed a very simple CNN with just one convolutional layer and one pooling layer. The parameters of this CNN are displayed in Table 1. The architecture is displayed in Figure 3.

Layer	**Parameter**	**Activation**
Convolutional	(64, 2)	ReLU
Pooling	2	-
Flattening	-	-
Fully-connected	-	-

Table 1: Parameters of our most simple CNN.

The convolutional layer features 64 filters of size 2. Smaller filters will capture more local information, but will fail to capture features spanning the entire matrix. Using larger kernel sizes, however, will often significantly increase the training time. This simple model acts as a baseline for our predictions and so we are not concerned about choosing kernels that will lead

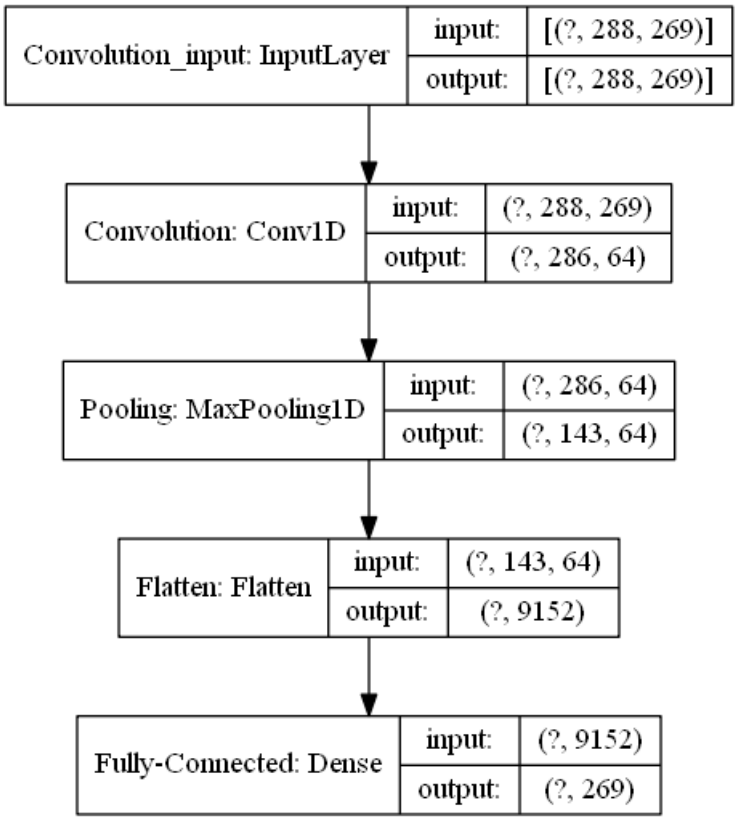


Figure 3: The layers of our initial CNN with their respective parameters. The '?' in the first dimension of each layer corresponds to the batch size which is specified in training. In this diagram the second parameter for the input layer is 288, this is because in the example displayed here the sequence length was chosen to be 288 time steps. This dimension varies depending on the sequence length, and in turn affects parameters in subsequent layers.

to the most accurate results. We choose a small kernel size in order to quickly train our model over a sufficient number of epochs.

Another important feature of CNN architecture are activation functions. Convolutional operations are non-linear, and so we introduce activation functions to add a non-linear property to the function. Without the activation function, the CNN is unable to learn the complex spatio-temporal features present in a road network. The activation function used on the convolutional layer here is a rectified linear unit (ReLU). Calculating the gradients of ReLUs is much easier than most other activation functions, such as tanh, and as a result networks using ReLU train significantly faster [12].

Neurons in a CNN consist of weights and biases, and the values of the weights and biases heavily influence the ability of our network to perform its specific task. The process of finding the optimal values for the weights and biases is known as optimisation. The majority of optimisation

techniques are variations of gradient descent [13]. The optimiser used for our network, and perhaps the most commonly used optimiser in CNNs today, is Adam [14]. Adam builds upon other gradient descent methods by utilising the first and second order moments of the gradients, and in general has faster convergence than similar optimisation methods such as stochastic gradient descent and RMSProp.

Similarly to the convolutional layer, the pooling layer has filters of size 2. The output of each pooling filter is the maximum number within a 1×2 region, hence why the second dimension of the outputs of this layer is half that of the inputs.

The pooling layer is followed by a flattening layer which converts the outputs from the pooling layer into a vector. This vector acts as the input for the fully connected layer.

The network ends in a fully connected layer which maps the extracted features to a predicted speed. The output is a 1×269 vector containing the predicted speed on each road for the next time step in the sequence.

While our baseline CNN has each type of component necessary for a complete network, it is often beneficial to have multiple types of each layer in order to form a hierarchical feature extraction process [4].

The main structure of our deeper neural networks is inspired by an architecture developed in [15]. While the network here takes spatio-temporal images as its inputs and uses 2D convolutional layers, the spatio-temporal features present in the images should be very similar to those in the matrix we use as inputs, and thus this architecture should also perform well for our task. The paramaters of this network are displayed in Table 2, and the architecture is shown in Figure 4.

Our previous network features one fully-connected layer at the end with 269 outputs corresponding to each road segment. In our adapted deeper network, we include an additional fully-connnected layer before the final one which has 500 outputs. This additional fully-connected layer is used for the same reason that nearly all multi-layer perceptrons or feed-forward networks use multiple layers [16]. By using more fully-connected layers, we are applying an additional function to the features extracted from our convolutional layers.

Our convolutional layers all feature kernels of size 3, followed by pooling layers of size 2. While there are several options for these two parameters, some of the most successful networks developed use these same parameters [17]. Our convolutional layers have 256, 128, and 64 channels respectively, and all feature ReLU activation functions.

Layer	Parameter	Activation
Convolutional	(256, 3)	ReLU
Pooling	2	-
Convolutional	(128, 3)	ReLU
Pooling	2	-
Convolutional	(64, 3)	ReLU
Pooling	2	-
Flattening	-	-
Fully-connected	-	-
Fully-connected	-	-

Table 2: Parameters of our second CNN. This network is almost identical to the network developed in [15] with the main differences being the use of 1D convolutional layers rather than 2D, and an additional fully connected layer at the end of the network.

While the deep network architecture proposed above may prove to be very effective in extracting features from the speed data alone, it may still struggle to make accurate predictions due to having access to no other information. While traffic most likely follows daily cycles, there are many other factors that will likely affect how traffic one day is going to differ from traffic on another day. Unfortunately our previous networks wont be able to address this issue, as their only inputs are the previous speed readings. Therefore by including extra information in our network, and altering the inputs, we are likely to increase the accuracy of our predictions.

We initially decide to include information regarding what day of the week we are attempting to make predictions for. An extra input is included in the network which is a $n \times 1$ vector, where n is equal to the number of time steps in our traffic history, and each element of the vector is an integer from 0-6 corresponding to the day of the week at each time step. As the day of the week only changes one every 24 hours, and not every 10 minutes which is the interval between our time steps, each integer in our vector of weekdays appears in groups of 144.

We apply this method of using both the speed and weekday inputs to both of our initially proposed networks. The architecture of our deeper network when adapted to take both inputs is displayed in Figure 4.

The features are extracted from each input separately, and then concatenated. A flattening layer is applied to concatenated features, which transforms the features into a vector to be

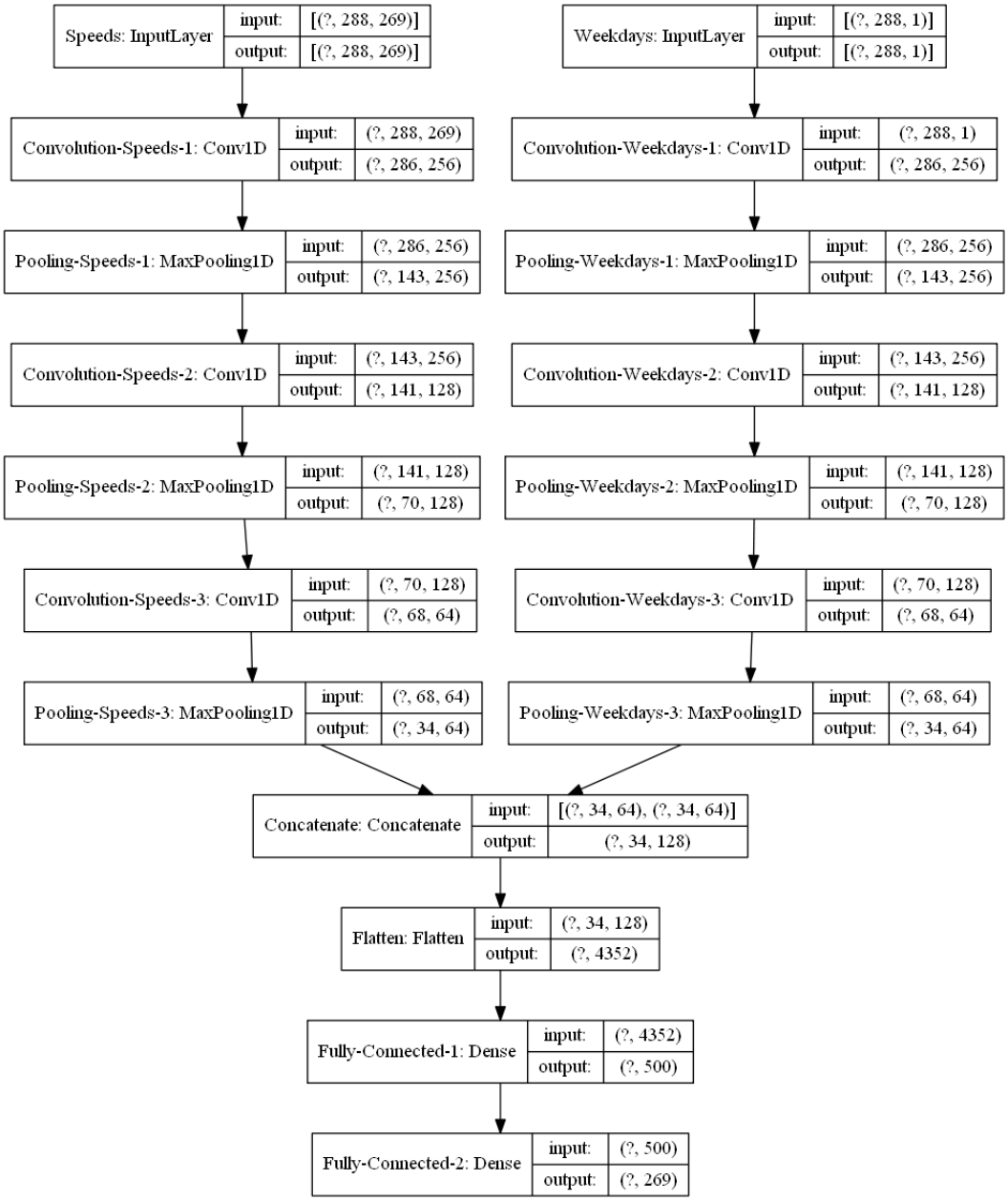


Figure 4: The architecture of our deeper CNN with an extra channel for a second input that contains information with the day of the week at each time step. This example shown is for a network that takes sequence lengths of 288 as its inputs.

passed through a fully-connected layer(s) where the speed predictions are outputted.

The network architecture before the concatenate layer is symmetrical. This is done so that the dimensions of the outputs from the final pooling layers are the same, which is necessary for the features to be concatenated.

We can even take this process a step further and concatenate together three networks. For our third network we include information on the exact time of day. While we should expect the

network to roughly determine what time of day it is by extracting relevant features from the sequence of speeds, there may be certain points of the day where the sequence does not follow such a distinct pattern. Therefore the network may sometimes misunderstand which point of the day the traffic history lies at, and this will severely affect its future predictions. This will be especially true when the history is of a shorter time period. The time information is included in the same form as the weekdays information; an $n \times 1$ vector, where n is equal to the number of time steps in our traffic history, and each element of the vector is an integer from 0-143 corresponding to the time of the day. The values range from 0-143 because there are 144 ten minute intervals within a 24 hour period.

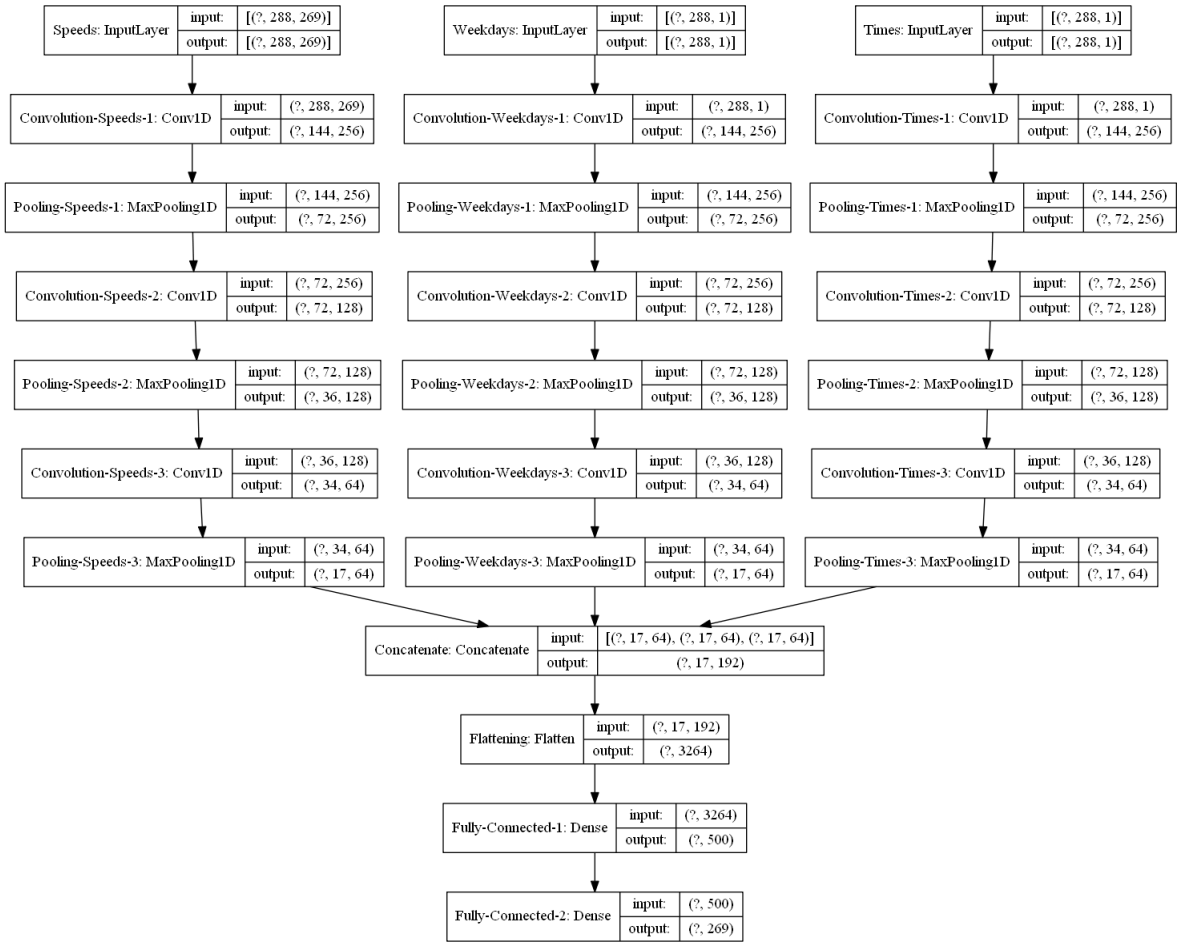


Figure 5: The architecture of our most complex CNN, which takes three inputs. The same three convolutional layers are applied to each input, and their extracted features are concatenated together to eventually produce a 1×269 vector of speed predictions for each road. This example is also shown for a network that takes sequence lengths of 288 as its inputs.

As before, we apply this method to both our baseline and deep networks. When applied to our deep network we obtain the architecture shown in Figure 5, which is our most complex

neural network developed for our traffic prediction task. For this model the parameter size is very large, and so we include strides of 2 on the first convolution layers in order to speed up the training times. Padding is also included to prevent the dimension size dropping too much from the addition of strides.

3.2 Recurrent Neural Network

RNNs are a type of feed-forward networks adapted to deal with sequential data, which our traffic data is a perfect example of. Like CNNs, RNNs have also proved that, when optimised, they too are able to make accurate predictions on traffic speeds [18].

RNNs in their most basic form suffer from short term memory issues, so instead we attempt to make our traffic predictions using an adapted RNN called an Long Short Term Memory (LSTM) network [19]. It is especially important that we retain memory over a longer sequence for our task because we expect the traffic to follow daily cycles; where one full day is equivalent to a sequence length of 144.

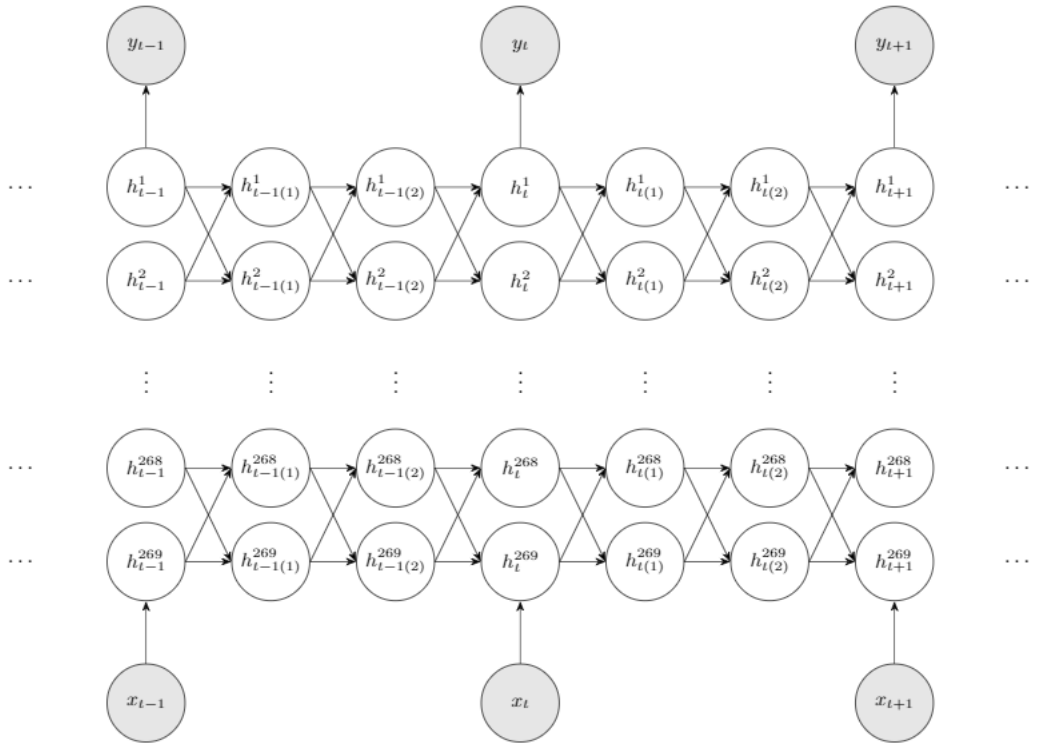


Figure 6: The architecture of our RNN with 2 LSTM layers. There are 269 memory units in both hidden layers for each time step, where each individual unit is connected to every unit in the subsequent layer. The memory units retain information from all previous time steps.

As with developing a CNN, it is often beneficial to seek previous networks used for similar

tasks to find the optimal architecture for an RNN. The structure of our RNNs is inspired from the models developed in [20]. Each road segment in the network is treated as a memory unit, and therefore the LSTM layers in our RNNs contain 269 memory units. Three different RNNs were developed and tested on with 2, 4 and 6 LSTM layers respectively. However, the individual layers in each RNN are identical, containing the same number of memory units (269). Each network ends in a fully-connected layer which maps the outputs of the final LSTM layer to a 1×269 vector of speed predictions for each road in the next time step. At each LSTM layer we have included a dropout factor of 0.2. This means that 20% of the outputs at each layer (randomly selected) are set to zero. Dropout is a technique introduced to prevent overfitting [21]. The full structure of our RNN is illustrated in Figure 6.

3.3 Capsule Neural Network

While CNNs have continued to outperform our expectations in a wide variety of applications, like any other machine learning model they are not without their faults. The purpose of the pooling layers in CNNs is to extract the most significant features locally, and ignore any less significant features. As a result, valuable information is lost by only pulling out the neuron with the highest activation. Capsule networks (CapsNets) are a modern approach designed to deal with this issue [22].

CapsNets build upon CNNs by adding structure called capsules. These capsules replace the pooling layers in regular CNNs, and output a vector corresponding to the probability of a certain feature being detected. Each capsule layer in a CapsNet contains many capsules, which each correspond to a different feature in the input object. A key difference between capsule layers and pooling layers is that the capsule layer's activation outputs a vector, whereas a pooling layer applies a scalar operation; the output is simply the greatest number in a certain sized region. While CNNs are trained by backpropagation, CapsNets are trained by a process called dynamic routing [22].

As with any network type, there are infinite possibilities of architectures to choose from for a CapsNet. One problem CapsNets suffer with even more so than CNNs is parameter size. The parameter size grows significantly when replacing pooling layers with capsule layers, and the more capsules in each layer, the larger the parameter size. Therefore it is especially important for CapsNet to take measure to reduce the parameter size while keep predictions as accurate as possible.

We start with a basic structure which is very similar to our initial CNN. The architecture of our basic CapsNet is displayed below in Figure 7.

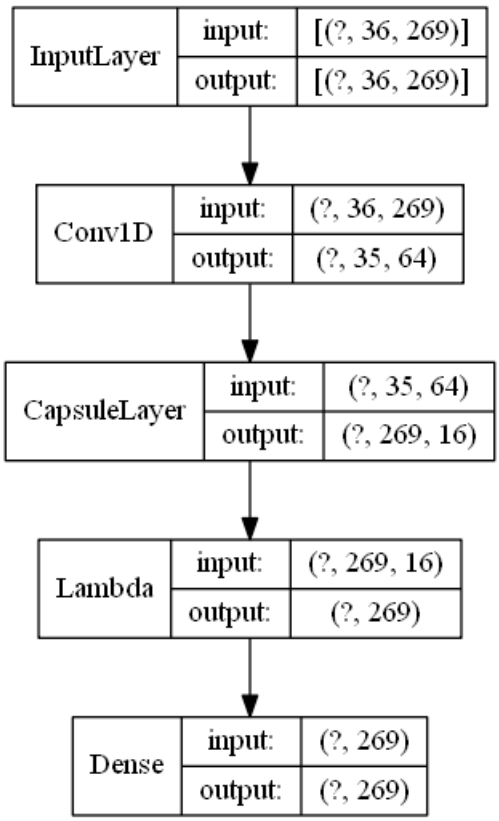


Figure 7: The architecture of our simple CapsNet, with the example shown designed to take inputs of 36 time steps. The Lambda layer is a special type of layer used in tensorflow [23] to bridge our custom made capsule layer to the fully-connected layer.

The structure follows on from our initial CNN. It features a single convolutional layer with 64 kernels of size 2. Here however, we have also included strides of 1, as the parameter size is much larger now we have a capsule layer. The pooling layer and flattening layers are removed entirely and replaced by a capsule layer. The capsule layer features 269 capsules (1 capsule per road segment), with a dimension of 16. As with any other network, the final layer is a fully-connected layer, which in our case maps the outputs of the capsule layer to a speed prediction for each road.

If we wish to gain more accurate predictions from a CapsNet, we must develop a deeper network. The architecture of our more complex CapsNet is inspired by a CapsNet that was developed to learn the spatio-temporal traffic interactions within the road network in Santander city of Spain [24]. The full architecture can be seen in Figure 8.

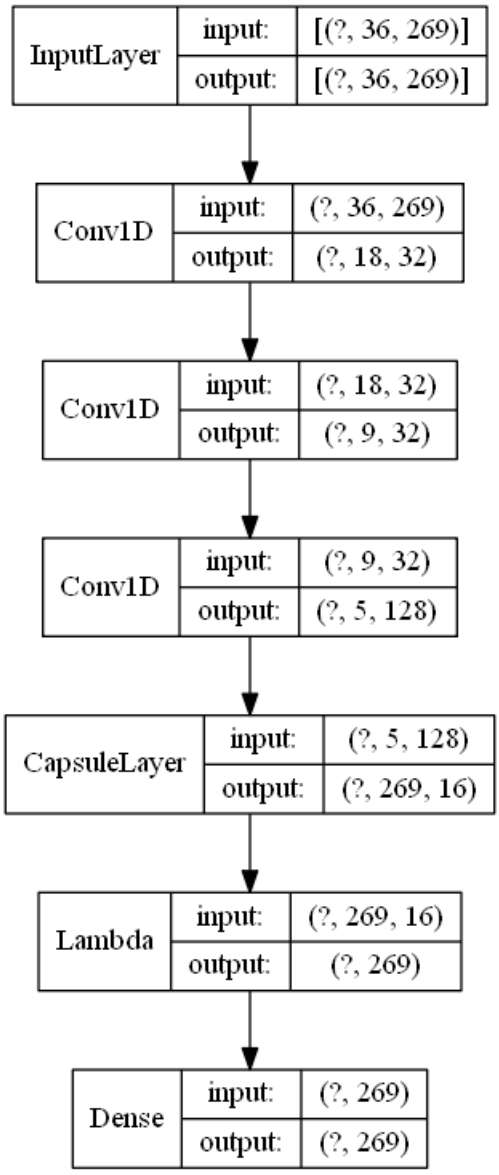


Figure 8: The architecture of our deeper CapsNet, heavily influenced by the architecture proposed in [24]. This network takes inputs of size 36×269 (36 time steps and 269 road segments).

4 Performance Validation

All the models were trained on data collected from 16 consecutive days between 00:00 on the 24$^{\text{th}}$ June to 00:00 on the 10$^{\text{th}}$ July in the year 2020. A weeks worth of data between the 12$^{\text{th}}$ 19$^{\text{th}}$ of July was used for testing the validation of our models. The majority of our models were trained and tested on two different sequence lengths:

- A shorter sequence of 36 time steps (equal to 6 hours was used). While in theory we could of attempted to use a much shorter history to train our networks on, some of the more complex networks, such as the three layer CNNs, reduce the dimension space by too much

(even when including padding) to allow a smaller input.

- A longer sequence length of 288 time steps (equivalent to 2 full days).

A validation set is pulled from the training set which is used during training to prevent overfitting.

For both sequence lengths we aim to predict speed readings for the next 144 time steps. Essentially we wish to predict the entire traffic sequence for the next 24 hours. Our networks achieve this by predicting speed readings for the next 10 minutes and then add this prediction on to their history in order to predict the speed readings in 20 minutes time. This cycle continues another 143 times until it has predicted the speed readings for exactly 24 hours in the future. This means that the 'history' used to predict readings more than 10 minutes in the future is not entirely historical traffic data, and is rather a mixture of true historical data and predicted history. As a result we should expect our networks to perform better at predicting traffic in the near future, where it has more true historical data in its inputs, and perform significantly worse when predicting evolution of traffic on a longer time scale, when its inputs will mainly consist of previous predictions.

We quantify the performance of our models in terms of mean square error (MSE). We calculate an overall mean square error for predictions on the next 24 hours, but we also consider the mean square error at various other time scales. This allows us to compare what time scales different models work more effectively on.

Before testing our models we construct a mean-average spatio-temporal image, displayed in Figure 7. This image displays the mean traffic density on each road at each 10 minute time interval. The overall MSE of this matrix of means is 23.6 (root mean square error (RMSE) of 4.86). This means when using the average speed values as predictions for future values, we can expect to be about ony 5 km/h off the true values. The relative mean square error of this matrix is 0.0270 (2.7%).The relative MSE is calculates as follows:

$$Relative\ MSE = \frac{\sum_{n=1}^{N} (\frac{y_n - \hat{y_n}}{y_n})^2}{N} \tag{2}$$

The relative MSE will often be more meaningful for our prediction validation as the range of speed readings is very high (from nearly 0 km/h up to over 100 km/h), and so we can not expect the same absolute mean square errors when speed readings are particularly high. One problem with the relative mean square error is that for some instances the speed recordings are missing,

and their values are taken to be zero. As division by a speed reading of zero is not possible, the relative error can not be calculated for that instance. We get around this by ignoring this instance from our sum of relative errors, and reducing N by 1 when calculating the relative MSE. Interestingly, and most likely due to this issue with missing data points, some models perform better in terms of regular MSE, while others perform better in terms of relative MSE. Another reason why some models may preform better in terms of regular MSE is because they may perform worse at predicting speeds when traffic density is highest (where speed readings are lowest), which will take a larger toll on the relative MSE than the regular MSE. For these reasons we also consider the regular mean square error when comparing all our models.

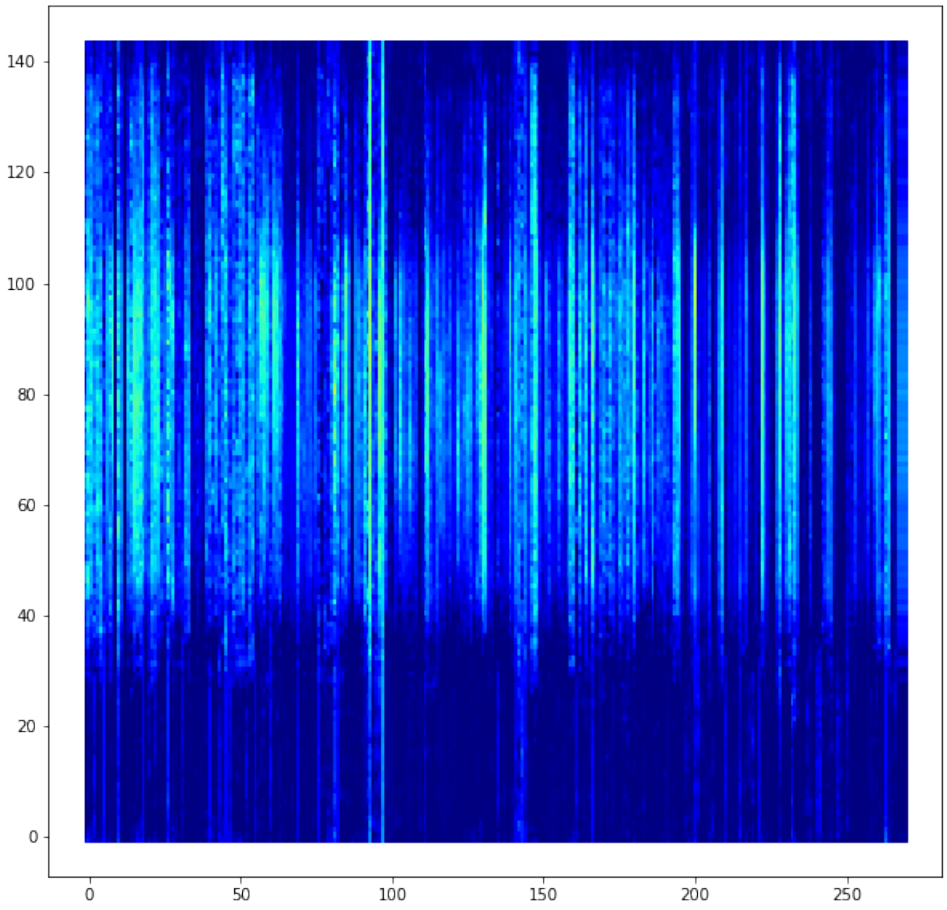


Figure 9: A spatio-temporal image to capture the mean traffic density on each road at each 10 minute interval of the day.

While we expect our networks to have poorer predictions over larger time scales, that is not the case for our mean-averaged matrix, which should have similar relative MSEs for predictions in the near and far future. However Table 3 displays how the relative mean square error changes depending on what time of day our prediction is for. We can see that the relative MSE is very

low at the start of the day, from around 00:00 - 8:00, but then increase rapidly and maintains a high value between 0.05 and 0.075 up until 20:00, when it begins to decrease. While the matrix of means has a good overall MSE, when we look deeper we see that it actually performs poorly during the day, and the main reason for its MSE is due to its predictions made during the night. At night, traffic is low, and so the speed will be very close to the free flow speed. As a result, fluctuations in speed are low at these times, and so the speeds will also be very close the average value. During the day when traffic is higher, speeds fluctuate more from the average value, hence why taking the mean value to predict the speed proves to be a poor method.

Time of day	**Relative MSE**
00:00 - 02:00	0.018162
02:00 - 04:00	0.007783
04:00 - 06:00	0.011986
06:00 - 08:00	0.033061
08:00 - 10:00	0.053838
10:00 - 12:00	0.061348
12:00 - 14:00	0.051616
14:00 - 16:00	0.052910
16:00 - 18:00	0.068350
18:00 - 20:00	0.052399
20:00 - 22:00	0.037681
22:00 - 24:00	0.026302

Table 3: Relative mean square errors at different times of the day when using the mean speed values to make predictions.

The reason for calculating the relative MSEs for this mean matrix is so we can determine how well our networks are performing. If our networks perform worse than this trivial method of taking the mean values, we can conclude that it is not an optimal method for predicting traffic flow.

Despite its simplicity though, this method does appear to achieve reasonable prediction accuracy; even at 16:00 - 18:00 where its accuracy is worst, its predictions are on average only about 7% off the true values. This method completely ignores any spatial correlations between the roads, and so the success of this method might indicate that there is very little correlation

at all between the roads. We construct a 269×269 matrix of Pearson correlation coefficients (PCC) at various time lags to determine the correlations between each road. The PCC for two road segments is calculated as follows:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{3}$$

Where X and Y are the vectors containing the history of speed recordings of the two road segments, cov is the covariance, and σ_X and σ_Y are the standard deviations of vectors X and Y respectively.

We calculate this matrix of PCCs at various time lags. The effect of traffic on one road will, in theory, often lead to traffic on another road, but this is not an instantaneous process. Instead the knock-on effect will be delayed, and so the correlation will not be observed at the same moment in time. The mean PCC at various time lags is displayed in Table 4.

Time lag	**Mean PCC**
No delay	0.350
10 minutes	0.348
20 minutes	0.345
30 minutes	0.342
40 minutes	0.338
50 minutes	0.334
60 minutes	0.329

Table 4: Mean PCC at various time delays.

From this data we observe that the average PCC between the roads decreases very slightly as time progresses, indicating that most of the correlation between the roads is shared in the present time, rather than the progression of traffic from one road to another over time. Despite this the average correlation is quite similar over the various time lags tested on, with the PCC maintaining around a value of 0.33-0.35. A PCC value in this range indicates there is very slight positive correlation shared between the different road segments. Figure 10 shows the recorded speeds of road segment 1 plotted against the recorded speeds of road segment 2. This graph agrees with the information suggested by our PCC tests; there is a very small amount of positive correlation.

Overall, the correlation is rather negligible, and this likely explains why the mean matrix

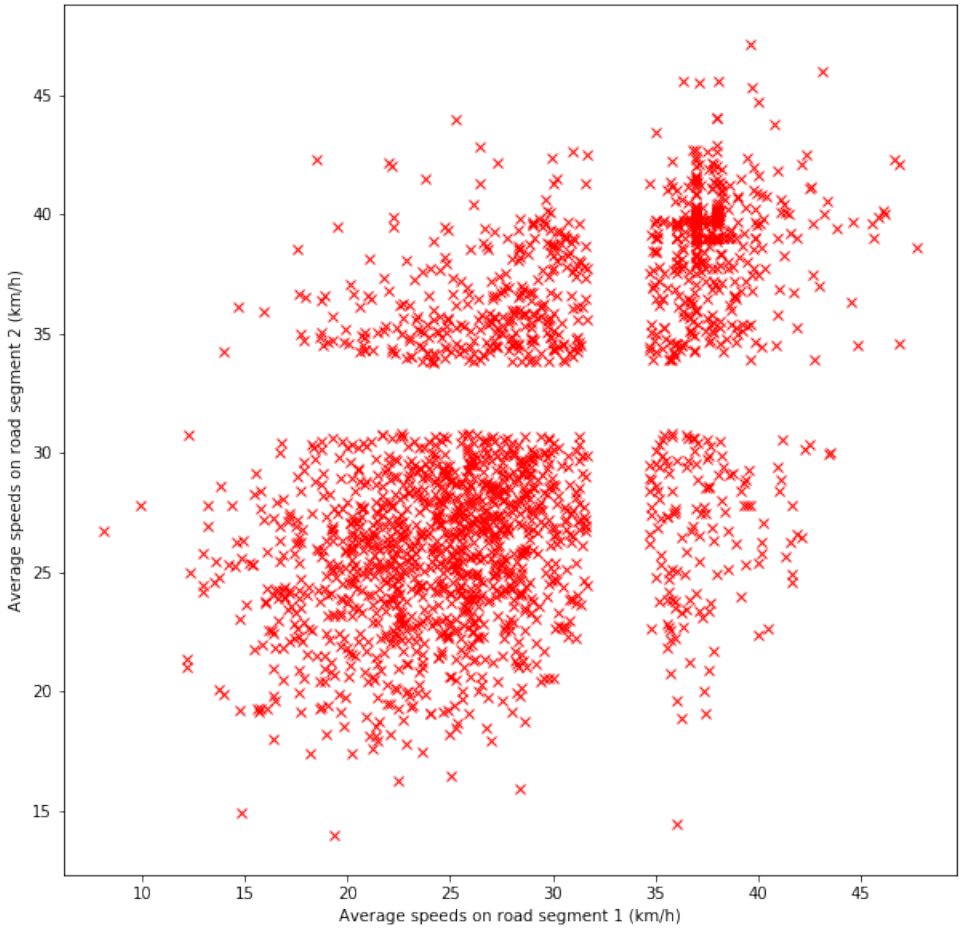


Figure 10: A plot to show the average speeds at different time steps on road segments 1 and 2. This graph suggests there is a very slight positive correlation. Another interesting observation in this graph is the empty bands at around 30-35 km/h. For some inexplicable reason there are no average speeds recorded in this speed range.

makes accurate predictions for the future traffic speeds. It is also unlikely that our networks will extract any significant features regarding the correlation between the roads, and the most noticeable features will be regarding the speed progression on each road from an individual perspective.

Our CNNs were all trained over 50 epochs, using a batch size of 144. The overall MSE and relative MSE of each network is displayed in Table 5.

For both sequence lengths tested on, the deeper models perform better. Adding an extra input channel also leads to a significant decrease in error. The models overall perform very well when using a sequence length of 288, but appear to perform significantly worse when using a sequence length of 36. This problem does however seem to be solved when adding the third input channel containing information on the specific time of the day. It is clear why this information

	Sequence Length			
	36		288	
	MSE	Relative MSE	MSE	Relative MSE
Basic CNN - 1 Input Channel	188.9	0.0815	52.9	0.0350
Deep CNN - 1 Input Channel	65.3	0.0384	28.7	0.0267
Basic CNN - 2 Input Channels	163.7	0.0706	25.9	0.0270
Deep CNN - 2 Input Channels	45.7	0.0427	24.8	0.0280
Basic CNN - 3 Input Channels	49.9	0.0300	24.5	0.0257
Deep CNN - 3 Input Channels	29.8	0.0349	23.0	0.0266

Table 5: Overall MSE and relative MSE of each network using both sequence lengths. All networks are trained on the same training sets and validation sets, over the same number of epochs and batch sizes, and errors are calculated using the same test set.

causes such as significant decrease in MSE, compared to the much lower effect it has on the same model tested on a sequence length of 288. When the sequence length is 288, the network is able to pick up the shifts in traffic over the past two entire days, however when the sequence length is only 36, it can only learn the evolution of traffic within the past 6 hours. Thus without the extra channel that feeds in the exact time of day, it is much harder for the network to predict what time of day it needs to predict the speed for, and in turn heavily impacts the speed it will output.

While it has been observed that all our models achieve greater prediction accuracy when they have more historical traffic data available to influence their prediction (a longer sequence), there are several benefits to using a model that utilises a shorter sequence length. Firstly, when the sequence length is shorter, the inputs for our networks are smaller, and in turn the total number of parameters in our network are much smaller. As a result the networks based around the shorter sequences train significantly faster. Secondly, and perhaps more importantly, less historical data is needed to make a prediction. Our models based around a sequence length of 288 require speed readings from every 10-minute interval within the past 2 days in order to make just a single set of predictions. However our models which utilise a sequence length of 36 only require data from the past 6 hours.

Table 5 displays the overall error for predictions made over the course of a day, but it doesn't indicate how the prediction accuracy varies depending on how far into the future our models

are making their predictions.

Table 6 shows how the error changes depending on how far in the future we are predicting for with our 3-input deep CNN. For our model that utilises a longer sequence length, we can see that predictions made at the 24 hour-mark are almost as accurate as the predictions made for the immediate future. For our network based around a sequence length of 36, we observe that the error decreases significantly as time progresses. However, what is also observed is that the error of initial predictions, made for the speeds in the near future, is comparable to the error observed from our 288 input size network. To summarise, while 6 hours of traffic history is not a sufficient predictor variable for traffic speeds in several hours time, it is sufficient for predicting traffic in the very near future.

	Sequence Length	
	288	**36**
Hours ahead	**Relative MSE**	
0-2	0.025884	0.028667
2-4	0.025752	0.029506
4-6	0.025696	0.029896
6-8	0.025616	0.030878
8-10	0.025500	0.032919
10-12	0.025408	0.034992
12-14	0.025484	0.037007
14-16	0.025524	0.038546
16-18	0.025743	0.040152
18-20	0.025954	0.041629
20-22	0.025989	0.042123
22-24	0.025974	0.042683

Table 6: This table displays how the relative error changes depending on how many hours ahead of the most recent speed recording the network is making its prediction for. The errors here are for the deep CNN with 3 inputs.

Overall our CNNs proved to be a viable method for predicting traffic speed, but the same could not be said for the RNNs. All three proposed RNNs turned out to be poor solutions to the traffic prediction task at hand. If we take their overall mean square error and relative mean

square error, we may be mislead into thinking they perform well. The three RNNs tested all scored mean square errors below 40 which is significantly better than the performance of some of the CNNs. However if we look at the spatio-temporal image representation of their predictions (Figure 11), we can clearly see why they are not suitable models for this task. Regardless of the time of day, the RNN will always output the same vector of speeds. Its prediction is effectively the average speeds for each road segment. Surprisingly, however, this results in a somewhat respectable MSE. As the RNN is trained in such a way to minimise the MSE, it is likely that it found predicting the speeds in this manner gave the lowest error.

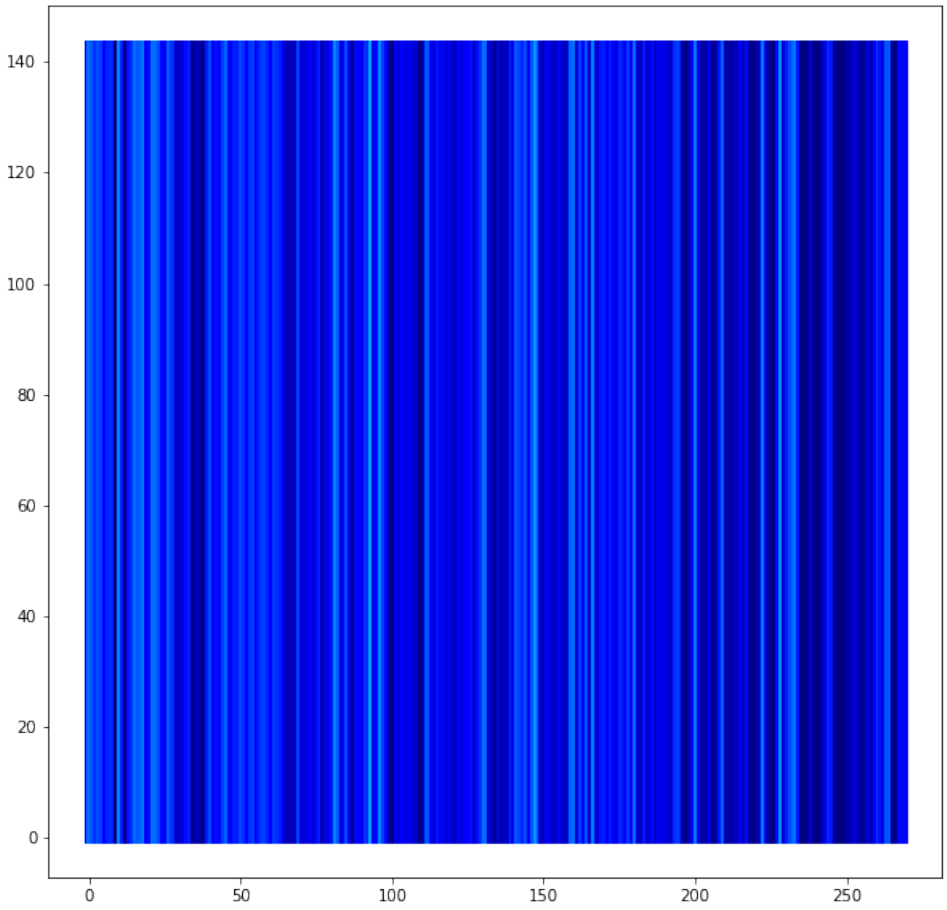


Figure 11: A spatio-temporal image of the predictions from our 2-layer RNN over a 24-hour period. We can see here that the RNN predicts the same set of speeds at every time step.

Due to the parameter size of the CapsNets, we could not use an input sequence of 288 time steps. Instead we only tested these networks on a sequence length of 36. As with the CNNs, both CapsNets are trained over 50 epochs, using a batch size of 144. The overall MSE and relative MSE for 24 hour predictions are presented in Table 7.

From the information in this table alone, we can see that the CapsNets perform relatively

	MSE	Relative MSE
Basic CapsNet	73.2	0.0461
Deep CapsNet	55.0	0.0556

Table 7: Overall MSE and relative MSE for both CapsNet models. Both models use the previous 6 hours worth of traffic data to make their predictions.

poorly when making predictions for the next 24 hours, with errors comparable to the deep CNN with 1 input channel and the same input size (36×269). However, like the CNNs used on this sequence length, while they perform poorly at predicting traffic speeds over a longer time span, they are much better at predicting traffic in the near future. Table 8 displays the error values according to how far ahead the networks are predicting.

	Network	
	Basic CapsNet	**Deep CapsNet**
Hours ahead	**Relative MSE**	
0-2	0.045970	0.029380
2-4	0.057609	0.032514
4-6	0.073825	0.037069
6-8	0.094395	0.045340
8-10	0.117584	0.056410
10-12	0.141777	0.068731
12-14	0.166597	0.077866
14-16	0.190754	0.079022
16-18	0.214604	0.074088
18-20	0.240320	0.065210
20-22	0.240320	0.053332
22-24	0.311183	0.041706

Table 8: This table displays how the relative error changes depending on how many hours ahead of the most recent speed recording the network is making its prediction for. The errors here are for the two CapsNet models.

The basic CapsNet performs rather lackluster compare to the other networks, but the deep CapsNet performs especially well for the first few hours. What else is interesting is that the

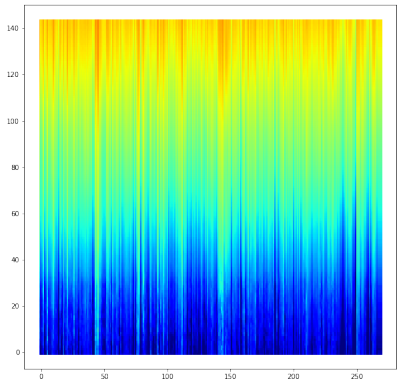
deep CapsNets error peaks at around 14-16 hours ahead and then decreases. This may be because the traffic generally follows a daily cycle, and close to 24 hours ahead, the traffic will closely mimic the traffic that occurred 24 hours ago, which is the original inputs of our network. However despite the model performing best within the first 2 hours it still performs worse than our CNNs, and the relative error is still lower than what we would achieve overall using the mean value to make our predictions. Yet there is one more area where the deep CapsNet holds an advantage over the other models. Previously we have only considered the effectiveness of our models for predicting traffic over the next few hours or next entire day, but what about on a even shorter time scale? Our networks make predictions for 10 minute intervals and use these predictions as inputs for subsequent predictions. Therefore it is only this first outputted prediction which actually receives an input consisting entirely of true historical traffic data. Table 9 contains the errors of our most successful networks on this initial 10 minute prediction.

	MSE	**Relative MSE**
Deep CNN - 1 input	25.4	0232
Deep CNN - 2 input	26.6	0.0216
Deep CNN - 3 input	25.1	0.209
Deep CapsNet	23.9	0.0207

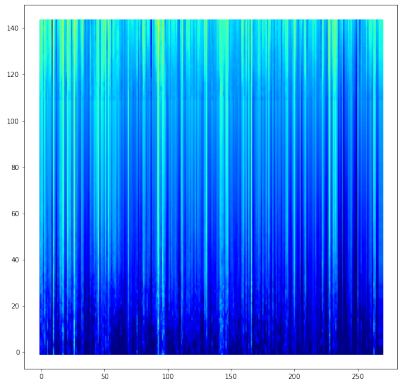
Table 9: MSE and relative MSE of the prediction of traffic speeds in the next 10 minutes for the top four performing networks.

Here we see that the deep CapsNet outperforms all other models for its initial prediction. This suggests that the CapsNet will be the optimal model for predicting traffic speeds if it is constantly updated with new data, and it is only necessary to predict traffic speed for the next singular time step. However, it lacks the flexibility of the CNNs, and is not as successful in using its initial predictions to make subsequent predictions for further time steps.

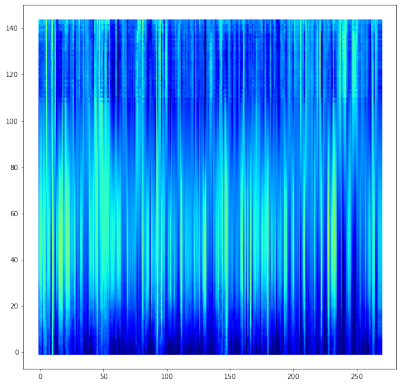
Pictured below is each model's spatio-temporal image representation of its traffic density predictions made for the 26^{th} June.



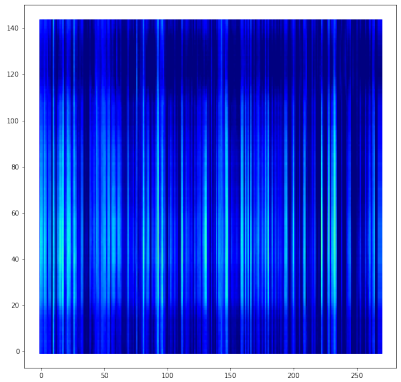
(a) Basic CNN with 1 input channel for speed data



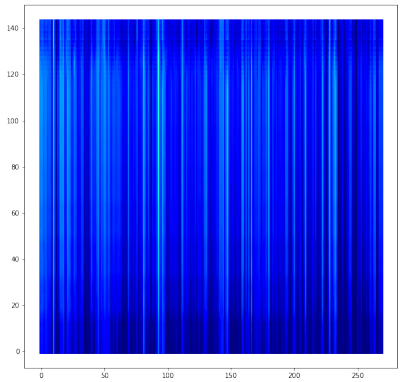
(b) Basic CNN with 2 input channels – speed data and day of the week



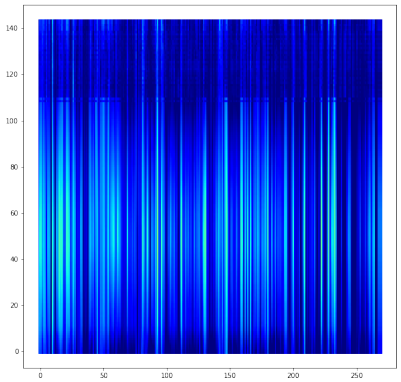
(c) Basic CNN with 3 input channels – speed data, day of the week, and time of day



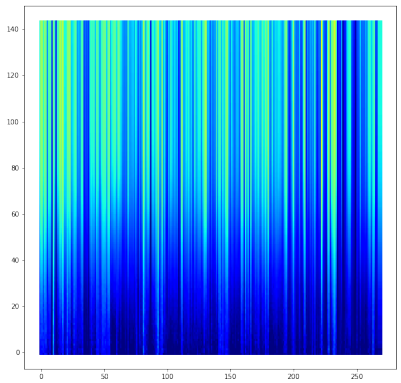
(d) Deep CNN with 1 input channel for speed data



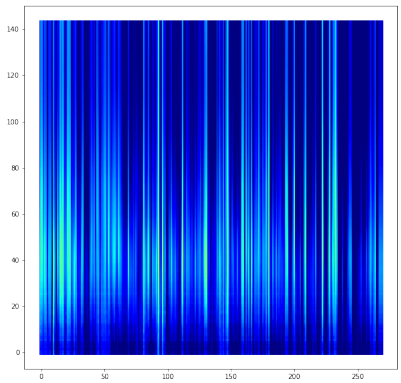
(e) Deep CNN with 2 input channels – speed data and day of the week



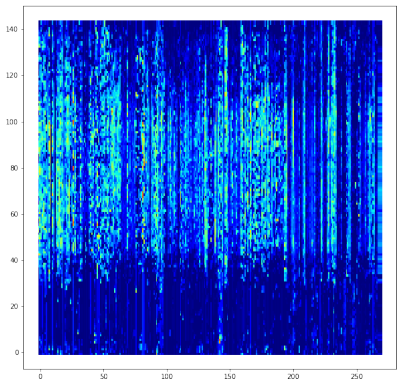
(f) Deep CNN with 3 input channels – speed data, day of the week, and time of day



(g) Basic CapsNet

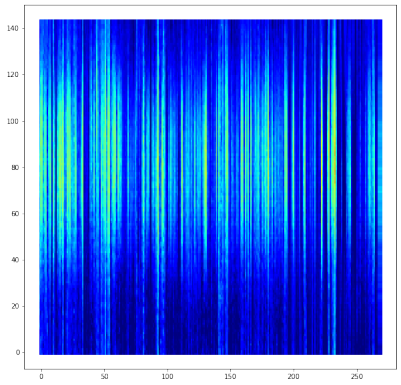


(h) Deep CapsNet

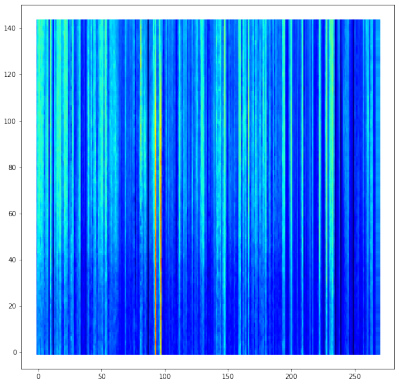


(i) True traffic density

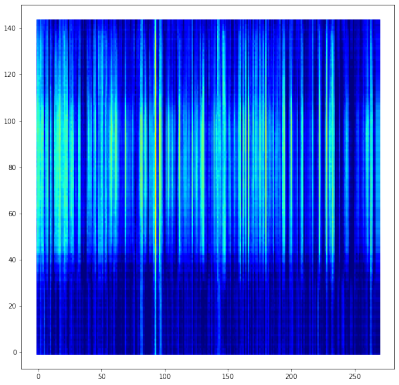
Figure 12: These images are produced from the predictions of our models which take inputs equal to the previous 6 hours worth of traffic data (36 time steps). The true traffic data is also pictured for comparison.



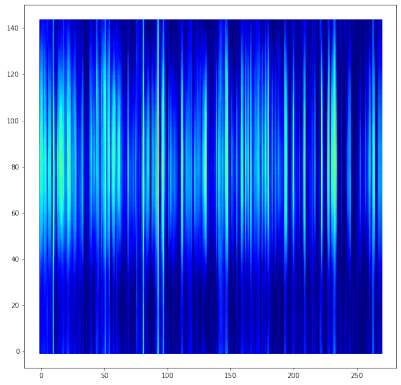
(a) Basic CNN with 1 input channel for speed data



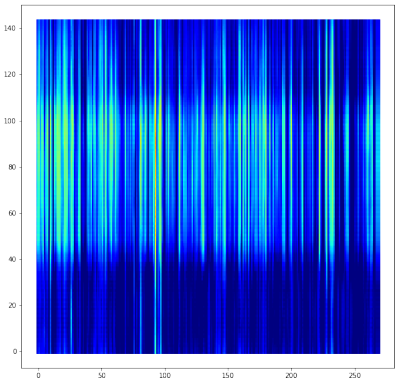
(b) Basic CNN with 2 input channels – speed data and day of the week



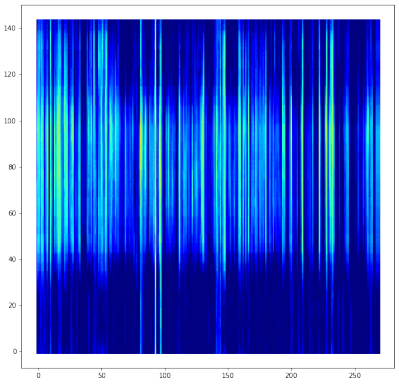
(c) Basic CNN with 3 input channels – speed data, day of the week, and time of day



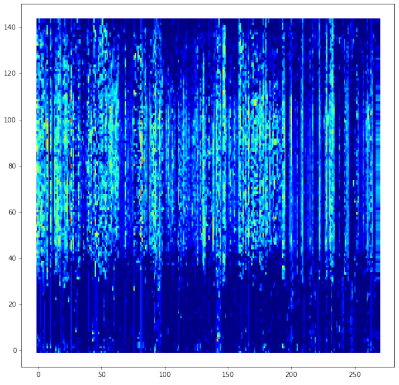
(d) Deep CNN with 1 input channel for speed data



(e) Deep CNN with 2 input channels – speed data and day of the week



(f) Deep CNN with 3 input channels – speed data, day of the week, and time of day



(g) True traffic density

Figure 13: These images are produced from the predictions of our models which take inputs equal to the previous 2 days worth of traffic data (288 time steps). The true traffic data is also pictured for comparison.

5 Conclusion

We have investigated the use of neural networks to predict the progression of traffic density in the road network surrounding a large city. This paper presents several different architectures of 3 main types of networks; convolutional neural networks, recurrent neural networks, and capsule networks. While RNNs, specifically LSTMs, proved to be a poor choice for predicting traffic speeds, the other two networks proved to be very effective when an optimal architecture is selected. CNNs are a flexible method, offering good prediction accuracy while maintaining a relatively fast training process. CapsNets proved to be almost as effective, if not more effective, at predicting traffic on short time scales, but their large parameter size means their training process is computationally much more demanding. As a result of this, it is not viable to build them around larger input sizes. The CNNs, on the other hand, can be designed to take either shorter or longer historical traffic data sequences as their inputs. Models built around a longer input size take longer to train, but gain a better picture of the daily temporal cycles present, and so perform much better at predicting traffic over the course of the next 24 hours. It is possible to incorporate additional input channels into our CNN containing additional information, such as the time of the day and what weekday it is. This additional information increases the parameter size, but also leads to an increase in prediction accuracy.

References

[1] World Health Organization Road Traffic Injuries, 2020

[2] E. Hatri et al. "Traffic management model for vehicle re-routing and traffic light control based on Multi-Objective Particle Swarm Optimization" (2017)

[3] C. Wu et al. "Travel-Time Prediction With Support Vector Regression" (2004)

[4] Y. LeCun et al. "Deep learning" (2015)

[5] P. Werbos "Backpropagation Through Time: What It Does and How to Do It" (1990)

[6] Y. Lv et al. "Traffic Flow Prediction With Big Data: A Deep Learning Approach" (2015)

[7] Y. LeCun et al. "Backpropagation Applied to Handwritten ZipCode Recognition" (1989)

[8] X. Ma et al. "Large-scale transportation network congestion evolution prediction using deep learning theory" (2015)

[9] S. Sarbour et al. "Dynamic Routing Between Capsules" (2017)

[10] `https://www.here.com/platform/traffic-solutions/real-time-traffic-information`

[11] S. Kiranyaz et al. "1D Convolutional Neural Networks and Applications – A Survey" (2019)

[12] G. Dahl et al. "Improving deep neural networks for LVCSR using rectified linear units and dropout" (2013)

[13] S. Ruder "An overview of gradient descent optimization algorithms"(2017)

[14] D. Kingma et al. "Adam: A Method for Stochastic Optimisation" (2015)

[15] X. Ma et al. "Learning Traffic as Images: A Deep Convolutional Neural Network for Large-Scale Transportation Network Speed Prediction" (2017)

[16] M. Gardner et al. "Artificial Neural Networks (The Multilayer Perceptron) - A Review of Applications in the Atmospheric Sciences" (1998)

[17] A. Krizhevsky et al. "ImageNet Classification with Deep Convolutional Neural Networks" (2012)

[18] Z. Pan et al. "Urban Traffic Prediction from Spatio-Temporal Data Using Deep Meta Learning"(2019)

[19] S. Hochreiter et al. "Long Short-Term Memory" (1997)

[20] Z. Zhao et al. "LSTM network: a deep learning approach for short-term traffic forecast" (2017)

[21] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" (2014)

[22] S. Sabour et al. "Dynamic Routing Between Capsules" (2017)

[23] `https://www.tensorflow.org/`

[24] Y. Kim et al. "A Capsule Network for Traffic Speed Prediction in Complex Road Networks" (2018)