TVE-F 18 007
Examensarbete 15 hp
Juni 2018

Sketch Classification with Neural Networks

A Comparative Study of CNN and RNN on the Quick, Draw! data set

Melanie Andersson
Maja Arvola
Sara Hedar

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Sketch Classification with Neural Networks - A Comparative Study of CNN and RNN on the Quick, Draw! data set

Melanie Andersson, Maja Arvola & Sara Hedar

The aim of the study is to apply and compare the performance of two different types of neural networks on the Quick, Draw! dataset and from this determine whether interpreting the sketches as sequences gives a higher accuracy than interpreting them as pixels. The two types of networks constructed were a recurrent neural network (RNN) and a convolutional neural network (CNN). The networks were optimised and the final architectures included five layers. The final evaluation accuracy achieved was 94.2% and 92.3% respectively, leading to the conclusion that the sequential interpretation of the Quick, Draw! dataset is favourable.

Handledare: Siri Persson
Ämnesgranskare: Niklas Wahlström
Examinator: Martin Sjödin
ISSN: 1401-5757, TVE-F 18 007

Contents

1 Populärvetenskaplig sammanfattning

Artificiell intelligens kan betraktas som ambitionen att efterlikna den mänskliga hjärnans förmåga till problemlösning samt att dra slutsatser i nya situationer. Artificiella neuronnätverk är ett område inom artificiell intelligens som består av självlärande algoritmer. Algoritmerna är inspirerade av hjärnans uppbyggnad och består av en struktur av så kallade neuroner i flera lager. Ett område som har ökat i popularitet under den senaste tiden är så kallad djupinlärning vilket syftar till neuronnätverk med många lager. Djupinlärning har visat sig vara väldigt effektivt inom många olika tillämpningar och utvecklingen av ny hårdvara har förkortat inlärningsprocessen väsentligt. Exempel på aktuella tillämpningar är självkörande bilar, talsyntes samt bildklassificering.

Den här studien består av att klassificera handritade skisser med hjälp av två olika typer av neuronnätverk. Skisserna hämtades från Googles dataset Quick, Draw! som samlades in via spelet med samma namn. De skisser som används i studien tillhör sju olika kategorier; ananas, flamingo, giraff, jordgubbe, krokodil, saxofon samt tvättmaskin. Nätverkens uppgift är att givet en ny skiss ange vilken kategori den tillhör och nätverken utvärderas sedan utifrån andelen korrekta klassificeringar. Det ena nätverket betraktar skisserna som en sekvens av penseldrag medan det andra nätverket betraktar skisserna som en färdig figur. Genom att konstruera, optimera och utvärdera nätverken dras slutsatser kring vilket betraktelsesätt som är det mest effektiva.

Det nätverk som betraktade skisserna som sekvenser presterade bäst och den högsta noggrannheten som uppnåddes i studien var 94.2%. Från detta drogs slutsatsen att det är bättre att betrakta skisserna från detta dataset som sekvenser än att betrakta dem som färdiga figurer vid användning av enkla neuronnätverk.

2 Introduction

According to Chollet, artificial intelligence can be described as "*the effort to automate intellectual tasks normally performed by humans*" [2]. Machine learning is a subset of artificial intelligence where the relationship between the input and output data of a system is modelled by extracting attributes from the input data. Artificial neural networks (hereafter neural networks) are a subset of machine learning which are loosely inspired by the structure within the human brain, using layers of neurons. Deep learning is a subset of neural networks where the neural network consists of many layers. See figure 1 for an illustration of the relationship between artificial intelligence, machine learning, neural networks and deep learning. Recent progress in hardware has enabled a rapid growth within deep learning. Deep learning has a wide range of successful applications such as computer vision, speech recognition and image classification [3].

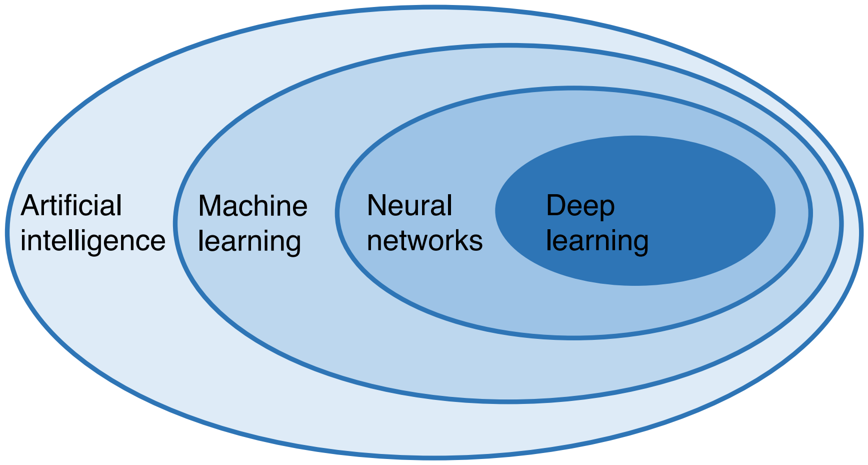


Figure 1: Venn diagram of the relationship between artificial intelligence, machine learning, neural networks and deep learning.

The aim of the study is to apply and compare the performance of two different types of neural networks on the Quick, Draw! data set. The data set consists of over 50 million sketches belonging to 345 different categories such as flamingo, crocodile and pineapple. The data set was collected through the game Quick, Draw! where the users were given 20 seconds to draw a sketch belonging to a specified category. The sketches are stored as vectors containing the strokes in the order they were drawn.

The two types of neural networks studied are convolutional neural networks (CNN) and recurrent neural networks (RNN). CNNs are typically used for image classification. 2D-convolutional layers, the type of layers used in the constructed CNN, interpret images as 2D-objects and preserves the grid-like structure of the pixels. RNNs are typically used for finding patterns in sequential data. There is a connection between the chain-like structure of the RNN and the sequential structure of the data it is suitable for.

When applied to the Quick, Draw! data set the CNN interprets the sketches as 2D-images while the RNN interprets the sketches as sequences of 2D-coordinates based on the strokes. RNNs are typically not used for image classification since images in general do not have an internal sequential structure. In this case the simplicity of the sketches allows for a sequential interpretation of the image according to how it was drawn; that is the strokes drawn and in which order they were drawn.

This study uses RNN for image classification which is not a common area of application for this type of network since the data usually does not have a sequential structure. The format of the Quick, Draw! data set enables the comparison of the performance of the two types of networks; CNN and RNN. The aim of the study is to, through applying both CNN and RNN to the data set, determine whether interpreting the sketches as sequences gives a higher accuracy than interpreting them as pixels.

3 Theory

3.1 Parametric Models

The aim of machine learning is to find a model that describes the relationship between the input and output data of a system by learning from the input data. The model f needs to be estimated in the following expression

$$Y = f(X) + \varepsilon, \tag{1}$$

where Y is the output data, f is the model, X is the input data and ε is an error term [4]. An example of a model could be Ohm's law which is a linear model that describes the relationship between the voltage and the current for a resistor. In this example the current could be considered as the input data and the voltage as the output data. Another example is the case of classifying sketches, as in this study, where the input data is the sketches and the output data is the class.

Parametric models are a type of mathematical model which describes the relationship between the input and output data using finite-dimensional parameters. Linear regression, logistic regression and neural networks are some examples of parametric models.

3.2 Supervised Learning

Within machine learning, the data that is used for finding the model is often split into training and evaluation data. Labelled data contains both the input and the corresponding output and can be used for finding the model, also known as supervised learning. Based on the model predictions can be made for new, unlabelled input data [1].

3.3 Classification Problems

Classification problems have a qualitative output, as opposed to regression problems where the output is quantitative. Qualitative variables take a discrete number of classes, such as a coin flip, while quantitative variables can take continuous values, such as the height of a person. Classifying sketches is an example of a classification problem.

3.4 Training the Model

To achieve a more accurate parametric model the parameters must be optimised through a training process. To evaluate the performance of the model the deviation of the prediction must be quantified. A loss function can be used to compute a numerical value, where a larger loss value corresponds to a larger deviation [2].

Cross-entropy is a loss function which is suitable for classification problems [5]. The reason behind this is that it takes into consideration "how wrong" the prediction was. A high probability for an incorrect prediction will give a high loss whilst a high probability for a correct prediction will give a loss close to zero. The cross-entropy is given by

$$L(x_i, y_i, \theta) = -\sum_{k=1}^{K} y_{ik} \log q_k(x_i; \theta), \tag{2}$$

where x_i is the i:th input data point, y_i the corresponding prediction, θ the parameters in the model, K the number of classes and q_k the estimated probability for the i:th data point belonging to the k:th class [4].

From the loss function a cost function can be defined as the average of the losses for all data points as stated below

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i, \theta), \tag{3}$$

where $J(\theta)$ is the cost function dependent on the parameters of the model, n the number of data points and $L(x_i, y_i, \theta)$ the loss function.

The parameters of the model are optimised by minimising a loss function using an optimisation algorithm. The aim of the algorithm is to find the minimum, often through an iterative process. The global minimum of a multi-dimensional function is difficult to find and therefore, a local minimum is considered to be sufficient. In each step of the training a loss is calculated, and the parameters are then adjusted to minimise the loss [4].

Stochastic Gradient Descent (SGD) is an optimisation algorithm which computes an approximation of the gradient of the cost function. A local minimum of the cost function can be found by taking a step in the direction of the negative gradient. SGD divides the data into so-called mini-batches and computes the gradient for each mini-batch, in order to reduce the computational time [4].

The size of the step is given by the so-called learning rate (lr). A higher learning rate corresponds to a larger step. If the learning rate is too low the model might not reach the minimum while a too high learning rate might not converge, as illustrated in figure 2 [4].

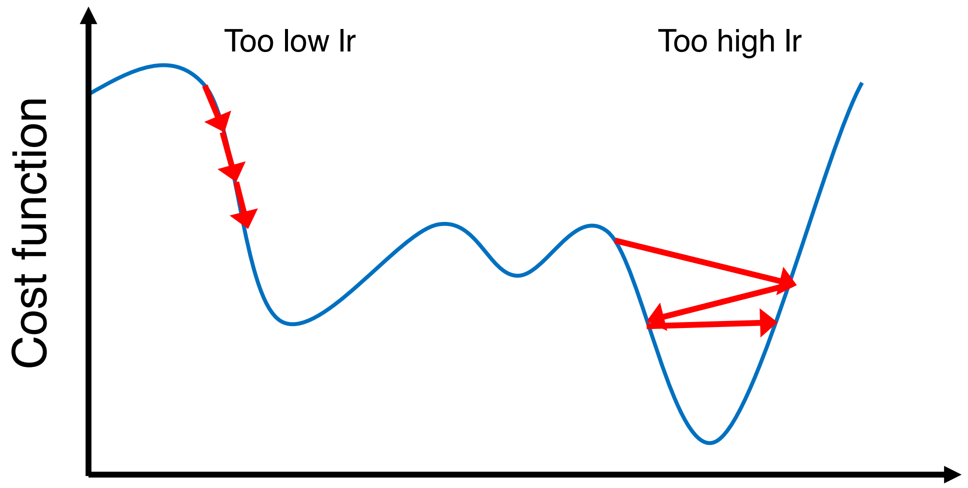


Figure 2: Illustration of how the learning rate affects the convergence.

The Adam optimisation algorithm is a version of the SGD where the learning rate is adjusted for each step. The algorithm is computationally efficient and requires little memory [6].

3.5 Neural Networks

A neural network (NN) is supposed to resemble the structure of the biological neural networks in the brain. It can be described as a composition of several linear models that are connected with activation functions, see section 3.5.1. The input data to one layer of the neural network is the output data from the previous layer. This way the data is processed through the layers and finally gives a result [7]. Neural networks can be used for both regression as well as classification [4].

Neural networks are composed of neurons that are connected with different weights which scale the input. The neurons are collected in layers and form the neural network which consists of several layers, some of them which are hidden; so-called hidden layers. A bias term, a constant, is added to shift the input to the activation function along the x_i-axis. For a neural network the weights and biases are the parameters which are optimised [4].

For a neural network with two layers the output from the first layer is given by

$$H = \sigma\,(W^{(1)T}X + b^{(1)T}), \tag{4}$$

where H is the output, σ the activation function, $W^{(1)}$ the weights, X the input and $b^{(1)}$ the bias. In matrix notation these variables can be expressed as stated below

$$b^{(1)} = \begin{bmatrix} \beta_{01}^{(1)} & \dots & \beta_{0M}^{(1)} \end{bmatrix}, \quad W^{(1)} = \begin{bmatrix} \beta_{11}^{(1)} & \dots & \beta_{1M}^{(1)} \\ \vdots & \dots & \vdots \\ \beta_{p1}^{(1)} & \dots & \beta_{pM}^{(1)} \end{bmatrix}, \quad X = \begin{bmatrix} X_1 \\ \vdots \\ X_p \end{bmatrix}. \tag{5}$$

The output from the second layer is given by

$$Z = W^{(2)T}H + b^{(2)T}, \tag{6}$$

where Z is the output from the second layer, H the output from the previous layer and $b^{(2)}$ the bias [4]. In matrix notation these variables can be expressed as follows

$$b^{(2)} = \begin{bmatrix} \beta_{0}^{(2)} \end{bmatrix}, \quad W^{(2)} = \begin{bmatrix} \beta_{1}^{(2)} \\ \vdots \\ \beta_{M}^{(2)} \end{bmatrix}, \quad H = \begin{bmatrix} H_1 \\ \vdots \\ H_M \end{bmatrix}. \tag{7}$$

The network described above is illustrated in figure 3.

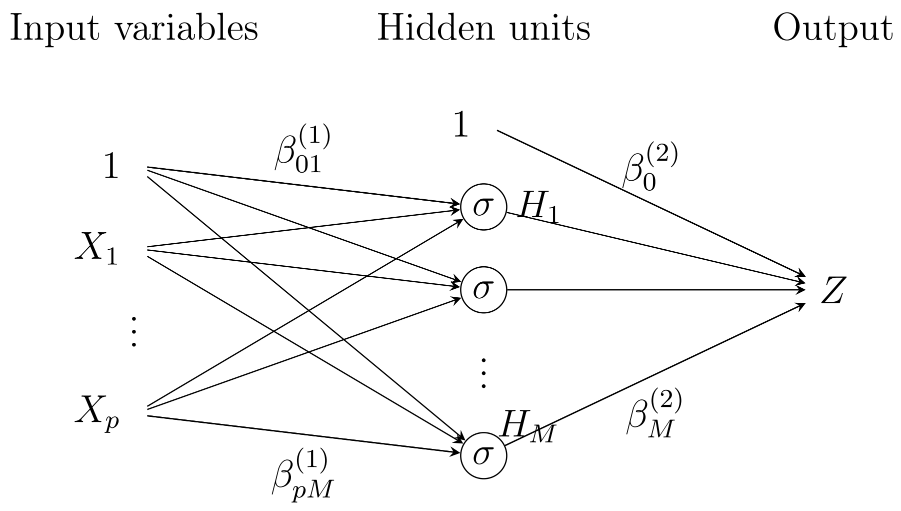


Figure 3: Neural network with two layers, where one is a hidden layer. Figure from [4] by courtesy of Niklas Wahlström.

3.5.1 Activation Functions

An activation function scales the input and thereby decides when a neuron is activated, that is passing information to the next layer.

The rectified linear unit (ReLU) is an activation function which has become increasingly popular due to its simplicity and high performance. The function is zero for all negative inputs and equal to the input if it is positive, as expressed below

$$\sigma(x) = max(0, x), \tag{8}$$

where $\sigma(x)$ is the activation function dependent on the input [4]. A graphical representation of this function can be seen in figure 4.

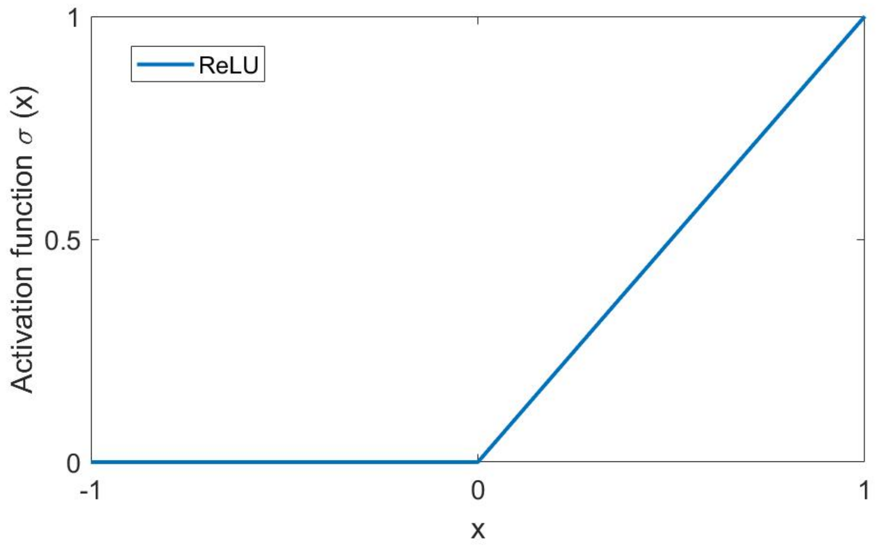


Figure 4: Activation function ReLU as a function of x.

When working with a classification problem the final result should be a class probability in order to achieve a qualitative output. In a neural network this can be done by using softmax as the activation function in the last layer of the network [4].

$$\text{softmax}(Z) = \frac{e^{Z_k}}{\sum_{l=1}^{K} e^{Z_l}}, \tag{9}$$

where Z is a vector containing the input to the function and $k \in [1, K]$ where K is the number of classes [7]. Softmax gives an output between 0 and 1 which can be interpreted as a class probability [4].

3.5.2 Fully Connected Layers and Dropout

In a fully connected layer, also-called dense layer, all connections to the output units are described by unique parameters contributing to a large number of parameters that have to be computed. Commonly fully connected layers are placed in the end of a neural network [4].

Adding so-called dropout to a layer removes a fraction of the neurons according to a specified percentage. Dropout prevents the model from overfitting, that is adapting too much to the training data, by making the result more random [4]. One sign of overfitting is a large difference between the training and validation loss, where the training loss is significantly smaller [8].

3.6 Convolutional Neural Networks

Convolutional neural networks (CNN) are a type of neural network typically used for data with a grid-like structure such as 2D-images. The structure of a CNN consists of several layers such as convolutional layers, pooling layers and dense layers. These layers are used for processing the input in order to produce the output.

As suggested by the name, convolutional neural networks are connected to the mathematical operation convolution. For a 2D-image the convolution operator is given by

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} \beta_{ij} \cdot I_{x+i-1,y+j-1}, \tag{10}$$

where I is the image and K a set of parameters (β_{ij}), known as a kernel, that has height h and width w. The kernel is stepped with a specified size (stride) and applied to all pixels of the image, as illustrated in figure 5.

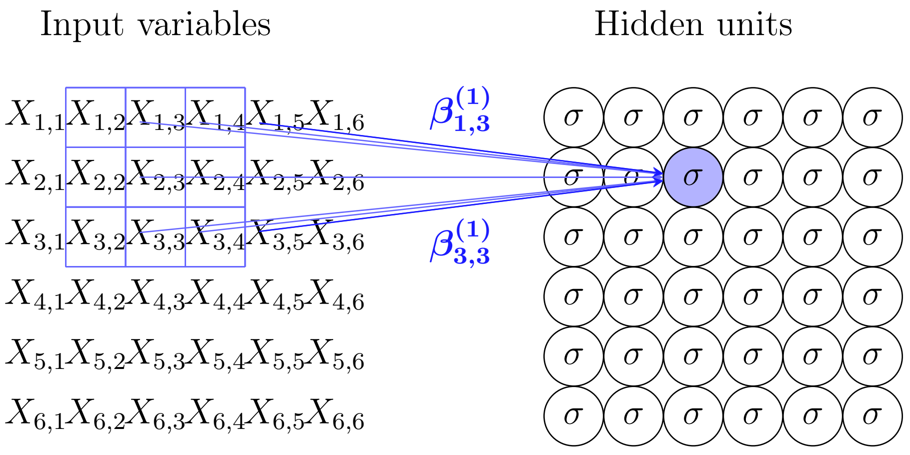

Input variables
Hidden units
$\boldsymbol{\beta}^{(1)}_{1,3}$
$\boldsymbol{\beta}^{(1)}_{3,3}$
$X_{1,1}$ $X_{1,2}$ $X_{1,3}$ $X_{1,4}$ $X_{1,5}$ $X_{1,6}$
$X_{6,1}$ $X_{6,2}$ $X_{6,3}$ $X_{6,4}$ $X_{6,5}$ $X_{6,6}$
σ

Figure 5: A convolutional layer with a kernel with parameters β_{ij} and size 3×3. Modified version of figure in [4], by courtesy of Niklas Wahlström.

The definition of the convolution used in CNNs is not equivalent to the mathematical definition, what is actually performed is a cross-correlation [10]. Cross-correlation of two functions $f(t)$ and $g(t)$ is the equivalent to convolving $f^*(-t)$ and $g(t)$, where f^* is the complex conjugate of f [11].

A CNN has at least one layer which uses convolution for the computations instead of the commonly used matrix multiplication. One of the advantages of convolutional layers is the so-called sparse interaction; using a smaller region of pixels for feature extraction instead of using all pixels separately. This reduces the number of parameters which improves the efficiency of the computations [3].

A convolutional layer reduces the number of parameters in another way as well, namely parameter sharing. In a fully connected layer all hidden units, which are the connections between the input and output, have unique parameters. This is an important difference compared to convolutional layers. Parameter sharing means that not all parameters need to be unique, instead the same set of parameters, the kernel, can be used in several different places of the neural network [3].

The computation of the output from a convolutional layer could be described as a convolution of the input and the kernel. The result is a so-called channel, which is a set of hidden units, containing the output. Using multiple kernels in a layer will increase the number of output channels with the same amount. In a CNN each layer depends on all channels from the previous layer which contributes to the complex structure of this type of network [4].

Figure 6 illustrates the structure of two convolutional layers.

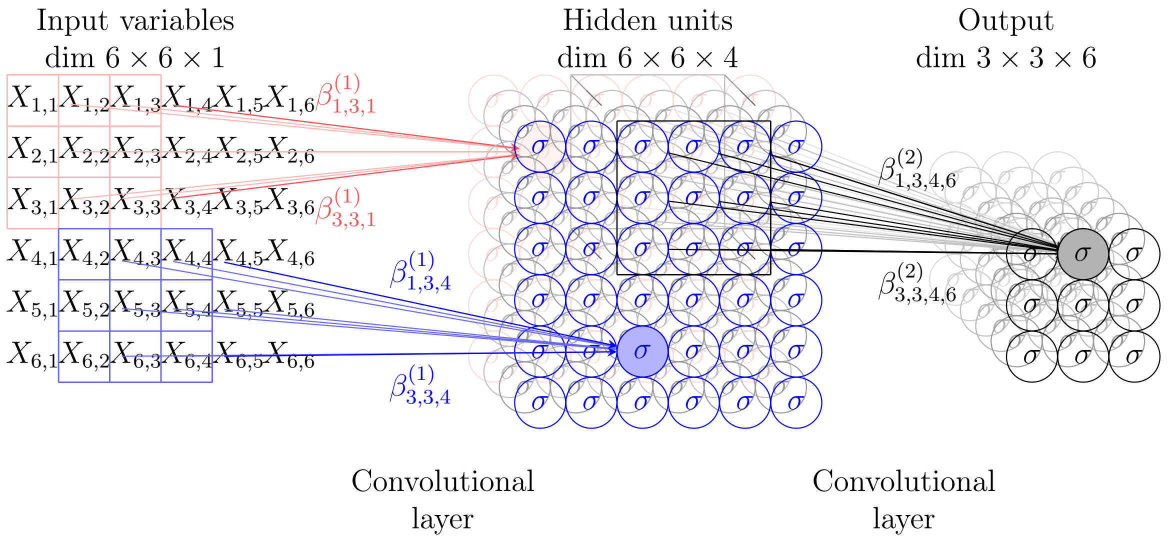

Input variables
dim $6 \times 6 \times 1$
Hidden units
dim $6 \times 6 \times 4$
Output
dim $3 \times 3 \times 6$
$X_{1,1}$ $X_{1,2}$ $X_{1,3}$ $X_{1,4}$ $X_{1,5}$ $X_{1,6}$
$X_{2,1}$ $X_{2,2}$ $X_{2,3}$ $X_{2,4}$ $X_{2,5}$ $X_{2,6}$
$X_{3,1}$ $X_{3,2}$ $X_{3,3}$ $X_{3,4}$ $X_{3,5}$ $X_{3,6}$
$X_{4,1}$ $X_{4,2}$ $X_{4,3}$ $X_{4,4}$ $X_{4,5}$ $X_{4,6}$
$X_{5,1}$ $X_{5,2}$ $X_{5,3}$ $X_{5,4}$ $X_{5,5}$ $X_{5,6}$
$X_{6,1}$ $X_{6,2}$ $X_{6,3}$ $X_{6,4}$ $X_{6,5}$ $X_{6,6}$
$\beta^{(1)}_{1,3,1}$
$\beta^{(1)}_{3,3,1}$
$\beta^{(1)}_{1,3,4}$
$\beta^{(1)}_{3,3,4}$
$\beta^{(2)}_{1,3,4,6}$
$\beta^{(2)}_{3,3,4,6}$
σ
Convolutional
layer
Convolutional
layer

Figure 6: An excerpt of a CNN architecture which consists of two convolutional layers. The input to the first layer has dimension $6 \times 6 \times 1$ and the layer has four kernels with size 3×3 and stride $[1, 1]$, two of these are marked in red and blue. These kernels produce four channels of hidden units. The hidden units are stored as $6 \times 6 \times 4$, which are the input to the second convolutional layer. The second convolutional layer has six kernels of dimension $3 \times 3 \times 4$, stride $[2, 2]$ and produces an output of dimension $3 \times 3 \times 6$. Modified version of figure in [4], by courtesy of Niklas Wahlström.

3.6.1 One-Dimensional Convolutional Layers

One type of convolutional layers is the 1D-convolutional layer. This type of layer can preserve grid-like structures of data with one dimension and can be interpreted as a feature extraction [12]. They can also be used for dimension reduction.

Figure 7 illustrates the inputs, x_i, being passed through a 1D-layer which produces the outputs y_i, where A is a 1D-kernel.

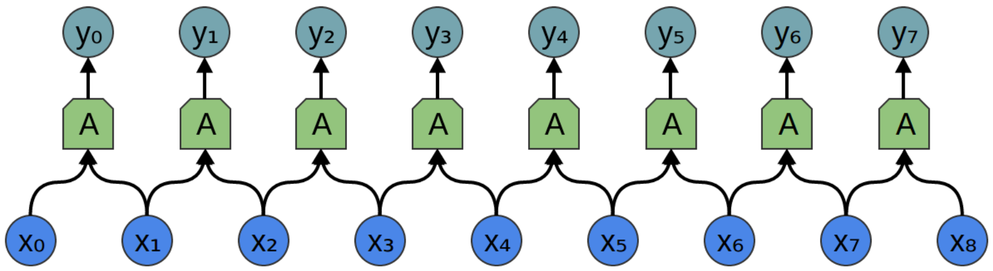


Figure 7: A 1D-layer with input x_i and output y_i where A is a kernel [13].

3.6.2 Two-Dimensional Convolutional Layers

Another type of convolutional layers is the 2D-convolutional layer. This type of layer stores the data in matrices which preserves 2D-grid-like structures [4]. These layers are often used for the application of image classification due to its ability to extract 2D-features, such as detecting edges in images [13].

Figure 8 illustrates the inputs, x_{ij}, being passed through a 2D-layer which produces the outputs y_{ij}, where A is a 2D-kernel.

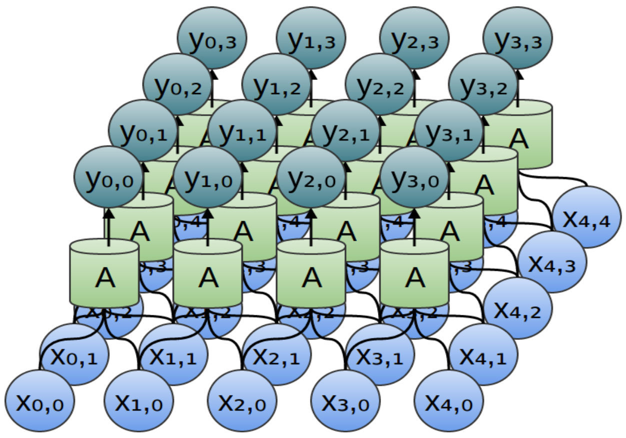


Figure 8: A 2D-layer with input x_{ij}, output y_{ij} where A is a kernel [13].

3.6.3 Condensing Information

Pooling layers are used to condense data by summarising a region of output data with some sort of statistical operation. This is done in order to reduce the number of parameters and thereby simplify the computations [9]. Max pooling is one of the most common pooling functions which condenses data by keeping the maximum value from the region which is being condensed, as illustrated in figure 9. Pooling layers are typically placed after a convolutional layer in order to condense the output [3].

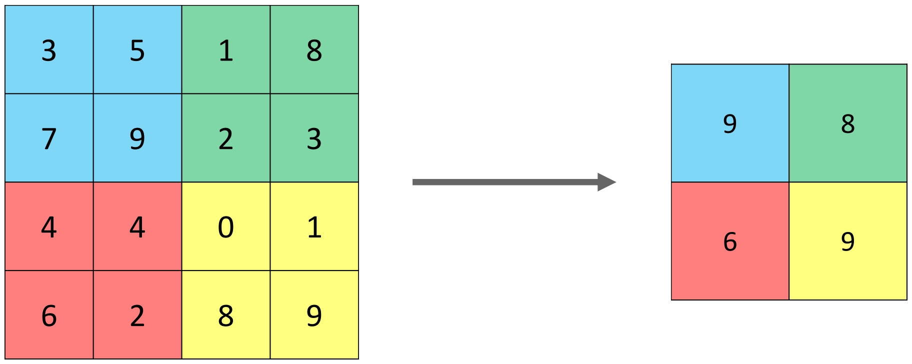


Figure 9: Max pooling of size 2×2 applied to a 4×4 matrix.

Another way of condensing data in a CNN is to apply the kernel to every other pixel for example, instead of to all pixels one by one. This is known as using a stride of two which reduces the number of rows and columns to half of the original number. The number of rows and columns can be reduced to the desired number by applying an appropriate stride [4].

3.7 Recurrent Neural Network

Recurrent neural networks (RNN) are a type of neural networks which are mostly applied on data with a sequential structure. The networks contain loops which allows information to persist and be passed between the different steps of the network. RNNs can be viewed as copies of the same network and a common visualisation can be seen in figure 10 [14].

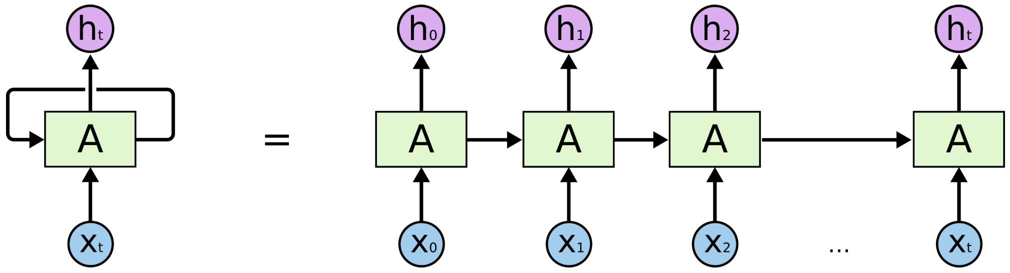


Figure 10: An unrolled recurrent NN, where A is a section of the network [14].

The chain-like structure of RNNs can be connected to the sequential structure of the data it is suitable for.

3.7.1 Long-Term Dependencies and LSTM

Standard RNNs have issues with learning so-called long-term dependencies in data. That is extracting information that has been processed a long time before the information is needed [14]. Mathematically the problem can be explained by the vanishing or exploding gradient problem. As the gradient propagates over many steps (as in RNNs) it may vanish or explode. If the gradient vanishes the learning process is affected and an exploding gradient may lead to instability [3].

Different versions of RNNs have been developed to be able to learn long-term dependencies. The idea behind gated RNNs is to create paths through time and that the derivatives of these paths should neither vanish nor explode. Gated RNNs include both LSTM (Long Short Term Memory) and networks based on the so-called gated recurrent unit (GRU) [3].

LSTM are specifically designed to be able to learn long-term dependencies. They have the same chain-like structure as standard RNNs where neural networks layers are repeated [14]. But instead of repeating a single layer the module in an LSTM contains four layers, see figure 11. The network can be viewed as having both an internal recurrence (in the LSTM-cell) and an outer recurrence, as other RNNs [3].

Today many applications of RNNs are done with LSTM since they have been shown to perform well in many areas [14]. A few successful applications of LSTM are; image captioning, machine translation, speech recognition and parsing [3].

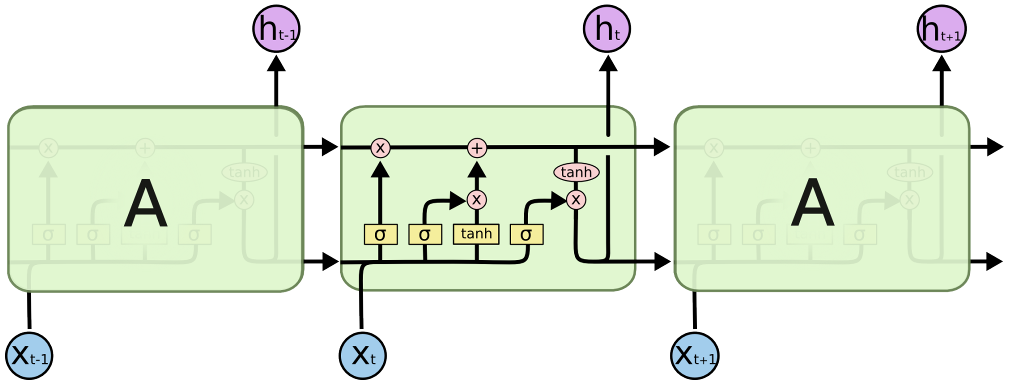


Figure 11: A LSTM module where A is a section of the NN, the yellow boxes represent the NN layers and the pink circles represent point-wise operations [14].

4 Data Set

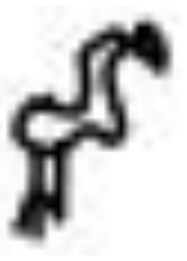
The Quick, Draw! data set provided by Google was created through the game Quick, Draw!. In the game the users were given 20 seconds to draw an object from a given category [15].

The complete data set consists of more than 50 million sketches of 345 categories. The raw data contains information regarding the category of the drawn object, a time stamp of when the drawing was created and the drawing itself along with other information [16].

The chosen categories from the data set were the following:

- Flamingo
- Giraffe
- Crocodile
- Pineapple
- Saxophone
- Strawberry
- Washing machine

Some examples of the sketches from the chosen categories are presented in figure 12-14. Since the data set has been created by different users the result varies.


(a)

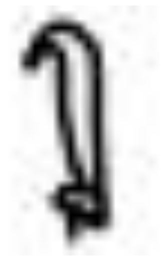
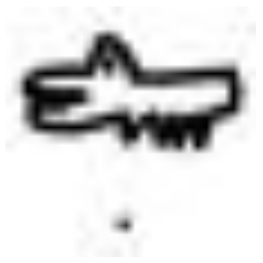

(b)

Figure 12: Two samples of flamingos from the Quick, Draw! data set converted into jpg-files of 28 × 28 pixels.


(a)

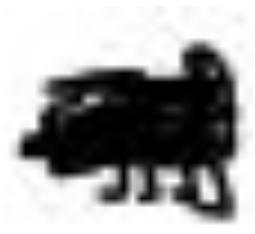
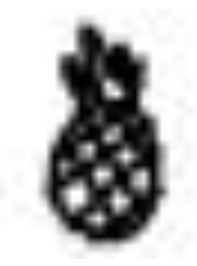

(b)

Figure 13: Two samples of crocodiles from the Quick, Draw! data set converted into jpg-files of 28 × 28 pixels.


(a)

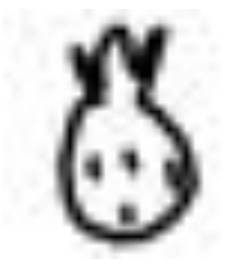

(b)

Figure 14: Two samples of pineapples from the Quick, Draw! data set converted into jpg-files of 28 × 28 pixels.

4.1 Data Processing

The raw data set was available in ndjson-format, containing a sequential representation of the sketches. The files contain information about which category the drawing belongs to, the country where it was drawn, a time stamp, whether it was correctly classified or not, a key-id and the drawing itself stored in vectors along with other information [16].

The data processing is based on a simplified version of the data set, also in ndjson-format. The drawings have been aligned to the top-left corner and uniformly scaled, along with other simplifications.

4.1.1 Convolutional Neural Network

The data set was also available as npy-files converted from the simplified ndjson-files. The drawings were aligned to the center and saved as 28×28 greyscale bitmaps [16]. The npy-files were converted into jpg-files before they were used for training the CNN. Some examples of generated jpg-files can be seen in figure 12-14.

4.1.2 Recurrent Neural Network

The ndjson-files were converted into TFRecord files in order to be implemented in TensorFlow. The conversion was executed according to the given instructions from TensorFlow's tutorial *Recurrent Neural Networks for Drawing Classification*. In the conversion the drawings and the corresponding categories were extracted [17].

5 Implementation and Optimisation

The open source software library TensorFlow was used for the numerical computations for both types of networks [18]. The design of the RNN was implemented using TensorFlow while the application programming interface (API) Keras was used to design the CNN. Keras can run on top of Tensorflow and includes several types of layers, activation functions, pre-trained models etc. [20].

During the optimisation process only one parameter was varied and the parameter which yielded the highest accuracy on evaluation data was kept. For the CNN the training data consisted of 3 000 sketches per category and 1 000 sketches per category for evaluation. For the RNN the training data consisted of 300 sketches per category and 100 sketches per category for evaluation. The final networks were trained on the same 10 000 sketches and evaluated on the same 3 000 sketches per category.

5.1 Convolutional Neural Network

The initial design of the CNN consisted of one 2D-convolutional layer and one fully connected layer with softmax as the activation function. For the other layers ReLU was used as activation function, Adam as optimisation algorithm and the loss function was calculated using cross-entropy. Data from the Quick, Draw! data set was converted to jpg-files and used as input data.

The order in which the parameters were varied during the optimisation process are listed below:

- Number of 1D-convolutional layers
- Number of fully connected layers
- Number of pooling layers
- Learning rate
- Dropout
- Number of epochs

The performance of the networks were evaluated based on the final accuracy on the evaluation data. Once the optimisation process was finalised the network with the best accuracy was trained and evaluated on a bigger data set.

5.2 Recurrent Neural Network

The design of the RNN is based on a TensorFlow tutorial [17]. The network from the tutorial consists of three 1D-convolutional layers, 3 LSTM-layers and one fully connected layer with softmax as the activation function. In the tutorial the network is trained on the Quick, Draw! data set and the input data is read as TFRecord-files, which is a file format used in TensorFlow.

During the implementation and optimisation the network from the tutorial was modified. The initial network that was constructed consisted of one 1D-convolutional layer, one LSTM-layer and finally a fully connected layer with softmax as activation function. For the other layers ReLU was used as activation function, Adam as optimisation algorithm and the loss function was calculated using cross-entropy.

The order in which the parameters were varied during the optimisation process are listed below:

- Number of 1D-convolutional layers
- Number of LSTM-layers
- Learning rate
- Dropout

The performance of the networks were evaluated based on the final accuracy on the evaluation data. Once the optimisation process was finalised the network with the best accuracy was trained and evaluated on a bigger data set and with an increased number of training steps.

6 Result

The result from the study is separated into two sections; one for CNN and one for RNN. The optimisation process is presented in tables and the evaluation of accuracy and loss is visualised in graphs.

6.1 Convolutional Neural Network

The initial CNN consisted of one 2D-convolutional layer and one fully connected layer with softmax as activation function. During the first part of the CNN optimisation the structure, i.e. number of different types of layers, was varied and the result is presented in table 1.

Table 1: Results from optimisation of CNN structure with the learning rate set to 0.001 and the number of epochs to 25. All structures end with a fully connected layer with softmax as activation function. The best result with regards to accuracy on the evaluation data is marked in green.

# Conv. layers	# Fully connected layer ReLU	# Pooling layers	Accuracy
1	-	-	0.8880
2	-	-	0.8999
3	-	-	0.8921
2	1	-	0.9066
2	1	1	0.9127

The final structure of the CNN consisted of two 2D-convolutional layers, one layer with max pooling, one fully connected layer with ReLU as activation function and one fully connected layer with softmax as activation function. The full architecture of the network can be seen in figure 15.

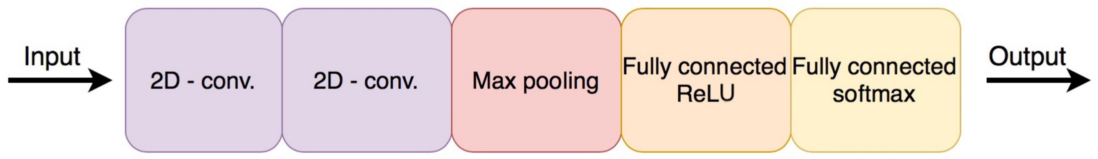


Figure 15: The final structure of the optimised CNN.

The structure above was used and the learning rate was varied between 0.0001 and 0.1, result is presented in table 2.

Table 2: Results from optimisation of learning rate on network with optimised structure and 25 epochs. The best result with regards to accuracy on the evaluation data is marked in green.

Learning rate	Accuracy
0.0001	0.8773
0.001	0.9127
0.005	0.9187
0.0075	0.9184
0.01	0.9196
0.05	0.1429
0.1	0.1429

For the smallest and two largest values the model has not converged while the other values appear to have yielded a stable accuracy around 91.5%. The largest learning rate which resulted in a stable accuracy, 0.01, was chosen to minimise the total training time.

Thereafter dropout was added to the network at two different positions. First the dropout was added after the first convolutional layer (Dropout 1) and varied. The best value was kept and dropout was added after the first dense layer (Dropout 2). Lastly only Dropout 2 was used and varied. The case of using no dropout was also included as reference. The results from varying the dropout at these positions is presented in table 3.

Table 3: Results from optimisation of dropout on network with optimised structure, optimised learning rate and 25 epochs. The best result with regards to accuracy on the evaluation data is marked in green.

Dropout 1	Dropout 2	Accuracy
-	-	0.9196
0.1	-	0.9157
0.15	-	0.9166
0.2	-	0.9084
0.25	-	0.9099
0.3	-	0.9071
0.4	-	0.8374
0.15	0.1	0.9031
0.15	0.15	0.8984
0.15	0.2	0.9077
0.15	0.25	0.9069
0.15	0.3	0.9049
0.15	0.4	0.8876
-	0.1	0.9076
-	0.15	0.8923
-	0.2	0.9039
-	0.25	0.9037
-	0.3	0.9056
-	0.4	0.8957

Since adding the dropout layers did not improve the accuracy of the network all dropout was removed. The network was then optimised with regards to number of epochs, the result is presented in table 4.

Table 4: Results from optimisation of number of epochs on network with optimised structure, optimised learning rate and no dropout. The best result with regards to accuracy on the evaluation data is marked in green.

Epochs	Accuracy
5	0.9161
10	0.9096
15	0.9101
20	0.9024
25	0.9196
30	0.9037
40	0.9121
50	0.9026
100	0.9117
500	0.9051

Increasing the number of epochs did not improve the accuracy so the original number of epochs, 25, was kept since it also yielded the highest accuracy. Lastly the CNN was trained on a larger data set and the result is presented in table 5.

Table 5: Results from training the optimised CNN on data sets of different sizes. The best result with regards to accuracy on the evaluation data is marked in green.

Training data/category	Test data/category	Accuracy
3000	1000	0.9161
10000	3000	0.9232

The result presented in table 5 is also visualised in figure 16.

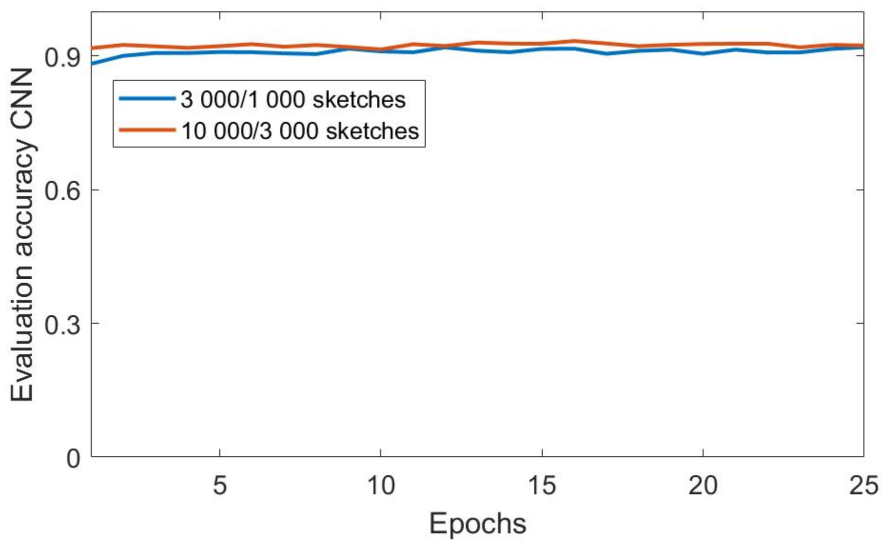


Figure 16: Evaluation accuracy of the CNN during the training process for the different sizes of data sets with 25 epochs.

Increasing the amount of data yielded a slightly higher accuracy of 92.3%. This is considered as the final accuracy and is compared to the final accuracy of the RNN. The evaluation loss of the CNN trained on the larger data set can be seen in figure 17.

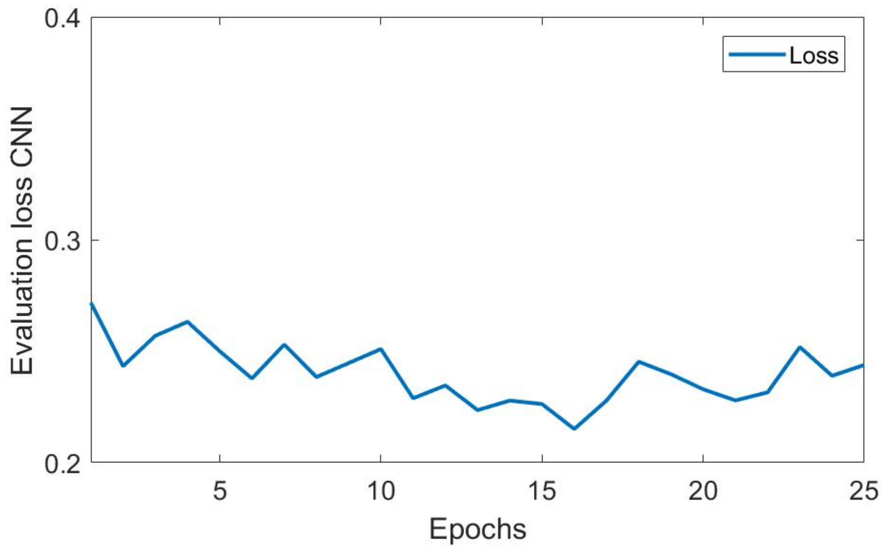


Figure 17: Evaluation loss of the CNN during the training process with 25 epochs and using the larger data set.

The loss decreases slightly during the training process and appears to have converged. The total training time for the CNN varied from 8.9 s to 10 min.

6.2 Recurrent Neural Network

The initial RNN consisted of one 1D-convolutional layers, one LSTM layer and one fully connected layer with softmax as the activation function. During the first part of the RNN optimisation the structure, i.e. the number of different layers, was varied and the result is presented in table 6.

Table 6: Results from optimisation of RNN structure with learning rate set to 0.0001. All structures end with a fully connected layer with softmax as activation function. The best result with regards to accuracy on the evaluation data is marked in green.

# Conv. layers	# LSTM layers	Accuracy
1	1	0.8410
3	1	0.8820
3	3	0.8760

The final structure of the RNN consisted of three 1D-convolutional layers, one LSTM layer and one fully connected layer with softmax as the activation function, see figure 18.

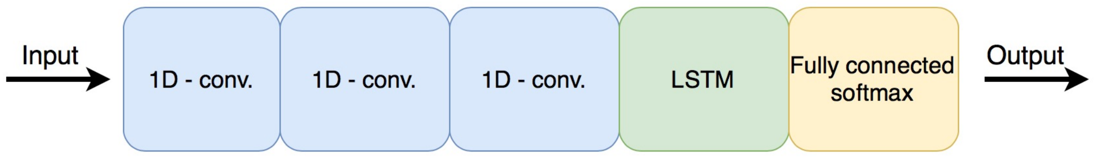


Figure 18: The final structure of the optimised RNN

Thereafter the learning rate was varied between 0.000075 and 0.001, the result is presented in table 7.

Table 7: Results from optimisation of learning rate on network with optimised structure. The best result with regards to accuracy on the evaluation data is marked in green.

Learning rate	Accuracy
0.000075	0.8700
0.0001	0.8820
0.0005	0.9020
0.00075	0.9080
0.001	0.1260

For the two smallest and the largest value the model has not converged while the other values appear to have yielded a stable accuracy around 90.5%. The largest learning rate, 0.00075, which resulted in a stable accuracy was chosen to minimise the total training time.

Thereafter dropout was added after each convolutional and LSTM layer and varied, the result is presented in table 8.

Table 8: Results from optimisation of dropout on network with optimised structure and optimised learning rate. The best result with regards to accuracy on the evaluation data is marked in green.

Dropout	Accuracy
-	0.9080
0.1	0.8889
0.2	0.9038
0.3	0.8829
0.4	0.8942

Since adding the dropout layer did not improve the accuracy of the network the dropout was removed. The network was then trained on larger data sets, the result is presented in table 9.

Table 9: Results from training the optimised RNN on data sets of different sizes. The best result with regards to accuracy on the evaluation data is marked in green.

Training data/category	Test data/category	Accuracy
300	100	0.9080
3000	1000	0.9380
10000	3000	0.9420

The result presented in table 9 is also visualised in figure 19.

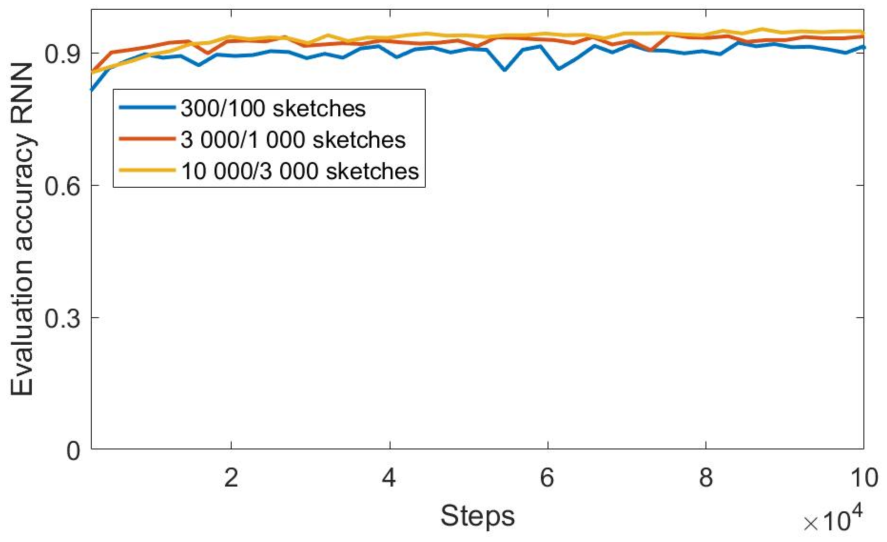


Figure 19: Evaluation accuracy of the RNN during the training process for the different sizes of data sets with 100 000 training steps.

Increasing the amount of data yielded a higher accuracy of 94.2%. This is considered as the final accuracy and is compared to the final accuracy of the CNN. The loss of the RNN trained on the larger data set can be seen in figure 20.

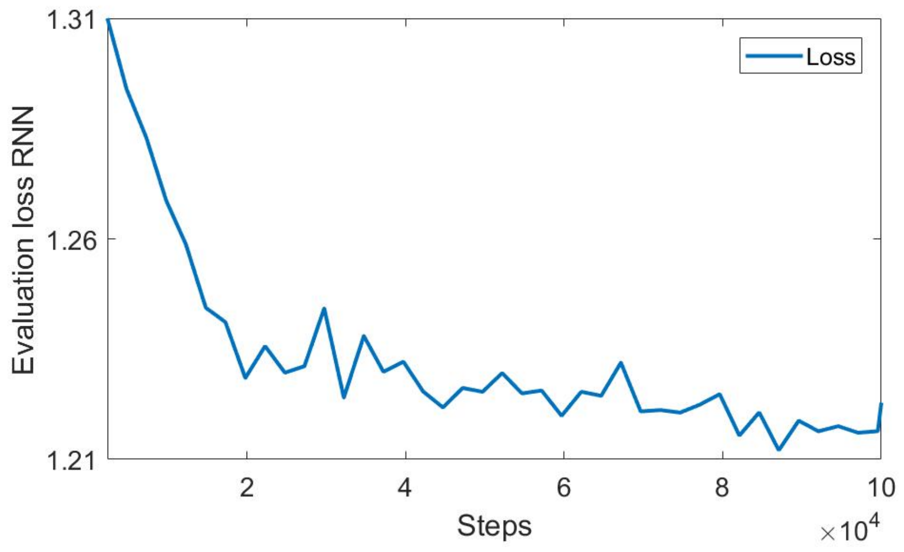


Figure 20: Evaluation loss of the RNN during the training process with 100 000 training steps and using the larger data set.

The loss decreases during the training process but does not appear to have converged completely. The total training time for the RNN varied from 6.5 h to 13.5 h.

7 Discussion

The performance of the networks was evaluated with respect to the final accuracy on the evaluation data. RNN performed best with an accuracy of 94.2% compared to CNN that achieved an accuracy of 92.3% when trained on 10 000 sketches and evaluated on 3 000 sketches per category. The total training time was significantly higher for the RNN, which varied from 6.5 h to 13.5 h, compared to the CNN which had a total training time that varied from 8.9 s to 10 min. Since the aim of the study was to evaluate the final accuracy of the networks the total training time was not taken into account when evaluating the performance.

One significant difference in the optimisation processes of the networks is that the amount of data for training and evaluation differed. The CNN is trained on a much larger data set during the optimisation process which gives a constantly higher accuracy than the RNN. For comparability the two final networks were trained and evaluated on the same, larger data. When trained on the same amount of data, the accuracy of the RNN surpassed the accuracy of CNN.

The evaluation losses of the final networks do not increase which is a clear sign that the models have not been overfitted. This might have been the reason why adding dropout did not increase the performance of the network. For the RNN the loss did not converge completely so increasing the number of training steps might decrease the loss further. Although cross-entropy was used as loss function for training both networks the evaluation losses might not be comparable in absolute terms due to differences in implementation.

In machine learning the result of the training process is affected by randomness. It is therefore appropriate to train the network several times using the same settings (iterating) and averaging the results to minimise the effects of the random properties. This methodology has not been applied in the optimisation process and the results might have affected the conclusions during the optimisation. The final networks might have had a different structure if this issue had been considered.

Interpreting the sketches as sequences using the RNN yielded a higher accuracy than interpreting them as pixels using the CNN. One possible explanation is the nature of the sketches and the data gathering through the game Quick, Draw!. When drawing a sketch it is common to begin with the most essential part and add the details later and the time limit in the game could add to this effect. This information is preserved through the sequential description since it is stored with respect to time while it is lost in the pixel format as only the finished image is considered.

8 Conclusions and Future Improvements

The RNN achieved a higher accuracy than the CNN. From this result it can be concluded that interpreting the sketches from the Quick, Draw! data set as sequences gives a higher accuracy than interpreting them as pixels when constructing fairly shallow neural networks.

In order to achieve a higher accuracy the amount of training and evaluation data per category could be increased. Another improvement could be to pre-process the data set by removing sketches which can be considered as noise, see appendix for example. Removing unnecessary image processing in the implementation of the CNN might also be a way of improving the results. More categories could be added and the performance of the networks in each category could be evaluated. To achieve a more reliable result one could perform an iteration in each step of the optimisation process and average the result.

9 Acknowledgements

Firstly, we would like to take the opportunity to express our sincere gratitude to our project mentor Niklas Wahlström, Post doctoral at Department of Information Technology, Division of Systems and Control for his patience, valuable guidance and for sharing his immense knowledge.

We would also like to acknowledge Precisit AB for giving us the opportunity to conduct the project with them.

Lastly, we are very grateful to our supervisor Siri Persson, Consulting Software Developer at Precisit AB, for her support and coaching.

References

[1] Kriesel D. *A Brief Introduction to Neural Networks* [Internet]. Version Zeta2, 2007 [Updated 2012-03-17; cited 2018-04-03] Available from: http://www.dkriesel.com/en/science/neural_networks

[2] Chollet F. *Deep Learning with Python* [Internet], Manning Publications; 2017 [cited 2018 Maj 17] Available from: https://manning-content.s3.amazonaws.com/download/4/02b02cf-c1ac-47ae-8ffd-25f85daad877/SampleCh01.pdf

[3] Goodfellow I, Bengio Y, Courville A. *Deep Learning* [Internet], MIT Press; 2016 [cited 2018 Mar 28] Available from: http://www.deeplearningbook.org

[4] Lindsten F, Svensson A, Wahlström N, Schön T. *Lecture Notes* [Internet]. Uppsala University Department of Information Technology [updated 2018 Feb 02; cited 2018 Mar 18] Available from: http://www.it.uu.se/edu/course/homepage/sml/literature/lecture_notes.pdf

[5] Dahal P. *Classification and Loss Evaluation - Softmax and Cross Entropy Loss* [Internet]. Deep Notes, 2017 Maj 28 [cited 2018 Apr 01] Available from: https://deepnotes.io/softmax-crossentropy

[6] P. Kingma D, Lei Ba J. *Adam: A Method for Stochastic Optimization* [Internet]. Proceedings of International Conference for Learning Representations, 2015, San Diego [updated 2017 Jan 30; cited 2018 Apr 17] Available from: https://arxiv.org/pdf/1412.6980.pdf

[7] Wahlström N. *Statistical Machine Learning 2018, Lecture 8 - Deep learning and neural networks*, [unpublished lecture notes]. Uppsala University; notes provided at lecture given 2018 Feb 22. [cited 2018 Apr 03] Available from: http://www.it.uu.se/edu/course/homepage/sml/lectures/lecture8_handout_updated.pdf

[8] Schön T. *Statistical Machine Learning 2018, Lecture 5 - Bias-variance trade-off, cross-validation*, [unpublished lecture notes]. Uppsala University; notes provided at lecture given 2018 Jan 31. [cited 2018 Maj 21] Available from: http://www.it.uu.se/edu/course/homepage/sml/lectures/lecture5_handout.pdf

[9] Karpathy A. *Convolutional Neural Networks (CNNs / ConvNets)* [Internet], Stanford University - Vision and Learning lab, [cited 2018 Maj 28], Available from: http://cs231n.github.io/convolutional-networks/#add

[10] Veličković P. *Deep learning for complete beginners: convolutional neural networks with keras* [Internet], Cambridge Spark Ltd [updated 2017 Mar 22; cited 2018 Maj 16] Available from: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

[11] Weissten E. *Cross-Correlation* [Internet], MathWorld - A Wolfram Web Resource [cited 2018 Maj 16] Available from: http://mathworld.wolfram.com/Cross-Correlation.html

[12] Ng A, Katanforoosh K, Bensouda Mourri, Y. *Networks in Networks and 1x1 Convolutions* [video on the Internet], deeplearning.ai [cited 2018 Apr 03] Available from: https://www.coursera.org/learn/convolutional-neural-networks/lecture/ZTb8x/networks-in-networks-and-1x1-convolutions

[13] Olah C. Cola's blog [Internet]. Christopher Olah. 2014-. *Conv Nets: A Modular Perspective*; 2018 Jul 08 [cited 2018 Maj 18] Available from: https://colah.github.io/posts/2014-07-Conv-Nets-Modular/

[14] Olah C. Cola's blog [Internet]. Christopher Olah. 2014-. *Understanding LSTM Networks*; 2015 Aug 27 [cited 2018 Apr 03] Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[15] Ha D, Eck D. *A Neural Representation of Sketch Drawings* Proceedings of International Conference for Learning Representations, 2018, Vancouver [cited 2018 Maj 22] Available from: https://openreview.net/forum?id=Hy6GHpkCW

[16] Jongejan J, Ray M, Hardmaru. *The Quick, Draw! data set* [Internet], Google Creative Lab; 2017 Maj 09 [updated 2017 May 25; cited 2018 Apr 17] Available from: https://github.com/googlecreativelab/quickdraw-dataset#license

[17] *Recurrent Neural Networks for Drawing Classification* [Internet], Tensorflow [updated 2018 mars 30; cited 2018 Mar 27] Available from: https://www.tensorflow.org/versions/master/tutorials/recurrent_quickdraw

[18] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C et al. *Large-Scale Machine Learning on Heterogeneous Systems* [Internet], Tensorflow; 2015 [cited 2018 Mar 27] Available from: https://www.tensorflow.org

[19] *TensorBoard: Visualizing Learning* [Internet], Tensorflow [updated 2018 Apr 28; cited 2018 Maj 21] Available from: https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

[20] Chollet F et al. *Keras* [Internet], 2015 [cited 2018 Mar 27] Available from: https://keras.io

10 Appendix

Figure 21: A sample of a crocodile that could be considered as noise in the data set.