Image Verification with Siamese Capsule Networks

Mark de Blaauw
Supervisor: Ali el Hassouni
Vrije Universiteit Amsterdam

February
2019

Abstract

Capsule networks (CapsNets) have been introduced by Sabour et al. (2017) to improve certain limitations in convolutional neural networks (CNNs). This resulted in CapsNets, which need less training data and are more robust to affine transformations on images than CNNs. This paper researches if the domain of one-shot learning is able to benefit from these improvements by incorporating the CapsNet model into a Siamese neural network structure, called a Siamese capsule network with magnitude (SCNM). Experiments using the Omniglot dataset resulted in an average improvement of 2.7% accuracy compared to a Siamese convolutional neural network (SCNN) on ten affine tranformed test sets. The SCNM structure has only roughly one-quarter the number of parameters compared to that of the SCNN, showing that the SCNM structure is significantly more robust than SCNNs on affine transformations and needs considerably less parameters to achieve this.

1 Introduction

An increase in computer power and data has led to an increase in the popularity of deep learning applications. This has resulted in state-of-the-art models that are reaching human-level performance. This is the case in computer vision, in which classification, recognition, and detection tasks are modelled with the use of CNNs. These CNNs use convolutions to detect patterns from images. They have a translation-equivariant nature. In other words, when the input shifts, the output also shifts but stays otherwise unchanged. However, CNNs do not posses other equivariant transformations, such as rotation (Goodfellow et al. (2016); Cohen et al. (2016)). Thus, data augmentation is applied to replicate equivariance to transformations (Linmans et al. (2018)). Pooling is introduced to reduce the number of parameters and make CNNs more translation-invariant for classification. These implementations improve the performances of CNNs but do not solve their problems completely and even introduce some other problems, such as a loss of relative spatial information (Honari et al. (2016)).

Recently, CapsNets were introduced by Sabour et al. (2017), which uses capsules and stores transformation (i.e., pose) information into these capsules. This makes the model transformation equivariant and as a result robust to transformations on input images. Hence, the necessity of data augmentation is less needed. Pooling is replaced by a so-called routing-by-agreement algorithm, which uses the pose information of local features to agree on the existence of an whole entity (i.e., higher-level feature representation). This means that CapsNets need less training data to reach the same performance as traditional CNNs, are robust to affine transformations, and take relative spatial relationships into account, as is shown by Sabour et al. (2017) on achieving state-of-the-art performance on both MNIST and affNIST datasets.

Another field that may benefit from these advantages is image verification. The major challenge in image verification is verifying a new class from small quantities of training data. It is called 'one-shot learning' when a single observation is used to verify a new class (Koch et al. (2015)). To do this, it is hypothesised that once a few classes in the same domain have been learned, some of this information can be utilised to make learning other classes more efficient (Fei-Fei et al. (2006); Lake et al. (2011)). One way to achieve this is to use Siamese convolutional neural networks (SCNNs), which offer state-of-the-art performance without imposing strong priors. O'Neill (2018) showed that Siamese capsule networks (SCNs) are capable of achieving the same performance as SCNNs. This paper researches if a different implementation of a SCN, named SCNM, is able to outperform both the general SCN structure from O'Neill (2018) and SCNN from Koch et al. (2015) with respect to robustness to affine transformations on images.

This is done by reviewing related work in Section 2. Section 3 explains what the Omniglot dataset is and how datasets are extracted for learning. Section 4 introduces methods of learning and finding optimised hyperparameters for the different models. Section 5 then explains the experimental setups and Section 6 showcases the results of the experiments. The paper concludes in Section 7 and 8 with, respectively, a discussion and conclusion.

2 Related work

Much work has been conducted in the field of one-shot learning using Siamese structures. This network was used by Bromley et al. (1994) to verify written signatures in 1994. To evaluate whether two signatures were the same, they calculated a cosine of the angle of two extracted feature vectors from two signatures. Around that time, the same Siamese structure was proposed for fingerprint identification by Baldi and Chauvin (1993).

Recently, the Siamese structure has been applied for face verification by Taigman et al. (2014). It achieved human-level performance on the 'Faces in the Wild' dataset and performed significantly better than previous methods. It uses a lot of prepossessing steps before the images are forwarded into the Siamese structure. The Manhattan distance is used to evaluate whether two extracted features lie close in their dimensional space and is followed with a single logistic unit to assess if the two input images are the same. The paper also mentions that it does not use max pooling layers at each layer because of information loss, which is a similar thought process used with CapsNets.

In Chopra et al. (2005), face verification is also applied using a Siamese structure. The Manhattan distance is used to evaluate the distance between two extracted features. However, a contrastive loss function is used, which drives the similarity metric to be small for a pair of matching input images and large for a pair of different images.

A loss function with the same idea is implemented by Schroff et al. (2015), which is called the triplet loss function. This paper uses a deep convolutional network in a Siamese structure without imposing a multi-stage prepossessing phase as in Taigman et al. (2014). It achieved state-of-the-art performance on the 'Faces in the Wild' dataset. The same ideas of a deep Siamese network were applied by Koch et al. (2015), who used the Omniglot dataset to train and evaluate the model. The Manhattan distance was used followed by a sigmoidal output unit to compare both input images. A regularised cross-entropy loss function was used for training the network, and it achieved state-of-the-art performance on the Omniglot dataset compared to previous methods.

Further, the paper by O'Neill (2018) is the only research that combines a Siamese structure with CapsNets. It does this by replacing the subnetworks of the Siamese structure with the CapsNet structure but excluding the reconstruction network. Finally, it uses a l2-normalised contrastive loss from Chopra et al. (2005) to maximise interclass variance and minimise intraclass variance.

We see that in many of the related works, the Manhattan distance is used as the distance metric in the distance layer of Siamese neural networks. As for the loss function, a standard cross-entropy is regarded or, in many cases, a contrastive loss function is used.

3 Dataset

The Omniglot dataset is used during the research. In this section, the dataset is described and how datasets are extracted to do training, validation and testing is explained.

3.1 Omniglot dataset

A dataset of characters was collected by Lake et al. (2011) to research applications for one-shot learning. The dataset consists of 50 different alphabets from around the world and even includes some alphabets from science fiction novels. The 50 alphabets have a total of 1600 different characters, in which every character has only 20 examples and for which all characters are drawn by 20 different persons. This means that only 20 training examples are provided for every character class. Hence, they are excellent for experiments in one-shot learning applications. The characters are depicted in greyscale with a size of 105 x 105 pixels. Some characters from the dataset are shown in Figure 1. Further, Lake et al. (2011) split the data into a 40 alphabet 'background set' and 10 alphabet 'evaluation set'. The use of these terms is maintained, wherein the test set creation refers to the 'evaluation set' and 'background set' refers to the creation of the training- and validation sets. Ten alphabets are randomly chosen from the 'background set' for the validation set. The other 30 alphabets are used to construct training sets.

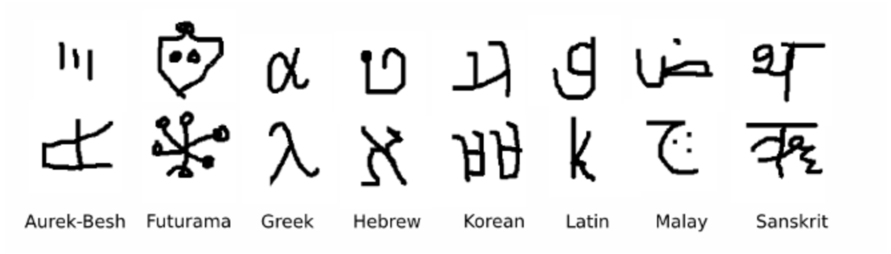


Figure 1: Two characters each from eight alphabets.

3.2 Dataset creation

The Omniglot dataset does not provide labelled data and therefore an algorithm is implemented to create training, validation, and test sets. The Siamese structure has an input of two images that are compared and assessed as to whether they are the same. Hence, datasets of pairs need to be constructed including a binary label. This is done by creating a balanced dataset for all three different sets and uniformly picking pairs. The pseudocode for building datasets is shown in Algorithm 1. The full code can be found in Appendix A.1.

Test sets are created to investigate robustness against affine transformations. In Figure 2 the transformations that were used are shown graphically. At first, the algorithm to produce the initial test sets that was mentioned previously is used. Thereafter, a uniform rotation is applied to both pairs independently, with a range of $[-20, 20]$ degrees. The pairs are then independent sheared[1]

[1]Information about shearing: `https://en.wikipedia.org/wiki/Shear_mapping`

Algorithm 1 Balanced dataset generation

```
1: Given a character dataset $N = \{I_1, ..., I_m\}$, with $j \in \{1, ..., m\}$, $I_j$ a character class and $m$ the total amount of different character classes. Then $I_j = \{1_j, ..., l_j, ..., 20_j\}$ is a set of drawn characters from the class $j$, with $l \in \{1, ..., 20\}$.
2: Generate $K$ balanced training pairs from the set $N$.
3: for $t = K$ do
4:     Randomly sample a character class $I_{j_1}$ from $N$.
5:     Randomly sample one character $l_{j_1}$ from $I_{j_1}$.
6:     if $t < K/2$ then
7:         Randomly sample a character class $I_{j_2}$ from $N$ while $I_{j_1} \neq I_{j_2}$.
8:         Randomly sample one character $l_{j_2}$ from $I_{j_2}$.
9:         Store the character pair $[l_{j_1}, l_{j_2}]$ with a negative label.
10:    else
11:        Randomly sample one character $l_{j_2}$ from $I_{j_1}$ while $l_{j_1} \neq l_{j_2}$.
12:        Store the character pair $[l_{j_1}, l_{j_2}]$ with a positive label.
13:    end if
14: end for
```

with a uniformly chosen shearing factor between $[-0.4, 0.4]$. This is done by multiplying the y coordinate by the shearing factor and adding that to the x coordinate. At last, horizontal and vertical translations are applied to the pairs independently, again uniformly, but with a dynamic range, such that the characters do not fall off the 105 x 105 image edge.

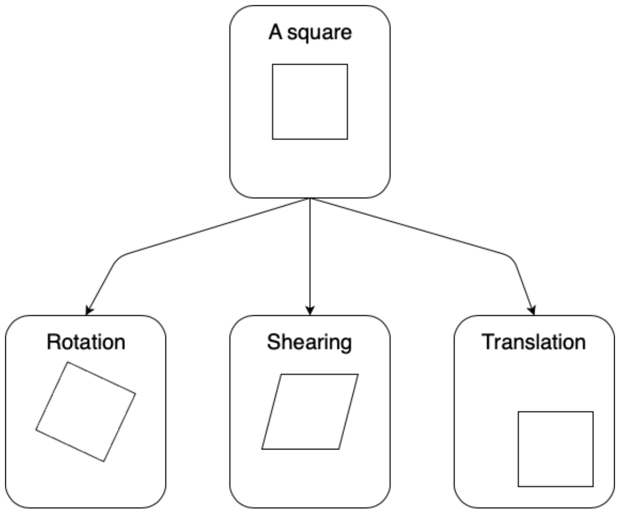


Figure 2: Shows graphically the three transformation functions that are applied on the images for the affine transformed test sets.

4 Methodology

This section explains the methods that were used during the experiments. It is essential to know how SCNNs work and how this structure can be improved by the capsule philosophy. Therefore, some information regarding capsule networks is explained in this section including possible structures that can incorporate the advantages of capsule networks into the Siamese network. Selecting appropriate hyperparameters to maximise accuracy is also an important part of machine learning models in general, and this section explains which methods were used to find these hyperparameters.

4.1 Siamese convolution neural network

Siamese neural networks were used in 1994 by Bromley et al. (1994) to verify written signatures. The network consists of two identical subnetworks that are joined by their output. The subnetworks extract features from two signature images and join both subnetworks by computing a distance metric of the neuron outputs of the highest-level feature representation. Verification is done by assessing a certain threshold for the output of the distance metric. In addition to applying Siamese structures to verify signatures, it has recently been successfully used for image verification by Taigman et al. (2014) and Schroff et al. (2015).

Further, in Figure 3, a Siamese neural network with one hidden layer and logistic output p is depicted. It shows the structure of two subnetworks that are joined with a distance layer. One important aspect to notice is that both subnetworks are symmetric in their structure and share the same weights. This ensures that two extremely similar images will lie close in the feature space, because both subnetworks compute the same function. Another advantage of symmetric twin networks is that it does not matter in which subnetwork either of the two images is imputed to get the same output.

Many variations are possible in Figure 3, such as different distance metrics, multiple hidden layers, activation functions, fully connected layers before and/or after the distance layer, and different loss functions. However, one undeniable improvement is the use of convolution layers, which have achieved excellent results in computer vision applications (Simonyan and Zisserman (2015); Krizhevsky et al. (2012)). This is because of their weight-sharing property, which greatly reduces the number of weights that need to be learned and consequently decreases overfitting. Hence, they have a built-in regularisation.

Another popular implementation is max-pooling. This is one specific tool from a set of subsampling tools, one that ensures that the network becomes invariant to small translations of the image input by reducing the resolution of the feature maps (Scherer et al. (2010)). Scherer et al. (2010) and Britz (2015) compared certain CNN structures with different pooling settings and datasets.

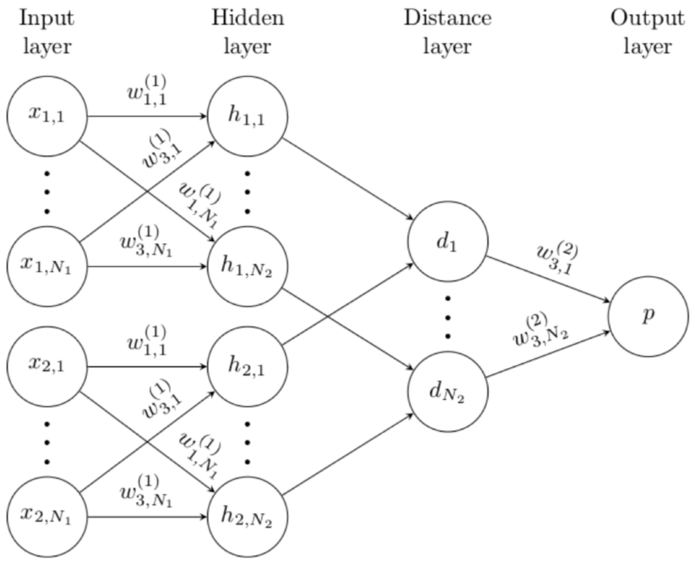


Figure 3: A simple Siamese neural network with one hidden layer, a distance layer, and logistic output.

They found out that max-pooling works better by faster convergence and lower error rates than subsampling and average pooling. Pooling is in most cases used together with convolutions, in which a convolution is performed first, followed by a pooling computation. This combination of convolutions and max-pooling, which are called convolutional neural networks (CNNs), is very successful and is also used during this research to implement a SCNN (Simonyan and Zisserman (2015); Krizhevsky et al. (2012); Koch et al. (2015); Taigman et al. (2014) Schroff et al. (2015)).

Different structures regarding the distance layer and loss function are also possible. In Figure 3, the structure shows a logistic output. A possible configuration is that of a L1 distance layer (Manhattan distance) with a weighted sigmoidal output p. The prediction vector would be given by $p = \sigma(\sum_{j=1}^{N_2} w_{3,j}^{(2)} |h_{1,j} - h_{2,j}|)$, with σ as a sigmoid function (Koch et al. (2015)). A natural loss for this structure would be a cross-entropy function (Koch et al. (2015)). However, other structures are also possible. A so-called constructive loss function is able to use any distance score and uses a margin to separate dissimilar input images as much as possible (Hadsel et al. (2005)).

Thus, many different variations are possible for the Siamese network. Some can be chosen empirically or by domain knowledge. For others, it is better to decide by rendering a hyperparameter search. However, there are some limitations in the general structure of CNNs that are inherited by SCNNs. Recently, a paper was introduced that tries to solve some of these limitations (Sabour et al. (2017)).

4.2 Capsule networks

It is first necessary to explain what those limitations are before speaking or mentioning any improvements that can be made. The structure of CNNs is built out of sequential layers of convolutions that preserve translation symmetry of the images. When the input shifts, the output also shifts but stays otherwise unchanged, or in other words, $\Phi(Tx) = T\Phi(x)$, where Φ is the computation of the convolution and T is the translation (Linmans et al. (2018)). However, these convolutions are only translation-equivariant, whereas there exist many more transformations that do not change the meaning of an entity that is presented in the image. For example, rotation of an entity in an image does not change the meaning of said entity. This, of course, imposes a problem and is countered by extending the training data by augmentation and extending the model to give it the ability to learn all these different transformations. However, this does not guarantee generalisation of transformations to the test set, as is depicted in Figure 4 (Linmans et al. (2018)). A better method is to construct a network that is not only translation-equivariant but also equivariant to other transformations.

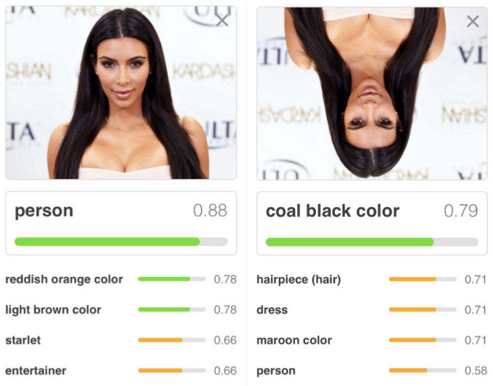


Figure 4: A CNN trained for classification is able to classify a person correctly on the left image. However, it is unable to classify the image correctly after a rotation transformation of 180 degrees on the image.

Subsampling or pooling also creates problems. It is a crude method to recognise objects without exactly knowing where they are, i.e., a method to make CNNs translation-invariant for classification (Hinton et al. (2011)). This means that the network loses spatial information of local features and consequently loses spatial relationships between local features. This is bad when, for example, dealing with facial identity recognition, which requires spatial relationships between eyes, nose, and mouth. An example is shown in Figure 5, where the relative spatial relationship between eyes, nose, and mouth are wrongly placed for a person.

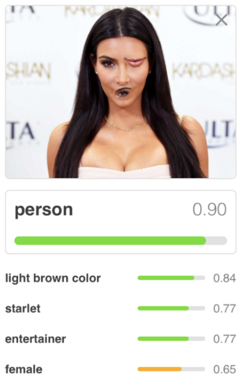


Figure 5: A CNN imputed with an image that has wrongly placed physical face features still classifies the image as a person.

To improve these flaws, the focus needs to lie on making the network not only translation-equivariant but also equivariant to other transformations and implementing a pooling scheme that does not lose spatial relationships of local features. Recently, a paper was introduced by Sabour et al. (2017) that aims to do that. The major difference that was proposed by Hinton et al. (2011) was to not only use scalars as output of neural activity, but to use vectors (capsules). These vectors represent different properties (referred to as instantiation parameters by Sabour et al. (2017)) of a particular entity that is present in the image. These properties can include the pose (position, size, orientation), deformation, velocity, texture, etc. One important and special property is the existence of the entity in the image, which in CNNs is referred to as the scalar neural activity output. In Sabour et al. (2017), the magnitude of the vector is used to replace the scalar neural activity of CNNs and the direction of the vector is used to present properties of an entity. A squash function is used, which is shown in Equation 1, to make sure the magnitude is between 0 and 1. Hence it represents a probability of the presence of an entity.

This structure allows a vector ($\vec{u} \in R^d$) to orientate through their respective d dimensional space with a fixed magnitude. This means that transformations are independent of scalar neural activity and hence achieve equivariance to transformations.

$$\vec{u} = \frac{\|\vec{v}\|^2}{1 + \|\vec{v}\|^2} \frac{\vec{v}}{\|\vec{v}\|} \tag{1}$$

The CapsNet is shown in Figure 6. A convolution is applied followed by a ReLu activation function. Thereafter, another convolution follows, which creates the capsules in the PrimaryCaps layer. In Figure 6, these exist out of 32 blocks of capsules $\in R^8$. Hence, the capsules are constructed of the feature maps from the second convolution. This also means that the instantiation parameters are learned by the model itself from the variations of the input images during training.

Routing-by-agreement is applied between the PrimaryCaps and DigitCaps layer in which predictions are made with lower-level features (capsules from PrimaryCaps) for higher-level features (DigitCaps capsules). When multiple lower-level features are in agreement to the same higher-level feature, a high-throughput (magnitude) is directed to that higher-level feature. In other words, the presence of the higher-level feature in the image is highly likely. The weights to make predictions from lower level to higher level features are learned with back propagation. The agreement algorithm between these layers has the same principles as k-means clustering and can be found in Sabour et al. (2017). Another property is that routing-by-agreement removes the local translation information and has the ability to store this information in one of the instantiation parameters of the capsules in the DigitCaps layer of Figure 6. From that point, it is very easy to make the network completely transformation-invariant by only considering the magnitude of the DigitCaps capsules. In this figure, only 10 capsules are left, all of which represent a specific higher-level feature. The magnitude of every capsule can be calculated to determine the probability of the higher-level feature in the input image.

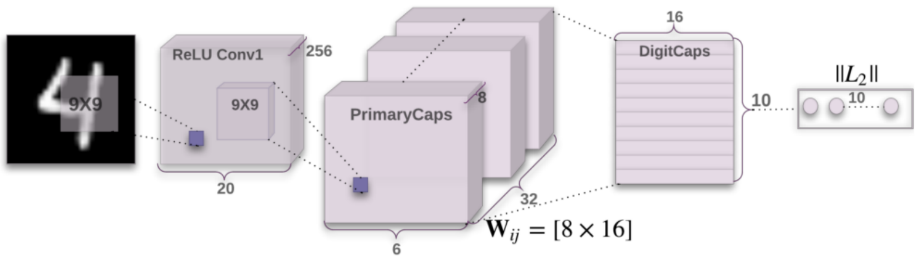


Figure 6: CapsNet architecture from Sabour et al. (2017) with which the MNIST dataset is learned and predicted. The routing-by-agreement is done between the PrimaryCaps and DigitCaps layer. The output of the routing-by-agreement is the DigitCaps layer with 10 capsules that refer to the 10 classes from MNIST.

4.2.1 Siamese capsule networks

The idea is to construct a combination of Siamese networks and capsule networks in such a way that the properties of equivariance and invariance of capsule networks are incorporated. In Figure 7, a picture of a deep Siamese neural network is shown. It exists out of two deep neural subnetworks. The idea is to replace both of these networks with a capsule network that exists out of some convolution layers, a primary capsule layer, and a routing-by-agreement layer without the reconstruction network that is used by Sabour et al. (2017). A similar structure was proposed by O'Neill (2018), where the Digitcap layer from Figure 6 is vectorised and imputed into the concatenation (i.e., distance layer) layer of Figure 7. The downside of this implementation is that the instantiation parameters are forwarded, wherein the Siamese capsule network loses its equivariance property after this layer. Rotating or other transformations on the input image will have a direct effect on the fully connected layers of the Siamese capsule network. Hence, another proposition is to forward only the magnitude of the DigitCaps capsules so that equivariance and the advantages of robustness against affine transformation might not be lost. This network is named the Siamese capsule network with magnitude or SCNM.

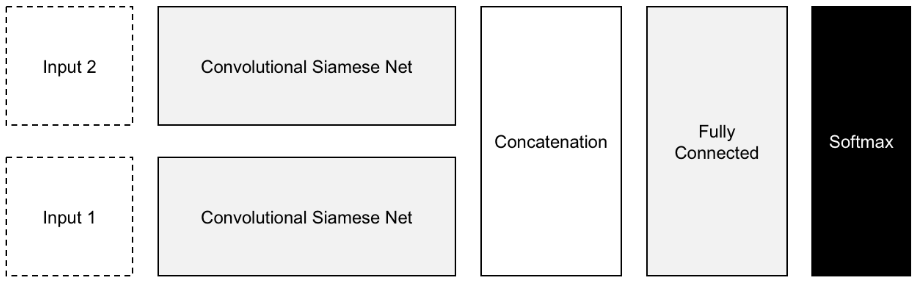


Figure 7: Architecture of a deep Siamese neural network with two CNN subnetworks.

4.3 Hyperparameter search

A hyperparameter search is very important in comparing and evaluating different models with each other. Melis et al. (2017) showed that in the neural language models community, older models seemed to outperform newer models by conducting a better and more extensive hyperparameter search. In this research, two algorithms are used to search these optimal hyperparameters.

4.3.1 Bayesian optimisation

Some of the most popular hyperparameter tuning methods are grid search and random search. However, these brute force methods are very expensive and waste time by trying out silly hyperparameter configurations. Bayesian search is a method that offers some of the most efficient approaches in terms of function evaluations and is therefore particularly useful when evaluations are costly, which is the case with deep learning models (Brochu et al. (2010)). Hence in this research Bayesian optimisation is used to derive optimal hyperparameter settings.

For the first N iterations, random search is applied, which is visible in Algorithm 2. A hyperparameter set is generated from prespecified boundaries and inserted into the function f, from where the function is evaluated (i.e., accuracy on validation set). These random iterations are used to build an initial Gaussian process (GP) in the next phase.

Then for the other $M - N$ iterations, the Bayesian loop from Algorithm 3 is used. Using the derived observations from Algorithm 2, a GP with which new observations are estimated is built. It returns a mean and variance of a univariate Gaussian at new point x. This means that $f(x) \sim N(\mu, \sigma^2)$, with estimated μ and σ^2. An acquisition function is used to derive which point x is valuable to evaluate next. For this, the expected improvement (EI) acquisition function is used and is defined as $u(x) = -EI(x) = -E[f(x) - f(x^+)]$, where x^+ is the best point observed so far with respect to $f(x)$. After the iterations are finished, the hyperparameter set x, which maximises the function $f(x)$, is chosen as the final hyperparameters.

Algorithm 2 Generate N random evaluations

```
1: for $i = 1 : N$ do
2:     $x_i = random(x)$ {Generate random hyperparameter set}
3:     $y_i = f(x_i)$
4: end for
```

Algorithm 3 Bayesian optimisation loop

```
1: for $t = N : M$ do
2:     Given observations $(x_i, y_i = f(x_i))$ for $i = 1 : t$ build a Gaussian process
       (GP) for the objective $f$.
3:     Find $x_{t+1}$ by optimising the acquisition function over the GP: $x_{t+1} = \underset{x}{\operatorname{argmax}}\, u(x)$.
4:     Sample the next observation $y_{t+1}$ at $x_{t+1}$ (i.e., $y_{t+1} = f(x_{t+1})$).
5: end for
```

		train/val/test size	Hyperparameter iterations	Epochs	Affine transformations on test set
SCNN	Experiment 1	10,000/3,000/3,000	40	10	No
	Experiment 2	10,000/3,000/3,000	40	10	Yes
SCN	Experiment 1	10,000/3,000/3,000	40	10	No
	Experiment 2	10,000/3,000/3,000	40	10	Yes
SCNM	Experiment 1	10,000/3,000/3,000	40	10	No
	Experiment 2	10,000/3,000/3,000	40	10	Yes

Table 1: Shows the experimental settings that are performed with the three different Siamese structures.

5 Experiments

Three Siamese neural network structures are constructed to research whether a different implementation of a SCN, named SCNM, is able to outperform an implementation that is based on O'Neill (2018)'s SCN structure and Koch et al. (2015)'s SCNN structure, with respect to robustness to affine transformations. In this section, the experimental setup and the precise use of models are explained.

5.1 Experimental setup

Two experiments are conducted, but first hyperparameter search is carried out for all three models. This is done with 40 iterations of the Bayesian algorithm from section 4.3, from which the first 10 iterations are generated with random Algorithm 2. In every iteration, 10,000 training pairs and 3,000 validation pairs are sampled using Algorithm 1. The model structures with optimal hyperparameters from Bayesian optimisation is depicted in the next two sections. The number of epochs is set at 10 for every training cycle. The main motivation to choose these values is computational costs.

Then, experiment 1 is conducted. The found optimal parameters are used and 10 datasets are constructed for both training and testing. Again, training sets with 10,000 pairs and testing sets (normal test sets) are used, with 3,000 pairs chosen from the evaluation set. Every model is trained and tested on each of these datasets. This means that the three different model architectures are trained on the same training and validated on the same test sets. This is chosen to control for experimental variability, such that factors as random dataset creation and random affine transformations do not affect the results between the three architectures.

The same data generating structure is applied to the second experiment that researches the robustness to affine transformations. However, after the 10 test sets (affine test sets) are generated, affine transformations are applied using the methods mentioned in section 3.2. In Table 1 the different settings of the two experiments are shown.

5.2 Siamese neural network architecture

To create a baseline, the same SCNN architecture from Koch et al. (2015) is implemented. It has 5 layers in both subnetworks, a manhattan distance layer to connect both subnetworks and a sigmoidal output unit. A regularised cross-entropy loss function is used. The learning rates are identical for every layer with a rate of 5.49e-5. The Adam optimiser was used to update the weights, for which $\beta_1 = 0.5745$ and $\beta_2 = 0.9999$. A batch size of 48 is used.

In both subnetworks, the first four layers have a convolution followed by a rectified linear (ReLu) unit. The ReLu units are followed by max pooling with a filter of 2 x 2 and stride of 2 for only the first three layers. The feature maps after the ReLu in the fourth layer are vectorised and imputed into the fifth layer, which is a fully connected layer with 1536 nodes and a sigmoid activation function. The regularisation is set at 4.8e-4. Then, the Manhattan distance of both outputs of the subnetworks from the fully connected layers are taken in the distance layer. This result is forwarded to the final sigmoidal output unit.

The filters and regularisation values, in the subnetworks, varies per layer. Every stride of the convolutions filters are set at 1. The convolution of the first layer has a filter of 5 x 5, 64 channels, and regularisation is set at 1.2e-6. The convolution of the second layer has a filter of 12, 112 channels and regularisation is set at 1e-8. The third layer has a convolution with a filter of 7 x 7, 96 channels and the regularisation is set at 2.73e-5. The convolution of the fourth layer has a filter of 2 x 2, 144 channels, and a regularisation set at 0.0056. In total, including shared parameters, this architecture has 15,976,705 parameters.

5.3 Siamese capsule network architectures

Two Siamese capsule networks are constructed. Both models use the exact same implementation from Sabour et al. (2017) until the DigitCaps layer in the Siamese subnetworks. Only the reconstruction layer is excluded. The first one is based on O'Neill (2018)'s structure, is called SCN and the structure is shown in Figure 8. The second network is the SCNM. It includes the magnitude (i.e., Euclidean distance) of the capsules from the DigitCaps layer. The structure of this network is shown in Figure 9. Both models have a regularised cross-entropy objective function, with regularisation in the ReLu Conv1, PrimaryCaps, and DigitCaps layer. The SCN also has a fully connected layer that is regularised. Both models are learned using the Adam optimiser with a batch size of 48 and equal learning rate for all layers.

5.3.1 Siamese capsule network with fully connected layer

The SCN in Figure 8 shows that the first convolution of a subnetwork has a filter of size 10 x 10, a stride of 3 and is followed with a ReLu unit. The regularisation is set at 8.097e-6. The PrimaryCaps layer is constructed by a convolution with a filter of size 14 x 14, 4 times 48 channels and a stride of 3. The regularisation for these weights is set at 1.4e-8.

The weight matrixes to predict the DigitCaps capsules with the PrimaryCaps have a dimension of 4 x 8. In total, every PrimaryCaps capsule makes 32 predictions, which is shown by the 32 capsules in the DigitCaps layer. The regularisation value for the weight matrix parameters is set at 4.705e-6. The routing-by-agreement makes two rounds to calculate the DigitCaps capsules.

The capsules from the DigitCaps layer are vectorised and imputed into a fully connected layer of size 64, with a regularisation value of 4.17e-4. From here on, the Manhattan distance is taken of the output of both subnetworks. This result is imputed into a sigmoidal unit. A learning rate of 0.0161 is used with β_1 is 0.9999 and β_2 is 0.9647. This SCN structure has a total of 6,403,713 parameters.

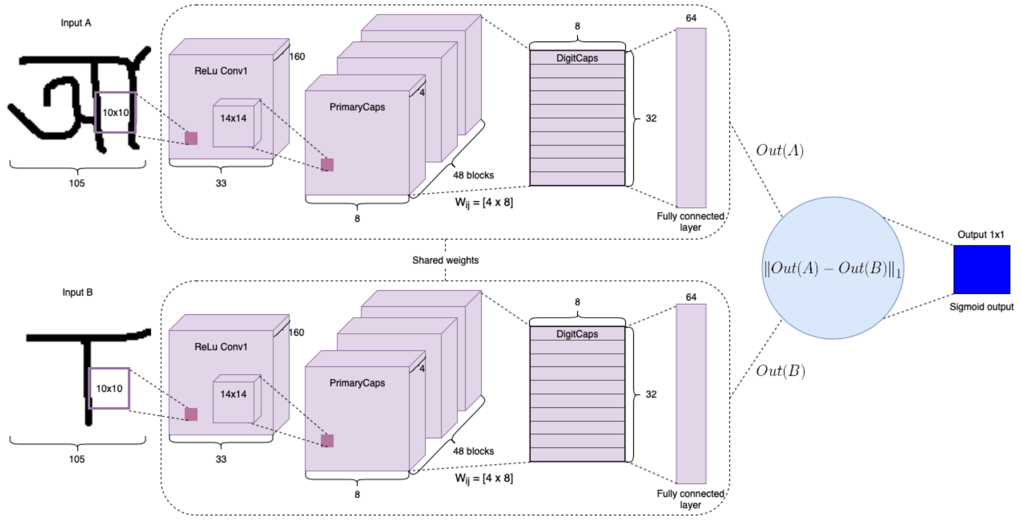


Figure 8: Structure of Siamese capsule network with a fully connected layer at the end of both subnetworks.

5.3.2 Siamese capsule network with magnitude layer

In Figure 9, the second model is depicted. It has in general the same structure as the model in Figure 8, but is different in the final layer of both subnetworks. Instead of a fully connected layer, it calculates the Euclidean distance of the DigitCaps capsules.

The first convolution has a filter of 15 x 15, a stride of 3, a regularisation of 1e-8 and is followed by a ReLu unit. The PrimaryCaps layer is constructed by a convolution with a stride of 2 and regularisation for these weights is set at 1.185e-5. The regularisation is set at 1e-7 for the weight matrix parameters. The routing-by-agreement algorithm makes four routing rounds. A learning rate of 0.00288 is used with a β_1 of 0.00986 and β_2 of 0.398. The model has 4,924,081 parameters.

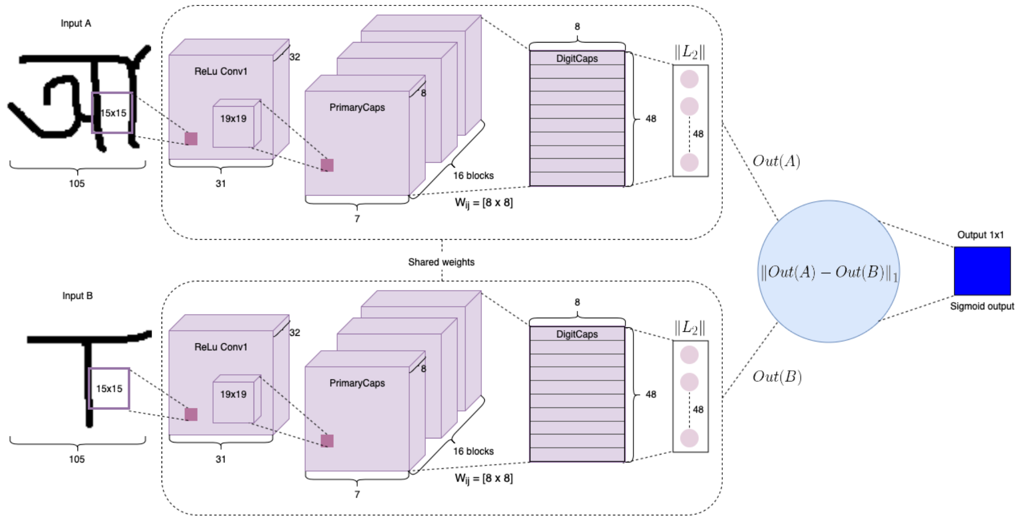


Figure 9: Structure of Siamese capsule network with a magnitude layer at the end of both subnetworks and without a fully connected layer.

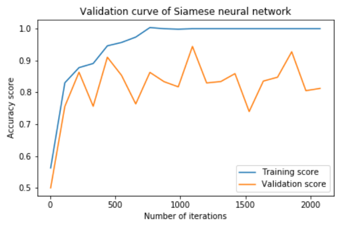
5.4 Evaluation measures

A simple accuracy measure is used for evaluation. The data pairs are equally distributed between negative and positive samples. Hence, accuracy is a fine measure to use. The non-parametric Wilcoxon matched pairs signed rank test is used to infer if two architectures are performing equal or different. Depending on the assumption, which is formed by graphical analysis, one-sided or two-sided hypothesis tests are performed.

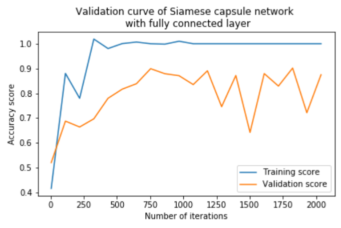
6 Results

This section gives the results of both experiments as well as statistical inferences between the model architectures.

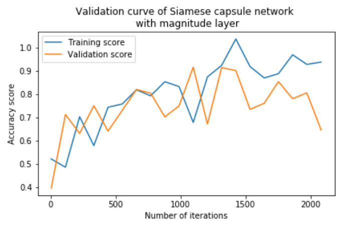
In Figure 10, some validation curves are shown. It is visible that all three structures are learning, as both training and validation accuracy increases with more iterations. However, it seems that there exists some overfitting with the SCNN



(a) SCNN



(b) SCN



(c) SCNM

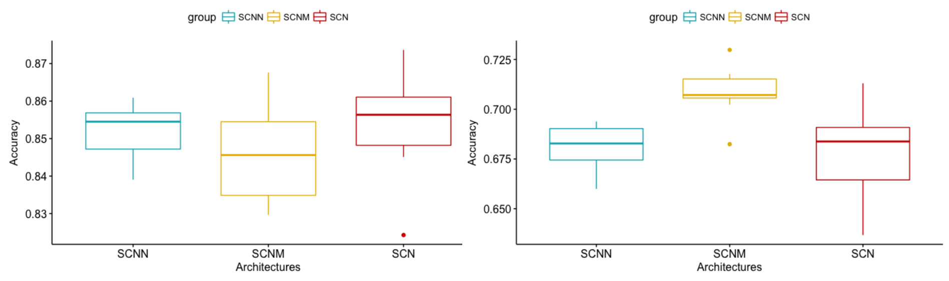
Figure 10: Validation curves for the three model architectures. Every iteration defines a batch calculation for either training and validation samples. Hence, variation is somewhat high. The data are smoothed for better visual representation.

and SCN architectures. This seems not to be the case for the SCNM architecture in Figure 10c.

6.1 Performance comparison

In Figure 11, the results on the 10 test set datasets are depicted. In Figure 11a, the results of the three architectures on the normal datasets are shown. It appears as though performance is approximately equal between the three structures. In Table 2a, and b, the results of the Wilcoxon rank tests are shown to see if one of the architectures performs significantly differently. For all the tests, an α of 0.05 is used.

The boxplots in Figure 11b shows the results of the models on the test datasets on which affine transformations are applied. There is an understandable decrease in performance compared to Figure 11a. The SCNM performs on average 2.7% and 3.0% better than the SCNN and SCN, respectively.



(a) Normal test set

(b) Affine transformation test set

Figure 11: Boxplot of the three architectures with the performance on the 10 normal and affine test sets.

Table 2: Statistical performance between the three architectures.

Test	**W**	**P-value**
SCNN = SCNM	39	0.28
SCNM = SCN	15	0.23
SCNN = SCN	20	0.49

(a) Statistical tests on the normal test datasets.

Test	**W**	**P-value**
SCNN < SCNM	0	**9.8e-4**
SCNM > SCN	51	**6.8e-3**
SCNN = SCN	28	1

(b) Statistical tests on the affine transformed test datasets. The first two rows are one-sided tests.

7 Discussion

Looking at the performance of the three architectures in Figure 11a, it shows they perform equally well. This assumption is strengthened by accepting the null hypothesis for the three tests in Table 2a. Also, it is assumed that the capsule network structures require less data to perform equally well compared to classical CNNs, because data augmentation is less necessary. We cannot really conclude this for the Siamese structures, because the three structures perform equally. However, the SCNN architecture has approximately three times as many parameters than the SCN and SCNM architecture. Hence, the Siamese capsule network structures are able to achieve the same performance with considerably fewer parameters. A side note is that it seems that the SCNN and SCN are slightly overfitting. Reducing this by using less convolutional layers for the SCNN or not using the fully connected layer for the SCN could have an impact on the conclusions.

In Figure 11b, the results of robustness against affine transformations is shown. Interestingly, the SCNM seems to do better than the other two structures. This is also validated by the tests in Table 2b, which show significantly better performance of the SCNM against the other architectures. Therefore, it looks as though the SCNM is more robust against affine transformations than the SCNN and SCN architectures.

These results look promising. However, in the original CapsNets paper, the CapsNets model was implemented including a reconstruction layer. This provided regularisation, but also enforced the network to store transformation information into the instantiation parameters. The SCNM and SCN structures do not allow for such a layer to work in the way it is implemented in the CapsNets model. Hence, if something can be found that works the same it can boost performance.

More research needs to be conducted to see if these results also apply with using more training epochs, more data, and different datasets. The obstacle at this moment is that capsule networks are very demanding on memory and are therefore not easy to scale, especially in the Siamese structure, for which hard boundaries needed to be set to not get out-of-memory issues.[2] Hence, research should also focus on making the capsule network model more memory efficient.

8 Conclusion

This research focuses on looking for a different implementation of a SCN that is able to outperform a SCN based on the structure from O'Neill (2018) and SCNN from Koch et al. (2015) with respect to robustness to affine transformations. The Omniglot dataset was used to investigate this, and a different implementation than SCNs was tested. This resulted in the SCNM structure that outperformed both the SCNN and SCN structure on robustness against affine transformations and it performed equally well on the test sets without affine transformations using roughly one-quarter the number of parameters compared to that of the SCNN.[3]

References

Baldi, P. and Y. Chauvin (1993), "Neural networks for fingerprint recognition."

Britz, D (2015), "Understanding convolutional neural net- works for nlp." `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/`. [Online; accessed 3-September-2018].

Brochu, E, V.M. Cora, and N Freitas (2010), "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning."

Bromley, J, I Guyon, and Y LeCunn (1994), "Signature verification using a siamese time delay neural network."

[2]An NVIDIA K80 GPU with 12 Gb of memory was used during this research.

[3]Tensorflow code is available on `https://github.com/mdeblaauw/research-paper`.

Chopra, S., R Hadsell, and Y LeCun (2005), "Learning a similarity metric disrciminatively, with application to face verification."

Cohen, T.S., M Geiger, and M Weiler (2016), "Group equivariant convolutional networks."

Fei-Fei, L, R Fergus, and P Perona (2006), "One-shot learning of object categories."

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016), *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hadsel, R, S Chopra, and Y LeCun (2005), "Dimensionality reduction by learning an invariant mapping."

Hinton, G.E., A Krizhevsky, and S.D. Wang (2011), "Transforming auto-encoders."

Honari, S, J Yosinski, P Vincent, and C Pal (2016), "Recombinator networks: Learning coarse-to-fine feature aggregation."

Koch, G, R Zemel, and R Salakhutdinov (2015), "Siamese neural networks for one-shot image recognition."

Krizhevsky, A, I Sutskever, and G.E. Hinton (2012), "Imagenet classification with deep convolutional neural networks."

Lake, Brenden M, R Salakhutdinov, Jason Gross, and Joshua B Tenenbaum (2011), "One shot learning of simple visual concepts."

Linmans, J, J Winkens, B.S. Veeling, T.S. Cohen, and M Welling (2018), "Sample efficient semantic segmentation using rotation equivariant convolutional networks."

Melis, G, C Dyer, and P Blunsom (2017), "On the state of the art of evaluation in neural language models."

O'Neill, J (2018), "Siamese capsule networks."

Sabour, S, N Frosst, and G.E Hinton (2017), "Dynamic routing between capsules."

Scherer, D, A Muller, and S Behnke (2010), "Evaluation of pooling operations in convolutional architectures for object recognition."

Schroff, F, D Kalenichenko, and J Philbin (2015), "Facenet: A unified embedding for face recognition and clustering."

Simonyan, K and A Zisserman (2015), "Very deep convolutional networks for large-scale image recognition."

Taigman, Y, M Yang, M.A. Ranzato, and L Wolf (2014), "Deepface: Closing the gap to human-level performance in face verification."

Appendix

A.1 Generating datasets

This section provides the full code of generating dataset pairs from the Omniglot dataset with balanced positive and negative pairs.

```
def create_train_data(size, s='train'):
    #get train data and shape
    X=data[s]
    n_classes, n_examples, w, h = X.shape

    #initialize 2 empty arrays for the input size in a list
    pairs=[np.zeros((size, h, w,1)) for i in range(2)]

    #initialize vector for the targets
    targets=np.zeros((size,))

    for x in range(size):
        #randomly sample one class (character)
        category = rnd.choice(n_classes,1,replace=False)
        #randomly sample one example from class (1-20 characters)
        idx_1 = rnd.randint(0, n_examples)
        pairs[0][x,:,:,:] = X[category, idx_1].reshape(w, h, 1)
        #randomly sample again one example from class and add last class
        # ..with modulo to ensure not same class pairs are created
        idx_2 = (idx_1 + rnd.randint(0, n_examples)) % n_examples
        #pick images of different class for 1st half and same
        # ...class for 2nd half
        if x >= size // 2:
            category_2 = category
            targets[x] = 1
        else:
        #add a random number to the category modulo n classes to ensure 2nd
        # ..image has different category
            idx_2 = rnd.randint(0, n_examples)
            category_2 = (category + rnd.randint(1,n_classes)) % n_classes
            targets[x] = 0
        pairs[1][x,:,:,:] = X[category_2,idx_2].reshape(w, h,1)

    return pairs, targets
```