Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Håvard Bjørnøy

Investigating the Effect of Samples per Class and Number of Classes for Capsule Networks' Performance

Master's thesis in Computer Science
Supervisor: Keith Downing
June 2020

NTNU
Norwegian University of Science and Technology

Håvard Bjørnøy

Investigating the Effect of Samples per Class and Number of Classes for Capsule Networks' Performance

Master's thesis in Computer Science
Supervisor: Keith Downing
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Summary

Image analysis is becoming ubiquitous in everyday services as for example unlocking your phone with face recognition, QR codes detection, and photo enhancer algorithms. Big industries like autonomous vehicles, autonomous warehouses, assembly lines and medical diagnosis tools are dependant on accurate robust solutions for their image analysis models. Different convolutional neural networks (CNNs) are the backbone of most of these modern applications. Despite CNNs great success, they have deficiencies in modelling spatial relationships between components and struggle to extrapolate on concepts like rotation. A new architecture, Capsule networks, aimed to tackle these deficiencies is proposed by Hinton et al. (2011) and Sabour et al. (2017).

A capsule network group neurons as units (capsules) that can represent if an object(or part of an object) exists as well as it's properties (rotation, hue, brightness). The capsules also have dynamic connections that *agree* when a capsule is related to a capsule in the layer above. These changes enable capsule networks to exploit spatial relationships between components of objects and generalize better on new instances with slightly different properties. If the capsule network generalizes better than regular CNNs, a promising application for capsule networks are small datasets. This thesis investigates capsule networks performance on subsets of the MNIST dataset with few samples per class, ranging from 1 to 100 samples per class. In addition, a study of capsule networks performance on datasets with 2 to 1623 number of classes is carried out on the Omniglot dataset, specifically to discover how the capsule network performs and scales to this challenge. In both experiments in this thesis, a CNN is used as a baseline.

The first experiment compared performance of the capsule network from Sabour et al. (2017) (CapsNet) and the CNN baseline from Sabour et al. (2017) on small datasets. The capsule network consistently outperformed the CNN baseline, in contrast with the results from previous discoveries by Schlegel et al. (2018) on the topic. The average difference in accuracy between the CapsNet and the CNN baseline for samples per class of 1,5,10, and 20 is 7%.

The second experiment compared the models from last experiment, modified for bigger input and outputs. The capsule network outperformed the CNN baseline on datasets with many classes. However, the CNN scaled much better w.r.t. model parameters than the capsule network. While the CNN parameters only increase by 3% from 2 to 1623 classes, the capsule network in this instance increased by 785%. The model was too big, so it had to be reduced in capacity to reduce the GPU memory so it would fit into limited resources of 16GiB GPU memory. For 400 classes, CapsNet achieved a 52.12% test accuracy, in contrast to the CNN baseline with a 20.63% test accuracy. The CapsNet model modified for 400 classes had 489% more model parameters than the compared CNN baseline. The unbalance in the number of parameters undercuts the side-by-side comparison of the models. Future work should investigate solutions to the scaling of capsule networks to many classes.

Sammendrag

Bildeanalyse er allestedsnærværende i daglige tjenester, for eksempel ansiktsgjenkjenning for å låse opp telefonen, QR-koder og algoritmer for bildeforbedring. Store industrier som autonome kjøretøy, autonome lager, samlebånd og medisinsk diagnose er avhengig av nøyaktige robuste løsninger for sine bildeanalysemodeller. Ulike convolutional nevrale nettverk (CNN) er sentrale komponenter i de fleste av disse moderne applikasjonene. Til tross for CNNs store suksess, har de problemer med modellering av romlige forhold og sliter med å ekstrapolere på konsepter som rotasjon. En ny arkitektur, kapselnettverk, som tar sikte på å takle disse manglene foreslås av Hinton et al. (2011) og Sabour et al. (2017).

Et kapselnettverk grupperer nevroner som enheter (kapsler) som kan representere om en gjenstand (eller del av et objekt) eksisterer så vel som dens egenskaper (rotasjon, fargetone, lysstyrke). Kapslene har også dynamiske forbindelser som er *enig* når en kapsel er relatert til en kapsel i laget over. Disse endringene gjør det mulig for kapselnettverk å utnytte romlige forhold mellom komponenter av objekter og generalisere bedre i nye tilfeller med litt forskjellige egenskaper. Hvis kapselnettverket generaliseres bedre enn vanlige CNN-er, er en lovende applikasjon for kapselnettverk små datasett. Denne masteroppgven undersøker ytelsen til kapselnettverk på undergrupper av MNIST datasettet med få prøver per klasse, fra 1 til 100 prøver per klasse. I tillegg blir en undersøkelse av kapselnettverkets ytelse på datasett med 2 til 1623 antall klasser utført på Omniglot datasettet, for å oppdage hvordan kapselnettverket presterer og skalerer til denne utfordringen. I begge eksperimentene i denne oppgaven sammenlignes kapselnettverk med en CNN grunnlinjemodell.

Det første eksperimentet sammenlignet ytelsen til kapselnettverket fra Sabour et al. (2017) (CapsNet) og CNN modellen fra Sabour et al. (2017) på små datasett. Kapselnettverket overgikk konsekvent CNN modellen, i motsetning til resultatene fra tidligere funn av Schlegel et al. (2018). Den gjennomsnittlige forskjellen i nøyaktighet mellom CapsNet og CNN modellen var på 7%

Det andre eksperimentet sammenlignet modellene fra forrige eksperiment, modifisert for større input og output. Kapselnettverket overgikk CNN modellen på datasett med mange klasser. Imidlertid skalerte CNN mye bedre m.t.p. modellparametere smmenligned med kapselnettverket. Mens CNN-parametrene bare øker med 3% fra 2 til 1623 klasser, økte kapselnettverket i dette tilfellet med 785%. Modellen var for stor, så den måtte reduseres i kapasitet for å redusere GPU-minnet, pga tilgjengelige ressurser. For 400 klasser oppnådde CapsNet testnøyaktighet på 52.12%, i motsetning til CNN-baseline med en testnøyaktighet på 20.63%. CapsNet-modellen modifisert for 400 klasser hadde 489% flere modellparametere enn den sammenlignede CNN modellen. Ubalansen i antall parametere gjør det vanskelig å sammenlignde modellene. Framtidig arbeid bør undersøke løsninger for skalering av kapselnett til mange klasser.

Preface

This thesis is written during the spring of 2020. It is my final assignment in the five-year master program *Computer Science* at the *Department of Computer Science (IDI)* at *Norwegian University of Science and Technology (NTNU)*. The thesis has forced me to dive into cutting edge knowledge in Machine Learning. It has been rewarding to see that many of the basic principles learnt throughout the years are relevant. However, I have also realized that it is important to see them questioned and challenged.

In my work for this thesis I get to acknowledge the revolutionary effect convolutional neural networks has had on the image processing field. At the same time as it is questioned for its exponential inefficiencies, and challenged with Capsule architecture and dynamic routing. In this thesis Capsule network is examined for it's promise and limitations with different applications.

I would like to show my appreciation to Keith Downing for his guidance throughout the thesis. By my side I had Hedda Hognedatter Bjørnebye Vik who had to listen to my thoughts and ideas. I thank her for her support and encouragement.

Håvard Bjørnøy,
Oslo, June 2020.

Table of Contents

Chapter 1

Introduction

This report looks into the progress in the Capsule network field, and its performance on datasets. Experiments are carried out on MNIST, a dataset of handwritten digits - as well as on a modified version of Omniglot, a dataset with 1623 different letters from 50 different alphabets. The aim is to investigate the performance of capsule networks on datasets with a small sample-size per class and on datasets with many classes.

1.1 Background and motivation

Many industries are now dependant on inferring knowledge from images or videos through object segmentation, tracking, and recognition; For example industries that work with autonomous vehicles, robotic assembly lines, robotic warehouse systems, medical diagnostics tools, or autonomous vacuum cleaners. There are functioning solutions to some of the applications in these industries, however, progress in the field of computer vision can improve efficiency and safety as well as save time and lives.

In 2012 there was a breakthrough in performance for Convolutional Neural Networks (CNNs), see the article by Krizhevsky et al. (2012). Following that, CNNs quite abruptly became the underlying model for best models in the majority of image and video applications. CNNs are flexible, simple, and most of all scales very well. They have many flaws, but flaws that are rather simple to overcome by augmenting the dataset to fit the application.

Despite all the successful industry applications there have been some critics of the architecture. CNNs fails to learn the spatial relationship between *higher-level features* like a whole car, and its *lower-level features* like its wheels, spoiler, and headlights. The consequence is that a CNN can classify an edited image of a car with the headlights as wheels, wheels as a spoiler, and a spoiler on the hood wrongly as a car. This happens despite the fact that the network may have seen 10 times more cars than a human. Unlike the network, a human can very quickly see that even though the edited ”car” has all the necessary components it does not have the correct spatial relationship between the components.

In the article written by Hinton et al. (2011), the idea of *capsules* was introduced to tackle the issue of spatial relationships. Capsules are more complex building blocks that can represent not only a feature but a feature's properties. This way the spatial relationship between higher or lower-level features can be represented. In the article by Sabour et al. (2017), dynamic routing between capsules (see Section 2.4.1) is introduced for the first time to update the weights between capsules so spatial relationships can manifest in a hierarchical structure between higher-level features and lower-level features. The practical effect of this change in architecture is a more data-efficient network that is robust to anomalous data. However, this comes at the cost of a more complex and slower to train network than other networks.

Capsule networks have so far shown great promise with performance close to other cutting edge models on the small, low-noise dataset MNIST (see Wan et al. (2013)). However, researchers that have applied the capsule network to noisy images of objects in natural environments have received poor results (see for example Xi et al. (2017)). Xi et al. (2017) and Rawlinson et al. (2018) points out that stacking more capsule layers decreases the performance. Recently, Rajasegaran et al. (2019) have overcome this challenge and increased performance on more complex data.

The thesis outline and explore the progress made in the field of capsule networks. The focus of the experiments in the thesis is the inherent limitations the original capsule network has with respect to data scarcity and a bigger output-space.

1.2 Research goals and questions

The goal of this thesis is to investigate the performance CapsNet has on datasets with few samples per class and with increasingly more classes. The following research questions (RQ) guides the thesis:

RQ1: Will CapsNet perform better than a CNN on a datasets with few samples per class?
RQ2: Will CapsNets perform better than a CNN on datasets with many classes?

1.3 Research approach

The capsule network explored in this thesis is modeled after the original model in the article by Sabour et al. (2017). Iwasaki (2018) has implemented a bare-boned GPU-enabled version of the capsule network in PyTorch. In this paper, that implementation is modified to dynamically fit differently sized inputs as well as differently sized outputs. A regular convolutional neural network modeled after Sabour et al. (2017) baseline is implemented as a baseline, alongside the capsule network, for comparison. The batch size as well as the regularization is adjusted to adapt to memory limitations and different regularization needs.

An analysis- and plotting toolbox is developed to analyze the performance of different models applied to different datasets. The models are trained to classify modified versions of the MNIST and Omniglot datasets. MNIST is a dataset with 28×28 images of handwrit-

ten digits, while Omniglot is a dataset with 105×105 images of letters from 50 different languages. The datasets are in turn modified to contain different numbers of samples per class and number of classes, and used in the experiments. The goal is that the results of the quantitative analysis can shed light on how well the capsule network scales to more complex issues.

1.4 Datasets

Except for the already introduced MNIST and Omniglot dataset that are used in the experiment part of the thesis, several datasets will be mentioned and referred to throughout the thesis. For readers who are not already familiar with these datasets, a short explanation of each of them is included here.

- MultiMNIST is a dataset with 70M 36×36 images two of digits from MNIST dataset superimposed on each other with 80% overlap. Published by Sabour et al. (2017)
- SVHN is a dataset with 99289 32×32 gray-scale images of StreetView House Numbers gathered and cropped from Googles street view. Published by Netzer et al. (2011)
- Fashion MNIST is a dataset of 70000 28×28 gray-scale images of fashion products of 10 different categories. Published by Xiao et al. (2017)
- SmallNORB is a dataset with 96×96 images of toy figures of 5 classes(four-legged animals, human figures, airplanes, trucks, and cars). The images were taken from many different viewpoints as well as different lighting conditions. Published by LeCun et al. (2004).
- CIFAR10 is a dataset with 60000 32×32 RGB images of 10 different classes including airplane, horse, dog, and truck. Published by Krizhevsky and Hinton (2009).
- ImageNet(ILSVRC) is a hierarchical image database with 15 million images with 22,000 categories. Annually a competition called ImageNet Large-Scale Visual Recognition Challenge(ILSVRC) is held using a subset of the ImageNet database; A dataset with 1000 classes, containing approximately 1.2 million natural images for training, 50,000 for validation and 150,000 for testing. The images have different resolutions but more than 256×256 pixels.. Published by Deng et al. (2009).

1.5 Contributions

The main contribution of this thesis is an insight into how capsule networks perform on datasets with different characteristics. The models and the analysis toolbox used in this project are in a public repository on Github, see Bjørnøy (2020). Instructions of how to set up the project with the correct packages are located in the `README.md` file in the repository.

1.6 Report overview

The thesis goes methodically over relevant background theory in Chapter 2. In Chapter 4 the state of the art results and insights relevant to the thesis will be briefly presented. The different models used in the experiments are explained in detail in Chapter 5. The experiments, including all the empirical data, are presented and commented in Chapter 6. In Chapter 7, a more in-depth discussion of the result will take place. Finally, the conclusions reached from the thesis is presented in Chapter 8.

1.7 Summary

CNNs are the industry standard for image analysis applications. However, their poor ability to learn spatial relationships between components of an object has driven the field to invent an alternative that tackles this shortcoming. The alternative, Capsule networks, show promise with regards to capturing spatial relationships. Nevertheless, Capsule networks have their own shortcomings that need to be explored more.

The goal of the thesis is to compare CNN's and Capsule network's performance on datasets with vastly different characteristics. Characteristics like how many samples exist per class, and how many classes there are. The approach is to add functionality to an already existing bare-boned implementation of a capsule network. The contributions from this thesis is an empirical comparative study of the CapsNet and the CNN architectures, as well as the open-source analysis toolbox used in the writing of this thesis.

Chapter 2

Background Theory

This chapter presents relevant background theory. Section 2.1 introduces artificial neural networks, while Secton 2.2 explaines regularization. These are fundamental consepts needed to understand convolutional neural networks and capsule networks, but readers who are well familiar with the topic are advised to skip these sections. Section 2.3 outlines convolutional neural networks, the predecessor of capsule networks. Finally capsule networks are presented in Section 2.4, before the chapter is summarized in Section 2.5.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are human-created neural networks. They were originally inspired by biological neurons in the human brain. A biological neuron is a processing unit, as illustrated in the top part of Figure 2.1. As shown in Figure 2.1, a biological neuron consist of a cell body with a nucleus, several filaments called dendrites, a single long filament called the axon connected to axon terminals and synapses connecting axon terminals to other neurons' dendrites. The reader is not expected to have a knowledge of how a biological neuron works in great detail; it simply functions as an analogy for those who have.

Artificial neurons, illustrated in the bottom part of Figure 2.1, are inspired by the biological neuron. The analogy is as follows: The dendrites together function as the input vector $\mathbf{x}$, where each dendrite i sends a scalar signal x_i from a lower level neuron i. The synapses are mimicked by weights in the matrix $\mathbf{W}$, with elements w_{ij}, and bias b_j. They decide what relation a higher level neuron j have to the input x_i by the affine transformation

$$a_{j|i} = w_{ij}x_i + b_j, \tag{2.1}$$

where $a_{j|i}$ simply is the output from the affine transformation. The cell body function

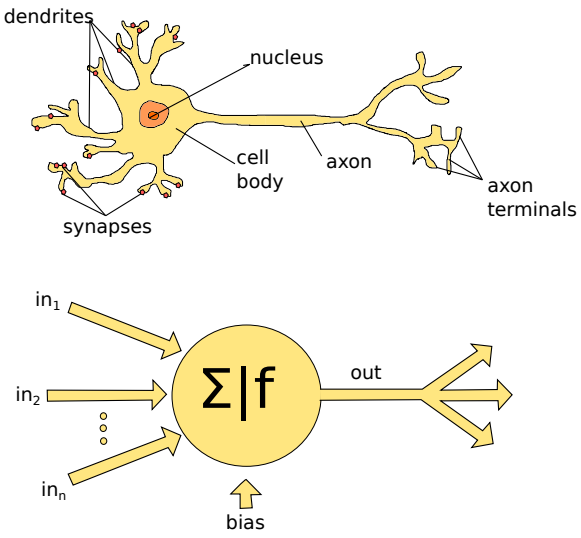


Figure 2.1: Biological neuron (top) and artificial neuron (bottom), the illustration points out the different parts and similarities between biological and artificial neurons. The flow of data is from the left(in_n:input, synapses and dendrites) to the right(out:output, axon terminals)

is modeled by a summation of $a_{j|i}$, where the output is z_j

$$z_j = \sum_i 1 \cdot a_{j|i} \tag{2.2}$$

and an activation function $f(z_j)$. The axon functions as an output h,

$$h_j = f(z_j), \tag{2.3}$$

a scalar that can be connected to other neurons. By stringing all of these operations together, one has what one would call a neural network *layer*. Although ANNs mimics some of the functions of biological neurons - it is a simplistic version, and in some dimensions, it differs. By stacking several layers together, and connecting the output of one layer to the input of another, a multi-layered neural network is formed. A neural network can be represented as a graph where each node is a neuron.

To be able to approximate non-linear functions, one needs to introduce non-linearity in the network; this is the task of the activation function. If the activation function is linear, the network can only express linear solutions. A popular activation function to use in neural networks is the rectified linear unit (ReLU: $a(x) = \max(0, x)$)

The loss function is a measure of the quality of the output. A typical scenario could be that you have input data X and labeled output data y; if so, one could do supervised learning. The goal of supervised learning is to learn from the input-output pairs to emulate the underlying model. Using the model, one could then predict the label $\hat{y}$ of input data. A typical loss function in the context of supervised learning is the mean squared error where the loss is the average of the sum of the squared residuals,

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2. \tag{2.4}$$

The idea is to adjust the weights in the network so that the loss is minimized. To do this, one needs the partial derivatives of the loss function w.r.t. the weights. The partial derivatives are found using backpropagation Kelley (1960). After that, an optimization algorithm, popularly the Adam optimizer Kingma and Ba (2014) is applied. In other words, the weights are updated iteratively in a promising direction by the optimizer, which calculates it with the partial derivatives, until a local optimum is reached.

The model of artificial neurons that are presented in this paper is the most widely used model. However, when designing artificial neurons and their interaction, there is a trade-off between how closely one wants to mimic biological neurons and performance. Biological neurons are complex, and with the complexity it is often more challenging to implement parallelized computing. Remarks on this topic are present in that of (Goodfellow et al., 2016, Chapter 1.2.1). For more detailed information about artificial neurons, loss functions, backpropagation, optimizers Goodfellow et al. (2016) is a good source.

The early deep neural networks had some problems that did not make them very robust. The deeper the networks became the more evident became the problems of vanishing and exploding gradients. In the early years, the $tanh()$, a function that can return very high gradients as well as near-zero gradients, was used as an activation function. When these gradients backpropagate with a fixed learning rate it is possible for the gradient to vanish,

making it hard to train the first layers. The gradient can also coincidentally explode, which can return NaN values because of numerical overflow.

ReLU, is an activation function that is known to combat both these problems because of it's stable derivative. ReLU combined with bad initialization of weights or high learning rates can render neurons useless because the weights are updated in such a fashion that the ReLU will never activate. If the ReLU never activates the derivative will always be zero making it impossible to update its weights. Today there are many different methods to combat these problems.

2.2 Regularization

When constructing a model, one wants the model to learn from a training dataset in such a way that the model can be applied to other, similar data points. However, there are many practical challenges when training a model. One of the bigger issues are cases where the model fits the data points instead of fitting the underlying model, called over-fitting. Overfitting generally occurs when one has a combination of a too small dataset as well as a model with high capacity. Regularization attempts to address the problem of over-fitting.

Regularization is any modification one makes to a model that intends to reduce its generalization error but not its training error. Generalization is a term that indicates how well a model performs on unseen data compared to the data the model trains on. Some regularization methods put constraints or penalties to a model, either incorporating prior knowledge or expressing a preference for simpler models.

2.2.1 L^1 and L^2 regularization

Both L^1 and L^2 regularization penalizes the parameters of the model using the norms with the same names. In a neural network, the weights are the parameters. The penalization, Ω, is added as a term in the loss function with a coefficient α which regulates its effect. The modified loss function, $\tilde{L}(\theta, X, y)$ is thus

$$\tilde{L}(\theta, X, y) = L(\theta, X, y) + \alpha\Omega(\Theta). \tag{2.5}$$

For L^2 regularization the penalization Ω is simply the 2-norm of the weights, which incentivizes the model to have low weights, adding bias for a simpler model. L^1 regularization uses the 1-norm, with a similar effect. However, since the L^1 derivative is constant, it forces more of the weights towards zero. This has the effect of making sparse weight matrices that can be leveraged to make learning algorithms faster.

2.2.2 Dropout

The term dropout refers to dropping out some units of the network. Dropout effectively removes some units of the network by multiplying the output with zero. Different units are randomly *removed* every run, given a user-defined probability. Dropout is computationally efficient, but still a great method for regularizing a model.

There exist multiple other regularization methods, such as data augmentation, multi-task learning, early stopping, sparse representations, and adversarial training, but only a

general understanding of the regularization concept and the methods are expected for this project paper.

2.3 Convolutional neural networks

Convolutional neural networks (CNNs) are networks in which the matrix multiplication in Equation (2.1) is replaced by a convolution in at least one of the layers. In its most general form, a convolution is an operation on two functions given a real-valued argument. In the context of neural networks a convolution, $K * I$, is more narrowly defined as

$$S_{ij} = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n). \tag{2.6}$$

Here, s_{ij} is the output of the convolution, K is the kernel and I is the input. S, I and K are matrices, so S_{ij} is the element on the ith row and jth column of S. The kernel's values represent the weights of the network. Figure 2.2 illustrates how a convolution from Equation (2.6) works. In the figure, a 2×2 kernel with values *w-z* is applied to a 3×4 matrix with values from a-l. The bottom part of the figure displays the resulting output, a 2×3 matrix.

The asterisk $*$ in Equation (2.6) denotes a convolution of the kernel K and Input I. Implementations of convolutions in neural networks popularly give the user control of different parameters that alter the behavior from the default convolutional operation. There is given a short introduction to some of the relevant hyperparameters one can change in the convolution module in the popular neural network library PyTorch (see Paszke et al. (2017a)). Read the Pytorch documentation and source code for 2D convolutional layers for more details. For a more in-depth explanation and great animations on this topic read Al-Rfou et al. (2016).

In the code in Figure 2.3, one can see the parameters defining the first layer of the LeNet-5 architecture (introduced in LeCun et al. (1998)). The *layer* object defined in Figure 2.3 can then later process the input, a 32×32 gray-scale image. Since a grey-scale image only has one *feature*, light intensity, the argument `in_channels` is set to 1. If the images had been in RGB-code, it would contain information in three features (red, green, and blue), meaning the input would have had three channels. The argument `out_channels` can be determined by the user, as any positive integer without any constraints. The amount of channels (both in and out) denotes how many *feature maps* there are in the input and output of the convolution. A feature map is the output S from Equation (2.6). Thus, layer defined in Figure 2.3 would initialize six kernels, which again produces six feature maps. The `kernel_size` defines the length of each dimension of the kernel. In Figure 2.2 you can see a convolution with `kernel_size = (2,2)`. The `kernel_size` controls how many weights there are in the kernel and indirectly how big a feature can be. It also affects how big the output feature map becomes.

The next argument in Figure 2.3, `padding`, is a method of adding additional border units to the input volume. With Zero-padding, the implementation in PyTorch would pad a number of extra zeroes around the volume. A 32x32 input with `padding=2` would become a 36x36 input.

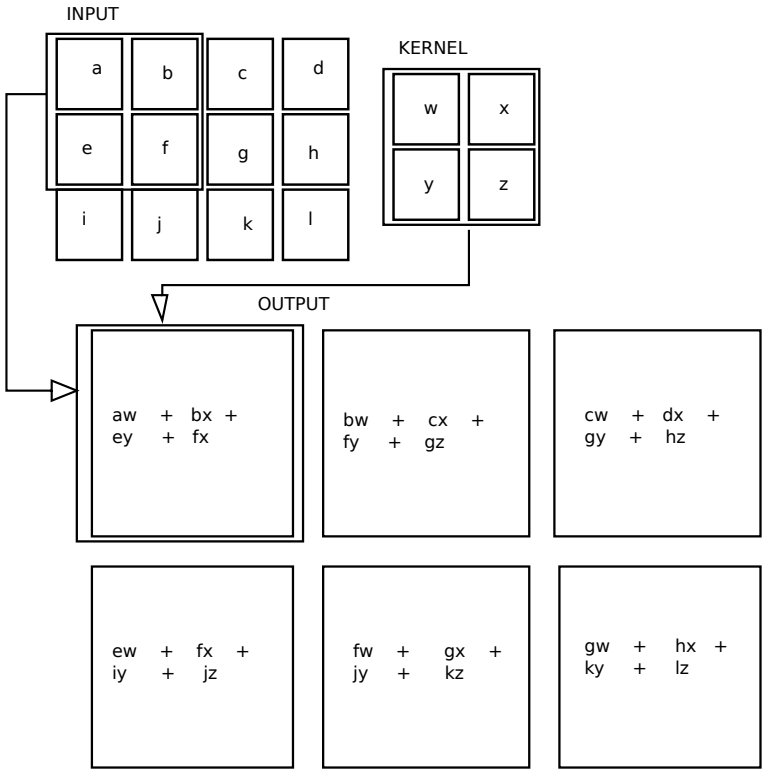


Figure 2.2: An example of a 2D convolution without kernel flipping. A 2x2 kernel is applied to a 3x4 matrix, producing a 2x3 matrix. The illustration is borrowed from Goodfellow et al. (2016)

```
import torch
layer = torch.nn.Conv2d(in_channels=1, out_channels=6,
                        kernel_size=(5,5),padding=0,
                        stride=1, dilation=1)
```

Figure 2.3: Example of how to create a convolutional layer in Pytorch Paszke et al. (2017a). It is an implementation of the first layer in the so-called LeNet-5 architecture from LeCun et al. (1998). The network will be explained in detail later in this section.

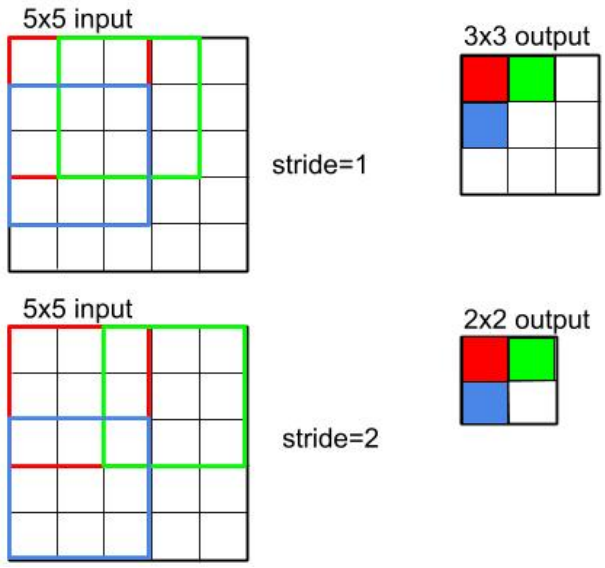


Figure 2.4: An example of the effect of different strides on output size and overlapping receptive fields. The colored boxes are the same 3×3 kernel being applied as it traverses through the input volume.

The effect of different number of strides is shown in Figure 2.4 and defines the *step-size* of the 3×3 kernel as it traverses through the input volume.

In Figure 2.2, `stride` is 1, thus the output becomes a 2×3 matrix. With `stride=2`, the output matrix would have been 1×2, completely neglecting i, j, k and l in the input in Figure 2.2. Thus, one can say that `stride` defines how often the kernel evaluates the input. This means it also affects the size of the output as well as how much the kernels overlap and evaluate the same values.

The final argument in Figure 2.3, dilation, is a way to ”inflate” a kernel by inserting spaces between kernel elements. The effective size of the kernel increases, even though it has the same amount of kernel elements. Explaining dilation and the previously explained arguments without the use of animations is difficult. For a more visual approach, the website created by Al-Rfou et al. (2016) has great animations accompanied by more textual explanation.

A prevalent architecture applied to images is a deep neural network alternating between convolutional layers and pooling operations. A pooling operation is a form of downsampling at a certain location with summary statistics (e.g., maximum, average) of the nearby values. Max-pooling, a popular type of pooling, returns the maximum output within a rectangular kernel. One have to set the `kernel_size` of the pooling layer in the creation of the `MaxPool2d()` object, as explained in Paszke et al. (2017b).

By using convolutional layers and pooling layers, the input is transitioned from high spatial information to low spatial information. Deep layers capture more conceptual features than shallow layers, which typically capture features like edges and textures. This

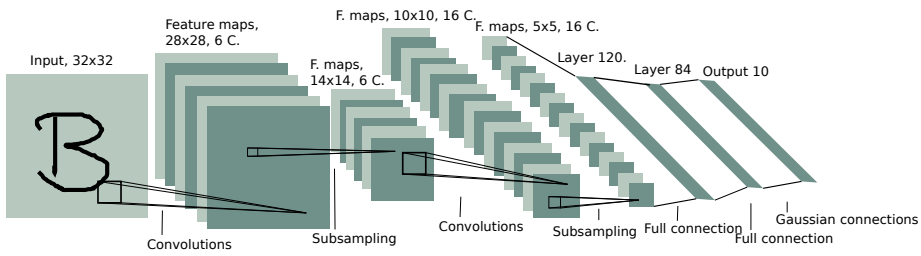


Figure 2.5: An example of a convolutional network, LeNet-5 LeCun et al. (1998). The illustration shows how an digit-instance propagates through the network as input and output of layers. The squares are 2 dimensional feature maps where "$Y \times Y$, Z C", with $A \times A$ in spatial dimensions and Z number of C channels. The three last layers are one dimensional layers with Z length.

process from small concrete detail-oriented features to more conceptual features is popularly referred to as feature extraction. The final layer is often fully connected and is used to do the inference/classification part. The division between feature extraction and inference is commonly used to give a holistic explanation of convolutional networks. However, it is debated how accurate this holistic explanation is.

In Figure 2.5, one can see the LeNet-5 Architecture, the network architecture that sparked some interest in the field with its classification of handwritten digits. LeNet-5 was one of the earliest successful applications of the convolutional network, when it was applied to images of handwritten digits. Each of the squares in Figure 2.5 is a feature map that represents the where in the image a certain feature is. LeNet-5 extract features by alternating applying convolutions and subsampling the input. The spatial dimensions are reduced from 32×32 to 5×5 while the number of channels increases from the original one gray tone to 16 channels. After that, the spatial dimensions are flattened as there are 2 fully connected layers and Gaussian connections. Gaussian connections function very similar to the more modern cross-entropy but use euclidean radial basis functions as a measure of cost.

The general idea of CNNs is that they mimic the human vision process with restrictive receptive fields like human eyes use in its visual processing. The architecture of convolutional neural networks is developed to model data that is assumed to have some properties. Neighboring values are assumed to be highly correlated, a property the network exploits. The assumption is valid for natural images, as neighboring values in an image are often highly correlated. Most natural time-series also have very correlated neighboring values. If convolutional neural networks are used on data where the assumption does not hold, it will underperform. Further, CNNs assume that a feature that is useful in one location is useful in several other locations. This assumption allows for parameter sharing. Parameter sharing makes the architecture much more memory efficient.

CNNs have had great success in the image-domain, as well as other domains. Al-

though, some moderate critics like Geoffrey Hinton have pointed out some of its weaknesses as in Hinton et al. (2011). CNNs are equivariant to translation. By adding pooling layers, they make the network somewhat shift-invariant. CNNs are also somewhat invariant to small changes in viewpoint. Humans are great at recognizing objects that are seen from a new viewpoint. Hinton et al. (2011) points out this seeming lack of awareness of orientation in CNNs, what he more generally calls *pose*. Pose information refers to 3D orientation relative to the viewer, but the pose also encompasses lighting and color. He thinks that the focus should on designing networks that aim for equivariance, disentangle instead of discarding. He addresses these problems in new research in his work on capsule networks, which will be explained in Section 2.4.

2.4 Capsule network

CNNs are known to be prone to fail to recognize entities with different rotation and lighting if it has not seen sufficient images in the training phase. This weakness is a motivation behind Capsule networks. Capsule networks can also use convolutions, but differs from CNN and other ANNs in two main ways; Neurons are grouped together in capsules which are updated as an unit and dynamic routing is introduced in relation to updating the capsules.

The concept of *capsules* was first outlined in a paper about transforming autoencoder (Hinton et al. (2011)). Dynamic routing, one of the central concepts of capsule networks, was presented in the article by Sabour et al. (2017). The article by Sabour et al. (2017) can be said to be the start of the field of capsule networks. The next year the same authors suggested modifying the representation of capsules and the routing algorithm for a performance boost in their paper titled *Matrix capsules with EM routing*, Hinton et al. (2018). The modifications are outlined in Chapter 4.

A capsule is a group of neurons that collectively produce an *activity vector*, where each neuron is an element in the activity vector. The activity vector make it possible to represent different *instantiation parameters*. By instantiation parameters, one means the properties that define the state of an instance of an entity. Properties may include instantiation parameters such as position, size/depth, rotation, deformation, lighting, hue, texture. The orientation of the activity vector represents the described state of the entity in the input, and is referred to as the *pose* of the entity. The length of the activity vector's length represents the probability that the entity exists in the input, equivalent to traditional neurons' scalar activation value.

A capsule is a group of neurons that collectively produce an *activity vector*, where each neuron is an element in the activity vector. The neurons together make it possible to represent different *instantiation value*(e.g. rotation, hue, lighting) of the entity in the input. The orientation of the activity vector represents the described state of the entity in the input. The length of the activity vector's length represents the probability that the entity exists in the input.

Traditional neurons have an activation value, a scalar, as an output. Capsules, on the other hand, have an *activity vector*. Neurons' activation value has a representative ability limited to signaling the probability of an entity existing or not. In the capsule, this probability of existence is represented by the length of the activity vector. The orientation

of the vector represents the *instantiation parameters*. By instantiation parameters, one means the properties that define the state of an instance of an entity. Properties may include instantiation parameters such as position, size/depth, rotation, deformation, lighting, hue, texture. The orientation of the activity vector will be referred to as the *pose* of the entity.

The motivation behind introducing CNN was to simulate the process of the biological visual cortex. Plain ANNs were bad at image processing. CNNs not only increased performance but lowered computational time. Capsules are not as rooted in biology compared to CNNs, but rather to humans' ability to understand that up-down car is a car even though one maybe never have seen a car of that type in that specific position. The ability to generalize is the driving force behind the introduction of capsules.

In Sabour et al. (2017), there are claims that CNNs' inability to deal with affine transformations in new inputs will be the architectures' downfall. To learn affine transformations, CNNs will have to replicate feature detectors on a grid that grows with the number of dimensions or increase the labeled dataset size in a similarly exponential way. The latter seems to be the strategy for many industry solutions. The capsule network is much slower computationally, but it does not suffer from the same exponential traits.

As previously mentioned one of the key part of capsule networks is the dynamic routing algorithm proposed by Sabour et al. (2017). The algorithm is outlined in Section 2.4.1. Furthermore, the capsule network applies a non-traditional loss function, margin loss, which is explained in Section 2.4.2.

2.4.1 Dynamic routing

The forward pass from lower capsule layer to higher capsule layer is called dynamic routing and differs from neurons forward passes in architectures like LeNet-5. The input $\mathbf{u}_i$ from the lower level capsule i is a activity vector whose norm is *squashed* between 0 and 1. The first operation is to apply transformation matrix $\mathbf{W}_{ij}$ on the activity vector $\mathbf{u}_i$ to calculate the *prediction vector* $\hat{\mathbf{u}}_j|i$,

$$\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i. \tag{2.7}$$

Analogous to traditional neurons this first operation is equivalent to Equation (2.1) for traditional neurons. In Equation (2.2), one can see the traditional neurons sum over all its contributions uniformly, whereas capsule networks suggest a bit more complex model. The sum s_j

$$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}, \tag{2.8}$$

takes the sum of the prediction vector $\mathbf{u}_{j|i}$ weighted by coupling coefficient c_{ij} for every capsule pair(i, j). The coefficients are calculated iteratively with the dynamic routing algorithm. Lastly the capsule network suggest to *squash* $\mathbf{s}_j$,

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1 + \|\mathbf{s}_j\|^2}\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}. \tag{2.9}$$

This transformation secures an activity vector $\mathbf{v}_j$ width a norm between 0 and 1 as an output. The squash function introduces non-linearity and takes the role closes to and tra-

ditional activation function from Equation (2.3). Sabour et al. (2017) reasoned the choice of the unprincipled non-linear activation simply by stating *"We leave it to discriminative learning to make good use of this non-linearity"*.

The coupling coefficients c_{ij} in Equation (2.9) symbolize agreement between capsule i and capsule j. The coupling coefficients from a capsule to all its parent capsules together sum up to 1, $\sum_{j=1} c_{ij} = 1$, forcing the capsules to prioritize its information sharing. This property is enforced by a *routing softmax*

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}, \tag{2.10}$$

where b_{ij} are the log prior probabilities that capsule i and j are coupled. Coupled is defined by the agreement $a_{ij} = \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$ between output $\mathbf{v}_j$ of capsule j in the layer above and the prediction vector $\hat{\mathbf{u}}_{j|i}$ made by capsule i in the layer below. The scalar output called agreement is treated as a log-likelihood and is added to b_{ij}. Iteratively the coupling coefficients are recomputed, the network computes another forward pass, and the log priors are learned as stated in the Algorithm 1. This process is called *routing-by-agreement*.

Algorithm 1 Dynamic routing algorithm (from Sabour et al. (2017))

procedure DYNAMIC ROUTING($\hat{\mathbf{u}}_{j|i}, l, r$)
  for capsule i in layer l and capsule j in layer (l + 1) **do**
    $b_{ij} \leftarrow 0$
  for r iterations **do**
    for capsule i in layer l and capsule j in layer (l + 1) **do**
      $\mathbf{c}_i \leftarrow \text{SOFTMAX}(\mathbf{b}_i)$ ▷ SOFTMAX($\mathbf{b}_i$) computes Eq. 2.10
      $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
      $\mathbf{v}_j \leftarrow \text{SQUASH}(\mathbf{s}_j)$ ▷ SQUASH($\mathbf{s}_j$) computes Eq. 2.9
      $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$
  return $\mathbf{v}_j$

2.4.2 Loss function

The suggested loss function for capsule networks in Sabour et al. (2017) is

$$L_k = T_k \ \max(0, m^+ - \|\mathbf{v}_k\|)^2 + \lambda \ (1 - T_k) \ \max(0, \|\mathbf{v}_k\| - m^-)^2. \tag{2.11}$$

where $T_k = 1$ iff class k is present, $m^+ = 0.9$, $m^- = 0.1$ and λ is a coefficient to down-weight the loss for absent classes. L_k is the margin loss for each high-level capsule, k, also called the *class capsule*. The length of these capsules output vectors $\|\mathbf{v}_k\|$ predict whether or not class k is present in the input. This marginal loss function enables the model to classify multiple classes. The total loss is the sum of the marginal losses for every last layer capsules. All the operations in the dynamic routing are differentiable, and the routing iterations can be unrolled into a directed differentiable graph; hence one can use backpropagation on capsule networks.

To force the capsules to encode the instantiation parameters of the classes, one can use reconstruction as a regularization method. The last layer of capsules encodes separate classes. The method masks everything but the activity vector of the correct class, which is then used as input to three fully connected layers as presented in Figure 5.2 in Chapter 5. The last layer has the same amount of logistic units as the input size. One can then encourage the model to recreate the input image by minimizing the mean squared error between the pixel intensities from the original image and the outputs from the reconstruction network. This *reconstruction loss* is added as a term in the loss function. The reconstruction of the input happens to also be a great tool to gain insight and to diagnose the model.

2.5 Summary

This chapter has introduced the core concepts ANNs, regularization, and CNN, as these are important to introduce capsule networks. Capsule layers can both be of convolutional nature or fully connected as plain ANNs. Capsules change the computational node, which usually is a neuron, to rather be a group of neurons that together model the instantiation parameters of an entity. These instantiation parameters can represent rotation, hue, and lighting of an instance within each capsule. The output of the capsule is an activity vector, where the direction of the vector represents the entity's state, and the length represents the probability of existence. The non-linear transformation in capsule network is not like traditional activation functions. The squash function squashes the length of the vector instead of the individual neuron outputs. Regularization of capsule networks is done by minimizing the difference between the reconstruction of the input from the correct capsule and the original image.

Chapter 3

Structured Literature review

Structured Literature reviews(SLRs) are important to make it clear why the papers referenced to in section 4 are included and to ensure that the work related to this thesis is of quality. The SLR is created as a guideline for the author and readers of this thesis on how to gather, filter out and choose literature. The guideline is formed from the top down. First, the general goal of the SLR is defined. Thereafter, a more concrete criteria checklist is created as a tool to filter papers more efficiently. Following that, methods to search and gather the information is elaborated on. This thesis touches on several branches of science, such as biological neurons, mathematical optimization and computer science. Different branches of science use different journals, therefore the search engine Google Scholar has been used to find relevant papers. The search-terms are created and designed to narrow the search as much as possible to minimize the time needed to review all the candidates. If there are not enough relevant papers from the narrow search a more expansive search with a less concrete search-term is used. The paper that introduced a functional Capsule network, *Dynamic Routing between Capsules*, Sabour et al. (2017), is central to the searches conducted whilst writing the thesis.

3.1 General goals

The general goal of the SLR is to gain a updated knowledge of the capsule network advances. There is extra focus on Capsule networks applied to datasets with larger images, many classes and datasets with few samples per class.

3.2 Criteria for relevance

The criterias are designed to accept papers that accept both general advances in the field as well as more specified applications. To make up for the differences, the criterias are parted into two categories. General criterias which apply to all papers and Special criterias that only apply to the different groups.

General criterias which apply to all papers:

- It should give insight into the behaviour and properties of capsule networks
- The methods should be understandable.
- It should have non-ambiguous interpretable results.

Special criterias which apply to papers in one of the following areas(Further development, large images, many classes or few samples per class):

- The paper includes a significant change of the capsule network architecture or routing algorithm, with originality. (Further development)
- The paper compares the their model to the original CapsNet with datasets covered in Sabour et al. (2017). (Further development)
- The paper apply a capsule network on a dataset with bigger images than 32×32. (Large images)
- The paper apply a capsule network on a dataset with more than 10 classes. (Many classes)
- The paper apply a capsule network on a dataset with less than 500 samples per class. (few samples per class)

3.3 Searching

3.3.1 Sources

There are several relevant repositories for Computer Science like Springer, ACM, IEEE, NIPS, ICLR, ICML and AAAI. Google Scholar is used for all searching. This is because there are so many journals, and several have bad search-engines. Google scholar on the other hand does not take any responsibility for the quality for its content. This puts the responsibility on quality assuring on the searcher.

3.3.2 Execution

The starting point for the search is of course the paper that introduced dynamic routing between capsules Sabour et al. (2017). The citations from this paper are assessed for relevance. All other searches are limited to papers that cites the original paper, which implicitly also restrict the search to all papers written in 2017 or later.

Search terms used for finding the articles most relevant to the Research questions.

- *(capsule AND "(many OR more) (Classes))*" 33 results
- *(capsule AND agreement AND ("samples per class" OR "small dataset" OR "data scarcity"))* 43 results

”agreement” was added to the last search-term to get under 50 candidates and filter out the papers that does not have an in-depth explanation of capsule networks.

The search results were first graded according to the criteria checklist based on the abstract. The papers with the highest scores were then skim-read and re-graded. Using the adjusted grade, a decision is made whether some areas of research were lacking in quality papers. If so some of these search-terms were made less concrete to include more papers were I felt more candidates was needed. The process of searching, picking out and assessing is a cyclic process.

Chapter 4

State of the art

This chapter outlines development of the capsule network idea and review literature that has applied capsule networks to different datasets. Section 4.1 goes deeper into the original Capsule articles, before further developments are presented in Section 4.2. Thereafter the focus shifts to applications on larger images (Section 4.3), non-image applications (Section 4.4), fewer samples per class (Section 4.5), and many classes (Section 4.6). At the end of the chapter there is a summary of all the strengths, limitations and nuances of capsule networks that was discovered in the literature.

4.1 Dynamic routing between Capsules

As mentioned in the background theory, Hinton et al. (2011) outlined how capsules can represent instantiation parameters, and how the length of the capsule vector could symbolize entity existence. They also formulated how, with a transformation matrix, one can calculate the prediction vectors for higher level capsule as in Equation (2.7). However, the transformation matrices had to be supplied externally in Hinton et al. (2011), limiting the use-cases for common image classification tasks. The dynamic routing between capsules presented in Sabour et al. (2017) formulated a method to train the transformation matrices as weights in a network.

The design of the capsule networks was inspired by inverse computer graphics rendering. Rendering is the process of producing images from a certain viewpoint given 3D models, textures and lighting conditions. It is calculated with transformation matrices that can perform scaling, rotation, translation, mirroring and shearing of objects. In capsule networks the inverse process is carried out by multiplying the transformation matrix with the capsule vectors to calculate the prediction vector in Equation (2.7). The purpose of the process is to make the network viewpoint invariant, which the authors think is a better solution than to account for every possible viewpoint of an object in a dataset- a non-trivial task. The capsule network proposed by Sabour et al. (2017), will from now on be referred to as *CapsNet*

CapsNet was implemented with focus on MNIST and MultiMNIST, but it was also

implemented with and without modifications on the CIFAR10, SVHN, and smallNORB datasets (see Section 6.1). The results from Sabour et al. (2017) on MNIST and MultiMNIST are presented in Table 4.1 with different number of routing iterations and with and without reconstruction loss. The results unambiguously favor including the reconstruction error. The number of routings have less of an effect without reconstruction, but the appendix of Sabour et al. (2017) provide additional reasoning on why 3 routing iterations is recommended for all experiments. It is shown to converge faster than fewer iterations and the average change of the priors b_{ij} in Equation (2.10) is very low after 5 routing iterations.

The baseline in Sabour et al. (2017) is not a state of the art network, but a vanilla CNN with three convolutional layers of 256, 256, 128 channels. Each layer has a 5×5 kernel with a stride of 1. The two last layers are fully connected layers connected by dropout to a softmax 10-output layer. The loss function used is cross-entropy. The CapsNet performs significantly better than this baseline on both MNIST and MultiMNIST.

Sabour et al. (2017) used the exact same CapsNet architecture on the smallNORB dataset and achieved 2.7% error rate, on par with state of art CNNs. A slightly smaller model was trained on the smaller SVHN dataset and achieved 4.3%. The network had 64 channels in the regular convolutional layer, 16 $6D$ convolutional capsules and finishing with $8D$ class capsules. A slightly bigger capsule network is applied to the CIFAR10 network. The solution presented used an ensemble of 7 models whom each focused on 24×24 patches of the input image. There were 64 instead of 32 convolutional capsules, and the RGB input image requires 3 input channels. The routing softmaxes were introduced to a none-of-above category to mitigate capsule networks tendency to model all the non-discriminatory background as well. The network achieved a 10.6% test error which is not state of the art, but as the authors point out it is on-par with the results presented by the first CNNs applied to CIFAR10.

Model	Routing iterations	Reconstruction	MNIST	Multi-MNIST
Baseline	NA	NA	0.39%	8.1%
CapsNet	1	no	$0.34 \pm 0.032\%$	NA
CapsNet	1	yes	$0.29 \pm 0.011\%$	7.5%
CapsNet	3	no	$0.35 \pm 0.036\%$	NA
CapsNet	3	yes	$0.25 \pm 0.005\%$	5.2%

Table 4.1: Results for the CapsNet from Sabour et al. (2017) with different number of routing iterations and with and without reconstruction loss included applied on different datasets. The results are presented as error-rates.

4.2 Further development

The authors of the original paper on dynamic routing later published a new paper on matrix capsules with EM Routing Hinton et al. (2018). The paper suggested changes to both the representation of capsules and on the method of dynamic routing. Instead of representing the capsule as a vector, a matrix was proposed. The matrix represent the pose of the

entity and each matrix has a separate logistic unit trained to represent the presence of the entity. Dynamic routing is based of a principle of routing-by-agreement. The agreement is modeled by the cosine distance between capsule vectors in Sabour et al. (2017), while it is formulated as Expectation Maximization(EM) of clusters in Hinton et al. (2018). The activated poses (transformed into vectors) of lower-level capsules represent data points while every higher-level capsule represent a Gaussian cluster. The architecture from Matrix capsules with EM-routing will from here on be called *EM-CapsNet*.

The authors claim they have overcome 3 deficiencies of the original CapsNet Sabour et al. (2017).

1. Using a logistic unit instead of the capsule vector length to represent an entity's existence, allows for loss functions that are optimized through the routing procedure. This is not possible with the capsule vector length as it must be squashed by an unprincipled non-linear function.

2. Using the negative log variance of a Gaussian cluster instead of the cosine distance between two capsule vectors as a formula for their agreement, improves the model's ability to distinguish between good and very good agreement. This is because the cosine distance saturates at 1(perfect agreement).

3. Using matrices with n elements requires n transformation matrices, while using vectors with n elements requires n^2 transformation matrices, improving the scalability of the architecture.

The EM-CapsNet was applied to the smallNORB dataset see LeCun et al. (2004). Their model with a test error of 1.4% improved on the previous best-known result of 2.6% (Cireşan et al. (2011)) by 45%. The CNN baseline constructed by Hinton et al. (2018) scored as low as 5.2%. The paper also experimented with a EM-CapsNet with cross-entropy loss. It collapsed in performance and did slightly worse than the baseline CNN.

The smallNORB dataset was also used to test the model's ability to recognize objects from never-seen-before viewpoints. Whereas the goal was to get an indication of whether the model have managed to extrapolate on rotational transformations. The baseline and the EM-CapsNet were trained until they had the same test accuracy on the familiar viewpoints, in an attempt to isolate their ability to generalize to new viewpoints. When tested on the new viewpoints, EM-CapsNet performed 30% better than the CNN baseline. EM-CapsNet is also applied to CIFAR10 and MNIST with minimal alterations. The network performed worse than the CapsNet architecture on both.

The architecture of CapsNet is criticized by Rawlinson et al. (2018) for it's unsupervised routing algorithm and supervised training of the network weights. Rawlinson et al. (2018) implies that this manner of training make deep capsule network architectures difficult to train. The paper suggests unsupervised training of capsules, which entail removing the margin loss as well as the masking of all but one capsule before the reconstruction network. This means that the previously capsule layer referred to as class-capsules, now all capsules represent the latent variables of the data (latent-capsules). This change enables the network to function as an autoencoder. As expected, there was an improvement in reconstruction loss, but the equivarient qualities, the corner-stone trait of capsules, collapsed. All capsules contributed to all the outputs. However, it is desirable that the capsules specialize in some way or form. To enable the capsules to specialize, an algorithm

that sparsefies the latent capsules activations was applied. Sparsefying the connections between capsules allows the latent capsules to represent and specialize in different subsets of features, making the network regain its equivarient abilities.

To classify the instances during testing, the outputs from the unsupervised trained sparse Capsule network (Sparse-CapsNet) were passed on to a Support Vector Machine (SVM). SVM is a popular algorithm for linear classification of clusters in multidimensional space. The SVM in Rawlinson et al. (2018) utilized the kernel trick with the popular non-linear radial basis function Boser et al. (1992), making it a non-linear classifier. Sparse-CapsNet performed vastly better than CapsNet on affNIST after training on MNIST. Sparse-CapsNet scored 99% accuracy on MNIST and 90.12% on affNIST, CapsNet scored 99.22% accuracy while it got 66% on the affNIST dataset. Sparse-Capsnet had at the time, the best testing accuracy on affNIST using only the MNIST dataset, and without extensive augmentation. Their results where surpassed by another capsule network the year after, by Kosiorek et al. (2019). They presented a testing accuracy of $92.2 \pm 0.59\%$. That said, affNIST is not a very popular benchmark dataset for testing this type of viewpoint-generalization.

Kosiorek et al. (2019) presents an unsupervised capsule autoencoder, which aims to utilize the capsule's ability to model geometric relationship between parts and wholes. The paper also presents state of the art results on unsupervised classification on SVHN and MNIST. It performs sub-par(33.48%) compared to state of the art(57.6%) on the CIFAR10 dataset. CIFAR10 Krizhevsky and Hinton (2009) is a dataset with complex, noisy, natural images. The authors of the unsupervised autoencoder claim that the cause for this sub-par performance is the model's inability to model background. However, despite capsule networks shortcomings on natural images, it seems like capsule networks' equivarience properties are well suited for the field of unsupervised learning.

Together with Rawlinson et al. (2018) , many papers support the claim that the original CapsNet cannot create significantly deep networks. Peer et al. (2018) and Xi et al. (2017) demonstrate it with respectively 6 and 3 layers of capsule layers. On both occasions the network suffered a total collapse in performance, scoring approximately 10% on the MNIST dataset. Since the MNIST dataset have ten classes, the performance is equivalent to random guessing.

Other papers not only support the conclusion that CapsNet cannot create deep networks, but present solutions to the issue. Both Peer et al. (2018) and Rajasegaran et al. (2019) suggests a substitute for the dynamic routing as described in Sabour et al. (2017). Both successfully train deeper capsule networks, but Peer et al. (2018) reports sub-par performance when increasing the depth. They observed near uniform coupling distribution, and managed to force a parse-tree coupling structure which enabled deep learning to some degree. A new architecture as well as a class-independent decoder is suggested by Rajasegaran et al. (2019). According to Rajasegaran et al. (2019), the changes that enabled deeper networks were localized routing in a convolutional framework as well as including skip connections as originally proposed by He et al. (2016). The new changes (in an 7-ensemble architecture) resulted in state of the art test accuracy amongst capsule networks for the different benchmarks datasets: CIFAR10(92.74%), SVHN(97.56%) and Fashion MNIST(94.73%). They also achieved a 68% reduction in the number of parameters.

Rajasegaran et al. (2019) back up the conclusion by Rawlinson et al. (2018) that an

class-independent decoder can represent features more efficiently without the constraint of modeling classes separately. In CapsNet's class capsules the different classes all tried to represent rotation, skewness and boldness for each digit in MNIST independently. These features are universal for all of the digits. By implementing class-independent capsules, the redundancies are removed and the capsules are more expressive.

4.3 Large and complex images

There are some successful applications of capsule networks on larger images, LaLonde and Bagci (2018) being one of them. They look into segmenting pathological lungs from large 512×512 CT scan images. LaLonde and Bagci (2018) presents a capsule version of the U-net Ronneberger et al. (2015), a popular architecture for segmenting large images. The capsule U-net provides slightly better segmentation accuracy than state of the art baselines, while reducing the number of parameters in the model with 95.6% compared to the regular U-net model Ronneberger et al. (2015). To reduce the number of parameters LaLonde and Bagci (2018) introduces locally constrained routing. They also implemented deconvolutional capsules as an alternative to regular deconvolutional layers. Two years after, more thorough research to back up the claims from LaLonde and Bagci (2018) and investigate the properties of capsules in image segmentation were published by LaLonde et al. (2020).

Medical imaging mostly have a uniform background to contrast the object imaged, and thus resemblance the other images that capsule networks have been successfully applied to. The results from LaLonde and Bagci (2018) show that capsules can be scaled up for quite high resolutions, especially since the limiting factor with processing large images is often the models parameters. For efficient training, a model must be trained be on one or more GPUs. So the fact that a capsule version of the U-net decreased the original network's size with 95.6%, shows great promise for capsules as a method of reducing the size of models, thus requiring less GPU memory.

The previous sections of this chapter have mostly focused on capsule networks tasked with images ranging from 28×28 to 96×96(smallNORB) resolution with 10 classes. The capsule networks have achieved state of the art results most of the datasets, except CIFAR10. The images from MNIST, SVHN and Fashion-MNIST have a flat background without to many details or variations. It is pointed out that CIFAR10 have a more natural background, a context with other objects and textures. In other words, the images are more complex. An example of an underlying pattern one does not want to classify by using is the following. Two of the classes in CIFAR10 is horse and airplane. Horses have more images with a green meadow background, while airplanes have more sky or buildings in the background. Having a way to deal with this background and model/discard it is important, and seemingly is one of the more pressing challenges of capsule networks.

Members of the image analysis community have requested a modification of capsule networks tailored for the ImageNet dataset. No evidence that a model with the traditional use of capsules that have been applied to the ILSVRC have been found. One paper, Zhang et al. (2018), presented results on ILSVRC, with a Capsule Projection Network(CapProNet). The network was implemented with different traditional CNNs like Resnet He et al. (2016) and DenseNet Huang et al. (2017) as backbones with capsules in

the final layer. The term capsules is loosened up the traditional definition of capsules as a grouping of neurons. This implementation of capsule subspaces borrows some ideas for capsules, but are more of an augmentation on CNNs. The CapProNet only used the capsule subspaces in the last layer. Subspace capsule share the regular capsules characteristics of:

1. A vector to represent a capsule
2. Vector length represent probability of entity existence
3. Vector orientation represent instantiation parameters

However, the capsule is not formed by grouping a specific set of neurons. The paper proposes to learn a set of capsule subspaces, then projecting a input feature vector onto them. The resultant vectors for the different capsule subspaces represent the capsule vectors. Zhang et al. (2018) claims $10 - 20\%$ improvement compared to different depth ResNets on ILSVRC as well as $5 - 7\%$ improvement to different depth Densenets. The implementation had only $< 1\%$ computational overhead compared to the original CNN network.

The work on capsule subspaces have since been elaborated on by Edraki et al. (2020). Edraki et al. (2020) incorporates the principle of capsule subspaces in the intermediate layers, apposed to only applying it in the last layer as in Zhang et al. (2018). The subspace capsules make the routing algorithm obsolete. Unfortunately the papers used different backbones for the experimental part, making it hard to comment on their relative performance. The subspace capsule network in Edraki et al. (2020) made significant improvement in performance compared to the baseline models in the three image related tasks: supervised classification, semi-supervised classification and generating high quality images on multiple datasets using a generative adversarial network framework proposed by Goodfellow et al. (2014). The experiments used different datasets and image sizes, ranging from CIFAR10's 32×32 images to ImageNets 256×256 images.

4.4 Non-image applications

The capsule architecture is to some extent designed to emulate inverse graphics, the process of decoding several 2D images from different viewpoints into a 3D understanding of entities. However, several papers have reported on capsules networks' promising equivarient properties. Motivated by this, other people have modified capsule networks, so that they can be applied to problems from other domains. An example is a 78-dimensional multivariate timeseries for diagnosing patients, extracted and processed from the dataset MIMIC-III (Johnson et al. (2016)). Capsule networks have also been applied to several text classification problems, like sentiment classification, question categorization, news categorization, review classification, opinion classification.

The EM-CapsNet modified for learning diagnostics on the MIMIC-III dataset, learned successfully, but was observed converging slowly by Bahadori (2018). Bahadori (2018) introduced spectral capsules for faster convergence. Spectral capsules calculates their vector-poses by performing a principal component analysis (PCA) of the weighted votes from the

capsules below. PCA is a linear calculation that is popularly used as a dimensionality-reduction tool, as it finds the axis in a multi-dimensional space that preserves most information if projecting onto that axis. The pose of the capsule is the vector that preserve the most variance in the data. The activation vector and agreement are then calculated by using the fraction of variance captured by the votes from the capsules below.

Setting the differences in pose representation and adjustments for time-series aside, the differences between EM-Capsnet and Spectral capsules are analogous to the difference between Gaussian mixture models and principal component analysis. Some experience working with these approaches is enough to explain why Spectral capsules are more robust in training; PCA cuts through much of the noise, and crudely explain its data in broad strokes. The final results of the Spectral capsules was an AUC of 0.8050, only slightly better than the EM-CapsNet, with an AUC of 0.8017. However, it is not possible to conclude whether the PCA approach shows merit in image analysis or other domains, as there is no evidence that the spectral capsules has been applied to the image analysis domain was found when researching material for this thesis.

There are examples of capsule networks being successfully applied to the domain of natural language processing (Srivastava et al. (2018), Xia et al. (2018), Kim et al. (2020)). In 2018 Zhao et al. (2018) presented results on par with state of the art models in many text classification applications like review classification, sentiment classification and question categorization. They tested their model on Reuters-21578, a dataset with a training set of single-labeled documents and a testing set of multi-label documents. It performed significantly better at the transfer between the task compared to the 9 different state of the art baselines. It has been observed that capsule networks have strong generalization abilities, and that could be a possible explanation for these results. Though Zhao et al. (2018) point out that the single-label to multi-label transitions from N label space to a 2^N labels space. To make up for an exponentially increasing label-space, one can gather more multi-label data or augment new data. Labeling data is quite labor intensive and augmenting documents are not as easy and effective as image augmentation. Since the capsules are thought to be more data-efficient, they should be well suited for multi-label text classification problems.

The discussed papers show the promise capsule networks have for a wide range of applications. Despite being engineered to perform well on visual or spacial tasks, the ability to describe a specific entity with several instantiation parameters instead of a boolean signal seem to fit a wide set of applications.

4.5 Few samples per class

The field of deep learning have been focused on learning large datasets, and are by many thought to easily overfit small datasets in many cases. This notion is challenged by Olson et al. (2018), whom applies neural networks to 116 real world datasets from the UCI Machine Learning repository, with hundreds of model parameters per observation. As one might expect, they find that regularization of the networks increase the performance, but lack of it does not lead to a collapse in performance in contradiction to the popular notion. Capsule networks has several times exemplified its comparably better generalization properties compared to regular CNNs. After training on MNIST it performed significantly

better on AffNIST, after training on smallNORB the capsule network excelled in performance on never-seen-before viewpoints. Even in the domain of of text classification it showed the best results when training on single-label data and testing on multi-label data, showcasing great generalization properties. In this section, the performance of capsule networks on small datasets is investigated.

In the time writing this thesis, the COVID-19 pandemic have over 4 million reported cases globally and regulations effect the global community and economy. CT scans of COVID-19 patients have typical features that can be recognized by doctors and artificial intelligence models. Because of their wide availability and fast turnaround time, CT-scans has the potential for being a complementary tool for diagnostics of COVID-19 and potentially similar use-cases. This is a problem capsule networks could be suitable for. However, it is often difficult assembling a good dataset for models to be trained on in the medical field; Health institutions have privacy standards that must be upheld and different countries, regions and hospitals have different formats and databases of data. Thus, large datasets are not easily available.

The dataset used in Mobiny et al. (2020) is a good example of data that is gathered from more than 5 sources, even using pictures in journals with downgraded resolution and yet has only 746 CT scans. The images are of either COVID or non-COVID class and the dimension of the images ranged from 153×120 to 1853×1458 pixels. Mobiny et al. (2020) solves the challenges of a scarce dataset by implementing a capsule network with improvements specific for the task, as well as a generative adversarial network for generating 900 new images. To regularize properly, they patched over parts of images that the network deemed to have discriminatory features. This forced the network to discriminate between COVID or non-COVID CT scans using all the available fine-grained telltale signs as well. The capsule network architecture with (0.96 AUC) and without (0.93 AUC) the patching of images beat the CNN baselines: Inception-v3 (0.89 AUC), DenseNet121 (0.90 AUC) and ResNet50 (0.88 AUC). The paper also includes a comparison with three Thoracic Radiologists where the network consistently outperformed all radiologists. At a 15.64% false positive rate the model achieve 95% true positive rate while the best Radiologist 85.11% true positive rate. This showcase that capsules can achieve great results on datasets without spacial rotation and transformations that capsules are designed to excel at.

Attempts at exploring capsule networks' data efficiency reports similar performance as CNNs. Schlegel et al. (2018) apply three versions of capsule networks together with two CNNs to the MNIST dataset with different number of samples per class. Their experiment trains the networks with different training set size (1,5,10,20,30,50,100 samples per class) until convergence. The authors attempt to replicate the models from Sabour et al. (2017) ($ConvNet1$ with $8.2M$ parameters) and Hinton et al. (2018) ($ConvNet2$ with $319K$ parameters). The third Capsule network was a minimized version of $ConvNet2$ ($Convnet2small$) with only $62K$ parameters. The CNN1 network has 3 convolutional layers, totalling $13.2M$ parameters, while the CNN2 network have 2 convolutional layers, totalling $1.1M$ parameters.

Capsnet1 performed similarly but consistently slightly worse ($0 - 10\%$) than CNN1 and CNN2. Both $ConvNet2$ and $ConvNet2small$ collapsed in performance, underperforming consistently on all number of samples under 101. Capsnet1 was designed to

perform on the MNIST dataset and performed slightly better than Capsnet2 on the whole dataset, but it was not expected to handle few datapoints so badly. After all CapsNet2 was concluded to generalize better by Hinton et al. (2018), and a decrease in number of parameters generally decrease overfitting which is a danger with small datasets. However the implementation detail of the Matrix capsules is somewhat lacking, and is possibly the reason for the errors.

4.6 Many classes

To the extent the research conducted for this thesis has uncovered, no capsule network have been applied to a dataset with more than a 1000 classes. Except from CapProNet, with their looser definition of capsules, no capsule network have performed well on a dataset with that many classes. It would be interesting to see CapsNet applied to a dataset with many classes, and with monotone background opposed to natural surroundings like ImageNet and CIFAR100.

4.7 Summary

There are many ways of implementing the general idea of capsules. CapsNet (Sabour et al. (2017)) is only one of them. Novel ways of representing the instantiation parameters and routing mechanisms to model agreement between higher and lower level capsule have been developed. EM-CapsNet Hinton et al. (2018) contributed with matrix instead of vector capsules which decrease the size of the needed transformation matrix, thus decreasing the number of parameters needed. The EM-CapsNet delivered 45% better accuracy than the state of the art on smallNORB. However, it performed worse than its counterpart, CapsNet, on both MNIST and CIFAR10.

Sparse unsupervised capsules is thought to generalize better by Rawlinson et al. (2018). Rawlinson et al. (2018) claim that the combination of CapsNets' unsupervised routing algorithm and supervised training is the reason the network structure cannot become very deep. Peer et al. (2018) and Xi et al. (2017) support the hypothesis as they have not been able to train the original CapsNet from Sabour et al. (2017) with more than 2 capsule layers without a collapse in performance. Deep capsule networks are possible in a convolutional framework with localized routing and skip connections, according to Rajasegaran et al. (2019). Peer et al. (2018) also show this, as they successfully managed to train deeper networks by forcing the network to create a parse-tree, but with diminishing results.

Sparse unsupervised capsules also introduced a class-independent decoder, that included an algorithm that sparsify couplings between capsules. The class-independent decoder enables more efficient decoding of features within the dataset, instead of representing rotation, lighting etc. for all n classes. The class-independent decoder was also used by Rajasegaran et al. (2019) with great success. Their model achieved state of the art results for capsule networks with CIFAR10 (92.74%), SVHN (97.56%) and Fashion MNIST (94.73%).

The capsule network idea has been applied to images as large as 512×512 in the medical imaging field with great success. However, subpar results was achieved when

capsule network implementations was applied to ImageNet, which contains large images, 1000 classes and many samples. Capsule networks also achieved subpar results on CIFAR10. It has been pointed out that it might be because of capsule networks tendency to want to model all the non-discriminative background. Both CIFAR10 and ImageNet are natural images with a wide range of backgrounds and context. Two similar implementations (Zhang et al. (2018); Edraki et al. (2020)) with a looser definition of capsules have achieved state of the art results on ImageNet and CIFAR10. Their definition of capsules does not statically group a set of neurons as a capsule. These implementations are a augmentation on existing CNNs inspired by the properties of capsules. That said, they present very promising results, especially given the very small computational overhead.

The idea of capsule networks, just like the idea of convolutional network before it, shows great promise outside the intended application on images. Capsule networks have been applied to multivariate timeseries and text categorization with success. In the case of text categorization, the capsule network implementation showed significant improvement over the baseline models when transferring from single-label to multi-label classification. This transferring between from different training data and testing data indicates great generalization abilities, and has been presented earlier on AffNIST (Sabour et al. (2017); Rawlinson et al. (2018)) and on new viewpoints in smallNORB Hinton et al. (2018). Success in classifying multi-label has also been done on MultiMNIST Sabour et al. (2017).

Capsule network achieved better results than CNN baselines on a COVID-19 CT scan dataset with few samples. The dataset does not have many viewpoints or other apparent features it is designed to perform well on. Intuitively the reason that capsules outperformed CNNs in this task would be the lack of a big dataset. The results from Schlegel et al. (2018) contests that hypothesis as their experiment tested CapsNet from Sabour et al. (2017) and EM-Capsnet from Hinton et al. (2018) on MNIST with samples ranging from one to a hundred. The EM-Capsnet did much worse than both the CapsNet and accompanying CNN baselines. CapsNet did not differentiate from the CNNs. Whether or not capsule have properties that make them more suited for scarce datasets is still under the lupe, and will be expanded on in this thesis.

There are not many examples of capsule network(traditional definition) applied to image dataset with very many classes. Therefore this is and interesting research topic that will be expanded upon in this thesis.

Chapter 5

Models

This chapter presents the architecture of the baseline model as well as the capsule network model custom-built for the MNIST dataset. In addition, the section states relevant hyper-parameters and the environment in which the model is trained. The architectures are later modified to accommodate changes in data, those changes are covered in Section 6 and the modifications are explained in this section.

5.1 Baseline

For comparison, a baseline model is constructed. It is a standard CNN based on the baseline used by Sabour et al. (2017). The architecture is described as two parts, called the feature extractor part and the classifier part. The feature extractor consists of three convolutional layers with 256, 256 and 128 channels. Each layer has a 5×5 kernel with a stride of 1.

The classifier consists of 3 fully connected layers with 328, 192 and 10 channels. To transition from convolutional layers to fully connected layers, the classifier flattens the spacial information of the output from the feature extractor. The two first layers apply the ReLu activation function, while the last layer uses softmax as the activation function. Before the the last layer, the model applies a dropout with a rate of 0.5 on the activations. Cross-entropy is used as the loss function. The CNN model contain 13.3M parameters.

5.2 Capsnet

The CapsNet architecture is based on the original capsule network by Sabour et al. (2017) and is visualized in Figure 5.1. It consists of only three layers, making it one of the more shallow models compared to other state-of-the-art models in the image-classification domain. The first layer is a conventional convolutional layer with a 9×9 kernel, with a stride of 1 and applies the ReLU activation function to the output. This layer is defined as the feature extractor of the model. In Figure 5.1 the orange boxes visualize how the

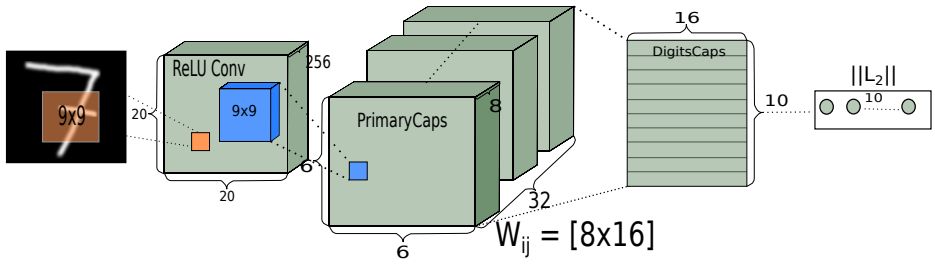


Figure 5.1: Architecture of CapsNet from Sabours *Dynamic Routing between capsules* Sabour et al. (2017)

layer maps from the 28×28 gray-scale image to a 20 feature map with 256 channels. The second layer, called *primary capsules*, consist of convolutional capsules with 9×9 kernel with a stride of 2. In Figure 5.1 the blue boxes attempt to visualize how the layer maps from the 256 20×20 feature maps to 32 channels of 6×6 $8D$ Capsules. The third layer, called *class capsules* is a fully connected layer between the 32 6×6 $8D$ Capsules and the 10 $16D$ capsules for each class. The class is predicted by the class capsule with the highest magnitude activation vector. The routing algorithm is only used between the capsule layers as there is no orientation in the scalar activations to agree on. The CapsNet model contain 8.2M parameters(regularization network included).

5.2.1 Regularization network

The regularization network is a fully connected network with 3 layers with 512, 1024, 784 neurons, in each of the respecting layers. The output of each layer is activated by the ReLU function except for the last layer's output which uses the sigmoid function. The output from the 784 neurons is mapped back into a 28×28 gray-scale image with the goal of being close to the original input image for the whole capsule network. The reconstruction network is optimized by minimizing the sum of squared differences between the outputs of sigmoid units and the pixels in the original input image. This reconstruction loss is added as a term in the loss function, Equation (2.11), for the capsule network. To prevent the term from dominating the loss function, it is regulated by the coefficient λ which is set to 0.0005.

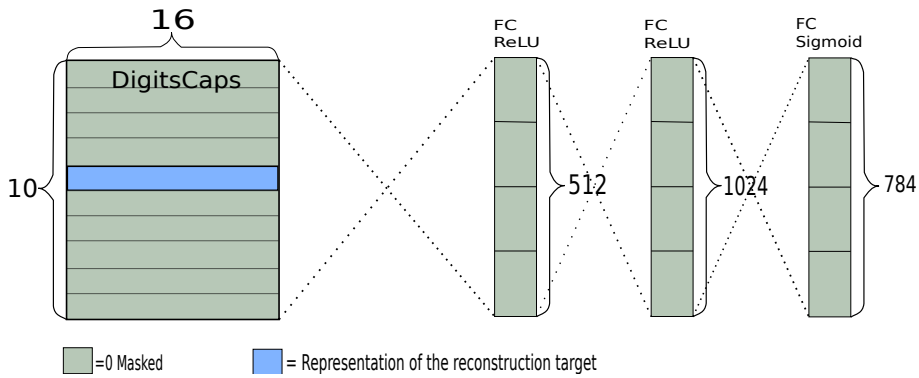


Figure 5.2: Architecture of Capsnets reconstruction network from Sabours *Dynamic Routing between capsules* Sabour et al. (2017)

Chapter 6

Experiments and results

This section presents the approach and result of two experiments. Both experiments compare a capsule network to a CNN baseline, as detailed in Chapter 5. First the datasets are explained in Section 6.1. Thereafter a brief explanation of the optimization algorithm used as well as relevant hyperparameters is presented in Section 6.2, followed by a statement of technology in Section 6.3. Section 6.4 presents an experiment with few samples per class carried out on MNIST. In Section 6.5, an experiment with a increasingly higher number of classes is carried out on a dataset assembled from the Omniglot dataset. The chapter is concluded by a short summary of the most important findings in Section 6.6.

6.1 Datasets

The MNIST dataset (LeCun et al. (1998)) contains 28×28 gray-scale images of handwritten digits (10 classes). The training set contains 50000 samples per class with 10000 samples in the test set per class.

The dataset is used in many of the papers published on capsule networks, including the original CapsNet. Therefore it is a familiar benchmark when testing performance of networks. It is also the dataset used by Schlegel et al. (2018), who ran an experiment similar to the one outlined in Section 6.4, "Experiment 1". Experiment 1 aims to reproduce the experiment conducted by Schlegel et al. (2018) and to supplement that experiment.

The Omniglot dataset (Lake et al. (2015)) contains 105×105 gray-scale images of 1623 different handwritten symbols/letters from 50 different alphabets. The training dataset has 12 instances per class while the testing set has 8 instances per class. The dataset is altered from the original hierarchical dataset by Lake et al. (2015). Only the images from Omniglot is used as the model's input, and only character ID numbers are returned. In other words, the model does not use the fact that characters are from different alphabets.

Several alphabets are related to others and have common letters with slightly different typography. An example being the letter "a"/"α", as shown in Figure 6.1. To distinguish between some of these instances can be very challenging. Yet all the models have to deal

Figure 6.1: The letter equivalent to the letter ”A” in four different alphabets sampled from the Omniglot dataset. The different alphabets in the Omniglot datasets have several similar letters which is a challenge to distinguish between, exemplified here by the letter A in Cyrillic, Old Church Slavonic, Greek and Latin.

with this obstacle. This means that the models will be restricted from reaching a very high score as many classes are near impossible to discriminate.

The Omniglot dataset is a suitable dataset for the experiment with increasingly higher number of classes (”Experiment 2”) not only because it has a total of 1623 classes; It also has images with monochrome neutral background with little discriminate details. It is desired to have this type of background as natural images hinder performance according to research discussed in Section 4.3. In Experiment 2, the idea is to isolate the effect of increasing the number of classes, which is possible using Omliglot.

When sampling classes from the Omliglot dataset to create the training set, an alphabet is chosen randomly. Thereafter, classes (letters) are chosen randomly from that alphabet, until the desired number of classes are selected. If one wants more classes than letters in the chosen alphabet, another alphabet is randomly chosen. This continues until the desired number of classes is reached. Remember that when choosing a letter from an alphabet, one gets 12 instances of that letter. The sampling is carried out in this manner to maximise the chance of including different letters, not similar letters from different alphabets as exemplified in Figure 6.1.

Augmentation of images is applied during training, and it is chosen to use the same augmentation as Sabour et al. (2017). The images are shifted randomly up to 2 pixels horizontally or vertically. The transformed images are not stored as a separate image-file or affect the number of samples per class. This transformation is to make sure the model becomes translation invariant as both datasets are centred and preprocessed well.

6.2 Optimization and hyperparameters

The models are all trained using an Adam optimizer (Kingma and Ba (2014)). The results are measured after the epoch which achieved the best test loss. Because of resource limitations, the models run for a set number of epochs. Thus, convergence is not guaranteed. The loss used by the optimizer is 1000 times lower than the losses reported in Section 6.4 and Section 6.5. This scaling is for easier visualizing of the results.

A suitable batch-size is defined for each experiment and is kept constant throughout the experiment. It is assumed that the batch-size has a rather small impact on the experiments, given the scope of the experiments. The learning rate (LR) is also set for the experiments, but is different for the capsule network and the CNN. Preliminary experiments to determine the LR are conducted and explained in Section 6.4.1. The reconstruction loss of the capsule network is scaled by a constant, to control the effect on the total loss. The constant is set to be 0.0005 in both experiments as in the original model (Sabour et al. (2017)).

6.3 Technology

The experiments are written in Python 3.6.8, using the deep learning framework Pytorch. Torchvision is used to download and load the MNIST dataset. The Omniglot dataset is downloaded from the Github page by Lake et al. (2015). Torchnet, Visdom, and TQDM together provide different means of visualizing the real-time progress and performance of the network during training. The results are calculated on a single Tesla P100 with 16 GB memory graphics processor unit. For installment of software and usage of program consult the `readme.md` file in the code repository (Bjørnøy (2020)).

6.4 Experiment 1: Few samples per class

Experiment 1 is designed to answer the first research question: *Will CapsNet perform better than a CNN on a datasets with few samples per class?* It is roughly modeled after the experiment in Schlegel et al. (2018). A capsule network and a CNN is trained on subsets of the MNIST dataset with fewer samples per class. This experiment is interesting as many real-life applications have few samples, an example being medical image diagnosis of very rare diseases. The model architectures are exactly as described in Section 5.

6.4.1 Hyperparameter and experimental setup

Preliminary experiments determine the learning rate (LR) for the different models, using one sample per class (SPC=1). In Figure 6.2, test loss of the CNN model with different learning rates during training is visualized. As one can see from Figure 6.2, a high learning rate of 0.001 results in a high test loss that oscillate around the same value. Further investigation also shows that the training loss behaves similarly, and that the test accuracy stays constant at approximately 10% (accuracy equivalent to random guessing). The most probable reason behind this behaviour is exploding gradients. From Figure 6.2, one can see that the CNN network produced best results with a LR of 0.0001. Furthermore, it had the

highest test accuracy as well as the highest training loss, indicating better generalization results as well. A LR of 0.0001 is therefore used for the CNN baseline in this experiment.

Figure 6.2: Test loss for CNN model trained for 500 epochs on MNIST with one sample per class on the MNIST dataset and a batch-size of 1. The different lines are the CNN model trained with different learning rates from 0.001 to 0.00001.

In Figure 6.3 results of the same preliminary experiment for CapsNet is presented. From Figure 6.3, one can see that a LR of 0.001 produces slightly lower test loss than a LR of 0.01, after a high number of epochs. However, one can see that the slope is steeper for the LR of 0.001, indicating that the model keeps on learning even after many epochs. Thus, a LR of 0.001 is preferred. An investigation of the training loss supports the conclusion; A LR of 0.001 produces consistently higher training loss than a LR of 0.01, indicating better generalization properties. Furthermore, in Figure 6.4, one can see a big difference in performance, favoring a learning rate of 0.001 for the CapsNet in this experiment.

The batch size (BS) is assumed to be independent of the type of model. Different batch-sizes are tested on the CNN model with SPC=1, the test accuracy can be seen in Figure 6.5. The model seem to be very insensitive to changes in batch-size. There is no obvious best choice. However, the model with BS=1 have not only the best result, but also beats the other models quite consistently the last 100 epochs(best in 69% of the measurements). Therefore a batch-size of 1 is chosen for this experiment.

The models are trained for 500 epochs on the MNIST datasets with 1,5,10,20,30,50 and 100 samples per class. Thereafter, the best test accuracy for each model is compared.

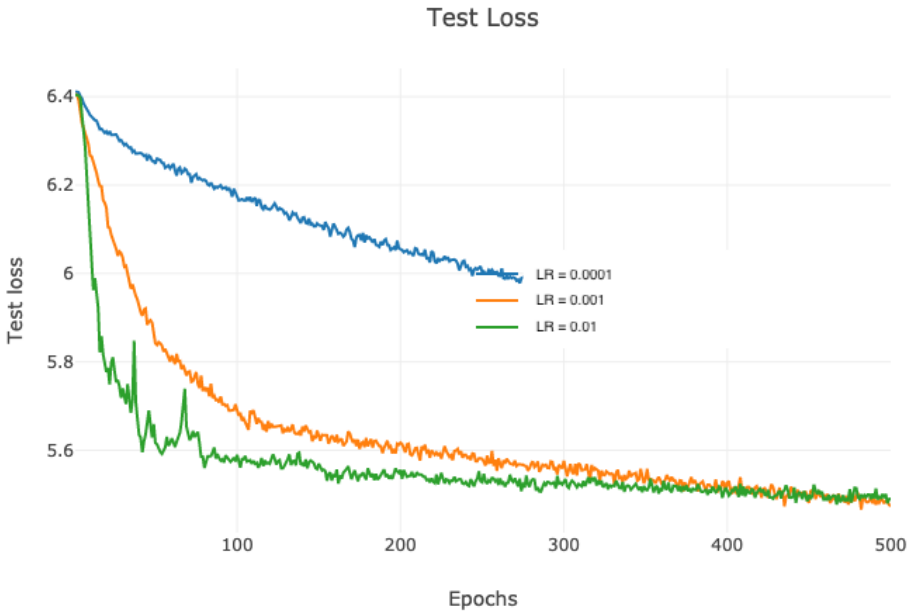


Figure 6.3: Test loss for CapsNet model trained for 500 epochs with one sample per class on the MNIST dataset and a batch-size of 1. The different lines are the Capsnet model trained with different learning rates from 0.01 to 0.0001.

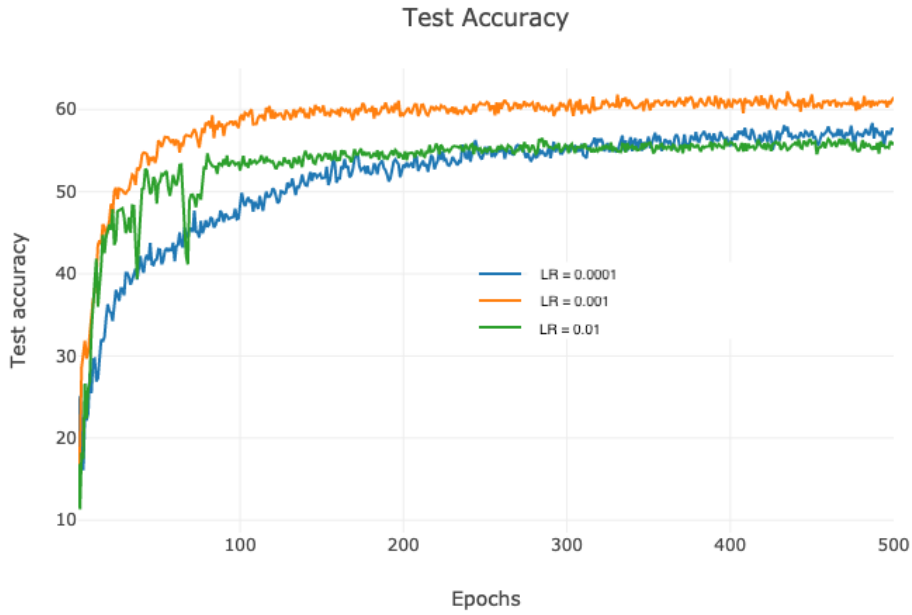


Figure 6.4: Test accuracy for CapsNet model trained for 500 epochs with one sample per class on the MNIST dataset and a batch-size of 1. The different lines are the Capsnet model trained with different learning rates from 0.01 to 0.0001.

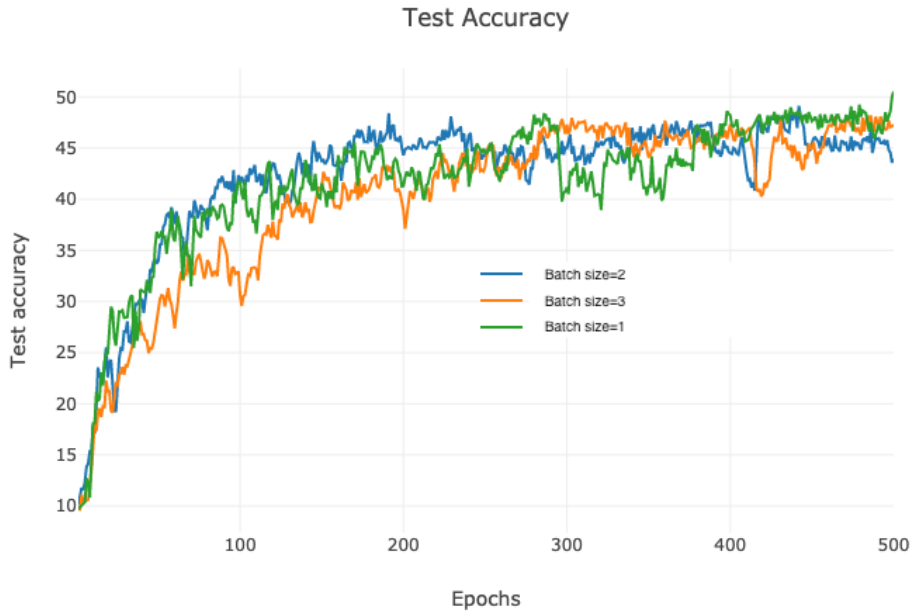


Figure 6.5: Test accuracy for CNN model trained for 500 epochs with one sample per class on the MNIST dataset and a learning rate of 0.0001. The different lines are the Capsnet model trained with Batch size equal to 1,2 and 3

6.4.2 Results

The results from Experiment 1 are presented in Table 6.1, Table 6.2, and Figure 6.6. Figure 6.6 simply displays the test accuracy from Table 6.1 and Table 6.2 for easier comparison.

Table 6.1 and Table 6.2 show that both models unsurprisingly get better test accuracy with more samples per class. It is expected that a network that generalizes well would get a slight increase in training accuracy as the CNN baseline has. However, surprisingly, the CapsNet achieves 100% train accuracy across the experiment. That suggests that the CapsNet might overfit the dataset even with 100 samples per class. This indicates that CapsNets can perform better with a higher reconstruction loss coefficient for datasets with few samples. This observation is backed up by the fact that the reconstruction loss flattens after 20 samples per class in the experiment, indicating that the reconstruction is not prioritised over correct classification of a few training samples.

Figure 6.6 show that the CapsNet outperforms the baseline CNN in every run. Comparing this to the results in Schlegel et al. (2018), their CNN1 model, which is also modeled after the baseline in Sabour et al. (2017), performed better for SPC=1 and SPC=5 and approximately on-par with our CNN for SPC=10,20,30,50,100. The results produced by the capsule network in Experiment 1, on the other hand, achieved great test accuracies. It performed better than both the Capsule network and CNNs in Schlegel et al. (2018), which suggest that even though the implementation is intended to be identical, the model might be sensitive to small deviations. The average difference in accuracy between the CapsNet and the CNN baseline for SPC of 1,5,10, and 20 is 7%.

CNN				
SPC	Tr.loss	Tr.acc	Te.loss	Te.acc
1	31.71	100.00%	42.35	50.53%
5	31.71	100.00%	36.98	75.54%
10	31.92	99.00%	35.13	84.07%
20	31.92	99.00%	34.16	88.66%
30	32.72	95.33%	33.94	89.70%
50	32.19	97.80%	33.12	93.46%
100	32.25	97.50%	32.82	94.80%

Table 6.1: Training and testing results of the CNN baseline on the MNIST dataset with different number of samples from each class. The experiment is carried out with a learning rate of 0.0001, and with BS=1.

Capsule						
SPC	Epochs	Tr.loss	Tr.acc	Rec.loss	Te.loss	Te.acc
1	500	44.88	100.00%	3.12×10^{-2}	54.66	61.67%
5	500	44.94	100.00%	2.82×10^{-2}	50.23	85.46%
10	500	44.72	100.00%	2.80×10^{-2}	48.60	87.37%
20	500	44.75	100.00%	2.70×10^{-2}	47.35	92.33%
30	500	44.73	100.00%	2.74×10^{-2}	46.57	94.45%
50	500	44.75	100.00%	2.71×10^{-2}	45.98	95.88%
100	500	44.74	100.00%	2.72×10^{-2}	45.74	96.96%

Table 6.2: Training and testing results of the CapsNet model on the MNIST dataset with different number of samples from each class. The experiment is carried out with a learning rate of 0.001, and with BS=1.

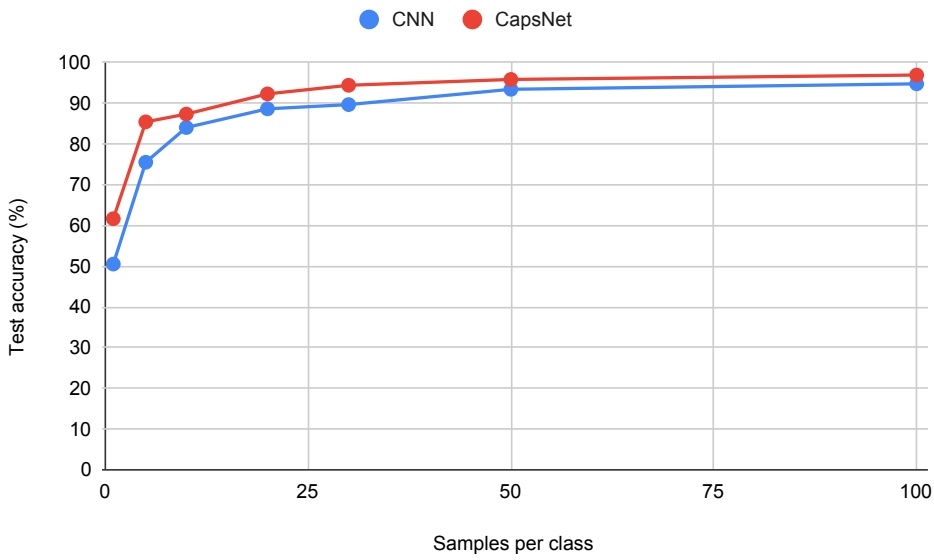


Figure 6.6: Test accuracy for CNN and CapsNet trained for 500 epochs with different number of samples per class on the MNIST dataset. The results are extracted from Table 6.1 and Table 6.1 for better comparison.

6.5 Experiment 2: Many classes

Experiment 2 is designed to answer the second research question: *Will CapsNets perform better than a CNN baseline on datasets with many classes?* The experiment is interesting as there aren't any studies that apply capsule network to image classification with more than 1623 classes. The experiment applies a modified version of the capsule network alongside a modified version of the baseline CNN on subsets of the Omniglot dataset with different number of classes.

6.5.1 Model adaptations to different sizes of input and output

During the course of the experiment 105×105 gray-scale images are used. The initial feature extractor in both the baseline model and the Capsnet are created for a fixed input size, and must therefore be modified to fit the 105×105 images. The baseline and CapsNet feature extractor have slightly different task. The baselines feature extractor maps the 28×28 input to 128 16×16 feature maps while the feature extractor in the Capsnet maps the input to 256 20×20 feature maps. It is decided to keep the dimensions of the mapping of the feature extractors constant. However, the feature extractors are modified to be compatible with the 105×105 input.

The CNN baseline feature extractor for 105×105 images has 3 convolutional layers with 256, 256 and 128 channels. The kernel size is 9×9, 9×9 and 6×6. The stride is 2, 2, 1. The feature extractor for 28×28 input contains 2.46M parameters, while the feature extractor for 105×105 input contains 6.51M parameters.

The CapsNet feature extractor for 105×105 images has 3 convolutional layers, all with 256 channels. The kernel size is 9×9,9×9 and 2×2. The stride is 2, 2, 1. The feature extractor for 28×28 input contains 0.02M parameters, while the feature extractor for 105×105 input contains 5.59M parameters.

In addition to being dependent on a fixed size of input, the models also return output of a given size. The experiment requires the networks to handle the wide span of 2 to 1623 classes, thus requiring the models to produce outputs of different sizes. To achieve that, the models' classifier must be modified so that the number of output neurons/capsules matches the number of classes

The baseline model is not notably affected by the modifications; They lead to a 3% increase in parameters. With a 10-classes output, the classifier require 10.8M parameters. With a 1623-classes output, the classifier require 11.1M parameters. Table 6.3 display the number of parameters required by different layers for two different outputs. The figure makes it clear that only the number of parameters in the final layer is affected by the modification.

The modifications have a bigger impact for the CapsNet, as Table 6.4 shows. For both models the parameters in the last layer grow linearly with respect to the number of classes. However, the magnitude of the number of parameters in the Capsnet is 3 times the magnitude of paramteres in the baseline (198 neurons for the baseline vs 147456 neurons in the CapsNet). This causes the total increase of parameters in the two networks to be very different with increasing number of classes. The difference is made clear by comparing Table 6.3 to Table 6.4.

CNN baseline classifier			
classes	1st layer	2nd layer	3rd layer
10	$10.75M$	$0.06M$	$0.00M$
1623	$10.75M$	$0.06M$	$0.31M$

Table 6.3: The table consist of the number of parameters required by the layers of the CNN baseline's classifier with respect to different output/categories to classify. The increase in output classes only affect the number of parameters in the last layer.

CapsNet classifier			
classes	Primary capsules	Class capsules	Reconstruction network
10	$5.31M$	$1.47M$	$11.91M$
1623	$5.31M$	$239.32M$	$25.12M$

Table 6.4: The table consist of the number of parameters required by the layers in the CapsNet's classifier with respect to different output/categories to classify. The increase in output classes only affect the number of parameters in the last layer, as well as the first layer of the reconstruction network.

The class capsules are already quite big before they are modified to handle approximately 160 times more classes. The model increases 785% in parameters when adapted to 1623 classes compared to 2 classes. The vast number of parameters when dealing with 1623 classes causes the model to try allocate more resources than available (16GiB) in the GPU. This is a practical problem with regards to running the model on the resources available in this thesis. To tackle this problem, an alternative model is used for the dataset with 1623 classes; The dimension of the capsules in the *Class capsules* layer is reduced to 10. This hampers the experiment which on all other runs will use 16 dimension *Class capsules* like in Sabour et al. (2017). Note that the decoder increase both due to increased input and output as it also needs to reconstruct a bigger image.

6.5.2 Hyperparameters and experimental setup

The number of model parameters (MP) for the models are reported in Table 6.5 and Table 6.6 as the different models scales very differently. The batch size is set to 7 across the experiment as the capsule network has limited GPU memory and the priority is to keep the class capsules dimensions as high as possible. It is assumed that the finding presented in Figure 6.5 holds, namely that the models are insensitive to the tuning of batch size. As in Experiemnt 1, the learning rate for the capsule network is set to 0.001, while the learning rate for the CNN is 0.00001. It is thus assumed that the optimal learning rate does not depend on the number of classes. Throughout the experiment, all classes contains 12 samples in the training set.

The experiment train the models for 200 epochs on samples of the Omniglot datasets with 2,10,100,400 and 1623 samples per classes.

6.5.3 Results

The results of the experiment are presented in Table 6.5 and Table 6.6. For easier comparison, the test accuracy is displayed in Figure 6.7. Furthermore, the number of parameters are visualized in Figure 6.8.

Table 6.5 and Table 6.6 show that the CapsNet once again scores very high on training accuracy compared to the CNN. It does not follow the trend in test accuracy like the results from the CNN. This indicates that the CapsNet not only overfits too some degree on small datasets, but also on larger datasets. However, this could be a affected by the few number of samples per class, as the Omniglot datasets only contains 12 training samples per class.

The tables, but more clearly Figure 6.7, show that the CNN baseline performs sub par compared to capsule network on all number of classes, except from on 1623 classes. When dealing with 1623 classes, the capsule network collapsed in performance. As explained previously, the capsule model was modified before it was applied to the dataset with 1623 classes due to limited computational resources. That is a very probable cause of the collapse. The capsule network provides much better test accuracy on 400 classes than the baseline CNN. On the other hand, the CNN scales much better to an increase in number of classes as its model has 79.5% less parameters than a capsule network for 400 classes and a whole 90.3% for 1623. Figure 6.8 illustrates the vast difference between the models.

The most notable result in this experiment is the capsule network's 52.12% test accuracy in contrast with the CNN baseline with a 20.63% test accuracy for 400 classes on the Omniglot dataset.

CNN						
NC	MP	GPU	Tr.loss	Tr.acc	Te.loss	Te.acc
2	$17.3M$	$1.51GiB$	6.803	100.00%	6.802	100.00%
10	$17.3M$	$1.51GiB$	31.74	100%	33.49	92.50%
100	$17.3M$	$1.51GiB$	86.33	64.33	90.57	45.75
400	$17.4M$	$1.52GiB$	124.5	29.47%	125.7	20.63%
$1623*$	$17.6M$	$1.53GiB$	159.6	3.877%	159.9	2.596%

Table 6.5: Many classes experiment. NC stands for number of classes, MP for million of parameters. The experiment is carried out with a learning rate of 0.00001. The (*) on NC=1623 is because the model trained only 100 epochs, the reason being limited resources.

Capsule							
NC	MP	GPU	Tr.loss	Tr.acc	Rec.loss	Te.loss	Te.acc
2	$23.0M$	$1.3GiB$	22.22	100.00%	3.73×10^{-2}	26.73	100.00%
10	$24.3M$	$1.4GiB$	44.85	100.00%	3.68×10^{-2}	46.47	93.75
100	$38.3M$	$2.9GiB$	76.71	99.92%	3.84×10^{-2}	78.39	74.00%
400	$85.0M$	$6.7GiB$	202.9	97.39	3.99×10^{-2}	204.7	51.12%
1623*	$180M$	$16GiB*$	538.7	0.1027%	4.11×10^{-2}	538.5	0.1078%

Table 6.6: Many classes experiment. NC stands for number of classes, MP for million of parameters. The experiment is carried out with a learning rate of 0.00001. The (*) on NC=1623 is because the model trained only 100 epochs, the reason being limited resources.

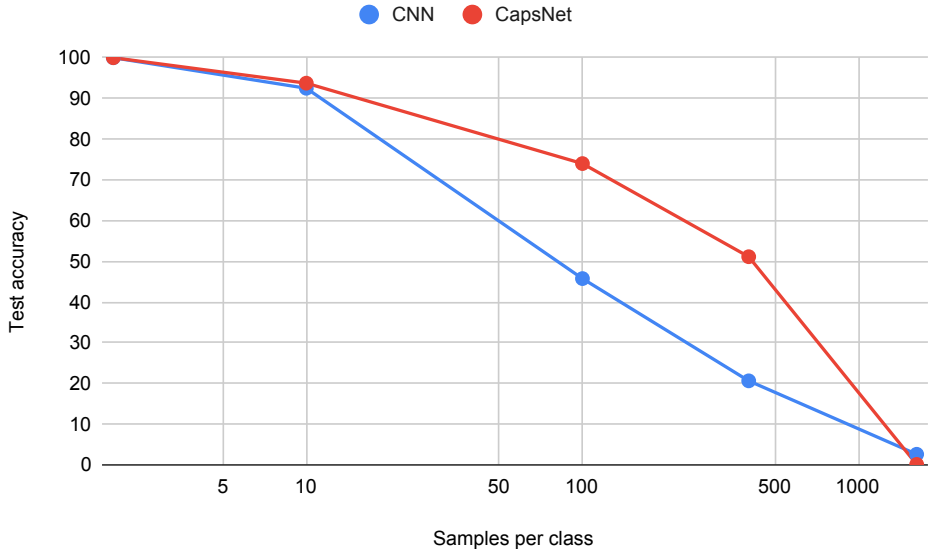


Figure 6.7: Test accuracy for CNN and CapsNet trained for 200 epochs with different number of number of classes on the MNIST dataset. The results are extracted from Table 6.5 and Table 6.6 for better comparison.(PS: the X-axis har nothing to do with samples per class)

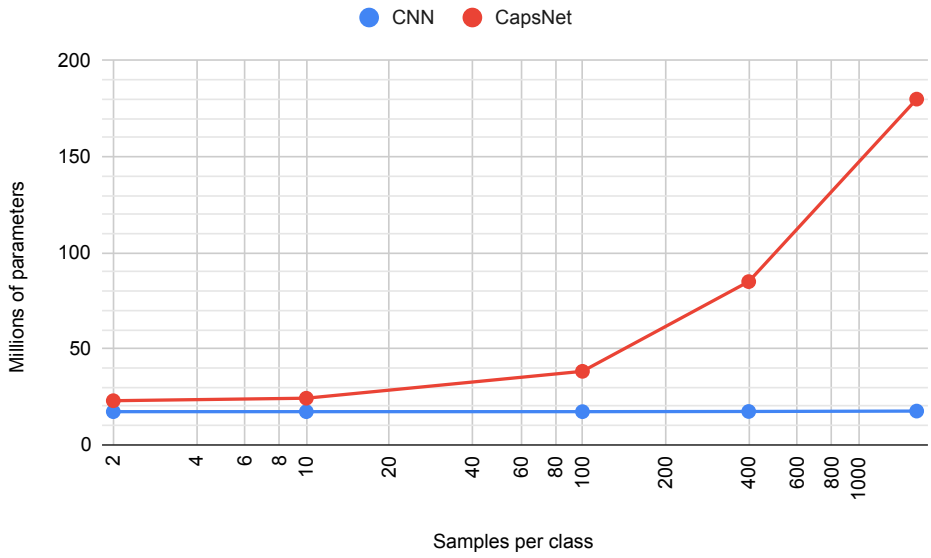


Figure 6.8: Number of model parameters for CNN and CapsNet trained for 200 epochs with different number of number of classes on the MNIST dataset. The results are extracted from Table 6.5 and Table 6.6 for better comparison.(PS: the X-axis har nothing to do with samples per class)

6.6 Summary

Preliminary experiments determined the learning rate in Experiment 1. The LR was set to 0.0001 for the CNN model and 0.001 for the Caps net. The batch size is assumed model-independent and insensitive to different datasets.

Experiment 1 shows that the implemented Capsnet outperforms the baseline CNN for all tested number of samples per class. The capsule network implemented in this thesis also outperformed the results given in Schlegel et al. (2018).

When planning experiment 2 the problem of large number of parameters in the capsule network for high number of classes had to be addressed. It was decided to adjust the final layer, so that it only had 10 dimensional capsules (instead of 16).

Experiment 2 reveals that the Capsnet outperformes the baseline CNN for all number of classes except for 1623 classes, for which the Capsnet collapsed in performance. The collapse might be due to the modification that was made to the final layer. The results from Experiment 2 highlights the computational challenges when using capsule networks on datasets with large number of classes.

Chapter 7

Discussion

This chapter presents a discussion about the major findings from the carried out experiments and how they answer the research questions. The limitations of the conclusion that can be drawn are considered, unexpected or inconclusive results are discussed and suggestions for further research are given.

The results from Experiment 1, presented in Figure 6.6, show that the capsule network consistently beats the test accuracy of the CNN baseline. Compared to the study by Schlegel et al. (2018), the CapsNet in this experiment did better for the dataset with 1,5, 10 and 20 samples, while the CNN baseline had approximately the same results but bigger deviations with fewer samples. A possible explanation is that the random samples matter more to the outcome when they are one of few contrary to one of many.

The experiment in the article by Schlegel et al. (2018) and the experiments in this thesis implemented a replica of the CapsNet from Sabour et al. (2017). The differences in results are biggest for few samples per class. But the results evens out at 100 samples. Possible explanations for the differences could be variations in implementation or the way they implemented early stopping and define convergence, a description of this is not included Schlegel et al. (2018). Another explanation is as mentioned the fact the datasets draws random samples that causes these semi-consistent differences.

Some hyperparameters in the models in Experiment 1 were determined by limited preliminary experiments. However, they were not tuned to fit each application. Other hyperparameters, as the reconstruction loss coefficient, were simply set. To find the optimal combination of hyper parameters crass-validation on a multidimensional grid of hyperparameters should ideally be carried out. However, as is the case with most models with many hyper-parameters, this is not feasible due to time and computational constraints. A thorough investigation of hyperparameters could lead to different results.

The results of Experiment 1 suggests that especially the reconstruction loss coefficient should be tuned. The experiment shows that the capsule model has a very high train accuracy compared to the baseline. This indicates that the CapsNet overfits on the training data. If that is the case, the results could benefit greatly from increasing the coefficient. However such an investigation is not within the scope of this thesis.

The results from Experiment 2, presented in Figure 6.7, indicate that the Capsule network outperform the baseline CNN for all number of classes except for 1623. However,the originally thought out experiment with 16 dimensional class capsules could not be performed on the 1623 classes. Instead the dimensions of the class capsules were reduced to 10, decreasing the models capacity of approximating functions. Setting aside the results from 1623 classes, the CapsNet achieved consistently better test accuracy on ≤ 400 classes compared to the CNN.

The hyperparameters in Experiment 2 is simply determined in the same manner as in Experiment 1. Thus, the previous discussion of hyperparameters in Experiment 1 holds for Experiment 2 as well. CapsNets training accuracy was also very high in Experiment 2 compared to the CNN baseline, indicating that an increase in the reconstruction loss coefficient would help the network perform better on many classes.

Whether a general capsule network architecture is a better option than CNNs for datasets with many classes is linked to a question of resource efficiency. The capsule network scales very badly as is, and guzzle GPU memory as shown in Table 6.6 due to the increase in parameters. The CNN's GPU memory load hardly increase with an increase $< 1\%$ from 2 classes to 400 classes, compared to CapsNet's 515% increase in GPU memory load. If one was to conduct an experiment with a CNN that has equally many parameters as the capsule network, the CNN would likely perform comparatively to the capsule network on dataset with many classes. Increasing expressive power (increasing the number of parameters) as a dataset becomes more complex is a textbook move.

The results from Experiment 2 imply that capsule networks are good at classifying datasets with many classes. As CapsNet's feature extractor consists of a regular convolutional network, it scales similarly to CNNs for bigger images. However, for datasets with many classes CapsNet scales much worse than CNNs. Related work that could help scale for bigger outputs are class-independant capsules in the last layer, similar to Rawlinson et al. (2018); Rajasegaran et al. (2019). The idea is that class-independent capsules can increase the expressive power of the channels as the same attributes like rotation does not need to be learned for each capsule. This way much less parameters would be needed in the last layer, while keeping, even increasing, the expressive power of the last layer, making scaling more comparable to the CNN baseline.

A possible explanation for CapsNets comparably better performance than CNN is that the there are only 12 samples per class in the Omniglot dataset. The results from the previous experiment suggests a capsule network performs better than the baseline CNN in the 10-20 samples per class range.

Chapter 8

Conclusion

Capsule networks are developed to tackle the problem of spatial relationships in images. The field is relatively new, but many promising applications, such as processing CT scans from COVID-19 patients have been seen. The thesis examined the performance of capsule networks on different applications, guided by two research questions.

Experiment 1 aimed to answer the first research question; *Will CapsNet perform better than a CNN on a datasets with few samples per class?* The Capsule network architecture from Sabour et al. (2017) performed better than the baseline CNN network on datasets with fewer samples. The test accuracy gap between the models was on average 7% for the lowest number of samples(1 to 20). These results are in conflict with other Schlegel et al. (2018).

Experiment 2 aimed to answer the second research question;*Will CapsNets perform better than a CNN baseline on datasets with many classes?* The modified CapsNet achieved a 52.12% test accuracy on the Omniglot dataset with 400 classes, in contrast with the CNN baseline with a 20.63% test accuracy. These results where overshadowed by CapsNets poor scaling to larger outputs. The CapsNet model modified for 400 classes is compared in this paragraph had 489% more model parameters than the compared CNN baseline. Because of this discrepancy between the models the results from this experiment is somewhat undermined.

A further investigation of capsule networks on datasets with few samples as there are conflicting view on whether capsule networks show promise in the field. In addition, a thorough investigation of the hyperparameters of capsule networks should be undertaken, however this requires large resources. It would also be interesting to further investigate capsule networks with fewer parameters for many classes, as the version testet in this thesis does not scale well.

Relevant work for addressing the scaling problem is the work of Bahadori (2018) and Rajasegaran et al. (2019), whom both suggested class-independent decoder.

Hopefully the experiments in this thesis can supplement the field in better understanding the properties of capsule networks with datasets with few samples and many classes.

Bibliography

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., Bengio, Y., Bergeron, A., Bergstra, J., Bisson, V., Bleecher Snyder, J., Bouchard, N., Boulanger-Lewandowski, N., Bouthillier, X., de Brébisson, A., Breuleux, O., Carrier, P.L., Cho, K., Chorowski, J., Christiano, P., Cooijmans, T., Côté, M.A., Côté, M., Courville, A., Dauphin, Y.N., Delalleau, O., Demouth, J., Desjardins, G., Dieleman, S., Dinh, L., Ducoffe, M., Dumoulin, V., Ebrahimi Kahou, S., Erhan, D., Fan, Z., Firat, O., Germain, M., Glorot, X., Goodfellow, I., Graham, M., Gulcehre, C., Hamel, P., Harlouchet, I., Heng, J.P., Hidasi, B., Honari, S., Jain, A., Jean, S., Jia, K., Korobov, M., Kulkarni, V., Lamb, A., Lamblin, P., Larsen, E., Laurent, C., Lee, S., Lefrancois, S., Lemieux, S., Léonard, N., Lin, Z., Livezey, J.A., Lorenz, C., Lowin, J., Ma, Q., Manzagol, P.A., Mastropietro, O., McGibbon, R.T., Memisevic, R., van Merriënboer, B., Michalski, V., Mirza, M., Orlandi, A., Pal, C., Pascanu, R., Pezeshki, M., Raffel, C., Renshaw, D., Rocklin, M., Romero, A., Roth, M., Sadowski, P., Salvatier, J., Savard, F., Schlüter, J., Schulman, J., Schwartz, G., Serban, I.V., Serdyuk, D., Shabanian, S., Simon, E., Spieckermann, S., Subramanyam, S.R., Sygnowski, J., Tanguay, J., van Tulder, G., Turian, J., Urban, S., Vincent, P., Visin, F., de Vries, H., Warde-Farley, D., Webb, D.J., Willson, M., Xu, K., Xue, L., Yao, L., Zhang, S., Zhang, Y. (Theano Development Team), 2016. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints abs/1605.02688. URL: `http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html`.

Bahadori, M.T., 2018. Spectral capsule networks .

Bjørnøy, H., 2020. Source code for the project. URL: `https://github.com/hbjornoy/capsnet`.

Boser, B.E., Guyon, I.M., Vapnik, V.N., 1992. A training algorithm for optimal margin classifiers, in: Proceedings of the fifth annual workshop on Computational learning theory, pp. 144–152.

Cireşan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J., 2011.

High-performance neural networks for visual object classification. arXiv preprint arXiv:1102.0183 .

Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L., 2009. ImageNet: A Large-Scale Hierarchical Image Database, in: CVPR09.

Edraki, M., Rahnavard, N., Shah, M., 2020. Subspace capsule network. arXiv preprint arXiv:2002.02924 .

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets, in: Advances in neural information processing systems, pp. 2672–2680.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

Hinton, G.E., Krizhevsky, A., Wang, S.D., 2011. Transforming auto-encoders, in: International Conference on Artificial Neural Networks, Springer. pp. 44–51.

Hinton, G.E., Sabour, S., Frosst, N., 2018. Matrix capsules with em routing https://openreview.net/pdf?id=HJWLfGWRb.

Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708.

Iwasaki, K., 2018. A barebones cuda-enabled pytorch implementation of the capsnet architecture in the paper "dynamic routing between capsules" by kenta iwasaki on behalf of gram.ai. URL: https://github.com/gram-ai/capsule-networks.

Johnson, A.E., Pollard, T.J., Shen, L., Li-wei, H.L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L.A., Mark, R.G., 2016. Mimic-iii, a freely accessible critical care database. Scientific data 3, 160035.

Kelley, H.J., 1960. Gradient theory of optimal flight paths. Ars Journal 30, 947–954.

Kim, J., Jang, S., Park, E., Choi, S., 2020. Text classification using capsules. Neurocomputing 376, 214–221.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Kosiorek, A., Sabour, S., Teh, Y.W., Hinton, G.E., 2019. Stacked capsule autoencoders, in: Advances in Neural Information Processing Systems, pp. 15486–15496.

Krizhevsky, A., Hinton, G., 2009. Learning multiple layers of features from tiny images. Technical Report. Citeseer.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

Lake, B.M., Salakhutdinov, R., Tenenbaum, J.B., 2015. Human-level concept learning through probabilistic program induction. Science 350, 1332–1338.

LaLonde, R., Bagci, U., 2018. Capsules for object segmentation. arXiv preprint arXiv:1804.04241 .

LaLonde, R., Xu, Z., Jain, S., Bagci, U., 2020. Capsules for biomedical image segmentation. arXiv preprint arXiv:2004.04736 .

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324.

LeCun, Y., Huang, F.J., Bottou, L., et al., 2004. Learning methods for generic object recognition with invariance to pose and lighting, in: CVPR (2), Citeseer. pp. 97–104.

Mobiny, A., Cicalese, P.A., Zare, S., Yuan, P., Abavisani, M., Wu, C.C., Ahuja, J., de Groot, P.M., Van Nguyen, H., 2020. Radiologist-level covid-19 detection using ct scans with detail-oriented capsule networks. arXiv preprint arXiv:2004.07407 .

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y., 2011. Reading digits in natural images with unsupervised feature learning .

Olson, M., Wyner, A., Berk, R., 2018. Modern neural networks generalize on small data sets, in: Advances in Neural Information Processing Systems, pp. 3619–3628.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A., 2017a. Automatic differentiation in pytorch .

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A., 2017b. docs for torch.nn. URL: `https://pytorch.org/docs/stable/nn.html#torch.nn.MaxPool2d`.

Peer, D., Stabinger, S., Rodriguez-Sanchez, A., 2018. Training deep capsule networks. arXiv preprint arXiv:1812.09707 .

Rajasegaran, J., Jayasundara, V., Jayasekara, S., Jayasekara, H., Seneviratne, S., Rodrigo, R., 2019. Deepcaps: Going deeper with capsule networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 10725–10733.

Rawlinson, D., Ahmed, A., Kowadlo, G., 2018. Sparse unsupervised capsules generalize better. arXiv preprint arXiv:1804.06094 .

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer. pp. 234–241.

Sabour, S., Frosst, N., Hinton, G.E., 2017. Dynamic routing between capsules, in: Advances in neural information processing systems, pp. 3856–3866. http://papers.nips.cc/paper/6975-dynamic-routing-between-capsules.pdf.

Schlegel, K., Neubert, P., Protzel, P., 2018. Comparison of data efficiency in dynamic routing for capsule networks .

Srivastava, S., Khurana, P., Tewari, V., 2018. Identifying aggression and toxicity in comments using capsule network, in: Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018), pp. 98–105.

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R., 2013. Regularization of neural networks using dropconnect, in: International conference on machine learning, pp. 1058–1066.

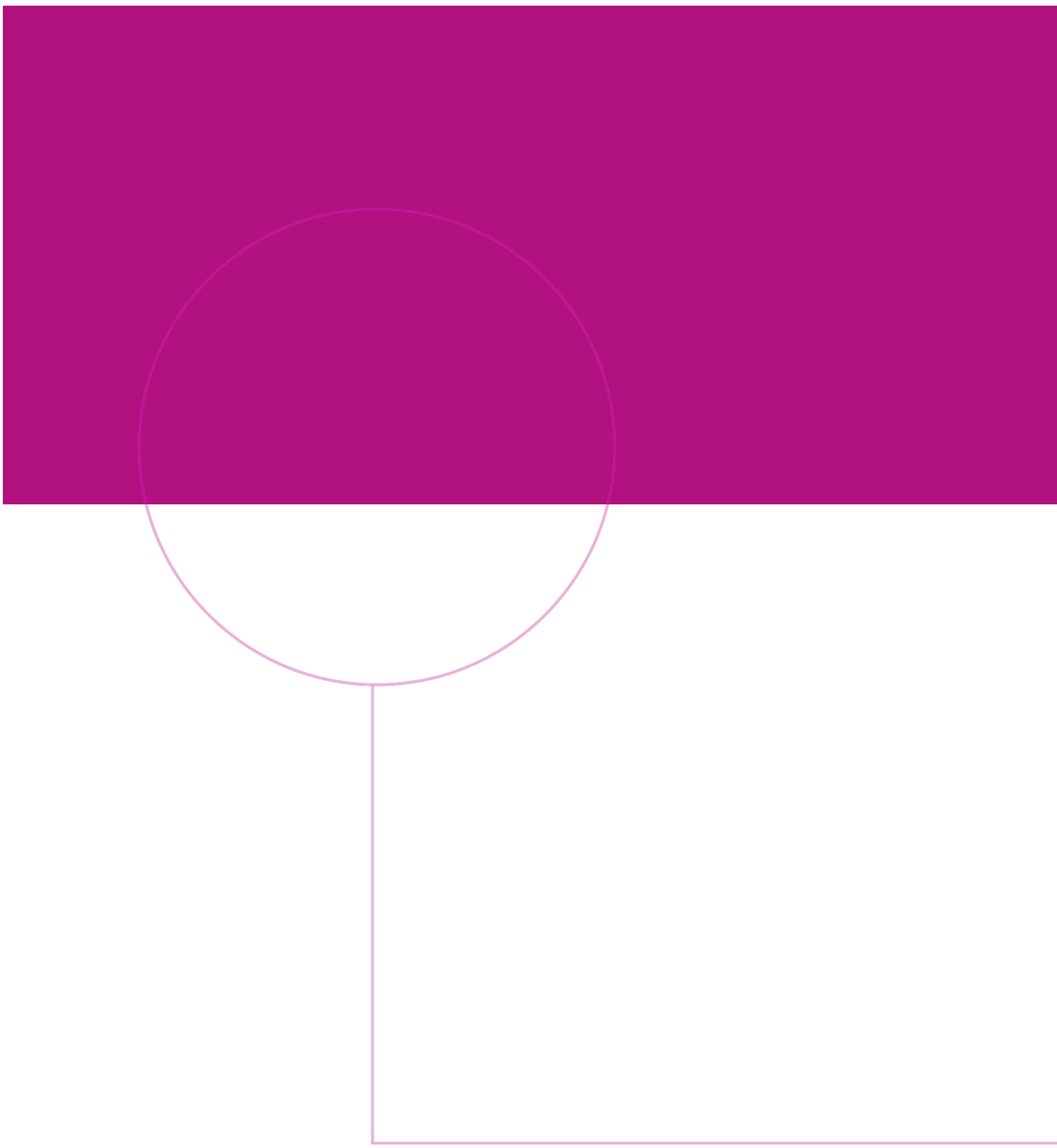
Xi, E., Bing, S., Jin, Y., 2017. Capsule network performance on complex data. arXiv preprint arXiv:1712.03480 .

Xia, C., Zhang, C., Yan, X., Chang, Y., Yu, P.S., 2018. Zero-shot user intent detection via capsule neural networks. CoRR abs/1809.00385. URL: http://arxiv.org/abs/1809.00385, arXiv:1809.00385.

Xiao, H., Rasul, K., Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 .

Zhang, L., Edraki, M., Qi, G.J., 2018. Cappronet: Deep feature learning via orthogonal projections onto capsule subspaces, in: Advances in Neural Information Processing Systems, pp. 5814–5823.

Zhao, W., Ye, J., Yang, M., Lei, Z., Zhang, S., Zhao, Z., 2018. Investigating capsule networks with dynamic routing for text classification. arXiv preprint arXiv:1804.00538 .



NTNU

Norwegian University of
Science and Technology