DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2018*

CapsNet Comprehension of Objects in Different Rotational Views

A comparative study between capsule and convolutional networks

Principal for this thesis was Sellpy (Sellhelp AB)
Supervisor at Sellpy was Max Berggren
Supervisor at CSC was Johan Gustavsson
Examinator was Elena Troubitsyna

MARTIN ENGELIN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Abstract

Capsule network (CapsNet) is a new and exciting approach to computer vision. In the small amount of research published so far, it has shown to be good at generalizing complex objects and perform well even when the images are skewed or the objects are seen from unfamiliar viewpoints. This thesis further tests this ability of CapsNet by comparing it to convolutional networks (ConvNets) on the task to understand images of clothing in different rotational views. Even though the ConvNets have a higher classification accuracy than CapsNets, the results indicate that CapsNets are better at understanding the clothes when viewed in different rotational views.

Sammanfattning

Capsule network (CapsNet) är en ny typ av neuralt nätverk för datorseende, som framförallt presterar bra även då bilderna är förvrängda eller sedda från obekanta vinklar. Den här uppsatsen testar CapsNets förmåga att förstå klädesobjekt sedda ur olika synviklar genom att göra en jämförelse med ConvNets. Resultaten visar att, trots att ConvNets har en högre exakthet i sin klassificering, är CapsNets bättre på att förstå kläderna sedda från olika synvinklar.

Contents

1 Introduction

Computer vision is an exciting subject whose aim is to enable computers to understand images, in order to automate tasks that the human visual system can do. It is a subject that has been explored ever since the late 1960's, yet computer vision is in need of improvement before it is on par with human vision.

The most successful approach to achieve computer vision today is convolutional neural networks (ConvNets). ConvNets use convolutional layers in order to obtain features of an image. The convolutional layers apply convolution with trainable filters on the input which outputs feature maps [8]. For example in face recognition a feature might be the mouth or the nose. These features are then passed to fully-connected layers that performs the classification of the image.

Although ConvNets perform very well in many cases [35], they have their limitations. One limitation is that they have no or little perspective of orientational or spatial relationships between the features of an object. As a response to this problem, Geoffrey Hinton has during several years proposed an alternative, to use networks consisting of capsules [12]. Hinton et al recently published an article presenting such a network, called Capsule Network (CapsNet) [33]. In CapsNets, the fundamental component of the network is not the traditional neurons but rather a capsule of neurons. A capsule is similar to a normal neuron but the main difference is that both the input and output of capsules are vectors rather than scalar values. This design enables CapsNets to not only detect features but also detect poses of the features and relations between them.

1.1 Principal

Despite the restrictions of computer vision today, it is still useful in different kinds of applications and companies. One such company is Sellpy. Sellpy is a company whose idea is to simplify the process of selling used items online, by offering a service that lists items, interact with buyers, as well as handle payment and delivery. Since Sellpy sells used goods the value for each item is smaller than that of any other normal e-commerce, which makes efficiency especially important. Therefore, the company uses machine learning.

When processing a new item a first step is to manually classify item characteristics, such as type, brand, material and color. These features are

then used in a machine learning model that predicts the price of the item. If a computer could find these classifications using images of the item, rather than it being done manually, it could be a source for increased efficiency of the listings.

The images in Sellpy's database are well photographed, with the object facing forward, in center with a white background. The object is then rotated and photographed from different angles. The image database can thus be split into two separate data sets, one with images of all rotations and one using only the images where the object is faced forward. For simplicity we'll call these two data sets SFF (Sellpy Face Forward) and SRO (Sellpy Rotated Objects).

1.2 Objective

When introduced, CapsNets had state-of-the art performance on the MNIST data set of handwritten digits. It showed especial good performance when tested on the affnist data set where the objects in MNIST had been randomly transformed [33]. It has also been tested on 3D objects in the smallNORB data set and performed great results [13]. This aim of this thesis is to test CapsNets on another type of data set. The data sets to be used contain images of clothes, and the focus will be to test its ability to understand the different rotational views in SRO.

1.3 Research Question

The images in SFF is not the kind of images where ConvNets normally have a problem and therefore not where CapsNets are expected to outperform ConvNets. CapsNets have shown to handle affine transformations of objects well [33] and can therefore be expected to perform better than ConvNets on the SRO data set.

Is capsule networks better than convolutional networks at recognizing clothes viewed in different rotational views? This will be answered by comparing CapsNet and ConvNet performances when using only images from the front of the object (SFF), and all horizontal views of an object (SRO).

2 Background

In this section a thorough background of fundamental rudiments of machine learning and neural networks is given. The ideas of capsule networks are later explained.

2.1 Machine Learning

Machine learning is a subject in computer science concerned with the implementation of computer software that can learn autonomously [14]. A supervised learning algorithm is an algorithm that given experience on performing a class of tasks with performance measures, improves performance with experience [8]. The goal of such algorithms is to approximate some function $f^*(x) = y$, where x is a set of data. The algorithm can be seen as a model of adjustable parameters that make up the function to be approximated.

2.1.1 Classification

There are many kinds of tasks that learning algorithms can be useful for. Two of the most common types of tasks are **classification** and **regression** [3]. Classification refers to the task of assigning a given input vector $\mathbf{X}$ to one of $\mathbf{K}$ distinct classes. The output can be either one of the distinct classes or a probability vector of the classes.

$$f : \mathbb{R}^n \to \{\mathbf{1},...,\mathbf{K}\}$$

Regression is similar to classification but instead of assigning the input vector $\mathbf{X}$ to a class $\mathbf{K}$, $\mathbf{X}$ is assigned to a real value $\mathbf{V}$.

$$f : \mathbb{R}^n \to \mathbf{V} \in \mathbb{R}$$

2.1.2 Training & Test

The process of giving a learning algorithm experience is called training. During the training phase the adjustable parameters of the model are determined based on a training data set, which is a data set of input vectors and their corresponding correct classification, often called labels.

Once a model is trained its performance is determined during the test phase, which uses a separate but similar data set to the training phase.

The action of mapping an input vector to a class is called inference and during testing the model uses inference on the test set in order to generate predictions. These predictions are then compared to the labels of the test set in order to calculate the performance of the model [3].

2.1.3 Overfitting

The aim of training a model is to minimize the error on the training set T, compared to the true function. As input vectors from T are processed, the adjustable parameters as well as the approximating function f^* of the model changes. Every possible training set T_i will generate a corresponding approximation function f_i. A model can be assessed using two terms: bias and variance.

The bias of a model is defined as the difference of the average output of the model given different input sets compared to the desired function. The variance is a measure of how much the model output varies between data sets. The total error of a model is minimal when its bias and variance are minimal [27].

At the beginning of the process of training a model, when the model has seen few input vectors, its bias will be high and its variance low. This is because the output will be far from the desired output and the data has not influenced the model much yet. Later during the training the model's bias will decrease as it learns the desired function. However if trained too long the model will learn the specific noise of the training set and its variance will increase. This situation is called overfitting, since the model fits the training set better than the true function [20].

2.1.4 Accuracy & Loss

The most common performance measure of a learning algorithm is its **accuracy**. Accuracy is simply the proportion of the test input data that is correctly classified by the algorithm. This is closely related to **error rate** which is the proportion of the input data that is incorrectly classified [8].

Another measure of a learning algorithm is its **loss**. A model's adjustable parameters are altered during training in order to estimate the underlying function in the best possible way. This process is only possible once it is defined what the best estimation is. The best estimation is defined as the estimation that minimizes the output of some loss function. The lower the

loss of a learning algorithm is, the more robust it is. [18].

A learning algorithm's accuracy and loss can be measured on both the training and the validation sets. These measures can be used to detect overfitting in a model. A model is most likely overfitted when the training measures are greater than the validation measures.

2.1.5 Performance Comparison

It is often useful to be able to compare the performance of two learning algorithms. A straight forward way to do this is to train both algorithms on the same training set, evaluate them on the same test set and then simply compare their error rates. This approach is however problematic since the test set is only a subset of the entire data set. Performing an evaluation using only this subset often leads to misrepresentation of the results [34][7].

A better approach to use when comparing two learning algorithms is **k-fold cross-validation**. This approach includes partitioning the entire data set D into k separate subsets T_1, ..., T_k. k trials are conducted where T_i is used as test set and the union of the other $T_j, j \neq i$ are used as training set [7][29]. This way the entire data set is used for testing the training algorithms while at the same time separating the training and test sets.

2.2 Neural Networks

An Artificial Neural Network is a computing system inspired by the biological neural networks that make up our brains. Such networks have been frequently used over the last decades for applications of pattern recognition, such as image and voice recognition [25] [19]. As a computing system, neural networks are used to approximate some function $f^*(x) = y$ by using a network of neurons in different layers [3].

2.2.1 Artificial Neuron

The nodes, or neurons, that are fundamental in a neural network are processing units. Neurons consists of three basic elements: **synapses**, an **adder** and an **activation function**. The synapses are the input to the neuron, and each synapse is characterized by its own weight. The output of a synapse is its input multiplied with its weight. The adder sums the output of the synapses, together with the neurons bias. Finally the activation function

squashes the sum into a certain interval, typically [0,1] or [-1,1]. Normal activation functions are ReLU and softmax [19][4].

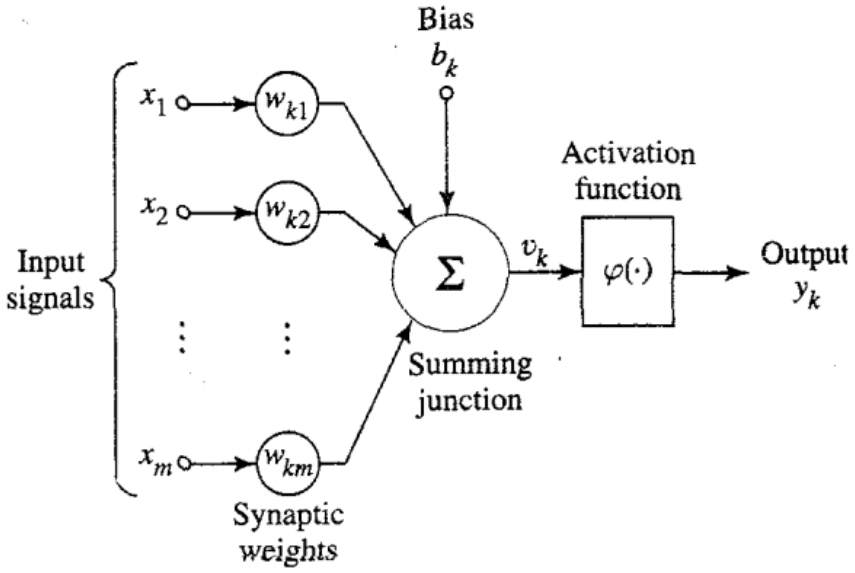

Figure 1: *Graphical represention of a neuron, adapted from* [9]

2.2.2 Feedforward Networks

One of the most useful neural network architecture is **feedforward networks**, which consist of multiple layers of neurons with adaptive weights. It is called feedforward because information is forwarded through the network from some input to some output without any backwards connections in the network [8]. Other than single-layer networks which are rarely used in practice, feedforward networks use so called hidden layers of neurons in between the input and output layers [11]. Such networks are often called Multi-layer Perceptrons [3].

A layer in a feedforward network is a list of neurons. The width of a layer is defined as the amount of neurons and its depth is determined by where in the network the layer is present. The neurons of a layer of depth n is typically connected to the neurons in the adjacent layers $n-1$ and $n+1$ [15].

In a mathematical sense every layer of a neural network is a function of

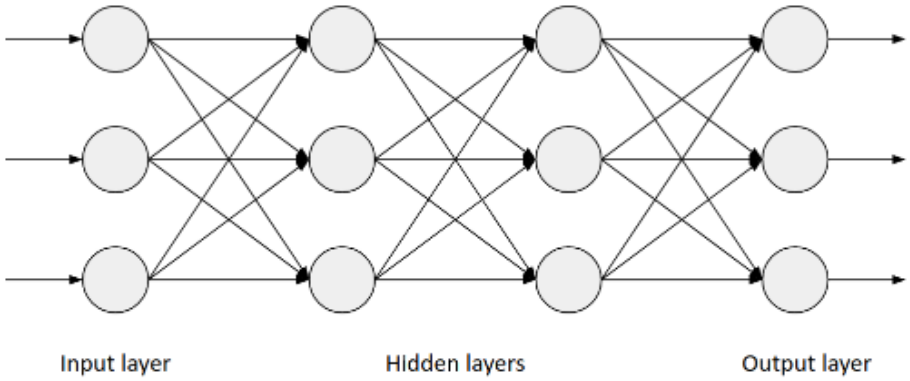

Figure 2: *A feedforward network with two hidden layers. Each layer has a width of three* [15]

the layer that precedes it. For a layer in depth n its function could look like

$$l_n = g_n(W_n^\top l_{n-1} + b_n)$$

The matrix W_n represents the weights of the connections between the neurons of layer $n-1$ and n. b_n is the bias for the specific layer. g_n is the activation function of the layer. In order to make the neural network approximate the desired function, the weights and the biases of the networks are altered by training [8].

The hidden layers of a neural network are the layers that determine its structure or design. The main considerations when determining the architecture of a neural network is its depth and width, where depth refers to the amount of hidden layers and width refers to the amount of neurons in each layer.

2.2.3 Training Neural Nets

The most common algorithm for training neural networks is backpropagation. It is conceptually simple and efficient, and often works in practical cases [27] [26] [30]. The simplest form of backpropagation networks is a feedforward network with both forward and backward connections. When a training input vector is inserted into the network's input nodes, the information is forwarded through the network until the predicted result y_p is presented

from the output nodes. The desired output y_d is compared with y_p using the network's loss function, whose result is forwarded backwards through the network. During the backwards pass of the information the weights of the network are adjusted in order to minimize the loss function, which is the aim of training the network [11].

2.3 Convolutional Networks

Computer vision is a subject whose aim is to enable computers to understand images, in order to automate tasks that the human visual system can do [2][32]. The most common and successful approach of computer vision today is neural network implementations [31].

Many neural networks have been developed for specific tasks. Such networks often have a lot of diversity when it comes to design choices. One type of specialized neural network is the **convolutional network**, often called CNN or ConvNet. Although it is specialized for the task of computer vision and image recognition, it has also proven useful in other applications such as speech recognition and image processing [37][5][23][1]. Goodfellow et al. describes ConvNets in the following way.

> *Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers* [8]

Convolution is an operator that takes two functions as input and outputs a single function that is a blend of the two input functions [40]. In neural networks convolution the two inputs are called input and kernel, and the output is referenced as feature map.

2.3.1 Motivation

Before convolutional networks, pattern recognition models often relied on manually crafted feature extractors. Feature extracting is an important part of models whose task is to recognize data such as images and speech, because of the large amount of input parameters. In order to automatically achieve such feature extraction with traditional feedforward networks, it would require a huge architecture that would be susceptible to overfitting and require vast amounts of training data [22]. The solution to this problem are convolutional networks.

There are three important features that a neural network gains from using covolution: sparse interactions, parameter sharing and invariant representation [8]. In traditional neural networks all neurons in one layer are connected to all of the neurons in the next layer. Sparse interactions mean that the network uses less connections which result in less computational power needed. Parameter sharing means that the same parameter can be used in multiple functions in the network. Finally invariance in a neural network refers to its capability to handle transformations of the data. That ConvNets for object recognition in images are invariant to translations means that the network can manage to recognize the object no matter where in the image it is.

2.3.2 Architecture

As with any kind of neural network the architecture of the network can vary significantly in for example width and depth. However most modern ConvNets have a similar structure to their design. They use three kinds of layers: convolutional, pooling and classifying [38].

A convolutional layer consists of a number of kernels or filters of a given size (K_x,K_y). Each kernel is shifted over the input image performing the convolution operation at each shift and finally outputting a feature map. The kernel's size and stride, i.e. the number of pixels the kernel skips in x- and y-direction between subsequent convolutions, determine the size of the feature maps [6]. For example as seen in figure 3, with an input image of size 7x7, 4 kernels of size 3x3 and a stride of 1 will output 4 feature maps of size 5x5.

The pooling layer takes the output of the convolutional layer and divides it into non-overlapping regions of a certain size. A pooling function is performed on each region, where the output is a pixel that is a summary of the pixels in the region [8]. The pooling layer gives the network faster convergence, improve generalization and gives the model feature invariance.

Convolutional and pooling layers can be stacked on each other in order to achieve complex feature extraction. Once the features are extracted, they are classified in a fully-connected layer [6].

One serious problem with deep neural networks in general is overfitting. A simple technique to prevent overfitting is dropout. Dropout layers are placed in between other layers, dropping random units during training [36]. It is also a common layer to use in convolutional networks.

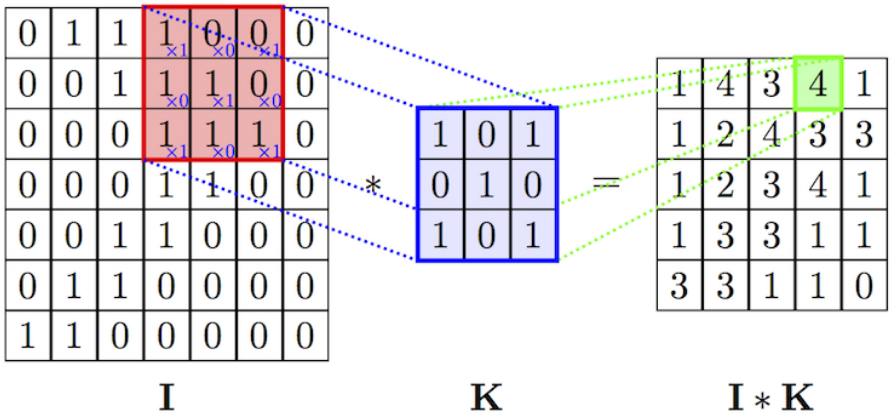

Figure 3: *Convolution operation in a convolutional layer. From left to right: Input image, convolutional filter, feature map*

2.3.3 Drawbacks

Although the convolutional network architecture is the best approach in image recognition at the time being, they perform poorly and use methods that are intellectually unsatisfying [12]. The main problem with convolutional networks is that they are misguided in what they are trying to achieve. By striving to achieve feature invariance, they lose important information of the poses of the features that might be crucial in the recognition process. This approach is incapable of recognitional tasks such as facial identity recognition, since it is not sufficient to identify a face based on the existence of features such as a nose, a mouth and two eyes. The spatial relationships between the features also need to be taken into consideration before establishing whether it is a face or not.

2.4 Capsule Networks

Due to the problems with convolutional networks, Hinton et al. proposed an alternative architecture in 2011 called *transforming auto-encoders* [12]. The main idea of the transforming auto-encoder is to exchange the fundamental building block of the network, from the traditional neuron to a "capsule" of neurons. A capsule is similar to a neuron but with the difference that both input and output of a capsule are vectors rather than scalar values. This design enables capsules to not only learn a specific feature but also recognize

its viewing conditions and deformations.

In 2017 Hinton proposed a network consisting of capsules called *Capsule Network*, or *CapsNet* [33]. The CapsNet presented in the article had state-of-the-art performance on MNIST (data set of handwritten digits) and a higher robustness to affine transformations compared to ConvNets. It also outperformed ConvNets at recognizing overlapping digits.

2.4.1 Capsules

A capsule is a group of neurons whose outputs represent the different properties of the same entity or feature [13]. One capsule encodes one entity, and its output contains both the probability that the capsule's entity is present and a set of "instantiation parameters" that may include the pose, texture and deformations of the entity [33].

When working properly the probability that the entity is present is locally invariant, i.e. it is not affected by the exact position within the limited domain covered by the capsule. The instantiation parameters however are equivariant. This means that as the viewing conditions of the entity change, the instantiation parameters change by a corresponding amount since they represent these viewing conditions [12].

Capsules can be implemented in many different ways. Below are three descriptions of such implementations.

2.4.2 Transforming Auto-encoder

The first architecture that used capsules as its fundamental building block was the transforming auto-encoder in 2011 [12]. In this early design a capsule use one of the values in its output vector to represent the probability that the entity exists, while the other values in the vector represent the instantiation parameters.

The transforming auto-encoder is the part of the network that generates the instantiation parameters to be used by the capsules. It was not a network built to recognize objects in images, but rather to take an input image of an object and a pose for the object, and output an image of the same object in the given pose.

The ideas about transforming auto-encoders were innovative and showed promise. Using capsules instead of neurons enabled an artificial network to easier understand the pose of the objects it tries to identify. One drawback

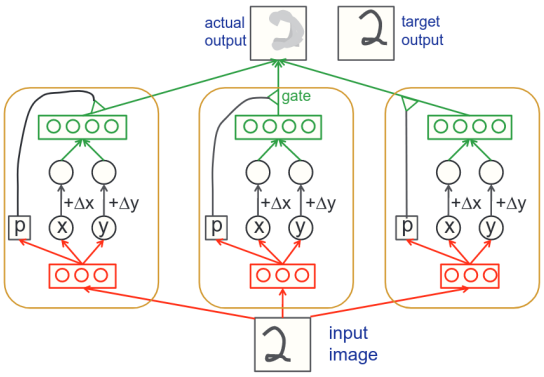

Figure 4: *Graphical representation of a capsule in the transforming auto-encoder from* [12]. *This specific capsule design's only instantiation parameters are its x and y position*

of this architecture however is that the transforming auto-encoder needs the pose of the objects to be supplied externally.

2.4.3 Vector Capsules with Dynamic Routing

In 2017 Hinton proposed a new architecture based on capsules. In this architecture, called CapsNet, the poses of the objects no longer need to be supplied externally. The values of the output vector of a capsule is its instantiation parameters, while the length of the vector represents the probability that its entity is present [33].

One primary idea of CapsNet is to exchange the pooling operation of ConvNets, which routes output from one convolutional layer to the next by making the neurons in the next layer ignore all but the most active features detected in the previous layer. Instead of pooling, a technique called "dynamic routing-by-agreement" is used. The algorithm routes output from capsules in a lower level to the relevant capsule(s) in a higher level. For example, a lower-level capsule that detects a nose is routed to the higher level capsule responsible for detecting faces etc [33].

For all but the last layer of capsules the output $\mathbf{u}_i$ of a capsule i_l in layer l is forwarded to each of the capsules in the next layer $l+1$. A capsule j_{l+1} in layer $l+1$ takes $\mathbf{u}_i$ and multiplies it with a corresponding weight matrix $\mathbf{W}_{ij}$.

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

The resulting vector $\hat{\mathbf{u}}_{j|i}$ is called a "prediction vector" because it can be interpreted as the capsule i_l's prediction of the transformation of the entity represented by capsule j_{l+1}. The prediction vector is then multiplied by a coupling coefficient c_{jk} determined by the routing-by-agreement algorithm. This algorithm calculates an agreement between the two capsules by doing a scalar product of the output of j_{l+1} and the prediction vector. If the agreement is large, the two capsules are relevant to each other and the coupling coefficient is increased. Otherwise the capsules are not relevant to each other and the coupling coefficient is decreased. By iteratively refining the coupling coefficients during training the relevant capsules are linked to each other [33].

This process is made for each output from layer l and the results are summed and used as the input $\mathbf{s}_j$ to a "squashing function".

$$\mathbf{s}_j = \sum_k \hat{\mathbf{u}}_{j|k} c_{jk}$$

The squashing function is the last calculation of the capsule and is similar to the activation function of a neuron. It "squashes" $\mathbf{s}_j$ so that its length is in between 0 and 1, in order for the length to represent probability.

$$\mathbf{v}_j = squash(s_j) = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$$

The output $\mathbf{v}_j$ of the squashing function, which is the output of the capsule as a whole, is then routed to the next layer capsules in the same manner as described above [33].

In the last layer of the network each capsule of the layer will represent one of the output classes of the network. The length of the last layer's capsules' outputs are used in order to make a prediction of the classification of the input image [33].

2.4.4 Matrix Capsules with EM Routing

In 2018 the authors who proposed the CapsNet in [33] presented a refinement of the architecture. The new architecture uses capsules whose inputs and outputs are both a matrix M and an activation probability a, rather than vectors, and the previous dynamic routing-by-agreement algorithm is

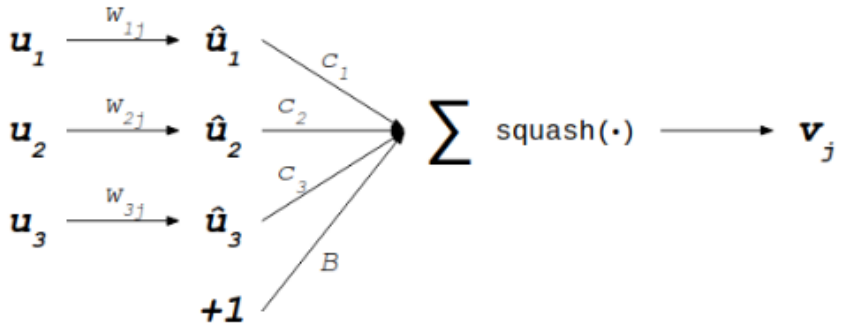

Figure 5: *Inner workings of a capsule*

exchanged with a form of expectation–maximization algorithm called EM routing [13].

The reasoning for exchanging the vectors to matrices is in order to make the transformation matrices in between capsules smaller. By using matrices for an output of size n, the transformation matrices can be made with n elements instead of n^2.

The probability of the entity represented by a capsule being present is no longer the length of its vector but a separate parameter a. This in order to avoid the squashing function which was not considered objective and sensible [13].

In the dynamic routing of [33] the agreement was measured as the cosine of the angle between two pose vectors. Although this is an implementation that works, it is not good at distinguishing between a quite good agreement and a very good one [13]. In order to correct this the EM routing algorithm was introduced, described below.

When the capsules i in a lower layer l has calculated their output matrices M_i and activation probabilities a_i these are used to cast a vote on the pose of each capsule j in the layer $l+1$ above. Each capsule i in layer l has a weight matrix W_{ij} to each capsule j in layer $l+1$ that is iteratively learned during the training of the network. The output matrices M_i are multiplied with the corresponding weight matrix W_{ij} in order to retrieve vote.

$$V_{ij} = M_i W_{ij}$$

Each capsule j in $l+1$ corresponds to a Gaussian, and the pose (converted to a vector) of each active capsule i in l corresponds to a data point. The

EM method fits data points into a mixture of Gaussian models. This is done in two steps. First the M-step update the Gaussian models and the activation probability a_j of the j capsule, using a_i, V_{ij} and an assignment probability that is initialized to $1/|j|$. The E-step recomputes the assignment probabilities of every data point to j. These two steps are run for a few iterations (normally 3) in order for each capsule j to form clusters and thereby finalize their outputs M_j and a_j [13].

2.4.5 Architecture

A CapsNet is a type of neural network that uses capsules instead of neurons as its fundamental building block. This does not mean that no neurons are used, it can still make use of traditional convolutional layers [33]. As described in previous sections a network of capsules can be implemented in many ways. The idea of capsules can also be used together with other ideas, as seen when implemented with non-parametric transformation networks [28].

Normally the features of the input is retrieved through a traditional convolutional layer. The convolutional layer's output is routed to a primary capsule layer which transforms the one dimensional representation of the data to the multidimensional representation in capsules [13][33]. The primary capsules are followed by one or more capsule layers, ending with a layer of capsules where every capsule represents a class to be classified.

2.4.6 Performance

In order to calculate the performance of a neural network model one must first have a data set on which to calculate its performance of. The performance of CapsNets are described below in comparison to baseline ConvNet on two data sets, MNIST and smallNORB, respectively (Figs. 6 & 7).

When first introduced, CapsNet accuracy was tested on the MNIST data set of handwritten digits [33]. MNIST has a training set of 60000 images for training and 10000 for testing. The digits are black-and-white and have been size-normalized and centered in an image of fixed-size (28x28 pixels) [24]. A CapsNet with vector capsules and dynamic routing had an error rate of 0.25 [33], which can be compared to the error rate of a deep ConvNet from 2011 of 0.35 [6].

In the article where matrix capsules were introduced [13] they were tested on the smallNORB data set. smallNORB contains images of 50 toys belong-

Figure 6: *Example images from the MNIST data set* [24]

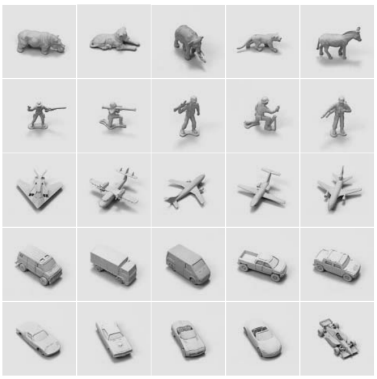

Figure 7: *Example images from the smallNORB data set* [21]

ing to five different categories. The items were photographed under different lighting conditions, elevations and azimuths [21]. The performance of CapsNets with matrix capsules were compared to a CapsNets with vector capsules as well as a baseline ConvNet.

Hinton et al. [13] also compared a matrix CapsNet and a ConvNet on a subset of smallNORB where the viewpoints of the objects were unfamiliar, i.e. not used for training. The two models have the same performance on images where the objects have familiar viewpoints. The results, presented in figure 8, shows that the CapsNet outperforms the ConvNet by far.

Model	Error rate
Matrix CapsNet	**1.8**
Vector CapsNet	3.2
Baseline ConvNet	5.2
Best ConvNet today [6]	2.56

Table 1: *The error rates of different neural network models on the small-NORB data set* [13]

Test set	Azimuth		Elevation	
	CNN	Capsules	CNN	Capsules
Novel viewpoints	20%	13.5%	17.8%	12.3%
Familiar viewpoints	3.7%	3.7%	4.3%	4.3%

Figure 8: *Comparison of the error rates of a matrix CapsNet and a similarly performing ConvNet on untrained viewpoints*[13]

2.5 Summary

The research area of computer vision is rapidly in progress with new and exciting ideas being presented frequently. One such idea is capsule networks which seems to have the potential to become the successor of convolutional networks as the best performing architecture in tasks of computer vision. Although the capsule network architecture is entirely new, it relies on the fundamental rudiments of machine learning and artificial neural networks presented earlier in this section.

Since the idea of capsule networks is so recent there has been little research on its performance on different sets of data. On a more specific level, it is of interest to further test capsule networks' ability to retrieve and detect the pose of objects in images, as it claims to be able to do. One way to do this is to use a data set of images of objects in different viewpoints, and split it up into two data sets: one containing every viewpoint of each object, and one containing only a single viewpoint of each object. A CapsNet should be better than a ConvNet at generalizing between the different viewpoints, and should therefore have a better performance than a ConvNet.

3 Method

This section explains the methodology of how the thesis tries to achieve its goal of determining how CapsNets perform on recognizing clothes in different rotational views. The models are presented as well as the data sets on which they are trained on.

3.1 Data sets

The data sets consist of images of 60000 second-hand clothing items photographed at standardized photography stations with a white background. Lower body clothing are photographed laying down on a table while upper body clothing are photographed dressed up on manikins. The photos are taken by human workers which might cause small variations in distances and angles of the images. The items are divided into 10 different types of clothing such as T-shirt, Dress, Shirt etc. These types represent the labels or classes of the data sets (Figure 9). The classes of the data sets are evenly distributed, i.e. there are 6000 of each type of item.

Figure 9: *Example image for every label in the data sets.*

All items are photographed in three rotational views, one from the front, one from the side and one from the back. The images are resized to 48x48 pixels without any other pre-processing involved. These images are used to generate two data sets, one including all rotational views of each item and

another including only the front view of each item. For simplicity we'll call these two data sets SRO (Sellpy Rotated Objects) and SFF (Sellpy Face Forward).

Figure 10: *Three high resolution example images from data set SRO. SFF would only use the leftmost image.*

3.2 Models

Six different models are used in the research, four convolutional networks and two capsule networks. Two of the convolutional networks, **CNN-1** and **CNN-2**, share a similar architecture as the one used for the smallNORB data set in [6]. These networks are henceforth refered to as the **independent ConvNets**. The other two convolutional networks, **CNN-3** and **CNN-4**, henceforth refered to as the **dependent ConvNets**, are designed to match the performance of the capsule networks on SFF. They are also used to test different types of convolutional networks in order to increase generality of the results. The two capsule networks consist of one vector and one matrix CapsNet, **V-Caps** and **M-Caps**.

3.2.1 CNN-1

CNN-1 is a convolutional network that has a similar structure as the one used for the smallNORB data set in [6]. It has six layers + three dropout layers in between each convolutional and fully connected layer (Table 2). The first layer is a convolutional layer with 300 filters of size 6x6 and a stride of 2. The next layer is a max pooling layer of size 2x2. Following is another convolutional layer with 500 filters of size 4x4 and a stride of 1, which is followed by another max pooling layer, this time with a size of 4x4. The second pooling layer is followed by a fully connected layer that contains 500

neurons. The last layer is a class layer that contains 10 neurons, one for each class. All convolutional and fully connected layers uses ReLU as activation function, except the last one that uses softmax.

Type	**Specifications**
Convolutional	300 filters, 6x6, strides 2
Max pooling	2x2
Dropout	0.2
Convolutional	500 filters, 4x4, strides 1
Max pooling	4x4
Dropout	0.2
Fully connected	500, ReLU
Dropout	0.3
Class	10, softmax

Table 2: *CNN-1 layers*

3.2.2 CNN-2

Because the matrix CapsNet is so computationally heavy to compute, it has a small size with only 100K trainable parameters. This can be compared to the Vector CapsNet which has 6.8M trainable parameters. In order to compare the matrix CapsNet to a ConvNet we will also use a smaller ConvNet, CNN-2, which has the same structure as the normal ConvNet, expect it has less width in its layers in order to match the matrix CapsNet's amount of trainable parameters.

CNN-2 only has 50 filters in its first layer, 70 in its second and its fully-connected layer has 150 neurons. All other parameters are the same.

Type	Specifications
Convolutional	50 filters, 6x6, strides 2
Max pooling	2x2
Dropout	0.2
Convolutional	70 filters, 4x4, strides 1
Max pooling	4x4
Dropout	0.2
Fully connected	150, ReLU
Dropout	0.3
Class	10, softmax

Table 3: *CNN-2 layers*

3.2.3 CNN-3

CNN-3 is a convolutional network that has somewhat similar convolutional layers to V-Caps, but altered in order to perform similar to the capsule networks on SFF. It starts with two convolutional layers with 30 and 50 filters of size 9x9 and a stride of 1. In between them is a max pooling layer of size 2x2, and they are followed by another max pooling layer of size 4x4. This is followed by a fully connected layer with 25 neurons. The last layer is a class layer with 10 neurons and softmax as activation function.

Type	Specifications
Convolutional	30 filters, 9x9, strides 1
Max pooling	2x2
Dropout	0.2
Convolutional	50 filters, 9x9, strides 1
Max pooling	4x4
Dropout	0.2
Fully connected	25, ReLU
Dropout	0.3
Class	10, softmax

Table 4: *CNN-3 layers*

3.2.4 CNN-4

This is another convolutional network that has been designed to match the performance of the capsule networks on SFF. CNN-4 has the same type of starting convolutional layers as M-Caps, the first one being 32 filters of size 7x7 followed by 64 filters of size 5x5, both with stride 2. Next is a max pool layer of size 2x2, followed by yet another convolutional layer with 64 filters of size 3x3 and a stride of 1. This is followed by a max pooling layer similar to the previous one, followed by a fully connected layer with 32 neurons. The last layer is the class layer with 10 neurons and softmax activation function.

Type	**Specifications**
Convolutional	32 filters, 7x7, strides 2
Convolutional	64 filters, 5x5, strides 2
Max pooling	2x2
Dropout	0.2
Convolutional	64 filters, 3x3, strides 1
Max pooling	2x2
Dropout	0.2
Fully connected	32, ReLU
Dropout	0.1
Class	10, softmax

Table 5: *CNN-4 layers*

3.2.5 V-Caps

The vector CapsNet used in this research has a similar structure as the one in [33]. It has 5 layers, starting with a regular convolutional layer with 256 filters of size 9x9 and a stride of 2. The next layer is a primary capsule layer which uses convolution of 256 filters of size 9x9 and a stride of 2 to create 1152 capsules with dimension 8. Next is a class capsule layer with 10 capsules with dimension 16, each one representing one output class. The last layer simple calculates the length of each capsule in the previous layer in order to output the likelihood of each class with the given input. Dynamic routing is used in between the primary capsules and class capsules.

Type	Specifications
Convolutional	256 filters, 9x9, strides 2
Dropout	0.2
Primary capsules	256 filters, 9x9, strides 2, dim 8
Dropout	0.2
Class capsules	10 capsules dim 16
Length	-

Table 6: *V-Caps layers*

3.2.6 M-Caps

The matrix CapsNet used in this research has the following architecture. The first layer is a regular convolutional layer with 32 filters of size 7x7 and a stride of 2. Next is another convolutional layer with 64 filters of size 5x5 and a stride of 2. The next layer is a primary convolutional capsule layer with 16 filters of size 1x1 and a stride of 1. All capsules in the network has a dimension of 3x3. After the primary capsules is a convolutional capsule layer with 16 filters of size 3x3 and a stride of 2. Next is another convolutional capsule layer, with 10 filters of size 3x3 and a stride of 1. The last layer is a class capsule layer with one capsule per class. The network uses EM-routing in between the capsule layers.

Type	Specifications
Convolutional	32 filters, 7x7, strides 2
Convolutional	64 filters, 5x5, strides 2
Primary capsules	16 filters, 1x1, strides 1, dim 3x3
Dropout	0.2
Convolutional capsules	16 filters, 3x3 strides 2, dim 3x3
Dropout	0.2
Convolutional capsules	10 filters, 3x3 strides 1, dim 3x3
Class capsules	10 capsules, dim 3x3

Table 7: *M-Caps layers*

3.3 Implementation

The software implementations uses Keras, which is a high-level neural networks API written in Python [16], using Tensorflow as backend. TensorFlow is an open source software library originally written by Google's Machine Intelligence Research organization, for numerical computation which is useful when implementing neural networks [39]. All training and validation of the models were conducted on a Amazon AWS cloud computing service running Ubuntu 16.04 and a Tesla K80 GPU.

3.4 Training

All networks have been trained using early stopping which is triggered when the loss of the validation set has not been lowered in 10 straight epochs. Due to limitations in time, a maximum number of epochs has been set to 50. The networks are trained using the ADAM optimizer [17] with a an initial learning rate of 0.001, which is multiplied by 0.96 after each epoch.

3.5 Analysis

All models are evaluated using 5-fold cross-validation where one subset is used for testing, one is used for validation and the remaining three are used for training. The division of subsets is randomly generated. Each model M_i therefore yields five separate results per data set, being its error rates. The average and standard deviation of these five results are presented as the result for each model and data set. In order to further establish the difference in results for M_i and every other model, a Student's t-test [7] is pairwise performed on M_i and all $M_j, j \neq i$. The t-test calculates the confidence that the null hypothesis, that there is no relationship between two measured phenomena, is rejected. It is a test that is widely used in this kind of research [29].

Once the results for SFF and SRO are collected, the t-test is used again to establish the confidence that a model performs similarly on the two data sets. Once this null hypothesis is rejected it is possible to determine the magnitude of the difference in results d_M between the data sets as the percentage difference between them. The d_M of each model is then used when comparing the models' ability to understand objects under rotation. A high d_M suggests that there is a big difference in performance between the two

data sets, and vice versa.

4 Results

In this section the performance of each neural network model is presented on both the SFF and SRO data set. The results presented for each model and data set are the average of five runs.

4.1 Front View (SFF)

Table 8 shows that the independent ConvNets (CNN-1 & CNN-2) have the lowest error rates of all models of 24.7% and 25.4%, respectively. The networks with the highest error rates are M-Caps and the dependent ConvNets (CNN-3 & CNN-4), with the same error rate of 26.8%. V-Caps has a slightly lower error rate of 26.5%.

Model	**Epochs**	**Time/epoch**	**Test error rate %**
CNN-1	30.4	44s	**24.9** ± 0.6
CNN-2	39.0	16s	25.4 ± 0.4
CNN-3	50.0	15s	26.8 ± 0.6
CNN-4	45.6	17s	26.8 ± 0.6
V-Caps	37.0	2m 22s	26.5 ± 0.4
M-Caps	40.2	5m 0s	26.8 ± 0.7

Table 8: *SFO error rate, average amount of epochs before stopping and time per epoch for each model*

Table 9 presents the confidence, in percent, that two models perform similarly on SFF. The dependent ConvNets and M-Caps are shown to perform similarly, having high percentage points. V-Caps is less similar to the ConvNets, which is especially distinct on V-Caps and CNN-4. Their error rates only differ 0.3 percentage points, but their confidence in rejecting the null hypothesis is 98.8%.

	CNN-1	CNN-2	CNN-3	CNN-4	V-Caps
CNN-2	3.2				
CNN-3	0.6	1.2			
CNN-4	0.2	0.0	93.5		
V-Caps	0.3	0.1	29.7	1.2	
M-Caps	0.1	0.6	98.9	96.2	43.3

Table 9: *Percentage of confidence that two model's perform similarly on SFF, according to a two-tail Student's t-test.*

4.2 Rotational views (SRO)

All models have a higher error rate on SRO than SFF. The independent ConvNets are still the ones with the lowest error rate, 27.0% and 27.9% respectively (Table 10). M-Caps have a lower error rate on SRO compared to V-Caps, 28.2% to 28.4%. The networks with the highest error rates are the dependent ConvNets, with error rates of 29.3% and 29.2%.

Model	Epochs	Time/epoch	Test error rate %
CNN-1	41.2	2m 10s	**27.0** ± 0.5
CNN-2	48.8	42s	27.9 ± 0.3
CNN-3	50	48s	29.3 ± 0.3
CNN-4	50	53s	29.2 ± 0.4
V-Caps	40.0	7m 44s	28.4 ± 0.4
M-Caps	45.2	15m 13s	28.2 ± 0.5

Table 10: *SRO error rate, average amount of epochs before stopping and time per epoch for each model*

Table 11 is the SRO equivalence of table 9. Most noteworthy is that the performance of M-Caps no longer is related to the dependet ConvNets.

	CNN-1	CNN-2	CNN-3	CNN-4	V-Caps
CNN-2	0.2				
CNN-3	0.0	0.0			
CNN-4	0.1	0.1	30.8		
V-Caps	0.8	5.0	1.7	1.6	
M-Caps	0.1	6.7	0.1	0.6	59.9

Table 11: *Percentage of confidence that two model's perform similarly on SRO, according to a two-tail Student's t-test.*

4.3 Time

In tables 8 and 10 we can see that the ConvNets take a lot less time to train. One example being CNN-2 and M-Caps that have similar amounts of parameters but a huge difference in time per epoch, taking over 20 times longer on SRO and 18 times longer for SFF. In spite of the slow training of M-Caps, it still has a higher error rate on both SFF and SRO.

4.4 Rotational ability

We see in table 12 that the independent ConvNets have the lowest error rates on both SFF and SRO. They however also have big d_M, or impairments, with error rates that are 8.4% and 9.8% higher on SRO compared to SFF. The CapsNets however have lower impairments, where V-Caps has an impairment of 7.2% and M-Caps has almost half of CNN-2's impairment, 5.2%.

The dependent ConvNets and M-Caps all share the same error rate of 26.8% on SFF, which makes them easy to compare. The ConvNets have 1.0 and 1.1 percentage point higher error rate on SRO compared to M-Caps, with impairments of 9.3% and 8.7%.

Model	d_M %	$\emptyset$ %
CNN-1	8.4 ± 2.1	99.9
CNN-2	9.8 ± 1.0	99.9
CNN-3	9.3 ± 2.7	99.8
CNN-4	8.7 ± 1.1	99.9
V-Caps	7.2 ± 1.9	99.9
M-Caps	**5.2** ± 2.3	98

Table 12: *Result table of each model on SFF and SRO. Impairment refers to the percentage impairment in error rate between SFF and SRO. $\emptyset$ refers to the percentage of confidence that the null hypothesis - that there is no difference in the performance between the two data sets - is rejected.*

The results presented so far only takes the average impairment into account, not the standard deviation. For example, the impairments of CNN-1 and M-Caps falls into each others' ranges, indicating that they might be related. In order to get a better understanding of how the impairment of the ConvNets and the CapsNets differ to each other, the impairment of every ConvNet is pairwise t-tested with the impairment of every CapsNet (Table 13). This test shows that the impairment of M-Caps is in fact not related to CNN-1, with a confidence of 96.6%. CNN-3 has the lowest confidence of rejecting the null hypothesis with M-Caps, with a confidence of 94.5%. V-Caps is more related to the ConvNets, since the null hypothesis cannot be rejected on a 5% confidence on any of them.

	V-Caps	**M-Caps**
CNN-1	57.3	3.4
CNN-2	5.0	1.1
CNN-3	34.5	5.5
CNN-4	12.7	4.3

Table 13: *Percentage of confidence that the impairment of two models are related, according to a two-tail Student's t-test.*

5 Discussion

This section aims to discuss and draw conclusions to the research question of how the performance of CapsNets compare to the performance of ConvNets on images of clothes in different rotational views. Limitations of the research and a reflection of the method is presented, as well as ideas for future research.

5.1 Error rate

All models have a similar error rate on both data sets, around 26% on SFF and 28% on SRO. This is a considerably higher error rate than the error rates of similar neural networks on other data sets of 3D objects [13]. A way to analyze a data set is to calculate a confusion matrix, which shows how a model predicts each label. Figure 11 and 12 show the confusion matrices of CNN-1 and M-Caps on SFF. These confusion matrices are very similar, indicating that the high error rates are more likely a consequence of the data sets rather than the design of the models.

The confusion matrices shows that the types of clothes that the models confuse with one another is intuitive. The confusion between jeans (2) and pants (1) is understandable and can be seen in the figures 9.1 and 9.2. The same goes for the upper body clothes. T-shirts (6) is easily confused with short sleeve tops (8), which is expected since they are both upper body clothes with short sleeves. Shirts (0), cardigans (5), long sleeve tops (7) and sweaters (9) are also easily confused with one another for the same reason. The confusions in the data sets are therefore the main cause of the high error rates, rather than the design or implementation of the models.

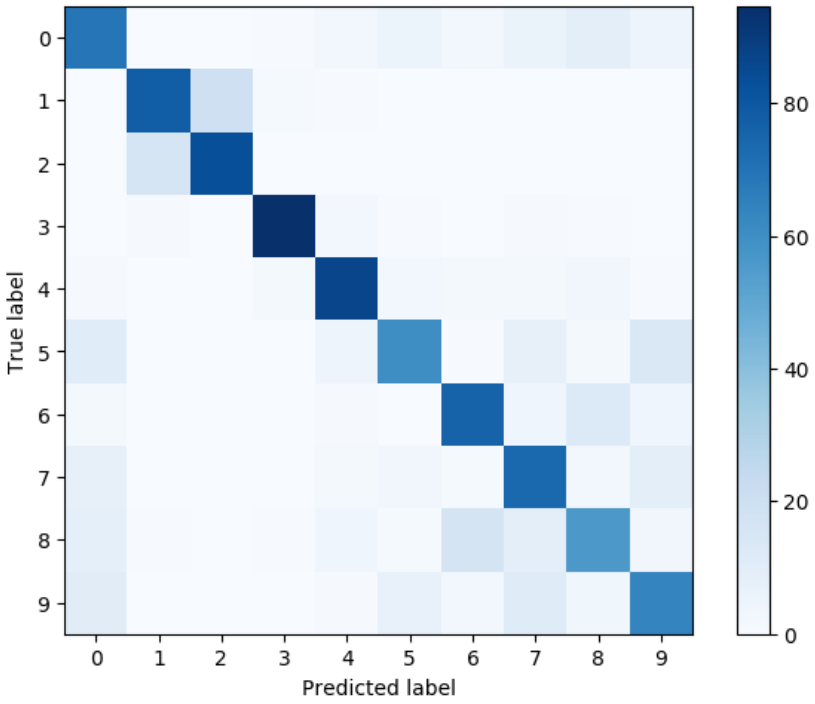

Figure 11: *Normalized confusion matrix of CNN-1 on SFF. Each number represents a label in the data set, as seen in figure 9.*

5.2 Rotational Comprehension

The error rate of every model is lower on SFF than on SRO, which is clear from the performance results as well as the fact that every model has a $d_M > 0$ (table 12). This is quite expected, since even though SRO contains three times more images than SFF, the additional images in SRO increase the variation of how each class is represented. This increase in variation is in this case bad in terms of classification, since the characteristics of a certain type of clothing is often more visible from the front than from the back. An example of this can be seen in figure 13.

The results however suggest that capsule networks, especially matrix CapsNets, are better than convolutional networks at understanding clothes in different rotational views, since they have a smaller average impairment in error rates between SFF and SRO. Table 13 shows that there is a statistical

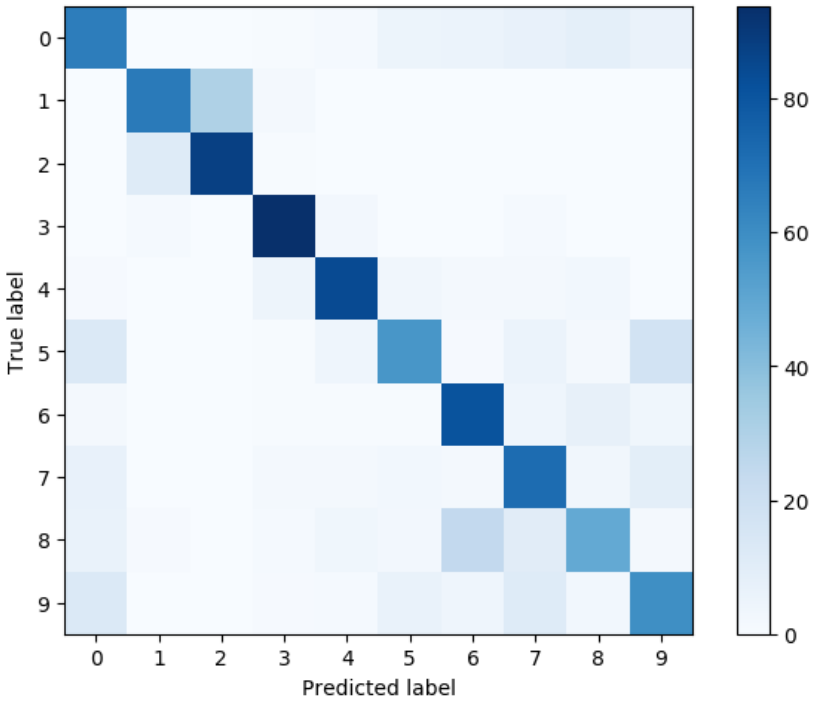

Figure 12: *Normalized confusion matrix of M-Caps on SFF. Each number represents a label in the data set, as seen in figure 9.*

difference in impairment between M-Caps and the convolutional networks. V-Caps however has a much smaller statistical confidence of having a different impairment compared to the ConvNets.

It is clear that there is a difference in performance between the two different types capsule networks. The impairment of the vector capsule network is lower than the convolutional networks but higher than the matrix capsule network. There is also little statistical confidence that CNN-1 and V-Caps have different impairments, with only a 42.7% confidence of rejecting the null hypothesis. This is in line with the expectations of the two architectures, since the matrix CapsNet architecture is created to be an improvement of the vector CapsNet [13].

That CapsNets have a better understanding of clothes in different rotational views is expected, since this is what they are typically supposed to be good at. M-Caps is also the only model that has a lower confidence of rejecting the null hypothesis than 99.8%. Although a 98% confidence should

Figure 13: *A cardigan and a sweater from SRO seen from the front and back*

not be considered low, it is still an indication that it is closer to performing similarly on SFF and SRO than the other models.

5.3 Applicability

Previous research on capsule networks have shown good performance in general, having accuracy that is on par with the best convolutional networks on several sets of data. This has not been the case in this thesis. CNN-1 has the best accuracy on both SFF and SRO, outperforming M-Caps with 2.1 and 1.2 percentage points. This while also being about 7 times faster to train.

Having a lower impairment is not very applicable, which makes convolutional networks more useful than capsule networks in this case. Since all models have a higher error rate on SRO than SFF, having a better comprehension of rotational views is not enough for the capsule networks to be useful. It is however easy to imagine other data sets where this better comprehension of rotational views leads to a better accuracy and therefore usefulness. Such data sets would probably include objects that are easy to classify no matter the rotational view, which is not always the case with certain types of clothes.

5.4 Limitations

The main limitation of the methodology is that it takes a lot of time to train neural network models on image data. This limitation has resulted in using smaller images, models and data sets. Downsizing the images to 48x48 means that a lot of information from the original images have been lost, which might result in worse model performance. The same is in some extent true for model size, bigger models have potential to perform better than smaller ones. The data sets of 60000 clothing items could have been

made bigger which is also a possible source of increasing performance and certainty in results.

Another result of this limitation is the use of k-fold cross-validation with $k = 5$ rather than the more commonly used $k = 10$. A smaller k leads to less trials and therefor less time to train. The downsides are however that the training set is only 3/5 of the full data size and there are less performance estimates [29].

5.5 Method reflection

The process of evaluating a CapsNet's ability to recognize complex 3D objects in different rotational views can be done in different ways. Perhaps the most straight forward way, the way used in this thesis, is to compare its recognitional performance to that of a more traditional ConvNet. There are however problems included when using this kind of methodology.

Neural networks are complex structures which are difficult to model without a certain level of trial and error. Small changes to the structure can result in big differences to performance [10]. When comparing two different types of architectures such as CapsNet and ConvNet, there is always a risk that the results of each model is more specific to the certain model rather than the type of architecture as a whole.

The convolutional networks used in this thesis have different structures, with different amount of convolutional layers and different sizes on their filters. Since these different convolutional structures have all gotten similar impairment results, we can with a certain amount of feasibility say that the results are general to the ConvNet architecture.

5.6 Future research

This work and previous have shown that capsule networks perform differently, and often better, than convolutional networks. In order to further understand the pros and cons of capsule networks, more research has to be made on different data sets and tasks. One important area of progress when it comes to capsule networks is that of time optimization. The results in this research has shown that training capsule networks can take over 20 times the time of training a similarly performing convolutional network. Being able to lower this training time of capsule networks would not only make it more available and useful, it would also simplify further research.

6 Conclusion

Capsule network is an interesting architecture that has shown great performance in several scenarios. In this thesis it has shown to be good at understanding clothes viewed in different rotational views. It has however also shown worse overall accuracy than convolutional networks, making it less useful in this certain case. The expectations that capsule networks are good at understanding the different viewing conditions of an object have however been met, making it a compelling area for further research.

7 Bibliography

References

[1] Ossama Abdel-Hamid et al. "Convolutional Neural Networks for Speech Recognition". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.10 (Oct. 2014), pp. 1533–1545. ISSN: 2329-9290. DOI: 10.1109/TASLP.2014.2339736. URL: http://ieeexplore.ieee.org/document/6857341/ (visited on 05/02/2018).

[2] Dana Harry Ballard and Christopher M. Brown. *Computer Vision*. 1st. Prentice Hall Professional Technical Reference, 1982. ISBN: 0131653164.

[3] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995. ISBN: 0198538642.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[5] Qifeng Chen, Jia Xu, and Vladlen Koltun. "Fast Image Processing with Fully-Convolutional Networks". In: *CoRR* abs/1709.00643 (2017). arXiv: 1709.00643. URL: http://arxiv.org/abs/1709.00643 (visited on 05/02/2018).

[6] Dan Ciresan et al. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: (2011). URL: https://www.aaai.org/ocs/index.php/IJCAI/IJCAI11/paper/view/3098 (visited on 05/02/2018).

[7] Thomas G. Dietterich. "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms". In: *Neural Comput.* 10.7 (Oct. 1998), pp. 1895–1923. ISSN: 0899-7667. DOI: 10.1162/089976698300017197. URL: http://dx.doi.org/10.1162/089976698300017197 (visited on 05/02/2018).

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (visited on 05/02/2018).

[9] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN: 0132733501.

[10] K. He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[11] Robert Hecht-Nielsen. *Neural Networks for Perception*. Harcourt Brace & Co., 1992. Chap. Theory of the Backpropagation Neural Network, pp. 65–93. ISBN: 0-12-741252-2.

[12] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. “Transforming Auto-Encoders”. In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by Timo Honkela et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 44–51. ISBN: 978-3-642-21735-7.

[13] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. “Matrix capsules with EM routing”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=HJWLfGWRb (visited on 05/02/2018).

[14] William L. Hosch. *Encyclopædia Britannica - Machine Learning*. URL: https://www.britannica.com/technology/machine-learning (visited on 01/22/2018).

[15] A. K. Jain, Jianchang Mao, and K. M. Mohiuddin. “Artificial neural networks: a tutorial”. In: *Computer* 29.3 (Mar. 1996), pp. 31–44. ISSN: 0018-9162. DOI: 10.1109/2.485891.

[16] *Keras*. http://keras.io/. 2018.

[17] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 05/02/2018).

[18] L. B. Klebanov, S. T. Rachev, and Frank J. Fabozzi. *Robust and non-robust models in statistics*. Nova Science Publishers, 2009, p. 317. ISBN: 9781607417682.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 05/02/2018).

[20] S. Lawrence and C.L. Giles. "Overfitting and neural networks: conjugate gradient and backpropagation". In: (2000), 114–119 vol.1. DOI: 10.1109/IJCNN.2000.857823.

[21] Y. LeCun, Fu Jie Huang, and L. Bottou. "Learning methods for generic object recognition with invariance to pose and lighting". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 2. Vol.2. June 2004, pp. 97–104. DOI: 10.1109/CVPR.2004.1315150.

[22] Yann LeCun and Yoshua Bengio. *The Handbook of Brain Theory and Neural Networks*. Ed. by Michael A. Arbib. Cambridge, MA, USA: MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. ISBN: 0-262-51102-9.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* (May 2015). URL: http://dx.doi.org/10.1038/nature14539 (visited on 05/02/2018).

[24] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[25] R. Lippmann. "An introduction to computing with neural nets". In: *IEEE ASSP Magazine* 4.2 (Apr. 1987), pp. 4–22. ISSN: 0740-7467. DOI: 10.1109/MASSP.1987.1165576.

[26] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018. URL: http://neuralnetworksanddeeplearning.com/ (visited on 05/02/2018).

[27] Genevieve. Orr et al. *Neural networks : tricks of the trade*. Springer, 1998, p. 432. ISBN: 3540653112.

[28] Dipan K. Pal and Marios Savvides. "Non-Parametric Transformation Networks". In: *CoRR* abs/1801.04520 (2018). arXiv: 1801.04520. URL: http://arxiv.org/abs/1801.04520 (visited on 05/02/2018).

[29] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Cross-Validation". In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 532–538. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565. URL: https://doi.org/10.1007/978-0-387-39940-9_565 (visited on 05/02/2018).

[30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Neurocomputing: Foundations of Research". In: ed. by James A. Anderson and Edward Rosenfeld. MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6.

[31] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575 (visited on 05/02/2018).

[32] T S Huang. "Computer Vision: Evolution and Promise". In: (Mar. 2018).

[33] S. Sabour, N. Frosst, and G. E Hinton. "Dynamic Routing Between Capsules". In: *ArXiv e-prints* (Oct. 2017). arXiv: 1710.09829 [cs.CV].

[34] Steven L Salzberg. "On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach". In: *Data Mining and Knowledge Discovery* 1.3 (Sept. 1997), pp. 317–328. ISSN: 1573-756X. DOI: 10.1023/A:1009752403260. URL: https://doi.org/10.1023/A:1009752403260 (visited on 05/02/2018).

[35] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556 (visited on 05/02/2018).

[36] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[37] Somsak Sukittanon et al. "Convolutional Networks for Speech Detection". In: International Speech Communication Association, May 2004. URL: https://www.microsoft.com/en-us/research/publication/convolutional-networks-for-speech-detection/ (visited on 05/02/2018).

[38] Michael T. McCann, Kyong Hwan Jin, and Michael Unser. "A Review of Convolutional Neural Networks for Inverse Problems in Imaging". In: *arXiv* (Oct. 2017). URL: https://arxiv.org/abs/1710.04011 (visited on 05/02/2018).

[39] *Tensorflow*. https://github.com/tensorflow/tensorflow. 2018.

[40] Eric W Weisstein. *Convolution*. URL: http://mathworld.wolfram.com/Convolution.html (visited on 05/02/2018).