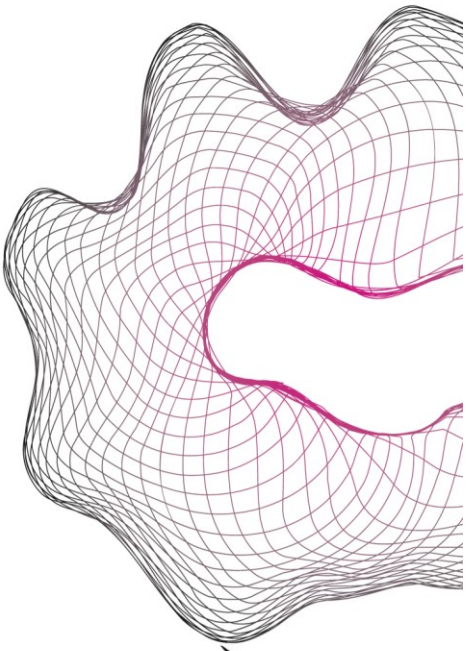
MASTER THESIS



EXTRACTING UNSEEN CLASSES FROM CAPSULE NETWORK FEATURE SPACE

Floor Couwenberg

FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF BIOMECHANICAL ENGINEERING

EXAMINATION COMMITTEE
DR. E.H.F. VAN ASSELDONK
DR.IR. M. VLUTTERS
DR.IR. D.C. MOCANU

DOCUMENT NUMBER
BE - 849

UNIVERSITY OF TWENTE.

11-04-2022

ACKNOWLEDGEMENTS

I have many people to thank for my graduation. Firstly, many thanks my daily supervisor Mark Vlutters for the support during this thesis, for having us team up at the new lab and for our philosophical talks that I enjoyed a lot. Thank you to Edwin van Asseldonk for providing your time as committee chair and the necessary feedback on my process and report. Similarly, I would like to thank Decebal Mocanu for joining the committee and providing valuable insight during the beginning of the thesis. And academics-wise, lastly I want to thank Ghada Sokar for helping me start my research into incremental learning.

Naturally, thank you to my partner Jaap who was instrumental in helping me through some rough times and who continues to love me still. To my family for always supporting me, special thanks to my dad for having many long, fun and informative talks with me about this subject and my brother for being so kind as to proofread everything.

Lastly, I want to thank some friends. Michelle, for sparring with me and helping me with the visualisation. Laura, for coming over and studying (and relaxing) together with me, you're definitely responsible for some of my most productive days. Rosalyn, for our frequent music quiz breaks during our online work sessions. Anouk, also for sparring with me and helping me to not feel crazy sometimes. Also thanks to Steyn, my study friends, and my housemates from La Vache Aviante. Finally, thanks to everyone who I have not named but has been there for me.

TABLE OF CONTENTS

1. GENERAL INTRODUCTION

Healthcare professionals are in short supply, a shortage that will continue to increase in the coming years [1]. A part of the solution for this problem might be found in autonomous robotics, by having a robot shouldering some of the care providers' diverse workload. Unfortunately, the current robotic systems are not yet autonomous enough to function in such an environment with a diverse set of tasks. Working in an environment with varying tasks demands knowing about and being able to identify a wide range of objects in the robot's surroundings, what they look like and e.g. how heavy they are.

While robots are not yet able to know about every object or task, deep learning (DL) techniques have at the very least provided a stepping stone to autonomy using either vision or another sensory input to obtain information. Deep learning refers to a type of learning by a computational model or *network* that incorporates multiple layers of 'neurons' to extract features of the input that it gets presented with. This is mostly done by using *convolutional layers*, where the input image is convoluted on pixel-level with certain kernels, revealing for example all edges in the y-direction. The connections between all the layers are weighted, and those weights get updated after every run through the network, called backpropagation. This method is a way of machine learning that most approaches the way humans learn and has provided promising results in fields such as computer vision, natural language processing and object recognition. [2][3]

A subfield of DL is *incremental learning*, which is what robots will need in order to function autonomously in any environment. Incremental learning can enable a robot to learn new information on the job, which eliminates the problem of having to teach the robot every possible thing beforehand. However, incremental learning approaches have so far been unsatisfactory for a real-world robotic application and continue to suffer from a problem called *catastrophic forgetting*. For every new task the network learns, it forgets part of the old tasks and performance therefore decreases over time. This happens because the weights between the layers are adapted for every new piece of information that is taught to the network, an adaptation that is often not beneficial for the previous information. [4-6]

Another issue autonomous robots or any applied neural network suffers from is generalization in computer vision. Currently, deep learning approaches have not been able to generalize as well as humans do, i.e. learn what one tree looks and be able to recognize all other trees as being a 'tree' as well. This lack of generalization is amongst other things caused by neural networks being designed to throw away position and orientation information. Since the network does not know either of those things, it cannot generalize to new points of view for objects in an image. [7]

The solution to both incremental learning and generalization issues that is proposed in this work is an approach using capsule networks. This type of network is designed specifically to keep position and orientation information and generalize to different points of view. The network consists of several convolutional layers, followed by two capsule layers. These capsule layers are not so different from the layers of other networks, except that the neurons are grouped together within several capsules. This grouping allows the network to keep position and orientation information. [7]

Additionally, the network proposed in this work uses a different algorithm between layers to form strong parts-to-whole connections, taking into account the orientation of the parts (features) with respect to the whole (object). This self-attention algorithm ensures that for an object in an image to be recognized as that object, multiple of the *capsules* in the layer below must agree on it being that object. [8]

This work furthermore proposes to use a fixed basic network, trained on a wide range of features (i.e. parts) before the weights are fixed. To that end, I assume that by knowing a set of varied features, any object can be learned from an image by simply combining the features in a different way than another object. By fixing the basic network, catastrophic forgetting is ruled out since the weights cannot be adapted anymore. The combining of features into new objects is be done by a clustering algorithm and combinations can be stored for future reference. This work aims to provide a proof of concept that any new object might be learned without changing the network, by learning new combinations of features.

The remainder of this thesis is structured as follows. In chapter two, the paper containing the methods, results and a discussion of the conducted research is presented. Chapter three provides a general discussion of this work. Lastly, additional figures are provided in the appendices.

2. RESEARCH PAPER

EXTRACTING UNSEEN CLASSES FROM CAPSULE NETWORK FEATURE SPACE

Floor Couwenberg

ABSTRACT

A shortage of healthcare professionals already is the norm, and the shortage will continue to increase even more in the next years. Autonomous robotics might be able to provide some relief for these professionals and our healthcare system. However, robots have not yet been able to achieve enough autonomy to function in such a setting with many unknowns. It has not been possible to teach robots everything they might need to know beforehand, prompting the rise of incremental learning. Incremental learning would allow the robot to learn on the job and therefore adapt to new situations, but most state-of-the-art methods still suffer from catastrophic forgetting. In this work, I argue that incremental learning can best be achieved by learning a broad basis of elements and combining these into new classes.

In this research, two capsule networks are built and investigated: Efficient CapsNet and Small CapsNet. In order to test the generalizing capabilities and the ability of this type of network to learn new information, MNIST data is augmented in several ways. The digits in the images are scaled and translated, a second digit is added to the image and one digit class is left out of training completely, to be later introduced during inference. Results are evaluated using test accuracies and losses, T-distributed Stochastic Neighbor Embedding and K-Means clustering.

The results indicate that Small CapsNet is able to learn a scaling and translation factor, but cannot yet generalize to data augmentations it has not seen before. In addition, both Efficient and Small CapsNet show that it is possible to learn a new class, based only on the information already known from other classes. The results indicate that a broad basis of classes is necessary, but that as long as the unfamiliar class contains similar elements it should be possible to combine these separate elements into a new class.

Keywords – Deep Learning, Neural Networks, Incremental Learning, Catastrophic Forgetting, Capsule Networks

1. INTRODUCTION

By 2030 the Netherlands will be dealing with a shortage of 102.600 care providers, divided over hospitals, nursing homes, home care, youth care, and social work [1]. While some solutions include attracting more young professionals to the discipline, another solution might be found in autonomous robotics.

Autonomous robots could be of added value in the field of healthcare by shouldering some of the care providers' diverse workload. Circa 50 percent of care providers in the Netherlands indicate that they regularly perform physically straining tasks requiring a substantial amount of strength [2]. Care robots could take over some of this heavy labor, relieving the care providers.

Various robots have already been developed for specific nursing tasks, such as robotic nursing beds and patient-lifting robots [3]. However, not many advances have been made in robotic systems that are capable of performing multiple nursing tasks (e.g. patient-lifting and carrying equipment), let alone systems capable of learning new tasks on the job.

For such a robot to be able to perform multiple tasks and even learn new ones, it needs to be able to deal with different unknowns. It needs to know about a wide range of objects, e.g. what they look like, what shape and weight they have. Next to that, it must be able to identify the objects from different viewpoints. Since it is not (yet) possible to teach a robot about every object in every situation beforehand, the robot will need to learn incrementally about objects and semantics in the field.

Vision is an important part of this problem since humans obtain most of their information about objects from what it looks like. As such, it makes sense that a robotic system must be capable of identifying objects based on vision. Within the field of machine learning, various solutions have been proposed to this image classification problem. With the rise of *deep learning* (DL) advancements in classification accuracy have been made [4, 5]. These improvements are mostly due to DL methods being able to learn features when provided with enough data, allowing for a bigger range of features than in conventional machine learning algorithms [5].

Although DL has provided excellent results on specific classification tasks, there are several downsides to conventional DL networks. When neural networks learn incrementally, most of them suffer from *catastrophic forgetting*. This means that they tend to 'forget' some of the information of the previously learned task. As a result, they can no longer accurately discern the difference between an image of class A and B when it has learned the second task consisting of classes C and D. While solving the problem of catastrophic forgetting has been investigated the last few years and network performance on incremental learning has increased, no widely adopted solution has been found yet [6-8].

Most current incremental learning solutions try to avoid catastrophic forgetting for example by storing part of the training data to reuse later [9, 10] or by generating sample images from old classes using techniques such as generative networks [11, 12]. Both approaches show a decrease in classification accuracy after more classes are incrementally added and both require large amounts of data in order to train. In addition, these approaches require the entire model to be retrained for every new class, something that is undesirable for a robot in the field.

Other incremental learning approaches [13, 14] therefore include memories only consisting of learned features and try to incorporate few-shot learning to limit the amount of training data needed. These cases also show deterioration of the accuracy between increments. Furthermore, it is unknown how well these methods generalize to new objects in completely different surroundings or from different points of view.

Most of these current incremental learning approaches use some form of Convolutional Neural Networks (CNNs) or another similar type of network that uses an operation called *pooling* [15]. Networks with a pooling operation look at features, or clusters of active pixels, and their orientations with respect to the image frame, which makes them inherently bad at generalization outside of the seen training data [16-18]. This can be attributed to the pooling layer, where clusters of neurons in one layer are combined into one neuron in the layer above [19]. While this pooling reduces the network size, it also makes the network invariant to the locations of the features. As a result, these networks merely encode whether a certain feature exists (anywhere) in the image and lose the information about the relation between different features [16]. Whereas in some cases this invariance is preferable, it gives rise to incidents

such as recognizing a Picasso as a human face, even though the proportions and locations of features such as the eyes and nose are all wrong. This makes these networks inherently bad at seeing Picasso's paintings, or any object that contains similar features to a previously learned object, as a separate class.

A network type capable of keeping orientation and position information is a capsule network [16]. This type of network has no pooling layers but instead combines the learned features into capsules of multiple dimensions. The output of these capsules is therefore a vector instead of a scalar, keeping the orientation and position information. The operation between capsule layers ensures that the network learns the relative orientations of features with respect to each other, instead of the image frame. By learning the relation between feature orientations, the network becomes viewpoint invariant.

In [16], it is argued that capsules aid in the generalizing capability of a neural network by encoding properties of each class in the dimensions of the capsules. By perturbing the capsule dimensions individually, they showed dimensions encoding for scale, thickness, translation, and more with the MNIST dataset [20]. These properties were learned by the network even when it was only trained with 2-pixel translations.

While the Capsule Network is promising, it is computationally expensive due to the specific routing operation between the capsule layers [16, 21]. The implementation of capsule networks used in [22] builds on this work and reduces the number of parameters of the network by performing a depthwise convolution and a self-attention routing instead of a dynamic routing between capsule layers. This network architecture makes it less computationally expensive.

Using a capsule network, features and their orientation can be learned and combined into higher-level concepts. Humans are good at generalizing to new objects because most of them exist of features that we have already seen. Simply learning the new configuration of features allows us to discriminate the object as a new class. I argue that the same can be done for machine learning. By learning a wide enough range of basic features using deep learning, new image classes can be added by learning a new configuration of the basic features.

The advantage of this approach is that the neural network can be frozen after initial training of basic features and combinations of these features may be saved in a database as the object classes. Similar to [13], the network is only used to extract features from a new image, after which the feature vector is used to create a new class or match the image to an existing class in the database. Using this method eliminates the problem of catastrophic forgetting and allows for incremental learning by adding new classes to the database.

This work aims to provide a proof of concept for incremental learning, using capsule networks to obtain information about MNIST [20] digit images. To this end, the generalizing capabilities of the network as implemented in [22] are investigated using several different augmented datasets. Furthermore, a smaller network is proposed and evaluated since such a relatively simple database is used. A smaller network might encode different information in the capsule dimensions than a complex network.

This paper is structured as follows. First, the network structures and experimental settings are described in section 2. The experimental results are presented in section 3, followed by a discussion of the findings in section 4. Finally, conclusions are drawn based on this work and future recommendations are made in section 5.

2. Materials & Methods

2.1 Standard network architecture

The basic network in this study, Efficient CapsNet, is comprised of four convolutional layers, a primary capsule layer and a digit capsule layer as specified in table 1 and illustrated in figure 1. The four convolutional layers are responsible for finding the features in the images and the depthwise convolutional layer assigns the high-dimensional features into capsules. The primary capsule layer consists of 16 capsules, the same amount used in [22]. The digit capsule layer contains one capsule for each class and is formed by self-attention routing as per [22].

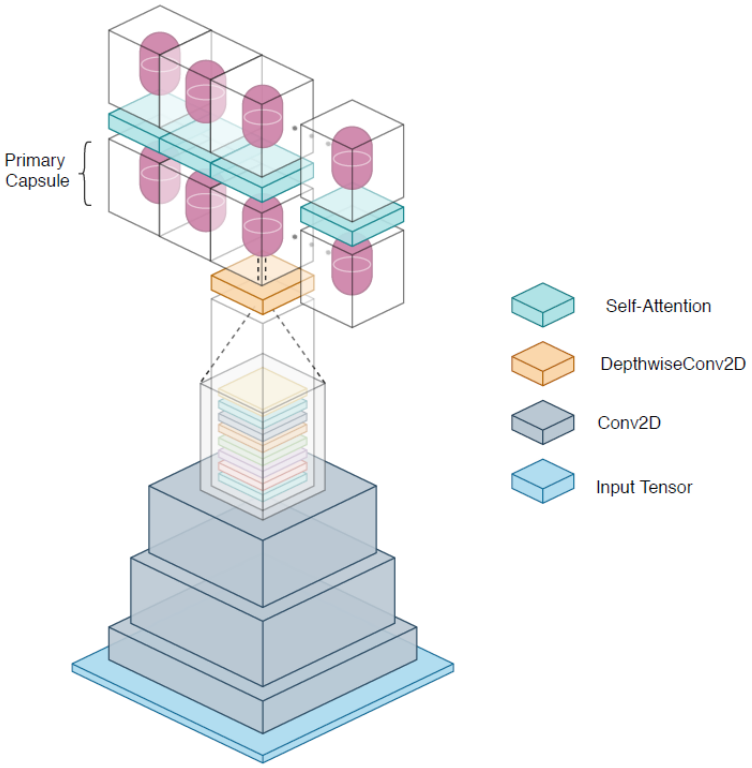


Figure 1: Schematic representation of Efficient CapsNet, from [22]. The convolutional layers detect local features and map them to a higher dimensional space. The depthwise convolution allows the formation of the primary capsules. The digit capsule layer is formed by a self-attention routing mechanism that correlates the primary capsule vector output to the digit capsules.

This self-attention routing mechanism follows the idea that the lower-level capsules of the primary capsule layer combine to form a whole in the digit capsule layer. It is very similar to a fully-connected layer because all ten digit capsules are a combination of all 16 primary capsules. However, which primary capsules have the most influence on a digit capsule is determined by this routing mechanism. Each primary capsule forms a prediction for each digit capsule, which are then compared to all other primary capsule predictions. If two or more primary capsules have similar predictions for a digit capsule, they are taken into account more for that digit capsule than other primary capsules. Each training batch, these coupling coefficients are combined with probabilities that a specific primary capsule belongs to a digit capsule. This allows primary capsules and digit capsules to form meaningful connections.

After the self-attention is done, classification takes place by taking the magnitude of the digit capsule output vectors $\|v_k\|$, where k is a digit capsule, followed by a Softmax operation.

During training, the digit capsule output is also fed through a simple decoder made up of ReLU activated fully connected layers to make image reconstructions, in order to compare them to the original images for the loss function.

Table 1: The network structure for both Efficient CapsNet and Small CapsNet.

Network	Convolutional layers				Fully connected layers		Primary capsule layer					Digit capsule layer		
	in_channels	out_channels	kernel size	stride	in_features	out_features	capsules	in_channels	out_channels	kernel size	dimensions	capsules	in_channels	dimensions
Efficient CapsNet	1 32 64 64	32 64 64 128	5 3 3 3	1 1 1 2			16	128	8	9	8	10	8	16
Decoder Efficient CapsNet					160 512 1024	512 1024 784								
Small CapsNet	1 32	32 32	5 3	1 2			8	32	4	11	4	10	4	8
Decoder Small CapsNet					80 512 1024	512 1024 784								

For some experiments, the network and decoder structure was adapted slightly to fit the larger image dimensions. These adaptations mainly were a larger stride or kernel in several layers. The specifics of this adaptation can be found in the appendix (A.1).

2.2 NETWORK AUGMENTATION

To investigate the influence of the size of the network on generalizability, amount of convolutional layers and capsule dimensions was reduced. This new network, Small CapsNet, is comprised of only two convolutional layers, a primary capsule layer and a digit capsule layer. Both capsule layers contain capsules with half the amount of dimensions as Efficient CapsNet has and the primary capsule layer only contains eight capsules instead of 16. The properties of each layer are summarized in table 1.

Similar to Efficient CapsNet, several changes in the network structure had to be made for some experiments to adapt to larger image dimensions. The network structure for these experiments can be found in the appendix (A.1).

2.3 DATASETS

Multiple datasets were developed to perform the desired experiments, an overview of these datasets is available in table 2.

Standard dataset

The MNIST dataset used in this research contains 60,000 training and 10,000 test images [20]. Unless otherwise mentioned, no alterations have been made to this dataset in training and testing. Images in the dataset are 28 by 28 pixels in size.

Scaling datasets

In order to investigate generalizability of both Efficient CapsNet and Small CapsNet, experiments on downscaled images were performed. For the scaling experiments a training set was developed based on the standard dataset with 2400/60000 images downscaled to 20 by 20 pixels. These downscaled images were then zero-padded back to 28 by 28 pixels. Following that, testing was done on the standard test set and on a scaled test set with only downscaled digits.

Translation datasets

In addition to scaling, the ability of Efficient CapsNet and Small CapsNet to deal with the translation of the digits was investigated. MNIST images were zero-padded to make the images 56 by 56 pixels, which allows the digits to be translated to four corners of the image. Each of the network types was trained on the left translation dataset (see table 2) and on a translation dataset with 2400/60000 (*1-in-25*) or 30000/60000 (*half-half*) images translated to the top right image corner.

The left translation dataset in this experiment is the dataset with the 56 by 56 pixel images, with the digit in the top left corner. The 1-in-25 and half-half dataset images have the digit in the top right corner (28 pixels to the right) for the amount specified (table 2). The translation was chosen to be 28 pixels because of the next experiment with two digits in the same image. Inference was done on either the left translation dataset or the right translation dataset.

The training time differs for the two networks in this experiment, with 50 epochs of training for Efficient CapsNet and 20 epochs of training for Small CapsNet. Efficient CapsNet was given more training epochs because preliminary results showed that with only 20 epochs the network was unable to correctly classify more than 50% of the digit images. In order to compare the two networks on generalizability, it was chosen to train Efficient CapsNet longer until a saturation of the classification accuracy was reached.

For this experiment, the adapted network structures were used to fit the larger image dimensions.

Two-digit dataset

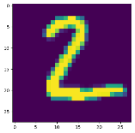
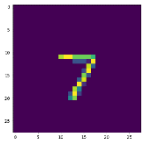
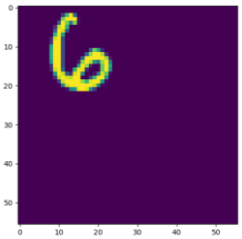
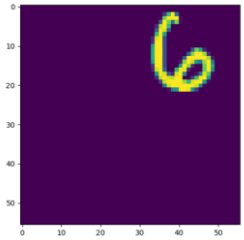
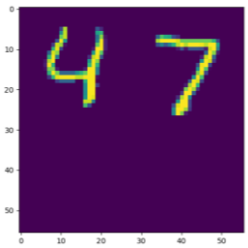
The third experiment introduces a second digit to the networks, in order to investigate whether the networks would be able to use their class knowledge to create a new class for two-digit numbers. For this experiment, the networks were trained on the same enlarged images as in the translation experiment. During inference, the second digit was introduced in the top right corner of the image.

For this experiment, the adapted network structure of Efficient CapsNet was used to fit the larger image dimensions. This experiment was not performed with Small CapsNet.

New-digit datasets

Lastly, the networks' ability to generalize the features they have learned to new objects was investigated. This was done by leaving either one or multiple digit(s) out of the training dataset and then introducing that digit during inference in the test dataset. These new-digit training datasets are named after the digit that was left out (e.g. minus-9).

Table 2: Overview of all dataset modifications.

Dataset name	Properties	Example of modified image
Standard dataset	MNIST images	
Scaled training set	2400/60000 downscaled images	
Scaled test set	10000/10000 downscaled images	
Left translation dataset	All images enlarged with digits in the top left corner	
Right translation dataset	All images enlarged with digits in the top right corner	
Half-half translation test set	5000/10000 images with the digit in the top right corner, the other images have the digit in the top left corner	
Four-quadrant translation test set	2500/10000 images with the digit in the top left corner, 2500 images with the digit top right, 2500 images with the digit bottom left, 2500 images with the digit bottom right	
1-in-25 translation training set	2400/60000 images with the digit in the top right corner, the other images have the digit in the top left corner	
Half-half translation training set	30000/60000 images with the digit in the top right corner, the other images have the digit in the top left corner	
Two-digit test set	All test images have two digits, one top left and one top right	
New-digit training sets: minus-X	All images with digit X are replaced with a randomly chosen other digit image	

2.4 TRAINING PROCEDURE

Each training session consists of a number of epochs, in this study networks were mostly trained for 20 epochs unless mentioned otherwise. During each epoch all 60,000 MNIST training images are passed through the network in batches of 100. After every batch, the total loss is calculated and backpropagated through the network to update the weights. This total loss consists of two parts, a margin loss and a reconstruction loss. The purpose of the margin loss is to force the capsules to encode information for one digit class only. This loss is calculated as per [16]:

$$L_m = \sum_{i=1}^{k} l_k\,, \qquad l_k = T_k \max(0, m^+ - \|v_k\|)^2 + \lambda(1 - T_k)\max(0, \|v_k\| - m^-)^2 \tag{1}$$

where k stands for each digit capsule and $\|v_k\|$ represents the magnitude of the output vector of each capsule of the digit capsule layer. As mentioned before, a large magnitude of a capsule corresponds to the certainty of that capsule that it is correct. The loss is calculated for each digit capsule and then summed, with $T_k = 1$ if capsule k corresponds to the present digit class. For each capsule that the digit class does not correspond to, the second term starting from the lambda ensures a higher loss. With $m^+ = 0.9$ and $m^- = 0.1$, the loss is tuned to the certainty of the correct and incorrect capsules. A correct capsule with large magnitude gets a loss close to zero, while an incorrect capsule with a large magnitude gets a large loss. The λ factor is set at 0.5 to ensure that the capsules are not faced with a loss too large in the beginning of training.

In addition to the margin loss, reconstructed images are made by a decoder in order to compute a reconstruction loss. The output of the digit capsule layer is masked such that only the output vector of the correct digit capsule is left. The vectors are concatenated, forming an array of 160 elements with 16 non-zero values (80 and 8 in case of Small CapsNet). This array is then fed through the decoder through three fully-connected layers, see the architecture in figure 2.

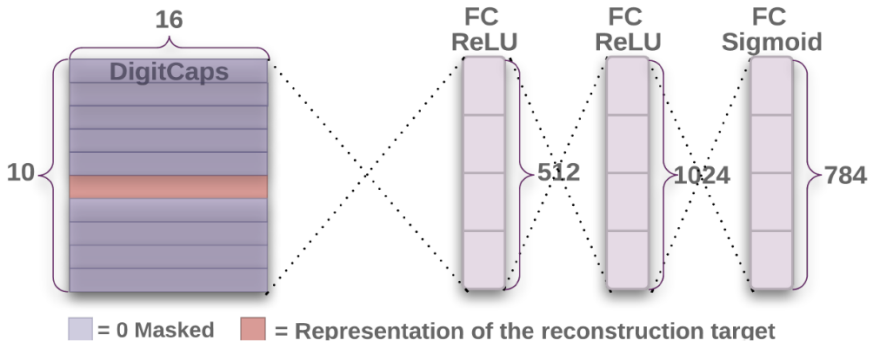


Figure 2: The decoder architecture, schematic from [16].

The output of the sigmoid layer is an image of the same size as the input images were, which can thus be compared to the original input image. The reconstruction loss is calculated using a mean squared error loss function:

$$L_r = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

where y is the original image and $\hat{y}$ the reconstructed image. The images are compared pixel-wise, with n the number of pixels in the image. The reconstruction loss as formulated in

equation 2 is multiplied with a regularization factor of 0.0005 to ensure that the margin loss is prioritized. The total loss is a sum of the margin loss and the regularized reconstruction loss.

$$L = L_m + 0.0005L_r \tag{3}$$

Training configuration

The networks were implemented in the Pytorch framework and the code was adapted from [22] to merge the self-attention routing of [21] with the capsule network basis of [16]. The optimizer used was the Adam optimizer. All training was done on a private PC with an Intel® UHD Graphics 630 GPU and an NVIDIA Quadro P1000 GPU. Unless mentioned otherwise, networks were trained for 20 epochs.

2.5 EXPERIMENT EVALUATION

The experiments were evaluated using several methods. Prediction matrices were made of each epoch of the training phase in order to visualize the network's performance during training. These matrices were constructed using the network's predictions and the true image labels. In addition, test accuracy and the test loss were used as a performance measure. Test accuracy is defined here as the percentage of correctly classified digits during inference. The test loss consists of the sum of the margin and reconstruction loss.

Furthermore, the digit capsule dimensions were perturbed individually before the image was reconstructed, providing insight into the features that the capsules dimensions encode. The individual dimensions were perturbed with steps of 0.05 in the range of $[-0.10, 0.10]$.

Lastly, the feature space was visualized by first reducing the dimensionality using T-distributed Stochastic Neighbor Embedding (t-SNE) [23] and then plotting the datapoints in a 2D graph. It is important to note that each run of the t-SNE algorithm provides a different mapping of the feature space and therefore can plot the same batch of images differently each time.

K-Means clustering

For the new-digit experiment specifically, the K-Means algorithm from the Scikit-learn python package [24] was used to perform clustering. K-Means is an algorithm that iteratively updates the centroids (center of data clusters) by minimizing the sum of the squared distances of data points to the centroids.

The network was trained on a minus-X dataset, while inference was done with the standard dataset but with a batch size of 1000 images. The clustering algorithm was given the information that there should be ten clusters for all ten digits. Clustering was done directly on the output of the digit capsule layer, in the high-dimensional feature space. To this end, the digit capsule output was concatenated to one 160-dimensional vector. Due to the larger batch size, clustering was performed on a 1000 test set images at the same time.

The evaluation of this method was done using t-SNE figures. After performing K-Means and having the algorithm assign each image a label, the dataset was visualized using t-SNE. To compare, another t-SNE figure was made with the true labels of each image. If K-Means has labelled the same images together that also share their true labels, that means that the clustering was successful.

3. Results

3.1 Baseline performance

Training on the MNIST database with Efficient CapsNet results in a test accuracy of 98.7%. This accuracy describes the percentage of images correctly predicted by the network. Figure 3 shows separate clusters for each digit, indicating that the network is able to learn distinctive features for each digit class. Several digits have been misclassified, mainly digits that are similar such as the four and the nine.

Additionally, figure 4 shows several digit reconstructions picked at random together with their true labels. Most of the reconstructions form a clear digit, but some digits have not been formed correctly. In this case, one of the sevens is of bad quality, possibly because of misclassification. Notably, within each digit class the reconstructed digits are similar in shape.

Efficient CapsNet on the standard test set

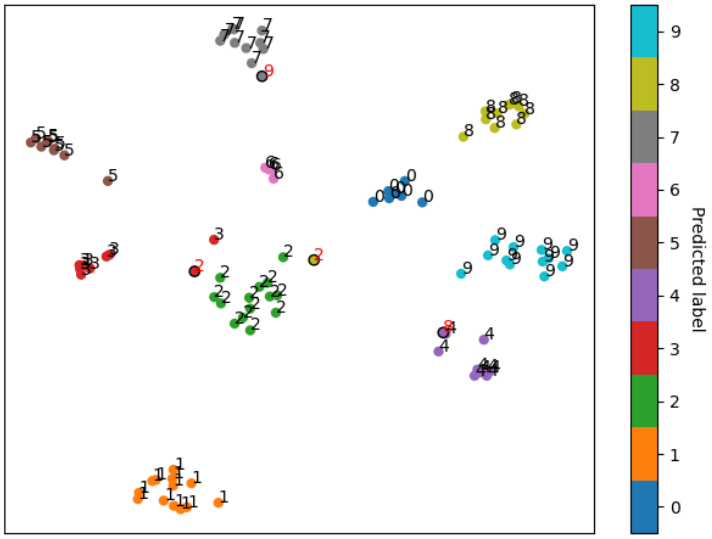


Figure 3: t-SNE mappings of the digit capsule feature space for Efficient CapsNet using the standard test set. The network was trained on the standard training set. The colours indicate the predicted label, while the numbers plotted are the true label. Datapoints with a black border and red annotation represent misclassified images.

Efficient CapsNet reconstructions

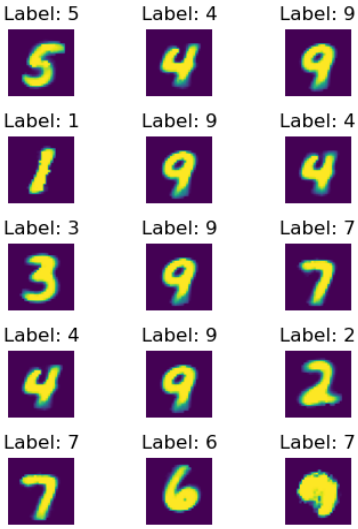


Figure 4: Reconstructed images of the test set and their true labels using Efficient CapsNet trained on the standard training set.

3.2 Performance on augmented data

Scaling

The performance of Efficient CapsNet on the scaled dataset is reported in table 3. When the network is trained with the standard dataset and inference is done with the scaled test set, the accuracy drops and the loss increases compared to the performance for the standard test set. However, while the overall accuracy is lower for the scaled training set, the drop in accuracy to the scaled test set becomes smaller. Interestingly, the accuracy is higher for the scaled test set when the network is not trained on any scaled images, compared to when the scaled training set is used.

Small CapsNet on the other hand shows a higher accuracy with the scaled training set both for the standard and scaled test sets. This would indicate that Small CapsNet benefits from

having several scaled images in the training set, whereas Efficient CapsNet performs poorer for it.

Table 3: Scaling experiment accuracies in percentages and losses for Efficient CapsNet and Small CapsNet, after training and inference on either the standard or scaled training sets.

Training / Test set	Efficient CapsNet		Small CapsNet	
	Accuracy (%)	Loss	Accuracy (%)	Loss
Standard / Standard	98.70	0.0466	94.18	0.5195
Standard / Scaled	93.02	0.3552	81.33	0.8337
Scaled / Standard	87.58	0.8258	95.07	0.5076
Scaled / Scaled	85.35	0.8725	87.62	0.7717

That Efficient CapsNet has more difficulty training with the scaled training set than the standard training set is supported by the prediction matrices in figure 5. These matrices show that the network trained on the scaled dataset has not reached the same classification accuracy as the network trained on the standard dataset. The network trained on the scaled training set is uncertain still about digits 1 and 8, confusing them with each other. This confusion is the cause of the lower accuracies reported in table 3.

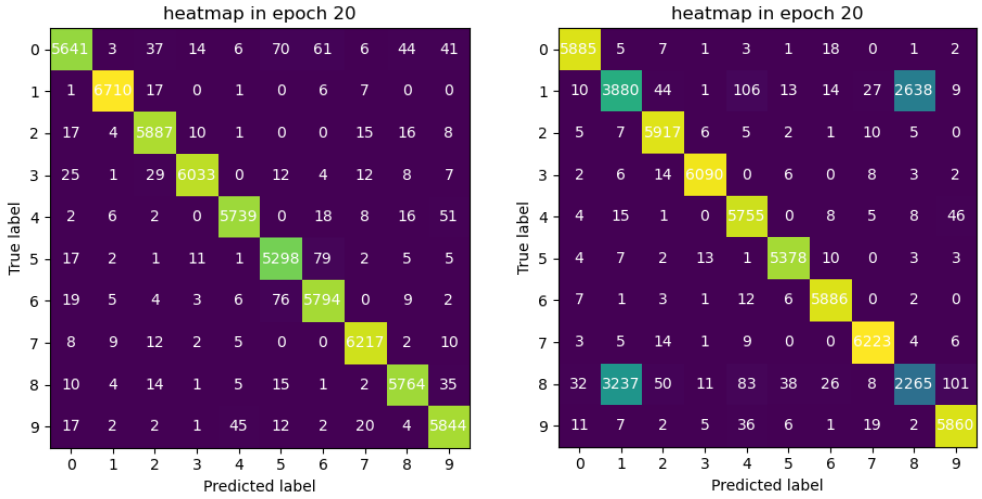


Figure 5: Prediction matrices showing the true and predicted labels in the last epoch of training. Left: Efficient CapsNet trained on the standard training set. Most of the true labels coincide with the predicted labels. Right: Efficient CapsNet trained on the scaled training set.

In addition to Small CapsNet showing an increase in accuracy when trained on the scaled training set, it also shows a scaling factor in the reconstructed images (figure 6). Efficient CapsNet does not show any scaling when the capsule dimensions are perturbed. These reconstructions indicate that Small CapsNet learns some scaling parameter when trained with the scaled training set, which might be why the accuracy is better than for Efficient CapsNet. However, it should be noted that not every digit reconstructed using Small CapsNet shows this scaling factor (see A.2).

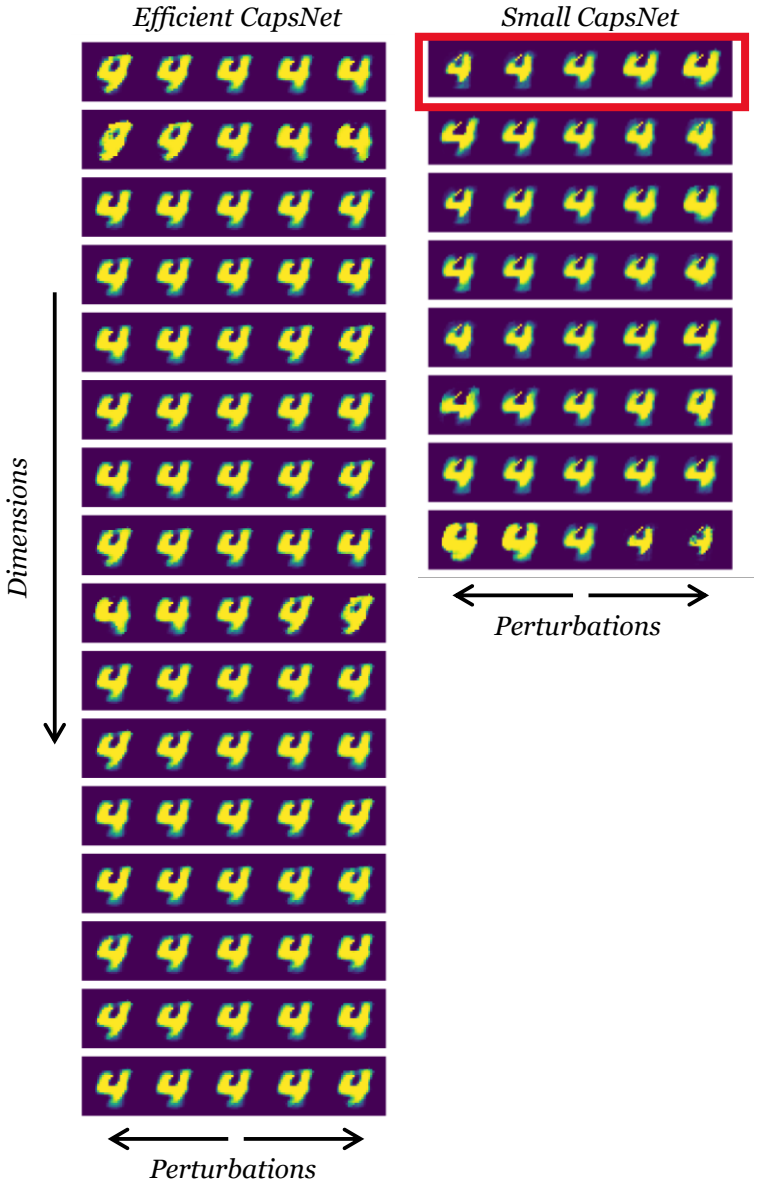


Figure 6: Reconstructions of a digit image during inference, using networks trained on the scaled training set. Each capsule dimension is perturbed individually (-0.10, -0.05, 0, +0.05, +0.10). Left: Efficient CapsNet. Right: Small CapsNet. The red box indicates a dimension where the scaling factor is visible.

Translation

Table 4 reports the accuracies and losses for the translation experiments, with the training sets being the 1-in-25 and the half-half training sets. Similar to the scaling experiment, Efficient CapsNet's accuracies drop and losses increase when more augmented images are used in the training dataset (half-half). This phenomenon further supports the notion that Efficient CapsNet does not benefit from seeing more augmented images in the training set.

The accuracies reported in table 4 for Small CapsNet are noticeably higher than for Efficient CapsNet, and the losses are lower. Small CapsNet especially shows a better performance than Efficient CapsNet on the half-half training set.

Table 4: Accuracies and losses for the translation experiments with Efficient CapsNet and Small CapsNet trained on either the 1-in-25 or the half-half translation training sets, resp.

Training / Test set	Efficient CapsNet		Small CapsNet	
	Accuracy (%)	Loss	Accuracy (%)	Loss
1-in-25/ Left	96.80	0.9442	98.22	0.0371
1-in-25/ Right	92.82	0.8712	97.07	0.0850
Half-half / Left	89.91	0.8481	98.08	0.0322
Half-half / Right	81.89	1.2525	98.23	0.0299

The t-SNE mappings for Efficient CapsNet in figure 7 clearly show two separate groups for each digit position. Nevertheless, the predicted labels show that the network classifies the top left digits in the same class as their corresponding top right digits. While there are more misclassified digits that with the standard test set, the distance in the feature space does not prevent the network from still recognizing the digits as the same.

However, the separation of the digit clusters within the groups is not as clear as for the standard test set, indicating that the network sees less differentiating features within the groups (positions). Additionally, when trained on the half-half training set, the network is not able to separate the digit clusters within the groups as clearly as when it is trained on the 1-in-25 training set. This is in accordance with the reported accuracies in table 4 that are lower for the half-half training set.

Efficient CapsNet trained on the 1-in-25 set *Efficient CapsNet trained on the half-half set*

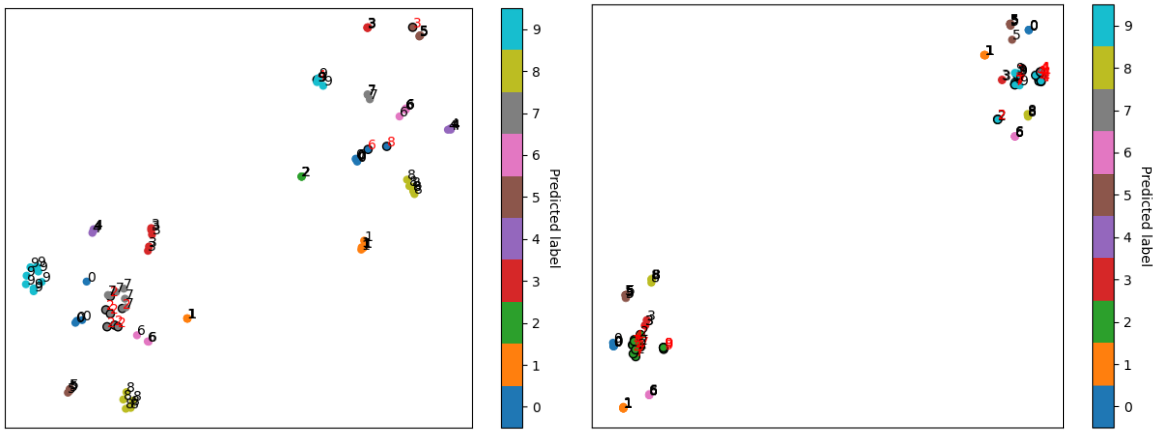


Figure 7: t-SNE mappings of the digit capsule feature space for Efficient CapsNet using the half-half translation test set. Colours represent the predicted label, while the annotation represents the true label. Datapoints with a black border and red annotation represent misclassified images. Left: Trained on the 1-in-25 training set. Right: Trained on the half-half training set.

However, while Efficient CapsNet exhibits two separate groups after being trained on translated images, Small CapsNet does not. Figure 8 illustrates that Small CapsNet groups the top-positioned digits together when trained on the 1-in-25 training set. Notably, there are two different clusters for most digits, but they are not in separate groups as with Efficient CapsNet. It is interesting that some digit classes do not get divided over two clusters, such as the nines and the ones in this example.

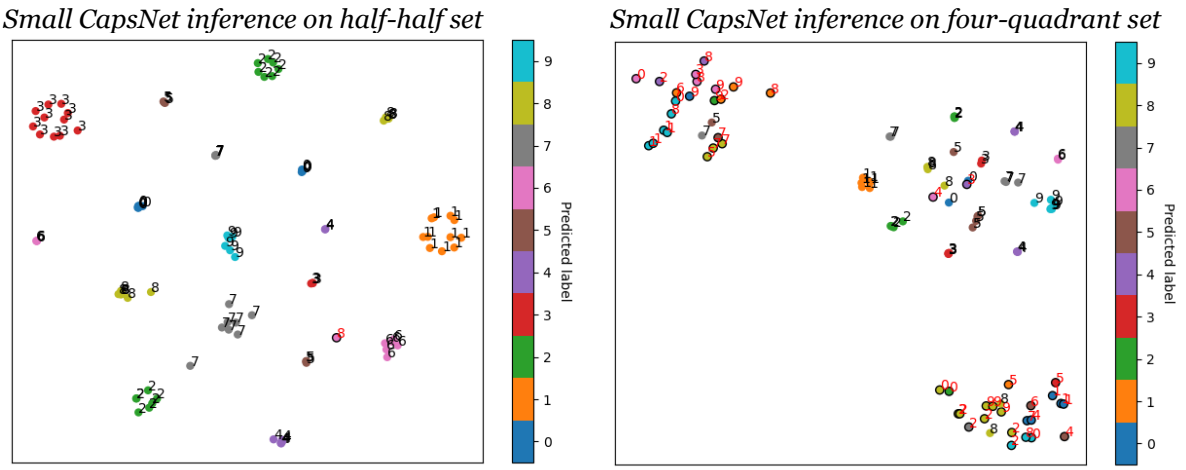


Figure 8: t-SNE mappings of the digit capsule feature space for Small CapsNet after training on the 1-in-25 translation training set. Datapoints with a black border and red annotation represent misclassified images. Left: Inference with the half-half translation test set. Right: Inference with the four-quadrant test set.

When inference is done with the four-quadrant test set with Small CapsNet, a separate groups do form. Presumably, one group is for the top left and top right digits, one for the bottom right digits, and one for the bottom left digits. The formation of additional groups with the four-quadrant test set indicates that Small CapsNet does not generalize to translations other than what it has seen during training. Additionally, most of the digits in either of the untrained positions are misclassified.

Lastly, the capsule dimensions after perturbation are shown in figure 9. The reconstructions made with Efficient CapsNet do not show any noticeable differences compared to reconstructions in other experiments. The digit is always depicted in the same corner and some effect of the perturbation can be seen regarding thickness and shape of the digit. The position of the digit in the reconstruction does change with the test image. It should also be noted that some of the reconstructions for the translated digits were empty, or of very poor quality.

On the contrary, the reconstructions made with Small CapsNet indicate that in multiple dimensions a translation factor is encoded. In the example shown in figure 9, five out of eight dimensions show a clear translation corresponding to a perturbation. The middle image shows the digit divided over the two positions, and depending on the perturbation the digit shifts to either the top left or top right position.

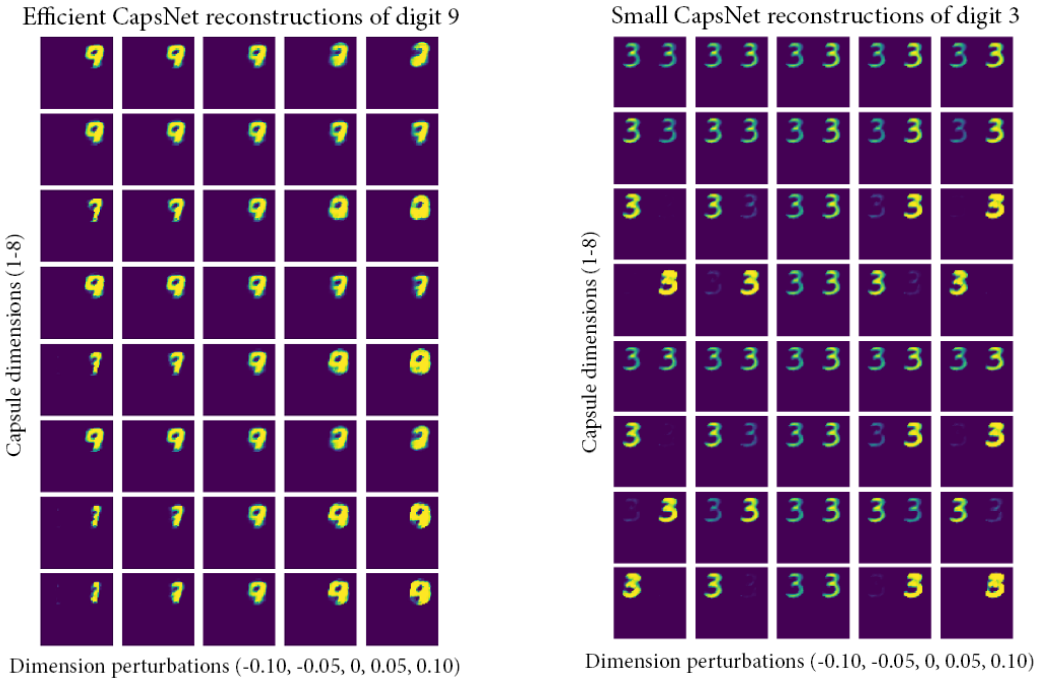


Figure 9: Reconstructions of perturbed dimensions of digit images after training both networks on the half-half translation training set and running inference with the half-half translation test set. Each row of images represents a digit capsule dimension. Each column corresponds to a perturbation value $(-0.10, -0.05, 0, 0.05, 0.10)$. *Left: Efficient CapsNet with digit nine. Right: Small CapsNet with digit three.*

Two-digits

Similar to the translated images, the two-digit images form their own separate group in the feature space, see figure 10. As with the translated images, this indicates that Efficient CapsNet understands that these two-digit images are different. However, within the two-digit group there is no definite separation such as in the standard dataset. Some of the images are clustered based on either the first or last digit, suggesting that the network does find some similar features.

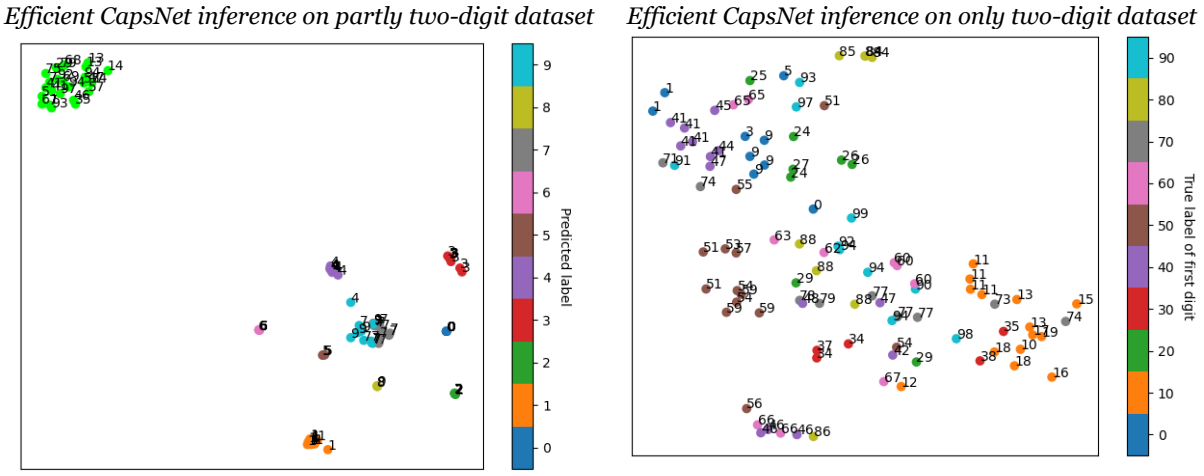


Figure 10: t-SNE mapping of the digit capsule feature space using Efficient CapsNet trained on the half-half translation dataset. Left: Inference is done with 25 of 100 being two-digit images. Two digit images are coloured bright green to contrast the one-digit images. Right: Inference is done with only two-digit images. The colour represents the true label of the first digit.

New-digit

Figure 11 shows the results of the new-digit experiment, with Efficient CapsNet either trained on a minus-9 or minus-5 training set. For both datasets, a separate cluster for the left-out digit can be seen with the ground truth labels in column A. Not all nine and five digit images are mapped into that cluster however, as there are outliers mapped near to other digit clusters.

Column B in figure 11 shows that the K-Means algorithm has given most of the images within the new-digit cluster the same label and therefore managed to find the new-digit cluster correctly, although the outliers are not accounted for. Examples where K-Means has not been able to find the new-digit cluster correctly can be found in A.3. Furthermore, in this case of the minus-5 dataset, the three and the eight are inseparable for the clustering method and become one large cluster attached to the new-digit cluster.

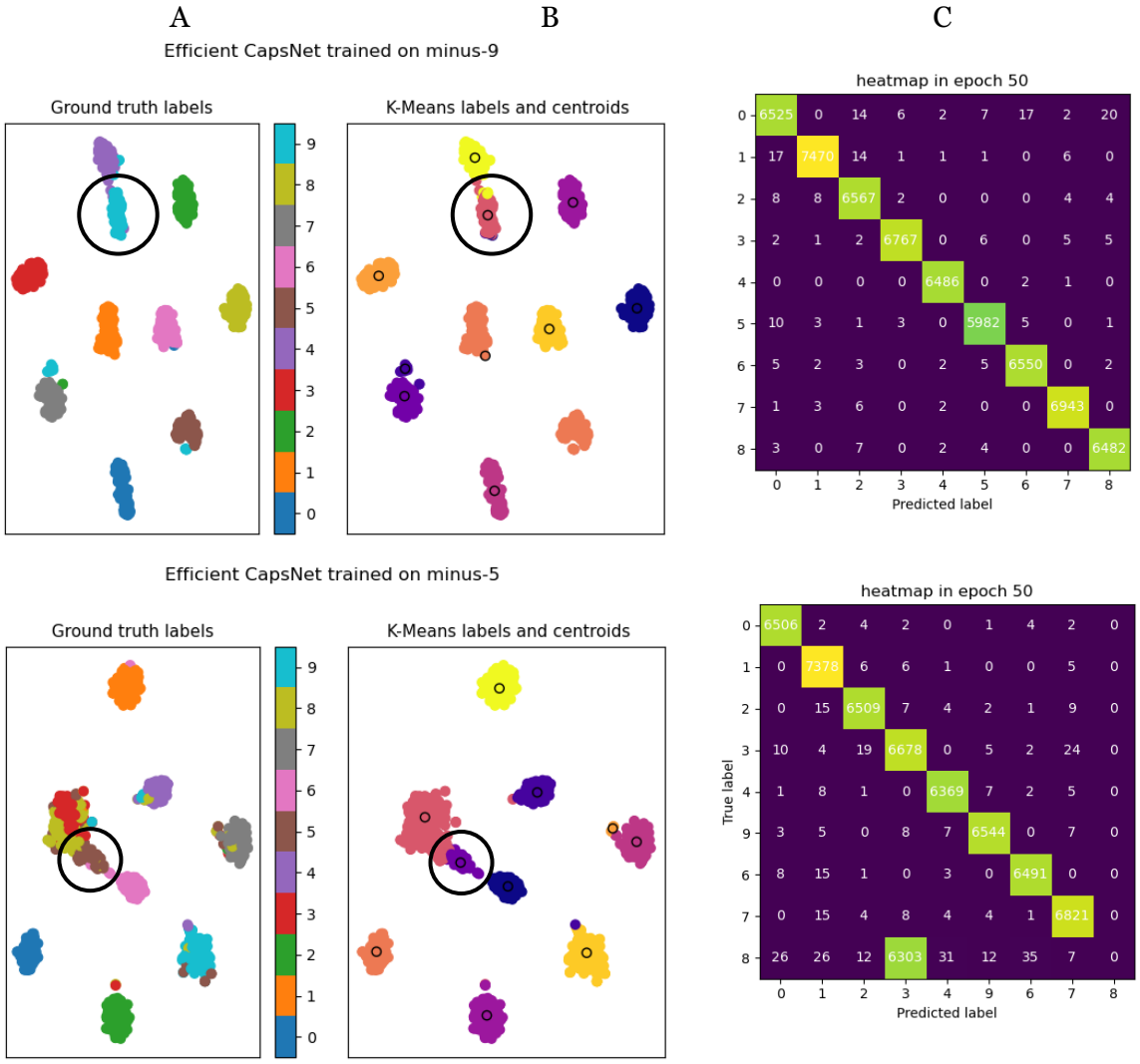


Figure 11: t-SNE mapping of the digit capsule feature space with or without K-Means clustering, after training Efficient CapsNet on a minus-X training set. Column A shows the mapping with the ground truth labels. Column B shows the mapping with the labels assigned by K-Means clustering. The colours of the K-Means labels are not the same as for the ground truth labels, because K-Means randomly initiates clusters and therefore labels, as such the colours are used to differentiate the clusters only. The large black circles indicate the cluster of the new digit. Column C shows the prediction matrices of the last epoch in the training phase. Top: Efficient CapsNet trained on a minus-9 training set, inference with standard test set. Bottom: Efficient CapsNet trained on a minus-5 training set, inference with the standard test set.

That the network is unable to correctly identify a three and an eight is also indicated by the prediction matrix at the end of training (column C of figure 11). This matrix shows that all eight digits are misclassified as threes. In case of the experiment with the nine left out, the prediction matrix shows no such misclassification problems. It should be noted that the problem with classification of the eight does not happen every time the network is trained on the minus-5 training set. In other training runs, other digits are harder to learn for the network, indicating that the initialization of the weights plays a role in which digits are harder to classify.

Finally, the new-digit experiment was also performed with Small CapsNet. During the training phase, the network learned to classify digits correctly quicker than Efficient CapsNet. Most predictions were already aligned with the true labels of the digits after the first epoch. As a result, there were no cases such as the minus-5 training set on Efficient CapsNet, where the network had a low classification accuracy after training for any specific digits.

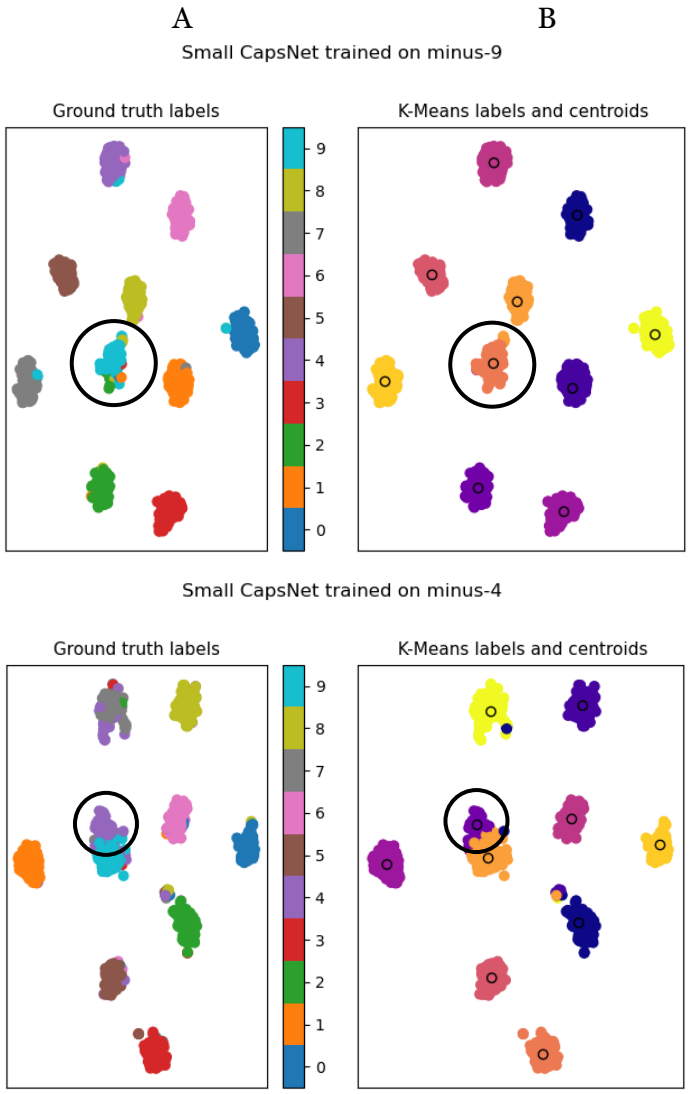


Figure 12: t-SNE mapping of the digit capsule feature space with or without K-Means clustering, after training Small CapsNet on a minus-X training set. Column A shows the mapping with the true labels. Column B shows the mapping with the labels assigned by K-Means. The colours of the K-Means labels are not the same as for the ground truth labels, because K-Means randomly initiates clusters and therefore labels, as such the colours are used to differentiate the clusters only. The large black circles indicate the cluster of the new digit. Top: Small CapsNet trained on a minus-9 training set, inference with standard test set. Bottom: Small CapsNet trained on a minus-4 training set, inference with the standard test set.

The minus-9 example in figure 12 is a completely separate cluster from the others. Interestingly, the cluster of fours with the minus-4 training set is intertwined with the clusters of nines and sevens respectively. This indicates that the network needs more information to distinguish a four from a nine or a seven, while this is not necessary for the minus-9 training set. Separate clusters for the fours, sevens and nines are still found by K-Means, and the centroids correspond to the middle of the clusters. In some cases for the minus-4 dataset, a separate cluster cannot be identified by K-Means, see A.4.

The difference between the minus-9 and the minus-4 datasets becomes clear in figure 13. The mean class probabilities show that most of the time, a four-digit image is classified as a nine for the minus-4 dataset. However, this is not seen for the nine-digit images, for which the probabilities are evenly divided between the digit capsules. This suggests that the ninth digit looks enough like all other digits to not be pulled towards one cluster in the feature space.

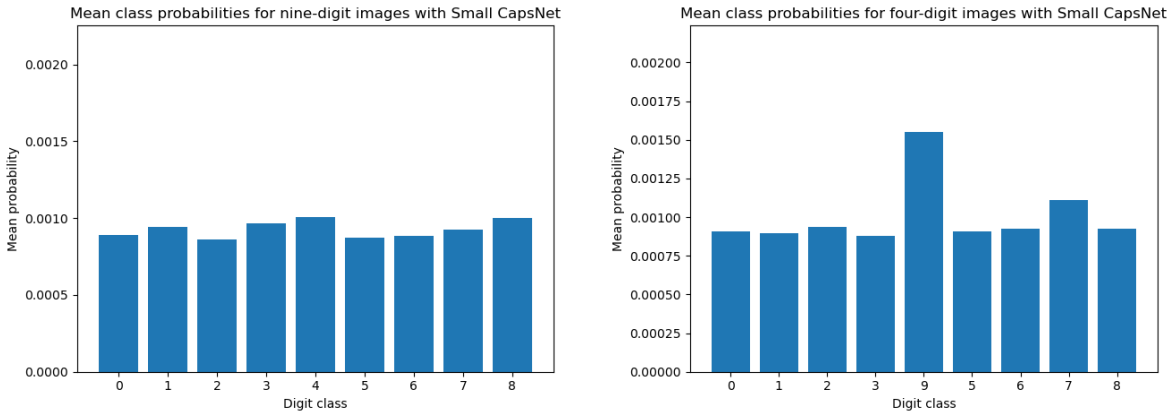


Figure 13: Bar charts of the mean class probabilities of the digit capsules. Left: mean probabilities for 94 nine-digit images. The ninth digit was left out of training. Right: mean probabilities for 104 four-digit images. The ninth digit here is encoded in the fifth capsule and the four was left out of training.

4. Discussion

The results have exposed several important differences between the two networks used in this study. Efficient CapsNet performs relatively well for most of the augmented datasets, but mostly produces lower accuracies and higher losses than Small CapsNet. In addition, Efficient CapsNet shows no indication of learning some scaling or translation characteristics, whereas Small CapsNet does. The new digit experiment results show that the clustering of a new digit class is possible, although not perfect. This applies to both Efficient and Small CapsNet. All of these results and their implications are discussed in more detail in this section.

As described in the introduction, one of the important factors aiding in incremental learning is the ability of neural networks to generalize outside of the training data. In general, the results show that Small CapsNet is better at generalization than Efficient CapsNet. This is suggested by both the scaling and the translation experiments, where Small CapsNet showed a scaling and translation factor and Efficient CapsNet did not.

Additionally, where the accuracies and losses did not vary much between training sets for Small CapsNet, Efficient CapsNet produced lower accuracies and higher losses for more complex training sets. It is clear that the larger network of the two has more trouble training when there are more augmented images in the training set, given its lower performance for the half-half translation training set than the 1-in-25 training set for example. The higher losses for the augmented test sets could indicate that the network has started to overfit the

data and is looking in too much detail at digit characteristics. This could also be why Efficient CapsNet does not show any scaling or translation factors in the capsule dimensions, even while the accuracy is still above 80 percent: the network is learning about details for each digit class, but as a result it does not see encompassing characteristics such as scaling and translation.

The fact alone that it was necessary with Efficient CapsNet to upscale the number of training epochs to reach an accuracy above 50 percent for the translation experiment suggests that the network structure is not working for these datasets. Small CapsNet is fully trained in each experiment after 20 epochs, without relinquishing its high performance rates. Network overfitting is a common problem and is also known to affect generalization capabilities [25]. Often this is caused by too many neurons (i.e. nodes) and therefore too many weights that can be tuned, allowing the network too much room to look at the details.

Where Efficient CapsNet is possibly too complex, Small CapsNet is not. It contains less primary capsules than digit capsules, forcing the network to more carefully choose the information it encodes. With the self-attention routing, primary capsules effectively 'vote' which of the ten digit capsules correspond with their output. But with only eight primary capsules, there are simply less votes, meaning that those votes must individually be more important. In a way, this is similar to the reasoning behind sparsity, where weights are penalized for being non-zero, forcing the network to use less space to encode the same amount of information [26, 27]. Since sparsity has been known to improve a network's generalizing capabilities, it would not be unreasonable to assume the same for Small CapsNet's architecture.

However, while Small CapsNet shows scaling and translation in the digit reconstructions, it does not yet seem capable of generalizing outside of the training data. The test with the four-quadrant translation test set illustrates this inability. The two new positions do form separate groups and show some clustering, but certainly not good enough for classification. The question rises how many examples of e.g. different positions the network would need before being able to generalize to any position, for any digit.

Furthermore, not every digit class shows the scaling and translation factors in the capsule dimensions of Small CapsNet. This indicates that part of the network still looks at the different scaling possibilities and the two possible digit positions as being separate features instead of a feature with multiple possibilities. The network seems to see a small digit and a normal-sized digit, and similarly a left digit and a right digit, instead of a digit that can be small or normal-sized, and left or right. The reconstructions for the translation experiment of Efficient CapsNet are more alike these Small CapsNet cases, where the position of the digit is always the same as in the original image it is reconstructing.

This dependence of the position of the digit on the original image suggests that part of Small CapsNet and the whole of Efficient CapsNet first look at the position of the digit, before even determining what the digit class is. On the other hand, the rest of Small CapsNet seems to first look at what digit class it is and only after that what the digit's position is. Otherwise the digit would not be visualized as being divided over the two corners in the reconstruction. Looking at the position first is more complicated, since then the rest of the digit must still be identified. In this way, the network must basically learn everything twice for every digit class: once for the digit translated to the left, and once for the digit translated to the right. This would explain why the digit clusters are so far apart in the feature space for Efficient CapsNet, and not for Small CapsNet. For the same reason, it would explain why Efficient CapsNet needs more training time than Small CapsNet.

Both Efficient CapsNet and Small CapsNet do show signs of generalization in the reconstructed digit images. Most of the reconstructed digits look alike within their classes, indicating that the network learns a general representation of each digit class. This was not the case in other capsule network implementations [16, 22], where each digit reconstruction was more like its original image. The cause of this difference could be the network architectures, or the value of the reconstruction loss. The low value of this loss ensures that good reconstructions are not a priority.

Reconstructed images were sometimes empty or of poor quality for translated digits, while the reconstructions of scaled and standard images were of good quality and identifiable. At least one issue for the decoder with the translated images is that three-quarters of the image is zero. Since the reconstruction loss is an MSE loss, the network can easily minimize the loss by giving every pixel a zero-value. If it does that, already more than three-quarters of the reconstruction pixels equal the original image pixels, resulting in a low loss. This can be achieved early on in the training process, practically leaving the network with only the margin loss to learn the digit features. This could also contribute to the long training time of Efficient CapsNet.

A second important aspect of incremental learning is being able to learn new classes or information, naturally. The two-digit experiment does not provide any proof that Efficient CapsNet might be able to learn two-digit numbers when it already knows all one-digit numbers. However, this is not surprising considering the fact that the network did not learn to see the concept of a digit as a whole. It sees edges and corners, groups of active pixels, but it has not been taught to look at space between entire groups of active pixels. For humans it is easy to see that there are two objects in the image, but a network is not able to do this without extra help. An example of a network that is able to localize and classify objects is YOLO [28], but it was trained to also predict bounding boxes and constrained to predict only one class per specified grid cell in the image. Efficient CapsNet could probably recognize the two digits separately if trained with the same specifications as YOLO, but it still would not learn a new class for each two-digit number.

On the other hand, the new-digit experiment does suggest that a capsule network might be able to learn new classes incrementally. The results of the experiment show that it is possible to recognize new digits as being new and different from all the other known classes. Both networks tested in this work enabled the clustering of the new digit images together in some cases, though not all.

Additionally, the new-digit images are not separated from the other digits as well as these other digit clusters are separated from each other. This might be attributed to a number of reasons, one of them being that K-Means is not very well suited for high-dimensional data clustering since it works based on Euclidean Distance, and distance starts to mean less and less with each dimension added [29]. Another explanation could be that the networks have not been given enough training time, especially Efficient CapsNet. Lastly, K-Means offers a variety of possible settings that have not been tweaked at all in this work, which could also result in a better performance.

Similar to the translation experiment, Efficient CapsNet needed more training time with the new-digit experiment to reach a similar accuracy to Small CapsNet with the standard training epochs. Even after giving Efficient CapsNet more epochs, it still has trouble correctly identifying each digit class. This is odd, given the fact that it has one less digit to train. While it is likely that which digit is harder for the network to classify is dependent on the weight initialization, it is unlikely that this is also the reason for the network struggling with the classification in general. Each training run with the minus-5 dataset resulted in classification issues with other digit classes. It is more likely that this difference compared to the standard

dataset has something to do with the missing digit. With a digit class left out of training, the network simply has less features to use for the recognition of the digit classes.

Interestingly, having less features to use for training does not seem to have an equal effect for each digit that is left out. When looking at the results for Small CapsNet, it is clear that when trained on the minus-4 dataset, the clustering is less successful because the four-digit images are always attached to the nine-digit clusters. However, this is not the case when the roles are reversed and the nine is left out of training. The nine-digit images do not lie close to the four-digit clusters in the feature space. This difference is clearly explained in the bar charts, which strongly indicate that the nine simply shares its features with the other digits and the four mostly shares features with the nine. This would indicate that a wide range of basic features is important and also does the job (for the nine). The MNIST dataset clearly does not provide a wide enough range of features to recognize the four as something separate.

5. Conclusion and Recommendations

This work presents a proof of concept for an incremental learning approach that is based on capsule networks. The approach consists of a fixed basic network, trained such that it contains a wide range of features, and a clustering method to combine these features into a new class or concept. To this end, the generalizing capabilities of the networks were investigated, and the ability to discern new classes from trained ones.

The scaling and translation experiments have provided valuable insights about the generalizing capabilities of the two networks Efficient CapsNet and Small CapsNet. Small CapsNet in particular shows promising results with regards to generalization, with the presence of scaling and translation factors in the capsule dimension reconstructions. As expected, Small CapsNet performed better than Efficient CapsNet on this dataset (MNIST), the latter possibly being too large and overfitting on the data.

Generalizing to new positions or scales or any other slightly changed class characteristic is necessary to be able to recognize new objects regardless of viewpoint. Ideally, these characteristics would be universal and encoded in the fixed basic network of the incremental learning approach. The results have shown that while some classes did seem to contain a universal translation or scaling factor, not every class did and generalization outside of the training data was not seen at all. This is a problem that needs solving before this approach can work on a large scale.

The second part of the approach is to use known features from the fixed network to learn new classes. The two-digit experiment did not provide proof for this method, mainly because the networks were not trained to also see separate objects in the same image. Without additional training procedures, the network will not be able to define a class for a two-digit number.

However, the new-digit experiment did suggest that learning new classes is possible with this approach. The results revealed that both networks are able to cluster new images, separately from other classes. The differences between the results for the different minus-X datasets provide more proof that a wide range of features enables the clustering of new images into a new class.

In conclusion, the results of this work suggest that having a fixed basic network before clustering the features into new classes could work as an incremental learning approach. It is proven that new classes can be discerned from the ones the network has trained on and the first steps towards generalization have been made.

Recommendations

Naturally, there is room for improvement and further research on this specific topic. Several suggestions are discussed here.

First of all, this research was still limited with respect to dataset variety and augmentation, leaving room for improvement. More types of data augmentation should also be tested, including for example a smaller translation. The translation used in this work was relatively large, which probably made it more difficult for the networks to learn. A smaller translation might even lead to the networks being able to generalize to other translation directions too.

A question that this work has not answered with regards to the data augmentation is how many augmented images are actually needed for the network to learn some scaling or translation parameter? Now, mostly one in 25 training images was augmented, but it might be more beneficial to have either more or less augmented images.

Another aspect that undoubtedly affects the generalizing capabilities of the networks is the loss function. As such, a different loss function or even a different approach to the combination of the reconstruction loss and the margin loss might help the network generalize better.

The two-digit experiment in this work pointed out that just because a network is able to extract features and combine them, does not mean that it can understand higher level concepts without help. While the network might be able to learn new classes, it will still need help finding objects in images, i.e. segmentation. An approach that might be suitable is one from [30], where the network is two-part. First, the network identifies the different parts in order to then assign them to wholes.

As mentioned, K-Means clustering is not well-suited for high-dimensional data, so it would be beneficial to try out other clustering algorithms. The new-digit experiment results might improve with dimensionality reduction first or using a different clustering algorithm altogether. Another benefit of a different algorithm might be that it is not necessary to provide the number of clusters you're expecting to see. Hardcoding this number is naturally not desirable for an autonomous system.

Additionally, the new-digit experiments were only performed with one digit left out of training, but it is not yet known how this experiment would go if more than one digit is left out. Would the second new digit also be separate from the first new digit and all other digit clusters? Answering this question is particularly important, because if the networks are unable to discern between two different new classes the whole approach does not work. Admittedly, MNIST might not be the best dataset to test this out with, since it only has ten classes. Leaving two out already might compromise the wide range of features that the fixed basic network is supposed to have.

Lastly, both networks were only trained for either 20 or 50 epochs. While the dataset is not that complicated, it should at least be investigated if this training time is the right one. This should however be treated carefully because as mentioned networks might start to overfit if trained too long. Next to the training time, learning rate and the optimizer were also chosen arbitrarily and could possibly be helpful in tweaking the performance.

REFERENCES

1. Ministerie van Volksgezondheid Welzijn en Sport, *Nadere toelichting arbeidsmarktprognose*. 2020. Available from: https://www.rijksoverheid.nl/ministeries/ministerie-van-volksgezondheid-welzijn-en-sport/documenten/publicaties/2020/11/09/nadere-toelichting-arbeidsmarktprognose.
2. Centraal Bureau voor Statistiek, *Fysieke arbeidsbelasting werknemers; beroep*. 2021. Available from: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/84435NED/table?ts=1625052796876.
3. Jiang, J., et al., *Research progress and prospect of nursing robot.* Recent Patents on Mechanical Engineering, 2018. **11**(1): p. 41-57.
4. Krizhevsky, A., I. Sutskever, and G.E. Hinton, *Imagenet classification with deep convolutional neural networks.* Advances in neural information processing systems, 2012. **25**: p. 1097-1105.
5. Alom, M.Z., et al., *The history began from alexnet: A comprehensive survey on deep learning approaches.* arXiv preprint arXiv:1803.01164, 2018.
6. van de Ven, G.M. and A. Tolias, *Three scenarios for continual learning.* ArXiv, 2019. **1904.07734**.
7. Ramasesh, V.V., E. Dyer, and M. Raghu, *Anatomy of Catastrophic Forgetting: Hidden Representations and Task Semantics.* arXiv e-prints, 2020. **2007.07400**.
8. Hadsell, R., et al., *Embracing Change: Continual Learning in Deep Neural Networks.* Trends in Cognitive Sciences, 2020. **24**(12): p. 1028-1040.
9. Rebuffi, S.-A., et al. *icarl: Incremental classifier and representation learning*. in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017.
10. Castro, F.M., et al. *End-to-end incremental learning*. in *Proceedings of the European conference on computer vision (ECCV)*. 2018.
11. Xiang, Y., et al. *Incremental learning using conditional adversarial networks*. in *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019.
12. Ven, G.M.v.d. and A. Tolias, *Generative replay with feedback connections as a general strategy for continual learning.* ArXiv, 2018. **abs/1809.10635**.
13. Ayub, A. and A.R. Wagner. *Tell me what this is: Few-shot incremental object learning by a robot*. in *IEEE International Conference on Intelligent Robots and Systems*. 2020.
14. Zhang, C., et al. *Few-shot incremental learning with continually evolved classifiers*. in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.
15. Masana, M., et al., *Class-incremental learning: survey and performance evaluation on image classification.* arXiv preprint arXiv:2010.15277, 2020.
16. Sabour, S., N. Frosst, and G.E. Hinton, *Dynamic Routing Between Capsules.* ArXiv, 2017. **abs/1710.09829**.
17. Azulay, A. and Y. Weiss, *Why do deep convolutional networks generalize so poorly to small image transformations?* arXiv preprint arXiv:1805.12177, 2018.
18. Engstrom, L., et al., *A rotation and a translation suffice: Fooling cnns with simple transformations.* 2018.
19. LeCun, Y., et al., *Handwritten digit recognition with a back-propagation network.* Advances in neural information processing systems, 1989. **2**.
20. Lecun, Y., et al., *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 1998. **86**(11): p. 2278-2324.
21. Mukhometzianov, R. and J. Carrillo, *CapsNet comparative performance evaluation for image classification.* arXiv preprint arXiv:1805.11195, 2018.
22. Mazzia, V., F. Salvetti, and M. Chiaberge, *Efficient-CapsNet: Capsule Network with Self-Attention Routing.* arXiv preprint arXiv:2101.12491, 2021.
23. Van der Maaten, L. and G. Hinton, *Visualizing data using t-SNE.* Journal of machine learning research, 2008. **9**(11).

24. Pedregosa, F., et al., *Scikit-learn: Machine learning in Python.* the Journal of machine Learning research, 2011. **12**: p. 2825-2830.
25. Zhang, C., et al., *Understanding deep learning (still) requires rethinking generalization.* Commun. ACM, 2021. **64**(3): p. 107–115.
26. Ng, A., *Sparse autoencoder.* CS294A Lecture notes, 2011. **72**(2011): p. 1-19.
27. Louizos, C., M. Welling, and D.P. Kingma, *Learning sparse neural networks through $ L_0 $ regularization.* arXiv preprint arXiv:1712.01312, 2017.
28. Redmon, J., et al. *You only look once: Unified, real-time object detection*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
29. Aggarwal, C.C., A. Hinneburg, and D.A. Keim. *On the Surprising Behavior of Distance Metrics in High Dimensional Space*. 2001. Berlin, Heidelberg: Springer Berlin Heidelberg.
30. Kosiorek, A.R., et al., *Stacked capsule autoencoders.* arXiv preprint arXiv:1906.06818, 2019.

3. General Discussion and Conclusion

The aim of this study was to provide a proof of concept for a novel incremental learning approach based on a fixed capsule network and a clustering algorithm. The concept rests on the assumption that with a wide enough range of features, any object can be learned using different combinations of the fixed features. To investigate whether the concept has merit, the generalizing capabilities of two capsule networks were investigated, and the ability to discern new classes from trained ones.

With regards to the networks' ability to generalize, the results indicate that the smaller network, Small CapsNet, performs better than the larger network, Efficient CapsNet. On the augmented scaled and translated datasets, Small CapsNet obtains higher accuracies than Efficient CapsNet. In addition, it shows a scaling and translation factor in some of the digit classes, which is expected if a network is to generalize outside of training data. However, while the scaling and translation factors might attribute to the higher accuracies, it does not yet allow Small CapsNet to generalize to new scaled or translated images.

The difference in performance between the two networks suggest that a smaller network is beneficial, while a larger network might be too complex. The larger a network is, the more space (i.e. neurons) it has to look too much at the details, resulting in a network that is very capable when used on exactly what it was trained for, but not new information.

Small CapsNet continues to outperform Efficient CapsNet in the new-digit experiment. Efficient CapsNet needs more than twice the training time of Small CapsNet, even though it is a larger network and has one digit less to train. This is another indication of the network simply being too large, since Small CapsNet does not seem to have any problems with training.

When looking at the incremental learning capabilities of the networks, both show promising results. The clustering algorithm succeeds in some cases in finding the new-digit images and is able to keep them separate from the other digits. This suggests that learning about a new object in an image, using only features of objects the network already knows is possible. How well different new digits are separated from the old digits also supports the assumption that the basic network needs to contain a wide enough range of features for it to be able to learn a new set of features.

In conclusion, this work has shown that having a neural network learn about new objects using a fixed set of features is possible. Naturally, more research is needed about the best training datasets for these kinds of experiments, the most ideal network architecture and clustering algorithms.

General References

1. Ministerie van Volksgezondheid Welzijn en Sport, *Nadere toelichting arbeidsmarktprognose*. 2020. Available from: https://www.rijksoverheid.nl/ministeries/ministerie-van-volksgezondheid-welzijn-en-sport/documenten/publicaties/2020/11/09/nadere-toelichting-arbeidsmarktprognose.
2. Deng, L. and D. Yu, *Deep Learning: Methods and Applications.* Foundations and Trends in Signal Processing, 2014. 7(3–4): p. 199-201.
3. LeCun, Y., Y. Bengio, and G. Hinton, *Deep learning.* nature, 2015. **521**(7553): p. 436-444.
4. van de Ven, G.M. and A. Tolias, *Three scenarios for continual learning.* ArXiv, 2019. **1904.07734**.
5. Ramasesh, V.V., E. Dyer, and M. Raghu, *Anatomy of Catastrophic Forgetting: Hidden Representations and Task Semantics.* arXiv e-prints, 2020. **2007.07400**.
6. Hadsell, R., et al., *Embracing Change: Continual Learning in Deep Neural Networks.* Trends in Cognitive Sciences, 2020. **24**(12): p. 1028-1040.
7. Sabour, S., N. Frosst, and G.E. Hinton, *Dynamic Routing Between Capsules.* ArXiv, 2017. **abs/1710.09829**.
8. Mazzia, V., F. Salvetti, and M. Chiaberge, *Efficient-CapsNet: Capsule Network with Self-Attention Routing.* arXiv preprint arXiv:2101.12491, 2021.

APPENDICES

A.1 NETWORK STRUCTURES FOR THE LARGER IMAGES

Table 5: The network structure for both Efficient CapsNet and Small CapsNet for the translated image datasets with 56 by 56 pixels.

Network	Convolutional layers				Fully connected layers		Primary capsule layer					Digit capsule layer		
	in_channels	out_channels	kernel size	stride	in_features	out_features	capsules	in_channels	out_channels	kernel size	dimensions	capsules	in_channels	dimensions
Efficient CapsNet	1 32 64 64	32 64 64 128	5 3 3 3	1 1 2 2			16	128	8	11	8	10	8	16
Decoder Efficient CapsNet					160 512 1024	512 1024 3136								
Small CapsNet	1 32 32	32 32 32	5 3 3	1 2 2			8	32	4	12	4	10	4	8
Decoder Small CapsNet					80 512 1024	512 1024 3136								

A.2 SCALING FACTOR SMALL CAPSNET

As mentioned in the results, not all digit classes show a scaling factor when Small CapsNet is trained on the scaled training set. Figure 14 gives several examples of classes where no scaling factor is visible.

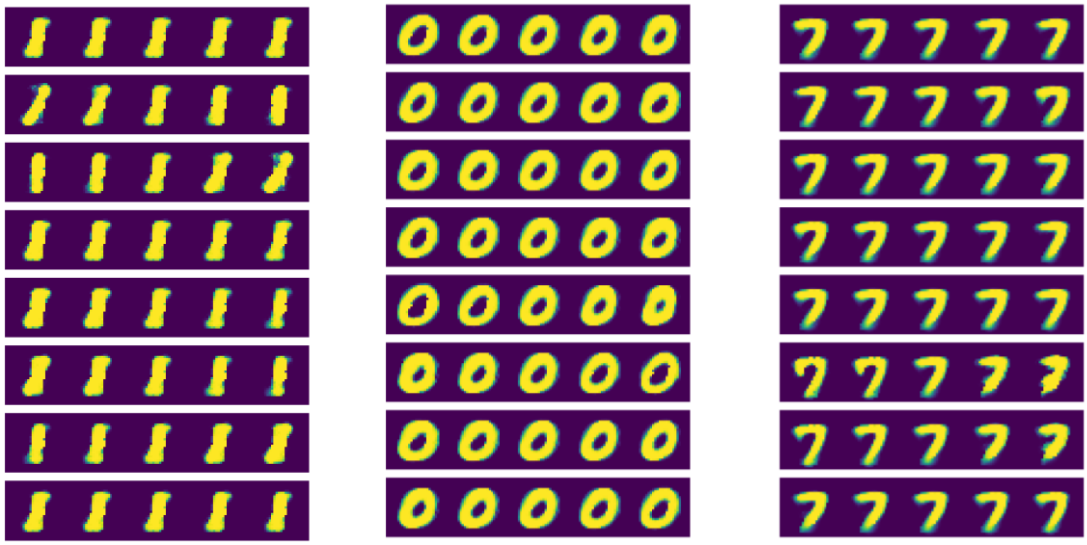


Figure 14: Reconstructions of digit images during inference, using Small CapsNet trained on the scaled training set. Each row represents a capsule dimension. Each column corresponds to a perturbation value (-0.10, -0.05, 0, +0.05, +0.10).

A.3 ADDITIONAL NEW-DIGIT FIGURES FOR EFFICIENT CAPSNET

For some batches of the new-digit experiment, the clustering did not go successfully, as shown in figure 15. In these cases, the K-Means algorithm was unable to identify the new-digit images separately from the rest.

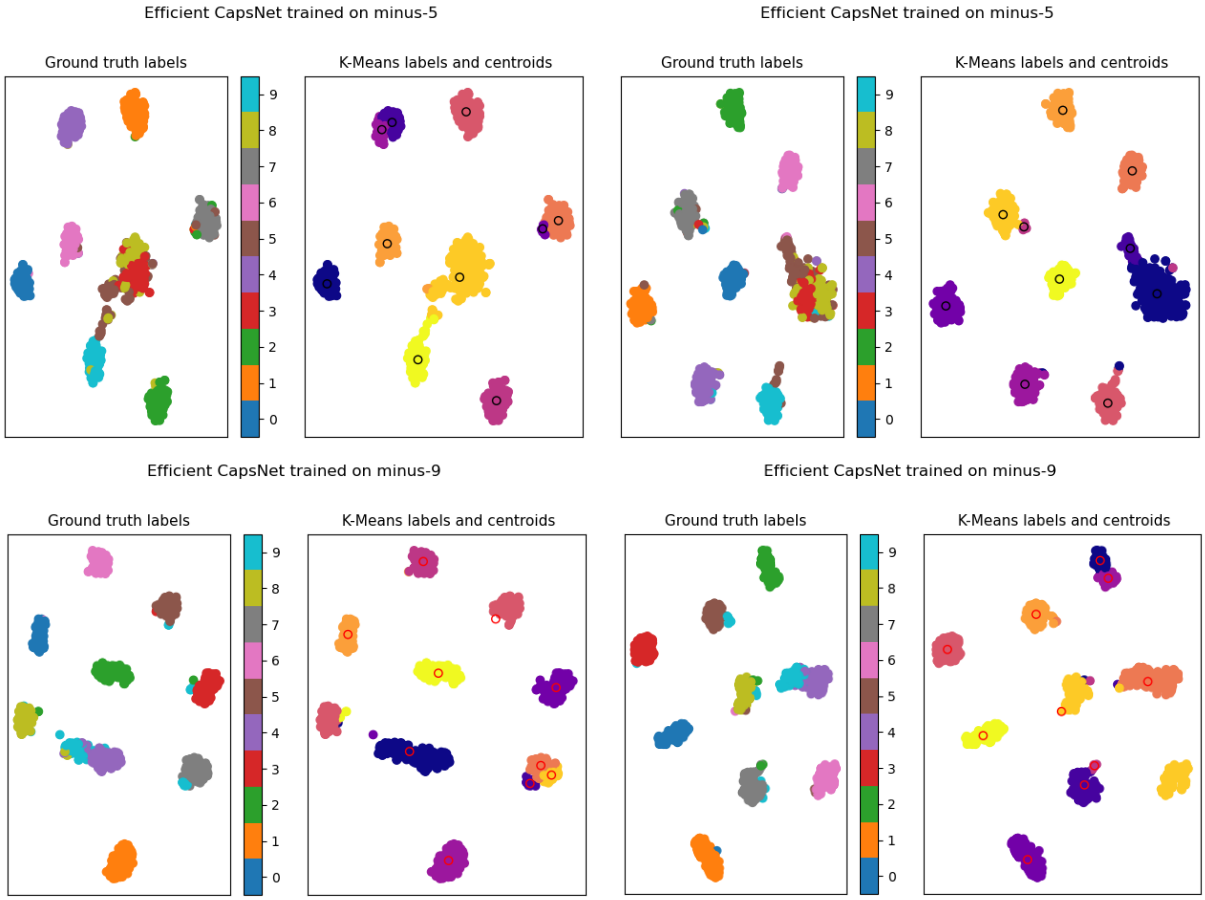


Figure 15: t-SNE mapping of the digit capsule feature space with or without K-Means clustering, after training Efficient CapsNet on a minus-X training set. All four images are examples of new-digit cases where K-Means has not been able to label the new-digit images together.

A.4 ADDITIONAL NEW-DIGIT FIGURES FOR SMALL CAPSNET

For some batches of the new-digit experiment, the clustering did not go successfully for Small CapsNet either, as shown in figure 16. In these cases, the K-Means algorithm was unable to identify the new-digit images separately from the rest.

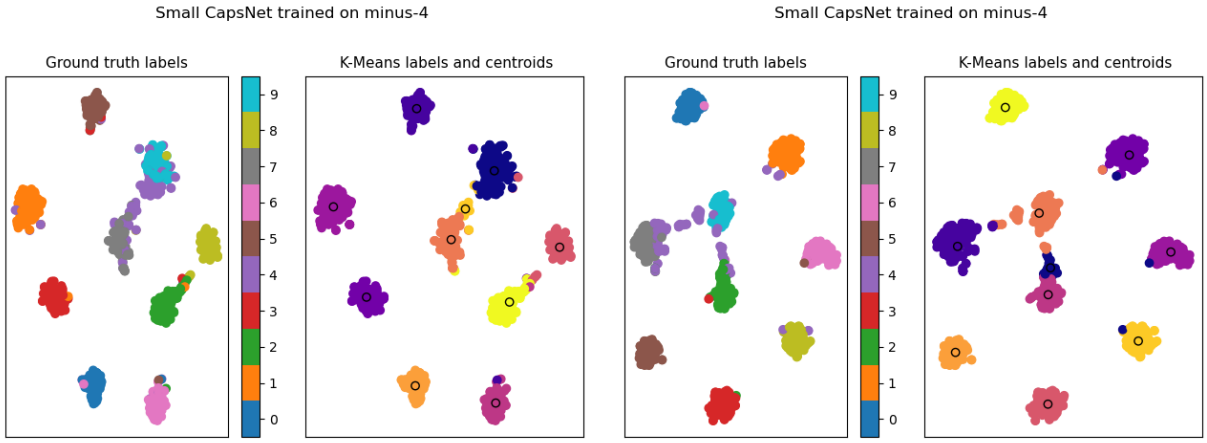


Figure 16: t-SNE mapping of the digit capsule feature space with or without K-Means clustering, after training Small CapsNet on a minus-X training set. The two images are examples of new-digit cases where K-Means has not been able to label the new-digit images together.