Optimizing Capsule Networks

by

Pouya Shiri

B.Sc., Shahid Beheshti University, 2015

M.Sc., University of Tehran, 2018

A Dissertation Submitted in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical And Computer Engineering

Optimizing Capsule Networks

by

Pouya Shiri

B.Sc., Shahid Beheshti University, 2015

M.Sc., University of Tehran, 2018

Supervisory Committee

Dr. Amirali Baniasadi, Supervisor

(Electrical And Computer Engineering)

Dr. Nikitas J. Dimopoulos, Departmental Member

(Electrical And Computer Engineering)

Dr. Nishant Mehta, Outside Member

(Computer Science Department)

ABSTRACT

Capsule Network (CapsNet) [1] was introduced in 2017 as the new generation of the image classifiers to perform supervised classification of images. It incorporates a new structure of neurons which is called a capsule. A capsule is basically a vector of neurons and serves as the basic computation unit in CapsNet. CapsNet has obtained state-of-the-art testing accuracy on the task of classifying the MNIST digit recognition dataset. Despite its fundamental advantages over CNNs, it has its own shortcomings as well. CapsNet provides a relatively high accuracy in classifying images with affine transforms applied to them and also classifying images containing overlapping categories, compared to CNNs. Unlike CNNs, CapsNet creates the representation based on the part to whole relationship of the features of different levels. As a result, it comes with a more robust representation of the input image. CapsNet could only get reasonable inference accuracy on small-scale datasets. Also, it only supports a limited number of categories in the classification task. Finally, CapsNet is a relatively slow network, which is mostly due to the iterative Dynamic Routing (DR) algorithm used in it. There have been several works trying to address the shortcomings of CapsNet since it was introduced.

In this work, we focus on optimizing CapsNet in several aspects: the network speed i.e. training and testing times, the number of parameters in the network, the network accuracy and its generalization ability. We propose several optimizations in order to compensate for the drawbacks of CapsNet. First, we introduce Quick-CapsNet (QCN) network with our primary focus on the network speed. QCN makes changes to the feature extractor of CapsNet and produces fewer capsules compared to

the baseline network (Base-CaspsNet) [1]. It performs inference 5x faster on small-scale datasets i.e. MNIST, F-MNIST, SVHN and CIFAR-10. QCN however loses testing accuracy marginally compared to the baseline e.g. 1% for F-MNIST dataset.

Our second contribution is designing a capsule-specific layer for the feature extractor of CapsNet referred to as Convolutional Fully-Connected (CFC) layer. We employ the CFC layer into CapsNet and call this new architecture CFC-CapsNet. CFC layer is added on top of the current feature extractor to translate the feature map into capsules. This layer has two parameters: kernel size and the output dimension. We performed some experiments to explore the effect of these two parameters on the network performance. Using the CFC layer results in reducing the number of parameters, faster training and testing, and higher test accuracy. On the CIFAR-10 dataset, CFC-CapsNet gets 1.46% higher accuracy (with baseline of 71.69%) and 49% fewer number of parameters. CFC-CapsNet is 4x and 4.5x faster than Base-CapsNet on CIFAR-10 for training and testing respectively.

Our third contribution includes the introduction of LE-CapsNet as a light, enhanced and resource-aware variant of CapsNet. This network contains a Primary Capsule Generator (PCG) module as well as a robust decoder. Using 3.8M weights, LE-CapsNet obtains 77.21% accuracy for the CIFAR-10 dataset while performing inference 4x faster than CapsNet. In addition, our proposed network is more robust at detecting images with affine transformations compared to CapsNet. We achieve 94.37% accuracy on the AffNIST dataset (compared to CapsNet's 90.52%).

Finally, we propose a deep variant of CapsNet consisting of several capsule layers referred to as Deep Light CapsNet (DL-CasNet). In this work, we design the Capsule Summarization (CapsSum) layer to reduce the complexity of the proposed deep network by reducing the number of parameters. DL-CapsNet, while being highly

[1] We refer to the first CapsNet [1] as Base-CapsNet.

accurate, employs a small number of parameters compared to the state-of-the-art CapsNet based networks. Moreover DL-CapsNet delivers faster training and inference. Using a 7-ensemble model on the CIFAR-10 dataset, we achieve a 91.29% accuracy. DL-CapsNet is among the few networks based on CapsNet that supports the CIFAR-100 dataset (68.36% test accuracy using the 7-ensemble model) and can process complex datasets with a high number of categories.

Contents

List of Tables

List of Figures

PREFACE

The following is the list of Pouya Shiri's publications at the University of Victoria in chronological order:

- Ramin Sharifi, Pouya Shiri, and Amirali Baniasadi. 2020. Zero-skipping in CapsNet. Is it worth it? In Proceedings of 35th International Conference on Computers and Their Applications (EPiC Series in Computing), EasyChair, 355–361. [4]

- Pouya Shiri, Ramin Sharifi, and Amirali Baniasadi. 2020. Quick-CapsNet (QCN): A Fast Alternative to Capsule Networks. In 2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA), 1–7. [5]

- Pouya Shiri and Amirali Baniasadi. 2021. Convolutional Fully-Connected Capsule Network (CFC-CapsNet). In Workshop on Design and Architectures for Signal and Image Processing (14th Edition) (DASIP '21), Association for Computing Machinery, Budapest, Hungary, 19–25. [6]

- Ramin Sharifi, Pouya Shiri, and Amirali Baniasadi. 2021. PrunedCaps: A Case For Primary Capsules Discrimination. In 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), 1437–1442. [7]

- Pouya Shiri and Amirali Baniasadi. 2021. LE-CapsNet: A Light and Enhanced Capsule Network. In 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), 1767–1772. [8]

- Pouya Shiri and Amirali Baniasadi. 2022. Convolutional Fully-Connected Capsule Network (CFC-CapsNet): A Novel and Fast Capsule Network. Journal of Signal Processing Systems. [9]

- Mohammad Chegini, Pouya Shiri and Amirali Baniasadi. 2022. RFNet: Fast and Efficient Neural Network for Modulation Classification of Radio Frequency Signals. ITU Journal on Future and Evolving Technologies (ITU J-FET). (Accepted, awaiting publication)

- Pouya Shiri, Ramin Sharifi and Amirali Baniasadi. 2022. Resource-Aware Capsule Networks. Deep Learning Applications, Volume 4. (Accepted, awaiting publication)

- Pouya Shiri and Amirali Baniasadi. 2022. DL-CapsNet: Deep and Light Capsule Network. In Workshop on Design and Architectures for Signal and Image Processing (15th Edition) (DASIP '22), Association for Computing Machinery, Budapest, Hungary. [10]

In [5], [6], [9], [8] and [10], Pouya Shiri conducted the research, analyzed the results, and prepared the manuscript under the guidance of Dr. Amirali Baniasadi.

In [4], [7], and Resource-Aware Capsule Networks, Pouya Shiri and Ramin Sharifi had collaboration in conducting the research, analyzing the results and preparing the manuscript under the guidance of Dr. Amirali Baniasadi.

In RF-Net, Pouya Shiri and Mohammad Chegini had collaboration in conducting the research, analyzing the results and preparing the manuscript under the guidance of Dr. Amirali Baniasadi. This paper is the outcome of a competition titled "Lightning-Fast Modulation Classification with Hardware-Efficient Neural Networks" hosted by Xilinx in the International Telecommunication Union (ITU) ML/AI in 5G Challenge.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Dr. Amirali Baniasadi for his consistent advice and support maintaining me on the right track. I would also like to thank Dr. Nikitas Dimopoulos for his helpful insights during our weekly meetings.

DEDICATION

To my lovely parents

Chapter 1

Introduction

Image Classification is one of the fundamental problems in Computer Vision. This problem is defined as categorizing a set of images into predefined classes. Several other computer vision tasks including object detection and semantic segmentation include the task of image classification as a sub-task. This task can be very challenging due to several factors, including the ever-increasing complexity we witness in modern datasets, the variation in the viewpoint of the object and the high intra-class variations [11].

Previously, the task of image classification was solved by extracting handcrafted features using feature descriptors and feeding them to a trainable classifier. A major drawback of this method was the fact that the accuracy was dependent on the feature extraction, and developing such extractor was a difficult task [12]. Deep Learning has overcome this by using several layers including non-linear data processing, and automatic feature extraction and transformation to provide solutions for pattern recognition and classification of data. Convolutional Neural Networks (CNNs) [13] have now become the leading deep learning method for most computer vision tasks including image classification [14].

The advances in Graphical Processing Units (GPUs), availability of large datasets, and the introduction of better algorithms all helped to a rise in popularity of CNNs [15].

These networks consist of several convolutional layers followed by fully-Connected (FC) layers [14]. We explain CNNs, their advantages and drawbacks and some of the mostly used layers in Chapter 3. Here we mention some key points about CNNs. Back in 2015, these networks started achieving exceptional performance in classification. For instance, AlexNet [16] and GoogLeNet [17], winners of the ImageNet classification competition (ILSVRC), were both CNN architectures. VGG [15] , which is another CNN, was the second-best classifier in the 2014. These types of networks have been making a significant progress since the introduction of AlexNet. EfficientNet and its variations have been able to achieve more than 90% top-1 accuracy [18] on the ImageNet dataset (compared to the 63.3% of AlexNet).

Classifiers based on CNNs, are networks consisting of layers of different types. Starting from the first layer, each layer aims to make a smaller abstraction of data coming from the previous layer such that the final layer can determine the category of the object available in the image. One of the fundamental layers used in CNNs is the Pooling layer. This layer reduces the input feature map size by summarizing it into a smaller feature map. There are two reasons for using such layers. Firstly, they adds a bit of translation invariance to the network. To be more specific, due to the use of these layers, a slight variation in the location of the object will not affect the decision of the classification. Secondly, they reduce the size of feature map and hence the number of parameters and the amount of computations. However, this layer loses information by dropping some values and summarizing the feature map, and this could be considered as a drawback of CNNs.

Another disadvantage of CNNs is that they do not consider the relationship be-

tween features of different levels. For instance, a portrait image of a face includes the face itself as a feature with the highest level. This feature consists of several lower level features such as lips, eyes, nose and so on. The spatial relationship between these low-level features and the high-level feature is not discovered and extracted by a CNN. As a result, if the location of the low-level features is changed, the network is likely to assign the image to a class which it actually does not belong to.

CapsNet was designed to address the above drawback [1]. CapsNet, its architecture and its advantages and drawbacks are explained in Chapter 3. Since its introduction, CapsNet has been a hot research topic [2, 19, 20, 21]. A CapsNet is a network that uses capsules as the basic unit of computation (in CNNs, the basic unit is a neuron). A capsule is a set of neurons building vectors (instead of neurons in CNNs). Although the concept was introduced earlier [22], there was no practical training mechanism for a network using capsules. In [1] the authors proposed an algorithm called "Routing by Agreement", or "Dynamic Routing" (DR) to make training capsules possible. Unlike CNNs, CapsNet takes into account the hierarchical relationship between low-level and high-level features using the DR algorithm.

CapsNet has several advantages over CNNs. While CNNs only provide a list of probabilities to categorize an image, CapsNets provide a vector for each category. Each dimension of these vectors correspond to an instantiation parameter of the object (e.g. rotation, skew, etc) of a specific class. In addition, CapsNet performs better in classifying image with overlapping categories and it is more robust to applying affine transformations to the input images compared to CNNs [1]. For instance, CapsNet can classify a rotated object by just being trained on the non-rotated input with a higher accuracy compared to CNNs.

Despite the advantages, CapsNet is a slow network compared to CNNs [23]. It cannot still perform on large-scale datasets with a high number of categories such as

ImageNet. In addition, it cannot supersede state-of-the-art CNNs in terms of network accuracy on small-scale datasets.

In this work, we focus on optimizing the performance of CapsNet. The performance of a network is evaluated using different metrics:

- The inference accuracy. This is an indicator of how the trained network generalizes to unseen data.
- The number of parameters. This metric represents the number of trainable parameters in the network. This serves as an indicator of how much memory footprint a network requires to perform the training and inference.
- The network training and inference time. This metric represents the amount of computations performed by a network to generate predictions from input images.

We take the first network based on Capsule Network [1] as the baseline and refer to it as Base-CapsNet. We primarily focus on making it faster and introduce Quick-CapsNet (QCN), which produces a significantly fewer number of capsules compared to Base-CapsNet. QCN builds on producing a fewer number of capsules, which results in a faster network. QCN achieves this at the cost of obtaining a slightly lower inference accuracy (68.17% on the CIFAR-10 dataset, compared to 71.69% of CapsNet). Inference is 5x faster on MNIST, F-MNIST, SVHN and Cifar-10 datasets. We also further enhanced QCN by employing a more robust decoder (the class-independent decoder) instead of the default decoder to further improve QCN. Quick-CapsNet (QCN) serves as a faster alternative to CapsNet, which can be a starting point to develop CapsNet for fast real-time applications.

To further improve the performance of CapsNet, we introduce a layer referred to as Convolutional Fully-Connected (CFC) layer and integrate it into CapsNet to

propose CFC-CapsNet. This network also takes advantage of a class-independent decoder. CFC-CapsNet is not as fast as QCN, but it is still faster than Base-CapsNet by 4.5 times. In addition, it obtains a higher accuracy, and includes up to 50% fewer parameters compared to Base-CapsNet. We also perform a detailed analysis of the CFC-CapNet to investigate how the performance changes by changing the parameters of the CFC layer.

As our next step towards improving CapsNet, we introduced the Primary Capsule Generator (PCG) module, and integrate it into CapsNet to introduce Light and Enhanced Capsule Network (LE-CapsNet). The PCG module generates a few robust capsules that can lead to a a representation with high generalization ability. This network also includes a dropout method specialized for capsules (referred to as Capsule Dropout) to improve the generalization of the network and uses the class-independent decoder. LE-CapsNet obtains a higher accuracy compared to Base-CapsNet, includes 67% fewer parameters, and performs 4x faster than Base-CapsNet.

The next step of this research, was to introduce the Multi-Level Capsule Extractor (MLCE) module and the Capsule Summarization (CapsSum) layers and carefully integrates them into CapsNet to introduce Deep and Light Capsule Network (DL-CapsNet). This network not only acheives a high accuracy on small-scale datasets such as MNIST, F-MNIST, SVHN and CIFAR-10, but also is capable of handling CIFAR-100 as a dataset with highly varied background and a high number of categories (100 classes). With 67.56% test accuracy for CIFAR-100, DL-CapsNet is among the few variants of CapsNet that performs well on this complex dataset.

The rest of this thesis is organized as follows. Chapter 2 provides some of the related works on optimizing CapsNet. Chapter 3 provides background information about CNNs and CapsNet. Chapter 4 explains the proposed solutions in detail. The experiments and results are included in Chapter 5. Finally, Chapter 6 concludes this

thesis.

Chapter 2

Background

This chapter aims to explain some background information about CNNs and CapsNet and the advantages and drawbacks of them. First, CNNs are briefly introduced and the strength of CNNs is stated by mentioning some of the state-of-the-art classifiers. Next, the various types of layers used in CNNs are briefly described.

In the second part of this chapter, CapsNet is briefly introduced, and the advantages of this network over CNNs is shown. Next, the architecture of CapsNet is explained and the chapter is concluded with the short-comings of CapsNet, which motivated this work.

2.1 Convolutional Neural Networks

CNNs have been very useful and effective for image classification. They have achieved wonderful performance on tasks that previous classic classifiers had limited performance. CNNs have made classification easier than before, since hand-crafted features are no longer necessary. These networks, take advantage of the fact that the nearby pixels in the image (local regions) are strongly correlated, and that is not necessarily true for distant pixels (globally).

Examples of classification tasks that CNNs have been very successful at, include classification of well-known small-scale datasets. The details of the datasets mentioned here are explained at the end of this chapter. Modified CNNs have obtained test accuracies of 99.79% on MNIST digit recognition, 96.53% on CIFAR-10, 75.72% on CIFAR-100 (similar to CIFAR-10, but containing 100 categories) and 98.31% on SVHN (street-view house numbers) datasets [24].

In the following sections, we explain different types of layers and functions used in CNNs.

2.1.1 Training, Testing and Loss Function

The datasets on which classification task is performed, include images with ground-truth categories (labels) assigned to them. These datasets come with two sets of data: training and testing. The training set is used to progressively make the prediction provided by the model more accurate. Testing set is used to verify how accurate the model predicts the categories.

The aim of the classifiers is to come up with a model that predicts the class of the input image. This is achieved using the optimization process called training. The loss function represents how well the created model by the classifier can predict the input images while doing the training. The aim of the optimization process is to lower the value of loss function progressively. There are different types of loss functions. A loss function consists of a main loss term, as well as regularizer terms which will be explained later in this chapter.

2.1.2 Forward Pass and Back-Propagation

The input images are fed to a network consisting of several layers. Each layer consists of several parameters that need to be trained during the training process. These

parameters are called weights. A batch of images is processed through multiple layers and creates the predictions. This process is called Forward-Pass. Once the predictions are made, they are compared to the ground-truth and the loss term is calculated. Then the mentioned weights of the network are updated based on the gradients of the loss term with respect to each weight. The process of updating weights is called back-propagation. The term back-propagation is used because for calculating the gradient of the weights of a layer, the information from all next layers needs to be propagated and weights are updated in reversed order.

2.1.3 Convolutional Layer

This type of layer takes data from the previous layer and performs some element-wise multiplications. A convolutional layer includes two types of weights. The first types is called a kernel that is strided over the input and for each stride, the kernel is multiplied with the input element by element. The other type is called bias, which are independent of the input, and they are added to the input layer. These weights try to extract the relationship between neighbour pixels. In the 2-dimensional (2D) case, the kernel is a 3D grid which consists of multiple 2D weights. Each set of 2D weights are called a feature map. The number of feature maps is called the depth of the convolutional layer.

The size of the output is calculated using the following equation:

$$O = \frac{W - K + 2P}{S} + 1 \tag{2.1}$$

where O is the size (width) of output, W is the size of input, K is the size of kernel, and P and S denote the padding and the stride size of the layer. In some cases, zero values are added around the input layer. This is called padding and the number of

zero rows and columns added shows the parameter P in the equation.

The number of parameters added to the model by using a convolutional layer, can be calculated using the following equation:

$$N_P = W + B = K^2 \cdot C \cdot N + N \tag{2.2}$$

where N_P is the total number of parameters, W is the parameters due to the weights in the kernel, B is the parameters of the biases, K is the size of kernel, C is the number of input channels and N is the number of kernels (depth of the convolutional layer).

After most of convolutional layers, there is an activation function to add non-linearity to the model. Activation functions are explained later in this chapter. The process of stacking several convolutional layers is called encoding, as usually the input image is progressively encoded into a shorter representation.

2.1.4 Transposed Convolution Layer

Transposed convolution layer (also called Fractionally Strided Convolution layer) aim to reverse the process of convolution layers. As a result, they are sometimes called Deconvolution layers, but this naming is not correct. Because the output of transposed convolution layer is not the exact reverse of the convolution process. This layer adds padding in between the neurons as necessary to make a larger feature map. The layer contains a kernel and it performs the convolution operation on the enlarged input (by padding) to get an output which is larger than the input.

2.1.5 Fully-Connected (FC) Layer

Fully-Connected (FC) layers add a neuron called bias to the input layer, and connect all these neurons to the output layer considering a weight for each connection. This

type of layer is mostly used in the decoder to make the encoded data larger for reconstruction purposes. The number of parameters in FC layers depend only on the input and output size. FC layers include $(N + 1) \cdot M$ parameters, where N is the input size and M is the output size. FC layers are not used on large input sizes as it would add lots of parameters to the model.

2.1.6 Activation Functions

Activation functions add non-linearity to the model so that the output is not simply a linear combination of the input. There are several activation functions, with different applications. Some famous activation functions include Rectified Linear Unit (ReLU), Leaky ReLU, Exponential Linear Unit (ELU) , Sigmoid and SoftMax [25]. Each activation function has its own advantages and disadvantages.

There is a common problem in neural networks called the Vanishing Gradient problem. In some cases, the gradient will be small, and it effectively prevents the weights from changing their value. In the worst case, this problem might stop the neural network from training.

ReLU function

This activation function zeros out the negative values and keeps the positive values unchanged. The following equation shows the ReLU function:

$$R(z) = \max(z, 0) = \begin{cases} z, z > 0 \\ 0, z \leq 0 \end{cases} \tag{2.3}$$

ReLU activation avoids the vanishing gradient problem. This activation also includes simple mathematical calculation for the gradient (during back-propagation), so it is

not computationally expensive. However, ReLU has no upper limit for positive values so outputs might explode. In addition, using ReLU might result in the "dying ReLU" effect in which dead neurons will be present. A dead neuron has a zero gradient so it will not be updated anymore. Other variants of ReLU such as Leaky ReLU and ELU were later introduced to avoid the problem of dying ReLU. Leaky ReLU and ELU consider a small positive gradient for negative inputs to address the dying ReLU problem and give the neurons a chance to recover. The Leaky ReLU and ELU use linear and exponential functions for negative inputs respectively.

Sigmoid

Sigmoid activation outputs a value between 0 and 1. As opposed to ReLU, Sigmoid is continuously differentiable. The following shows the Sigmoid function:

$$S(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

Due to its limited output, Sigmoid does not have the exploding problem. It also has a smooth gradient. However, for very high or very low inputs, the output responds to input with a small gradient. This results in the vanishing gradient problem. In addition, since the output is not centered around zero, the gradients go far in different directions and this makes the optimization harder.

SoftMax

This activation function is applied to the final layer in the classification networks. This activation turns numbers into probabilities. The created probabilities sum to 1. The SoftMax function is shown in the following equation:

$$S(z_i) = \frac{e^{z_i}}{1 + e^{z_i}} \tag{2.5}$$

2.1.7 Pooling Layer

The function of a pooling layer is to reduce the size of input. This results in the reduction of number of parameters and computations. Pooling layer takes a single value for different square segments of the input and disregards the rest of values. The most commonly used pooling layer is max-pooling in which the maximum of the square window is taken for different strides. An advantage of max-pooling is to avoid the effect the slight changes on the input might have on the classification task. However, there is a loss of information wherever max-pooling is used.

2.1.8 Regularizers and Drop-out

Regularizers and Drop-out are used to avoid the problem of over-fitting in networks. Over-fitting is related to the generalization ability of the trained model. If the model performs well on training data but fails to perform the same on the test data, the model is biased on the training data and hence over-fitting has occurred. Two of the mostly used regularizers include weight-loss and reconstruction losses. These two regularizers are integrated into the model by adding terms to the loss function.

Weight-loss regularizer adds a penalty based on the value of weights. So, the model is forced to have smaller weights. Reconstruction loss method tries to reconstruct the input image, and compare how accurate the reconstructed image is in respect to the input image. This accuracy is considered inside the loss function. Drop-out method randomly drops some of the neurons in a layer. This technique results in having a model that is not biased on specific neurons. Once some neurons are randomly dropped, the model is forced to learn using the remaining neurons.

2.2 Capsule Network (CapsNet)

Capsule Network (CapsNet) groups the neurons into vectors and calls this new structure a capsule. CapsNet changes the basic unit of the neural networks from neurons to vectors, resulting in holding more information about the features available in the image in the last layer. Specifically, CNNs provide a single value showing the probability of the image to contain an object of a certain category. CapsNets instead provide vectors, whose normalized magnitude denotes the probabilities. The different dimensions of these vectors, have a correlation with different instantiation parameters of the object in the image (size, skew, etc.). The architecture of the first CapsNet introduced in [1] is shown in figure 2.1. There are two convolutional layers that extract features from the input image. The output of the feature extractor, is "reshaped" into vectors. To this end, the feature map is divided into equal groups, and each location in a group of feature maps, creates a single vector. As such, for each location, there are several vectors describing the feature available in a region of the image.

The created vectors are multiplied by weight matrices in a stage referred to as the Affine Transform Matrix Multiplication. These weights are part of the trainable parameters of the network. There are different sets of matrices for each category in the classification task. The total number of these matrices equals $N_C \cdot N_I$, where N_C is the number of categories, and N_I is the number of input vectors. These matrices learn the transformation between different features available in the input image, and the whole images, making CapsNet robust to affine transformations. The output of the matrix multiplication are the Primary Capsules (PCs). There could be several capsule layers after this point (only one in the original implementation). If u_i denotes the output of a capsule in the previous layer, and $\hat{u}_{j|i}$ shows the predictions of the

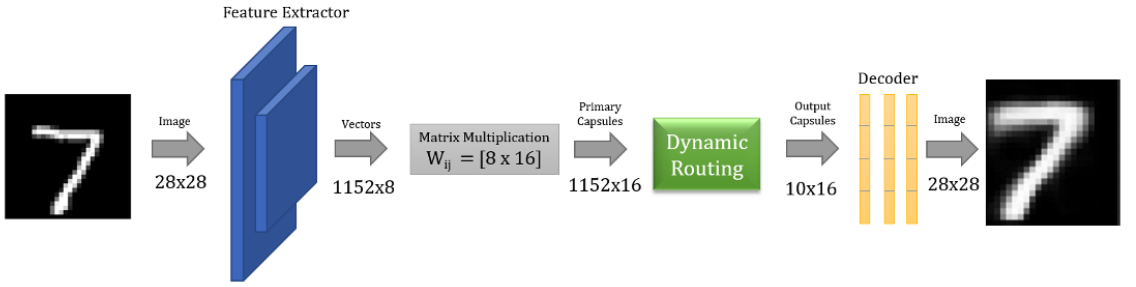


Figure 2.1: CapsNet architecture

current layer, we can write the following equation:

$$\hat{u}_{j|i} = W_{ij} \cdot u_i \tag{2.6}$$

where W_{ij} denotes the affine transformation matrix for the output u_i. The output of the current layer is calculated using the following equation:

$$s_j = squash(\sum_i c_{ij} \hat{u}_{j|i}) \tag{2.7}$$

where c_{ij} are coefficients determined by Dynamic Routing (DR) algorithm which is explained later in this chapter and squash is a non-linear activation function.

There are K output vectors in CapsNet where K is the number of classes in the classification task. The magnitude of each vector shows the probability that the entity represented by that capsule exists in the input image. Each dimension of this vector, is supposed to represent one instantiation parameter of the image e.g. skew and thickness. To realize the correspondence of vector length to the mentioned probability, a non-linear function is introduced called "squash" function. This function attempts to zero out short vectors and shrink the long vectors to a length of almost 1. This function is showed below:

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{2.8}$$

where j shows the index of capsule, s_j is its input vector, v_j is its output vector and $||s_j||$ denotes the magnitude of vector s_j.

2.2.1 Dynamic Routing Algorithm

The inference of the capsules in a layer from the previous capsule layer is performed using an iterative algorithm called Dynamic Routing. All capsules in the previous layer are connected to the capsules in the next layer (similar to FC layers in CNNs) and the weights of these connections are not determined by back-propagation, but by the DR algorithm. This algorithm is based on the level of agreement between the available predictions of the capsules on whether activate a specific capsule in the next layer or not.

Algorithm 1 Dynamic Routing Algorithm

```
1: procedure ROUTING(û_{j|i}, l, r)
2:     b_ij ← 0                                  ▷ for all capsules i and j in layers l and l+1
3:     for i iterations do
4:         c_i ← SoftMax(b_i)                    ▷ for all capsule i in layer l
5:         v_j = Squash(Σ_i c_ij û_{j|i})        ▷ for all capsule j in layer l+1
6:         b_ij = b_ij + û_{j|i}.v_j             ▷ for all capsule i and j in layers l and l+1
       return v_j
```

In this algorithm, b_{ij} are the log probabilities that capsules i and j from layers l and l+1 should be coupled. SoftMax function used here is based on the following equation:

$$c_{ij} = \frac{\exp b_{ij}}{\sum_k \exp b_{ik}} \tag{2.9}$$

DR algorithm is also called "routing by agreement" algorithm. This is due to the fact that routing is performed based on the agreement between the current output of

each capsule in the layer with the input predictions of the layer using the dot product between the two vectors.

2.2.2 Margin Loss

As explained in the previous section, the length of a capsule represents the probability that the capsule has predicted the entity corresponding to its class correctly. This fact is used to create a loss function for CapsNet called margin loss. There would be a high value for loss when incorrect predictions of all capsules. As a result, there is a separate loss term for each capsule. The following is the margin loss for each capsule:

$$L_k = T_k \max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) \max(0, ||v_k|| - m^-)^2 \tag{2.10}$$

where T_k is 1 if the entity is present, $||v_k||$ is the length of k-th capsule, λ is the down-weighting factor absent classes, and m^+ and m^- are used to remove predictions with very high or very low probability from the loss function.

2.2.3 Regularization by Reconstruction

The output capsules in CapsNet are used as an input to a decoder to reconstruct the input images. Figure 2.2 shows the CapsNet's decoder [1]. Comparing the reconstructed images with the input images adds an additional term to the loss function: reconstruction loss. The output capsules are masked before being fed to the decoder. During training, all output vectors except the one corresponding to the ground-truth label are masked (zeroed out). During inference, this masking is performed based on the magnitude of vectors: only the vector with highest magnitude is kept and the rest of the vectors are masked.

The decoder used in baseline CapsNet, includes several FC layers to decode the

output vectors back into the input image. The reconstruction is refrained from dominating margin loss by assigning a small weight to its term in the loss function (0.0005 in the baseline CapsNet).

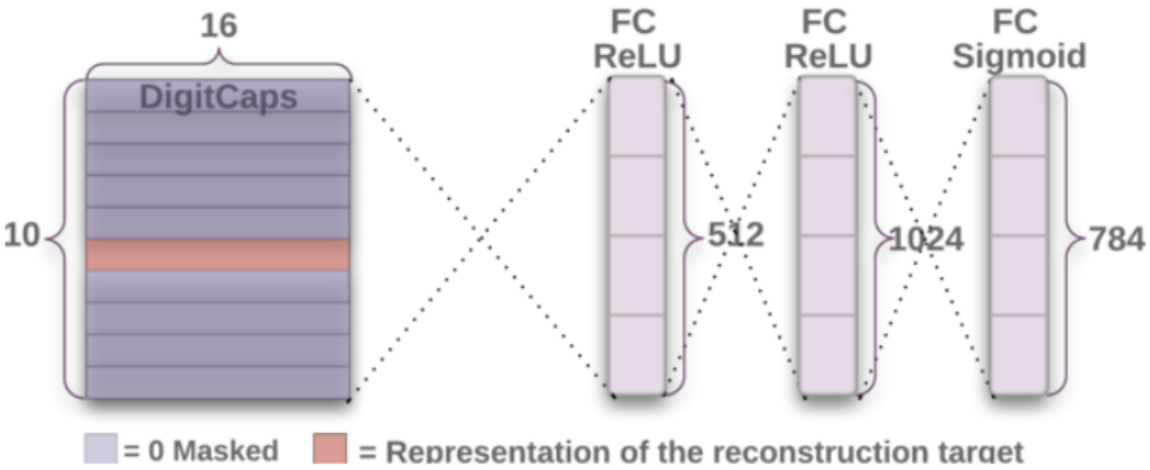


Figure 2.2: CapsNet Decoder [1].

2.2.4 Advantages of CapsNet over CNNs

The first drawback of CNNs, is ignoring the relationship between low-level and high-level features. This is due to the use of pooling layers. A high-level feature like a face consists of low-level features such as lips, nose, eyes and so on. CNNs do not consider the spatial relationship of these low-level features. Consequently, replacing the location of lips with the nose, would not affect the decision of the image being a face in CNNs.

CapsNet outputs a vector for each category in the classification task, while CNNs only provide the probability. As a results, CapsNet includes additional information (the pose of the object) as well the category it belongs to. The part-to-whole relationship between the different levels of features is obtained by CapsNet using the DR algorithm. Apart from this intrinsic advantage of CapsNet over CNNs, it is also more

robust to affine transformations applied to the input images. In other words, even without being trained on affine transformed image inputs, it can classify them better than conventional CNNs. This was verified in [1] by testing CapsNet and a conventional CNN on affined transformed MNIST dataset. The network is trained on raw dataset and tested on the transformed dataset. In addition, CapsNet performs better than CNNs in classifying images with overlapping categories. This was also verified in [1] by testing CapsNet on Multi-MNIST dataset. This dataset was generated by overlaying an input of one category on top of the other from a different category from the same dataset split (test or training).

To further elaborate on how CapsNet works, let us take the example of a face image. After using the low-level feature extractor and preparing the vectors, there are several vectors available for every region of the face image e.g. some around the nose, some around the lips. The weight matrices in the matrix multiplication stage are trained to find transformation between each of these vectors to the whole image, and the spatial relationship between these vectors is extracted during by iterating through the DR algorithm. The vectors "agreeing" on activating the face capsule [1], are assigned to a higher non-trainable weight in this algorithm to enforce their higher contribution in forming the face image.

2.2.5 Drawbacks of CapsNet

CapsNet has several drawbacks compared to CNNs:

- Slow Training and Testing
- Unable to classify complex datasets with high number of classes

CapsNet is a slow network compared to CNNs. This is due to the new structure of

[1] there is one output capsule for each category in the classification task

neurons as capsules and the different way capsules are processed. For example, DR algorithm is a computationally expensive operation which is repeated for 3 iterations in every forward pass of the network. The number of capsules directly influences the amount of computations in DR algorithm.

In CapsNet, there are K output vectors where K is the number of classes in the classification task. As a result, CapsNet contains too many weights for classification tasks with a high number of classes such as ImageNet. In addition, the baseline CapsNet is not capable of classifying complex datasets. There have been several works in literature trying to push the limits of CapsNet. Some of these works were reviewed in the previous chapter (related works).

Chapter 3

Related Work

This chapter covers some of the related works on CapsNet optimization. CapsNet has attracted a lot of attention since its introduction. There are several works trying to overcome the drawbacks of CapsNet. These works aim to improve CapsNet from different aspects:

- Improving the network accuracy
- Improving the network speed
- Optimizing the network volume and complexity

The first item focuses on making changes in the architecture to make CapsNet capable of classifying more complex datasets and push the limits of CapsNet in terms of generalization. The second item focuses on the network speed and CapsNet is changed to converge in fewer number of iterations or make the training and testing process faster. The third item focuses on making the network simpler and lighter. The volume of the network corresponds to the number of trainable parameters (weights) that exist in it.

In this Chapter, we divide the capsule-based networks into two groups: those with

a shallow feature extractor and those with a deep feature extractors. Base-CapsNet itself includes a shallow feature extractor (only two convolutional layers) and several other works also do not focus on improving the feature extractor, and instead focus on improving how capsules are processed in later stages of the network. However, there are works that focus on creating capsules using deep feature extractors, and hence they usually obtain a higher accuracy.

3.1 Networks with Shallow Feature Extractors

Xiang et al. introduce Multi-Scale CapsNet (MS-CapsNet) in [2] as an extension to CapsNet which creates more robust and efficient feature representations of the images. The feature extraction is different in MS-CapsNet with that of CapsNet: it is performed in multiple scales. The extracted features in different scales are then translated to capsules which form the first layer of capsules (Primary Capsules). They also introduce a new dropout technique to be used for capsules that increases the robustness of CapsNet.

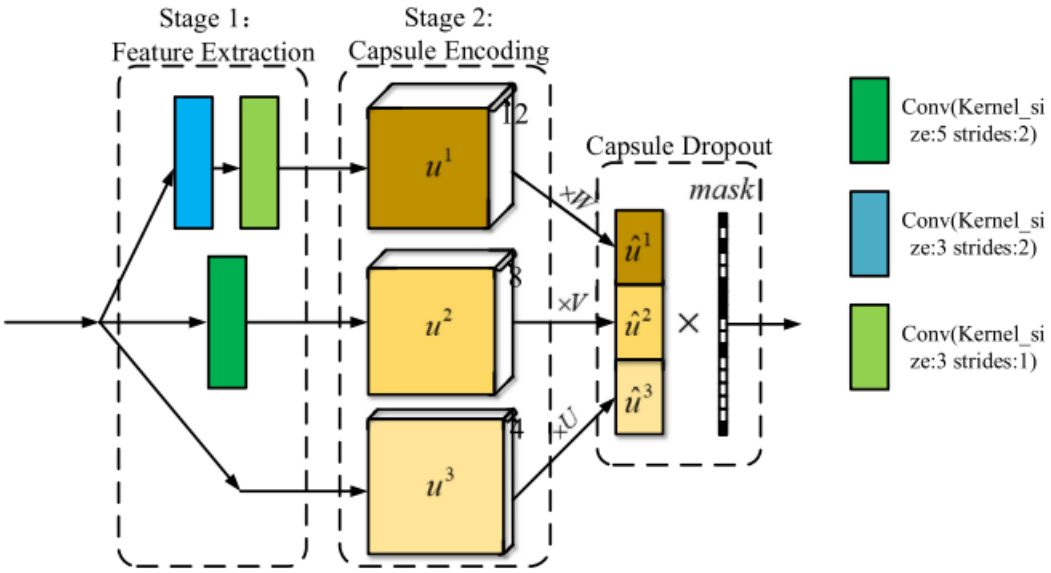


Figure 3.1: MS-CapsNet architecture [2]

Figure 3.1 shows the architecture of MS-CapsNet. As the figure shows the feature extraction consists of several layers each of which contains different convolutional layers with different properties. The capsule encoding creates capsules out of every layer of feature extraction. The dimensions of the created capsules are different for each output level of the feature extractor. As a result, different weight matrices are required for every level. In other words, matrices of different sizes are required to prepare the capsule predictions to take part in the dynamic routing.

After the capsule predictions are created out of different layers of feature extractor, the predictions are concatenated to create a single list to take part in the Dynamic Routing (DR) algorithm.

The next innovation of MS-CapsNet is the new dropout introduced. Dropout is used to make the model less dependent solely on individual predictions, and make the final decision based on all predictions. The dropout technique used in MS-CapsNet is to randomly remove some predictions (zero out the whole prediction) while performing

the training.

MS-CapsNet uses its own implementation of CapsNet on MX-Net and is tested on Fashion-MNIST and CIFAR-10 datasets. It gets a higher accuracy than the baseline CapsNet by 1.1% and 2.4% on the mentioned datasets respectively to reach to 75.7% for CIFAR-10 and 92.7% for Fashion-MNIST.

Rosario et al. [26] introduced MLCN which is a resource efficient variant of CapsNet which achieves higher accuracy and also has faster training and testing times. MLCN divides the primary capsules (PCs) into independent groups called lanes each of which is responsible to form one of the dimensions of the final digit capsules. Each lane has two properties: depth and width. The depth of a lane corresponds to the number of convolutions applied in that lane. The width of a lane corresponds to the number of filters per convolution.

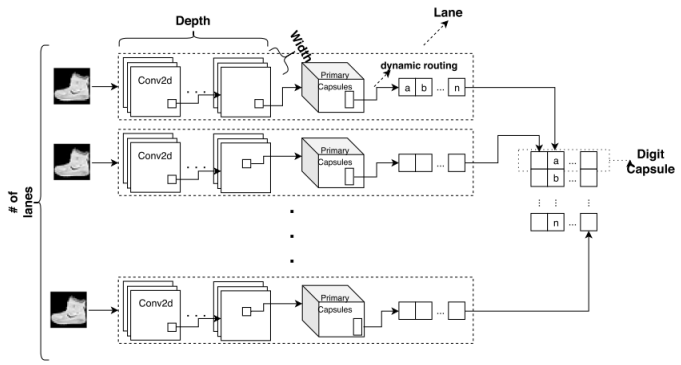


Figure 3.2: MLCN architecture

The advantage of MLCN over CapsNet is that it allows parallel execution of lanes, as PCs are constructed independently. In addition, the explainability of the network is improved as different features are associated to different sets of convolutions. Figure 3.2 shows the architecture of MLCN. As the figure shows, each lane is responsible for creating one dimension of the output. There are experiments done for different number of lanes (different dimensions of the digit capsule) i.e. 2, 4, 8, 16 and 32

lanes.

MLCN also integrates a new dropout technique (lane dropout) in which 10% of lanes are discarded randomly and the other 90% are used to calculate the classification. This method forces the lanes to contribute equally to the classification results.

There is an optimum number of lanes for different datasets and adding more lanes than the optimum values does not make the model better. MLCN has two configurations for lanes which differ in the properties of convolutional layers. MLCN is implemented on Keras-CapsNet code [27]. With 75.18% and 92.63%, it gets 8.82% and 1.33% higher accuracy than baseline CapsNet on CIFAR-10 and Fashion-MNIST datasets. It is also on average two times faster than CapsNet using the same number of parameters.

Amer et al. [28] propose Path-CapsNet which is a deep parallel multi-path variant of CapsNet. They make CapsNet deeper by including multiple pathways, and the resulting architecture obtains a better performance and also has fewer number of parameters. The idea of this work is that each PC is formed using a deep CNN, which is called a path. Path-CapsNet also introduces fan-in routing as a new routing strategy to replace the fan-out method originally introduced in [1]. It also integrates max-pooling and claims that it is not contradictory with the property of equivariance in CapsNet. Max-Pooling can be used to save parameters without losing performance and pose detection of CapsNet.

Layer	Type	Kernel	Padding	Stride	Output Channels
1	Conv	9	4	1	16
2	Conv	9	4	1	16
3	Maxpool	2	0	2	16
4	Conv	9	4	1	16
5	Conv	9	4	1	8
6	Maxpool	2	0	2	8

Figure 3.3: Layer specification of paths in Path-CapsNet

Path-CapsNet applies regularization by using a dropout method called DropCircuit which adapts dropout to multi-path architectures. It drops different paths ran-

domly so that the final output is not biased on specific paths. It is observed that the dropout method is not effective if there are only few lanes, since it might remove all the necessary information required for classification. The next innovation of this work is the new routing called fan-out routing. The fan-in routing is different to fan-out routing in the way of applying softmax. In fan-out routing, softmax is applied across connections coming out from a single PC while in fan-in routing, it is applied across connections coming into the same DigitCaps. Intuitively, in fan-out routing each part of an object is supposed to contribute more to a specific category. On the other hand in fan-in routing, different parts of the object have a different contribution for each category.

Figure 3.3 shows the layers constituting each path in Path-CapsNet. As this figure shows, each path consists of a deep CNN and it forms one of PCs. Path-CapsNet is tested on MNIST dataset using a different configuration i.e. the routing type, number of paths and whether or not using dropout.

Path-CapsNet claims that it has increased the representation power of PCs. Without fan-in routing and DropCircuit, it gets an accuracy of 99.56% on MNIST using 34% parameters of CapsNet.

Zou et al. [29] introduce Fast Convergent CapsNet (FCC) which is an extension of CapsNet which introduces a new activation function and also uses a regularization method. FCC makes the network converge faster and increases the ability of the network to generalize.

The activation function used in CapsNet is Squash and it works as follows:

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{3.1}$$

The alternative activation function introduced in FCC is instead based on the following equation:

$$v_j = (1 - e^{-a||s_j||^b}) \frac{s_j}{||s_j||} \tag{3.2}$$

This new activation function is better than squash function, because it makes the network converge faster, and it results in higher accuracy.

FCC also added a regularization term to the loss function of CapsNet: the minimum weight loss. This helps in avoiding over-fitting. The new loss function is in the following form.

$$Loss = \alpha L_M + \beta L_R + (1 - \alpha - \beta) \sum_i ||w_i||^2$$

where L_M and L_R denote the margin loss and the reconstruction loss respectively and the third term is the regularization term added to avoid over-fitting.

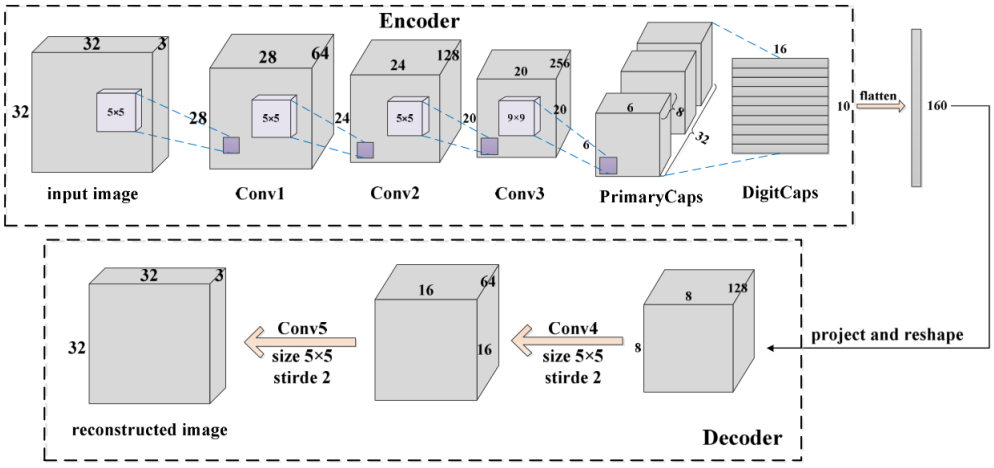


Figure 3.4: FSC-CapsNet architecture

Liu et al. [30] introduce FSC-CapsNet aiming to improve the performance of CapsNet on more complex datasets such as CIFAR-10. They modify both the encoder and the decoder of CapsNet. In the encoder (feature extractor), they add more convolutional layers and in the decoder they improve the performance of reconstruction by integrating fractionally-strided convolutional layers (deconvolution layers). FSC uses

no pooling layers in the proposed architecture. Figure 3.4 depicts the architecture of FSC. As the figure shows, there is one more convolutional layer compared to the baseline CapsNet and also the reconstruction network is now based on deconvolution instead of FC layers.

Liu et al. suggest that datasets that are more complex than MNIST require a deeper feature extractor and as a result, more number of convolutional layers. Also, the reconstruction for CIFAR-10 dataset is very blurry and is not comparable to that of MNIST dataset. So, they would like to make the reconstruction network more complex than three fully connected layers. FSC is tested on five complex datasets including CIFAR-10, CIFAR-100, Extended and Cropped Yale Face Database and BelgiumTS and it gets accuracy improvement for all cases. However, for CIFAR-10 it gets the highest improvement of 11% (with the baseline accuracy of 66.31%) compared to the baseline CapsNet.

FSC includes an analysis on number of convolutional layers in the extractor and the number of deconvolutions in the reconstruction part of the network. It turned out that the optimum numbers of layers are 3 and 2 for the encoder and decoder respectively. This configuration gets the best accuracy and the generalization performance. FSC also converges faster than baseline in all datasets except the Extended Yale Face Database. The images are reconstructed with a higher quality than the baseline. Even though for CIFAR-10 the reconstructions have a clearer outline, they are still very unclear and lack recognizable features.

Mobiny et al. [20] propose a fast variant of CapsNet (Fast CapsNet, hereby called FCNet) and apply it for the classification of lung nodules. CNN classifiers are common solutions for this problem. However, they require a large training set to provide a good generalization. FCNet improves the network training and inference time of CapsNet, so the proposed network can perform the classification with a higher ac-

curacy when fewer data is available. FCNet comes with a an alternative method for dynamic routing that leads to 3x speedup. In addition, since the current reconstruction network of CapsNet works poorly for lung nodule data, Mobiny et al. propose an efficient reconstruction method called convolutional decoder. This new reconstruction results in a higher classification accuracy.

Dynamic routing algorithm has a high computation load and it is not scalable. Consistent Dynamic Routing (CDR) solves this problem. In CDR, the coefficients of the capsules corresponding to the same pixels of the feature map are the same. In practice, CDR is a different organization of feature map to create the PCs. So there would be fewer capsules with higher dimensionality.

There is a high visual variation in the lung nodule data and the reconstruction of baseline CapsNet is not efficient for this data. FCNet comes with a convolutional decoder. This decoder takes the prediction vector of the predicted class, concatenated with the final output of CapsNet (target predictions) as input, and applies two fractionally-strided convolutions (deconvolutions) on it to reconstruct the input image.

Figure 3.5 shows the architecture of FCNet. As shown in the figure, the two changes made to the baseline CapsNet include the following:

- Different organization of the output of the feature extractor
- Different reconstruction method (changed input and deconvulotion)

Mobiny et al. also apply a 3D variant of FCNet on lung nodules dataset which has the same structure as the 2D variant, but inlucdes 3D convolutions and works on voxels. FCNet is compared with AlexNet, ResNet-50 and CapsNet on the lung nodules dataset. It gets the least error rate among all models both for 2D and the 3D model (11.45% and 9.52% respectively).

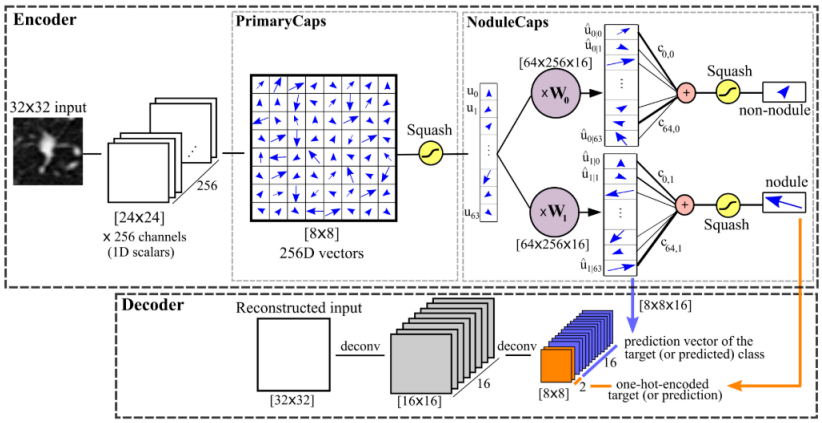


Figure 3.5: FCNet Architecture

Nair et al. [31] perform a stress test on CapsNet to verify its performance and level of expressiveness. They do so by applying CapsNet to high-scale datasets and exploring the sources of error in this network. The datasets in this work include MNIST, Fashion-MNIST, SVHN and CIFAR-10. This work mainly aims to verify the robustness of CapsNet to applying transformations to the input images. To this end, Nair et al. have modified all 4 mentioned datasets by augmenting them for testing purposes. The original datasets are used to train the CapsNet and the augmented datasets are used to test the network. The transformations applied to the datasets include rotation, shearing, translation and scaling. This work uses the Pytorch implementation of CapsNet.

The first comparison in this work is done between CapsNet and AlexNet [16]. The hyperparameters of CapsNet are the same ones in the original CapsNet paper [1]. CapsNet gets marginal improvement over AlexNet on MNIST and Fashion-MNIST (1.03% and 2.24% respectively) and substantial improvement on SVHN and CIFAR-10 (9.01% and 7.92% respectively). Nair et al. conclude that CapsNet is so far a good network as it has obtained a better accuracy using fewer number of parameters.

Nair et al. also conduct some experiments on the DR algorithm by exploring

the hyper-parameter "number of iterations in DR algorithm". They perform this investigation on all four datasets. Two iterations of DR performed very closely to three iterations, and even it some cases it performed better. This is in contrary with Sabour et al. [1], that claim 3 is the best choice for number of iterations. Nair et al. state that DR algorithm is too "abrup".

The next property of CapsNet inspected here, is the reconstruction. Nair et al. compare the reconstructed images with ground truth as the training goes on. They realize that for MNIST, there is substantial change in reconstruction from epoch 2 to 50. For Fashion-MNIST, the finer details of images are not reconstructed and this could be because of the shallow reconstruction network used. For SVHN, the reconstructions change from being non-informative to substantially appropriate and finally a gray rendering of the numbers. The network cannot reconstruct colors. Finally for CIFAR-10, the reconstructions are totally unrecognizable, except for the category of horse since the images in this category are more consistent.

Deliege et al. introduce HitNet [32] which includes a new layer referred to as the Hit-or-Miss (HoM) layer. The architecture of this network is shown in figure 3.6. HoM layer contains capsules that are trained to hit or miss a specific capsule based on a special loss function. This loss function is referred to as a "centripetal" loss function, because it is a piece-wise linear function of the predictions which is centripetal with respect to the correct capsules. This network also introduces the concept of ghost capsules that can potentially detect the mislabeled images or images that might need two labels in the training set.

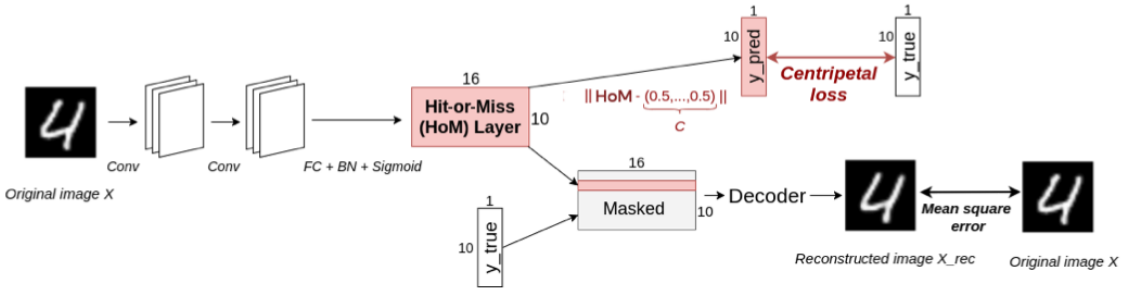


Figure 3.6: HitNet architecture

HitNet includes a shallow feature extractor to emphasize the potential of the HoM layer. This network replaces the squash activation with a Batch Normalization (BN) layer, followed by sigmoid activation to obtain HoM capsules. For each input image, HitNet aims the center of the feature space of the class that a capsule belongs to (hit), and capsules that are related to other calsses will be sent far away from the center of the feature space (miss).

3.2 Networks with Deep Feature Extractors

Rajasegaran et al. [19] propose DeepCaps as a deep variant of CapsNet. The two fundamental ideas behind DeepCaps include the following:

1. A novel dynamic routing algorithm based on 3D convolution

2. A class-independent decoder

DeepCaps gets better results than state-of-the-art works in CapsNet on CIFAR-10, Fashion-MNIST and SVHN datasets and it reduces the number of parameters by 68%.

The authors mention that the naive approach of simply stacking fully-connected capsule layers to come up with a deeper CapsNet has some drawbacks:

1. High computation cost and inference times due to the DR algorithm

2. Poor learning in the middle layers due to the small coupling coefficients

To reduce the computational complexity of CapsNet, authors propose some methods. The first one is to reduce the number of routing iterations in initial layers. The next proposal is to use 3D convolutional routing in the middle layers to reduce the number of parameters in the network by using the parameter sharing property.

In the feature extractor, DeepCaps uses skip-connections for reducing the effect of vanishing gradients. This network consists of three ConvCaps layers, one ConvCaps3D layer, one DigitCaps layer and finally the new reconstruction network which is based on deconvolution and is claimed to be class-independent. ConvCaps layers are the same as convolutional layers except the fact that their output is capsule-based. In other words, the output feature map is reshaped into vectors to be 4-dimensional instead of 3-dimensional ordinary feature maps. ConvCaps3D performs 3D convolutional operations and includes a novel 3D dynamic routing method.

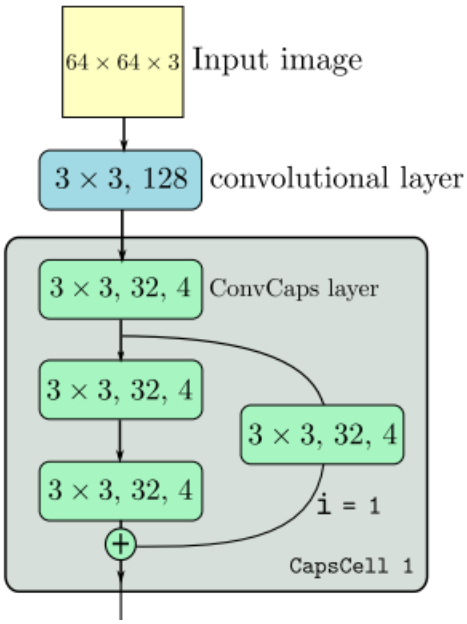


Figure 3.7: The first CapsCell in DeepCaps

The feature extractor in DeepCaps includes three CapsCells each of which includes a skip-connected convolutional layers stacked. The first CapsCell is shown in figure 3.7. There are three of these layers and then the final CapsCell which is different from the previous three in having ConvCaps3D operation. As figure 3.8 shows, there is an skip-connection from the output of the third ConvCaps which results in capsules being collected from both fourth and third ConvCaps layers. This skip-connection aims to contribute to the generalization of a model for a wider range of datasets. For instance, low level capsules from the first and second cell would be sufficient for simple datasets with poor information like MNIST. However, for more complex datasets like CIFAR-10 which has rich information, capsules generated using a deeper path are required.

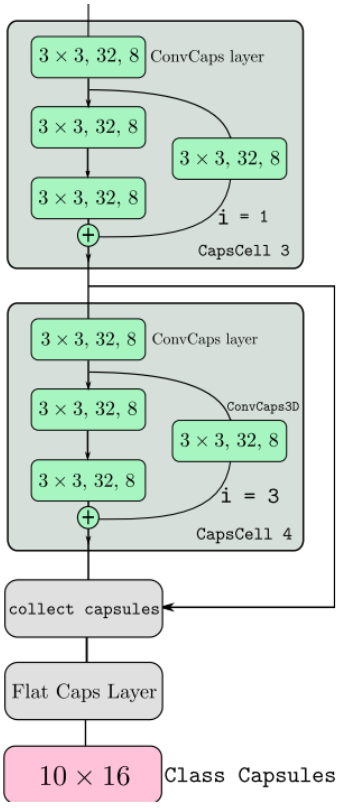


Figure 3.8: The final layers of DeepCaps

The next novelty in DeepCaps, is the proposed class-independent decoder. The

original decoder of CapsNet which includes two fully-connected layers, is class-dependent. The input of the two FC layers, is the masked output vector of the second capsule layer (DigitCaps). As figure 3.9 shows, the output includes b-dimensional vectors. All b values are zeroed out (masked) except the b values corresponding to the correct class (the ground truth) while training. While testing, the whole $a \times b$ values are used as the input to the decoder, and that is why the decoder is claimed to be class dependent. Instead in DeepCaps, there are only b values fed to the decoder: the b values corresponding to the ground truth class.

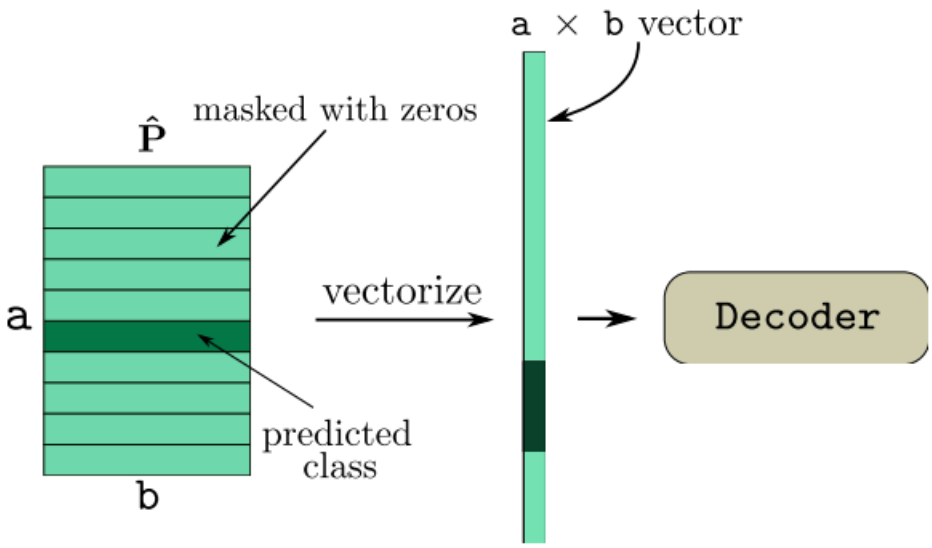


Figure 3.9: The original decoder input of CapsNet

Jasiwal et al. [33] propose CapsuleGAN, which uses CapsNet as the discriminator of GAN network while modeling image data. CapsuleGAN performs better than convolutional GAN at modeling the distribution of image data on MNIST and CIFAR-10 datasets using the generative adverserial metric (GAM). It also performs better at semi-supervised classification.

The margin loss of CapsNet is used instead of the binary cross-entropy loss of conventional discriminators for training. Visually and qualitatively, CapsuleGAN

generate cleaner and crisper images than convolutional GAN. Jaiswal et al. also perform the quantitative comparison between CapsuleGAN and convolutional GAN using GAM metric. In this metric, two GAN networks are given i.e. $M_1 = (G_1, D_1)$ and $M_2 = (G_2, D_2)$, then G_1 and G_2 are tested against D_2 and D_1 respectively. The ratios of the error of classification are calculated on real test dataset and also on generated samples. CapsuleGAN performs better than convolutional GAN on MNIST and worse than convolutional GAN on real test dataset of CIFAR-10.

CapsuleGAN and convolutional GAN are also compared in semi-supervised classification. To this end, 50,000 images are generated using GAN and CapsuleGAN.The Label Spreading algorithm is used with the generated images as the unlabeled examples and 100, 1000 and 10000 real labeled samples are used. The results show that CapsuleGAN performs better than GAN for all values of n and for both MNIST and CIFAR-10 datasets. So CapsuleGAN generated more realistic images than GAN and performs a better semi-supervised classification.

There are several works that come up with a different approach for routing the capsules between different capsule layers. For example, Hinton et al. [34] propose a new structure for CapsNet called EM-CapsNet which has a different routing algorithm. This variant of CapsNet consists of matrix capsules. Matrix capsules capture activations as well as a 4x4 matrix including pose information. The pose matrix defines the change of viewpoint of an object i.e. the translation and the rotation. Dynamic routing algorithm used in CapsNet is replaced with Expectation Maximization (EM) routing in EM-CapsNet. In EM routing, the part-whole relationship between capsules of the previous layer and the capsules of the next layer is obtained using the clustering method of Expectation Maximization. EM is used to group datapoints into Gaussian distributions.

The pose matrix is 4x4. So there would be 16 Gaussian distributions i.e. 16

mean values and 16 standard deviations. The activation of the capsules in the next layer (parent capsules) and their pose matrices are computed using the iterative EM routing algorithm. EM algorithm fits the capsules of the previous layer (datapoints) into a Gaussian Mixture Model (GMM). EM algorithm consists iterating two stages: E-step and M-step. Each datapoint has an assignment probability to a parent capsule. These probabilities are calculated in the E-step. The M-step updates the parameters of the Gaussian distributions based on the probabilities obtained in the E-step.

Huang et al. [35] propose Dual-Attention Capsule Network (DA-CapsNet) which takes advantage of the attention mechanism after the convolutional layers (referred to as Conv-Attention) and after the primary capsules (referred to as Caps-Attention).

Figure 3.10 shows the architecture of DA-CapsNet. Part A is used to extract features and applies the attention mechanism on the extracted features to make a robust representation of data. Afterwards, Part B transforms the primary capsules into AttCaps (capsules with attention mechanism).

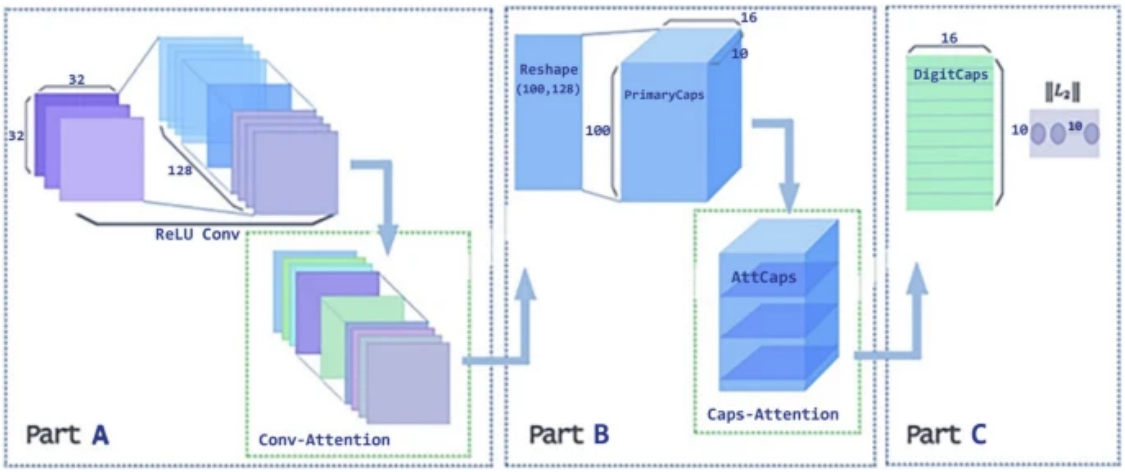


Figure 3.10: DA-CapsNet architecture

Chen et al. [36] propose a deep capsule network combined with a U-Net pre-processing module (DCN-UN) which attempts to improve the poor performance of

CapsNet on complex datasets such as CIFAR-10 and CIFAR-100. A convolutional capsule layer is developed based on local connections and weight-sharing strategies which allows reducing the number of parameters. In addition, this work employs a greedy strategy to develop the Mask Dynamic Routing (MDR) algorithm to further improve the performance of the network.

Ayidzoe et al. [37] introduce a less complex variant of CapsNet with an improved feature extractor. This network takes advantage of a Gabor filter and customized blocks for preprocessing the input data leading to the extraction of the semantic information of the image. This method results enhanced activation diagrams and learns the hierarchical information meaningfully.

Tao et al. [38] present an efficient and flexible network based on capsules referred to as Adaptive Capsule CapsNet (AC-CapsNet). This network replaces the primary capsules with an adaptive capsule layer. The adaptive values contain both the spatial information of each capsule vector and the local relationship among the neurons contained in the capsules.

Some of the related works, do not report the results for the conventional datasets used in the CapsNet domain, or only report some of the numbers. We have included the results of the related works in Chapter 5.

Chapter 4

Methodology

This chapter is dedicated to explaining each of the proposed networks in detail. We start with Quick-CapsNet (QCN) which aimed towards making CapsNet faster by reducing the number of capsules. Afterwards, we explain CFC-CapsNet which introduces the new CFC layer for improving the process of creating capsules. Next, we explain LE-CapsNet which improves the feature extractor by introducing the PCG module. The chapter is concluded with explaining our final work, DL-CapsNet which includes a novel deep architecture with only few number of parameters.

4.1 Quick-CapsNet (QCN)

The number of PCs directly determines the computation complexity. The affine matrix multiplication transform stage and dynamic routing become more computationally expensive as the number of PCs increases. Reducing the number of PCs in the baseline architecture, however, can reduce accuracy. So here our goal is to reduce the number of PCs while maintaining accuracy.

We replace the second convolution layer with a Fully-Connected (FC) layer. Intuitively, this translates to including the contribution of all neurons in the output feature

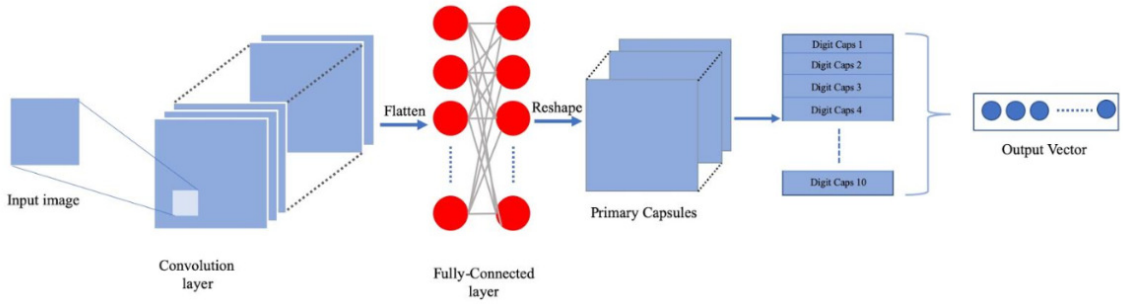


Figure 4.1: QCN architecture.

map of the convolutional layer. FC layer builds a representation that summarizes all of the neurons in the previous layer. Figure 4.1 shows the modified architecture. As presented, we feed the output of the first convolutional layer to an FC layer. The output of the FC layer is reshaped to create PCs, which are the inputs for the dynamic routing algorithm.

In the baseline CapsNet, there are 1152 8D PCs. We chose to minimize the number of outputs of the FC layer for two reasons. The first reason has to do with the intrinsic feature of FC layers. Since the number of parameters in FC layers depends on the number of inputs and outputs. Moreover, the input of the FC layer already has a significantly large number of neurons (being the output of a convolutional layer). Therefore we aim at having as few outputs as possible. This is to avoid creating a heavy network in terms of number of parameters. Secondly, as we presented in the motivation section, the fewer PCs we have, the faster the network becomes. However, there is a minimum number of PCs needed to achieve an acceptable level of accuracy. Based on our experiments, we found this number to be 4 for QCN.

4.2 CFC-CapsNet

We start with explaining the Convolutional Fully-Connected (CFC) layer and how it summarizes the feature map into significantly fewer number of capsules. Afterwards, we explain the other parts of CFC-CapsNet network, i.e. the decoder and the dropout layer.

4.2.1 Convolutional Fully-Connected Layer

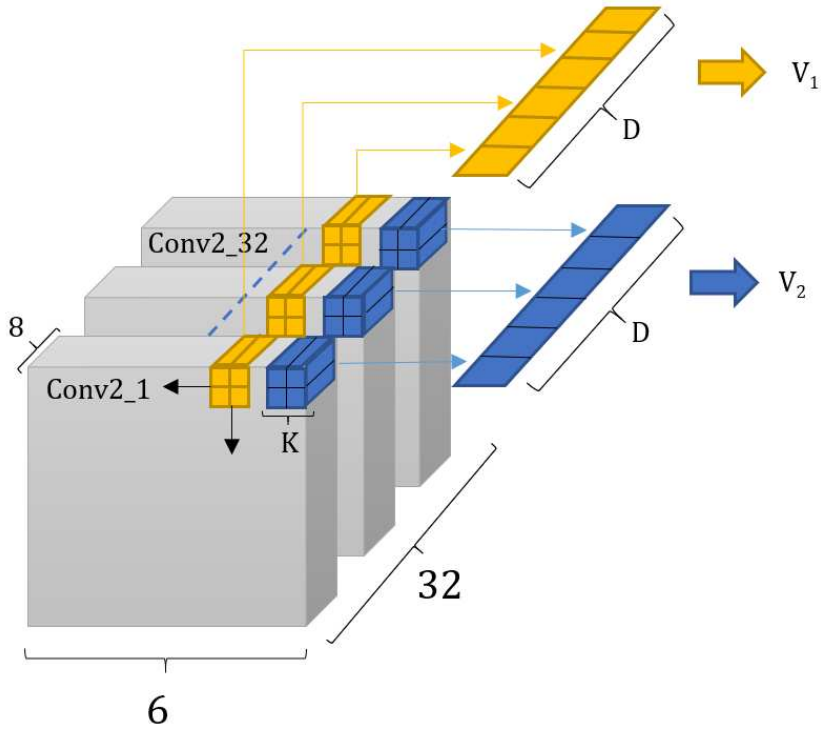


Figure 4.2: CFC Layer architecture

As mentioned in the Background section, in Base-CapsNet the output of the low-level feature extractor is reshaped a high number of vectors. Here we propose a layer referred to as the Convolutional Fully-Connected (CFC) layer to translate the extracted features to vectors. As Figure 4.2 shows, vectors corresponding to spatially correlated regions of the output activation are grouped together. The result is fed to

different FC layers, each of which produces a single vector.

The CFC layer has two parameters: kernel size (K) and the output dimensionality (D). The kernel size determines the size of each vector group to be summarized into a single vector. The higher choice of K, results in fewer generated vectors. The output dimensionality determines the number of elements in each generated output vector. The number of output elements in FC layers determines the output dimensionality.

The CFC layer is essentially the same as a convolutional layer, except for the fact that it does not implement the weight-sharing property. In convolutional layers, there is a single set of weights (the kernel) convolved with the input, whereas in the CFC layer, there are different kernels for each spatially correlated region of the input.

Now we formulate the CFC layer using a mathematical notation. As mentioned before, there is a low-level feature extractor. The output activation of this extractor is used to create the capsules. The activation is split to different chunks (C_m) based on the following equation:

$$C_m = (F_{abc})_{\substack{a\in[w,w+K] \\ b\in[h,h+K] \\ c\in[1,N]}} \tag{4.1}$$

where $m \in [1, (W-K+1)^2]$ and F is the output of the feature extractor, F_{abc} shows the neuron at the c-th feature map of F at the spatial position $x = a$ and $y = b$, N shows the number of feature maps, K is the parameter of the CFC layer and h and w are obtained using the following equations:

$$h = \frac{m}{W-K+1} \tag{4.2}$$

$$w = m\%(W-K+1) \tag{4.3}$$

where the percentage denotes the remainder (mod) operation.

Each chunk is then flattened. We refer to the flattened chunks as $C_F m$. Each flattened chunk is then fed to a fully-connected layer with weights W and the capsules are created based on the following equation:

$$V_m = C_F m \cdot W_m \tag{4.4}$$

where V_m denotes each capsule including D values.

4.2.2 Class-Independent Decoder

We modify the class-independent decoder explained in the Background section by making it deeper and use it as the decoder of CFC-CapsNet. The deconvolutional layers used in this decoder are listed in Table 4.1. It must be noted that for F-MNIST the output vector is projected to a 8x6x6 feature map. For SVHN and CIFAR-10, the output vector is projected to a 8x10x10 feature map to keep the decoder's structure consistent.

Table 4.1: Deconvolutional layers of the decoder

Layer	**Input Dims**	**Kernel**
1	8x6x6	128x3x3
2	128x8x8	64x5x5
3	64x12x12	32x5x5
4	32x16x16	16x5x5
5	16x20x20	16x5x5
6	16x24x24	16x3x3
7	16x26x26	1x3x3

4.2.3 Capsule Dropout

Following the capsule dropout method used in MS-CapsNet [2], CFC-CapsNet includes a capsule dropout method for improving the generalization ability of the network by regularizing the training process. To this end, we randomly drop some of the primary capsules and follow the DR algorithm using the remaining capsules. It is noteworthy that we do not drop the elements inside vectors, as it might change the direction of the vector and affect the representation. Instead, we drop the whole capsule. We have a single experiment for each capsule, whether to drop it or to keep it. Hence, we can use the Bernoulli distribution: the disregarding capsules is considered as a success, and a failure otherwise. In our dropout layer, we implement the inverted dropout method. In other words, if p is the drop probability, the primary capsules are scaled by $\frac{1}{1-p}$ during the training. All the neurons corresponding to these capsules are divided by $1-p$. This layer does not make any change to the capsules during inference.

4.2.4 Hard Training

We use the hard training method used in the code provided by the authors of DeepCaps [39]. In this method, first the network is trained using the default values for m_+ and m_- in the loss function. After the network becomes stable, it is trained using the new values for m_+ and m_-. These new values are tighter than the previous ones: the new m_+ is higher than the previous one, and the new m_- is lower than the previous one. In other words, the threshold for considering the predictions in the loss function is tightened and the training process is repeated. We chose the new values for m_+ and m_- equal to the reference method ($m_+ = 0.95$ and $m_- = 0.05$).

4.3 LE-CapsNet

In figure 4.3 we present the architecture of LE-CapsNet. This network consists of the PCG module to generate the primary capsules, the dynamic routing to infer the output capsules, and a class-independent decoder.

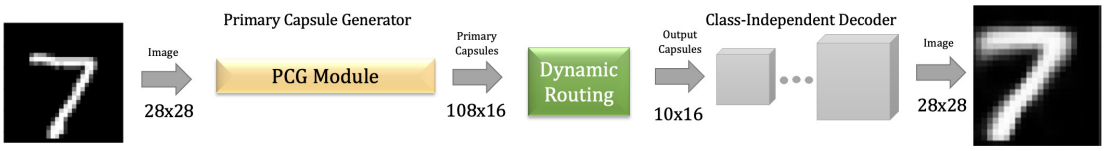


Figure 4.3: LE-CapsNet architecture

4.3.1 PCG Module

LE-CapsNet uses a Primary Capsule Generator (PCG) module to generate the primary capsules from the input image. As the figure shows, compared to CapsNet there is a significant reduction in the number of produced capsules using this module. For the MNIST dataset, the number of PCs is reduced from 1152 to 108 capsules [1]. Figure 4.4 shows the architecture of the PCG module. This module performs three steps. The first step is the conducts low-level feature extraction. In this step, we use a multi-scale sub-network instead of the two convolutional layers used in CapsNet. This sub-network consists of three scales, each with a different depth. The depth of each scale corresponds to the number of convolutional layers employed in that scale. Deeper scales are used to create representations of features of a higher level, whereas shallow scales correspond to low-level features. This helps in creating a better representation of the input images.

[1] The choice of dataset impacts these numbers as feature extraction results in extracting different amount of information (a different feature map size) for different datasets and based on the input image size.

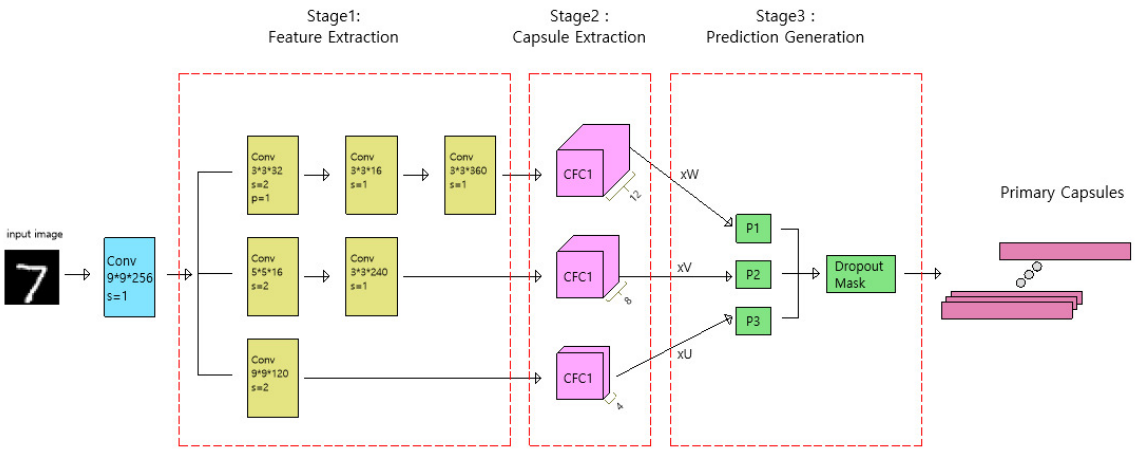


Figure 4.4: PCG module architecture

The second step in taken by the PCG module, and uses multiple CFC layers (one for each scale) to translate the extracted features to vectors. We modify CFC layers by varying the dimensionality of the output based on how deep a specific scale in the feature extraction is. For scales that extract features deeper, the output dimensionality of the CFC layer is chosen to be higher. This provides more outputs and hence better representation. In order to keep important information while summarizing the extracted information, we use more output neurons for deeper scales. For more shallow scales, we create a smaller summary of information, and therefore the output dimensionality is smaller.

The third step is also taken by the PCG module, and includes the affine matrix transformation and the capsule dropout. The output vectors of the second step are multiplied with the weight matrix (the affine transform matrix multiplication stage). The multiplication is done individually for each scale, to encode the spatial relationship between the vectors of each scale separately. Contrary to CapsNet, the generated primary capsules are a mixture of translated low-level, medium-level and high-level features. As such, we expect to create a better representation leading to a higher test accuracy. Note that there are three different sets of primary capsules generated by this method. The capsule dropout is explained in the next section in detail.

4.3.2 Decoder

LE-CapsNet also implements a class-independent decoder. This is the same decoder used in CFC-CapsNet This has two reasons: reducing the number of parameters and providing a more accurate reconstruction. This in turn helps creating a more powerful network in terms of classification accuracy and generalization ability.

4.3.3 Capsule Dropout

LE-CapsNet includes the same dropout technique used in CFC-CapsNet to regularize training the network.

4.4 DL-CapsNet

Figure 4.5 shows the architecture of DL-CapsNet. As the figure shows, the network consists of a convolutional layer, two normal CapsCells, the MLCE module, the DR algorithm and the class-independent decoder.

The network begins with a convolutional layer to extract very low-level features. The extracted features are reshaped to capsules. Afterwards, there are two normal CapsCells to create richer capsules. The output of the second CapsCell is fed to the MLCE module. This module is explained in detail in the next section. The capsules generated by the MLCE module are used to infer the output capsules of the network using the DR algorithm. Similar to CapsNet, the classification is done based on the output capsules. There are K capsules where K is the number of classes in the classification task, and the capsule with the highest length (L2 Norm) corresponds to the predicted class. These capsules are also fed to the decoder network.

4.4.1 Capsule Cell

Capsule Cells (CapsCells) were introduced by Rajasegaran et al. [19] as units including a combination of several Convolutional Capsule (ConvCaps) layers and a skip connection. The ConvCaps layer is a convolutional layer with its outputs reshaped to capsules. Figure 4.7 shows a CapsCell, which includes three ConvCaps layers. The output of the first layer is skip connected to the output of the last layer. This is done to avoid the problem of vanishing gradients. In addition, the skip-connection

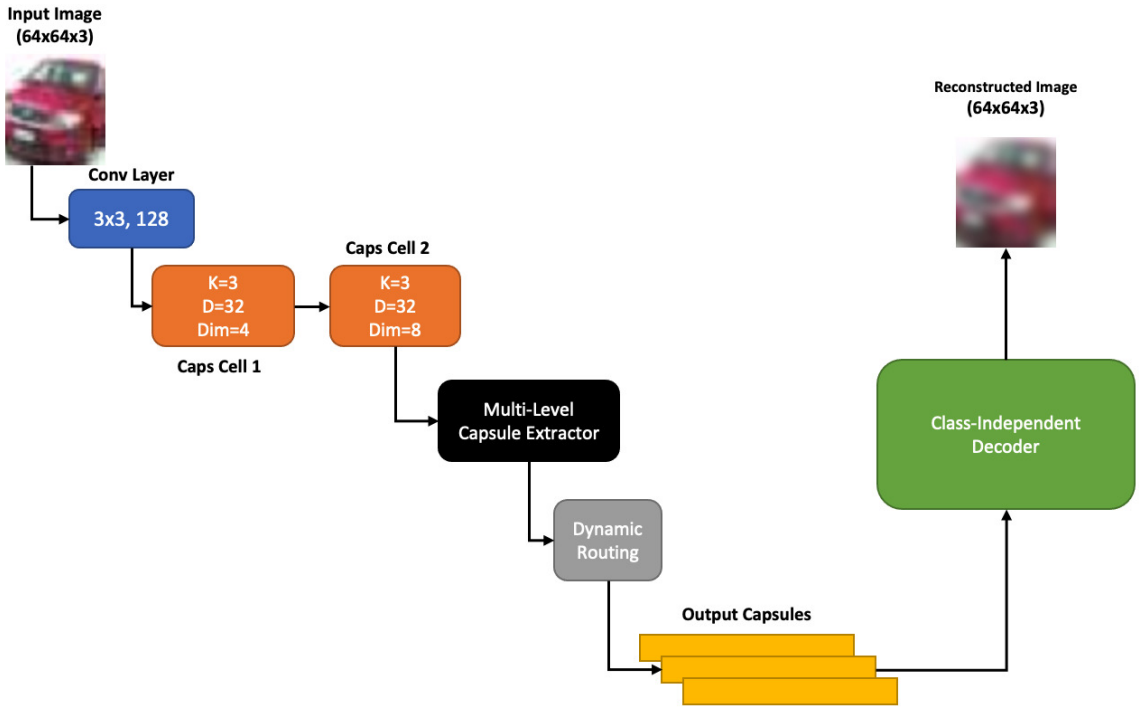


Figure 4.5: DL-CapsNet architecture.

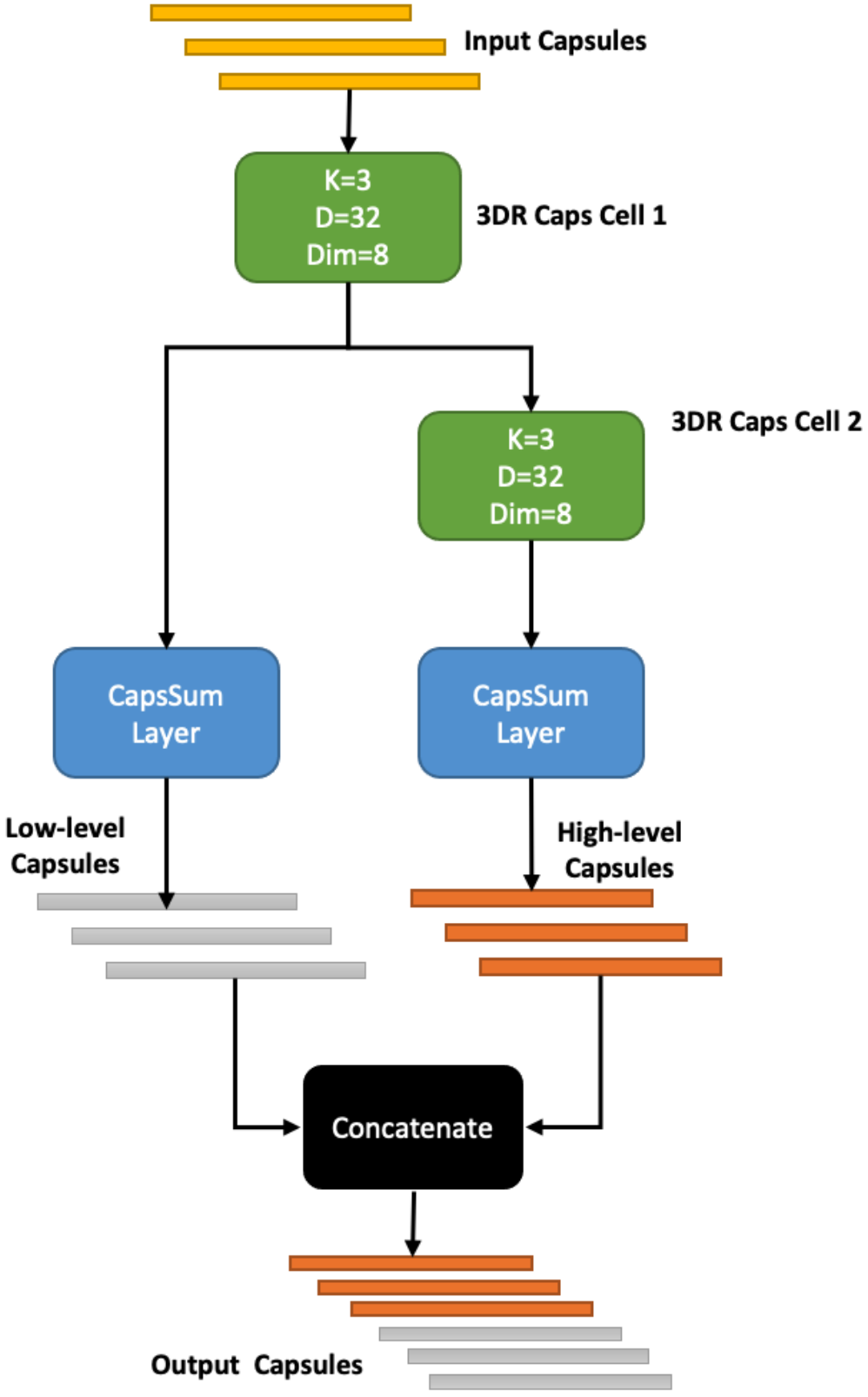


Figure 4.6: MLCE module architecture

helps route the low-level capsules to high-level capsules. The skip-connection either includes a ConvCaps layer, or implements the 3D dynamic routing (3DR) algorithm. The former is called a normal CapsCell and the latter is called a 3DR CapsCell.

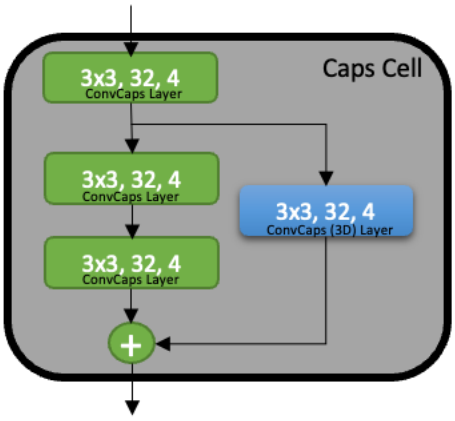


Figure 4.7: CapsCell Architecture

Each ConvCaps layer has 3 parameters: K or the kernel size, D or the number of values for each output vector (the dimensionality), and N_v or the number of vectors per spatial location of the output feature map.

4.4.2 Capsule Summarization (CapsSum) layer

In this work, we propose the CapsSum layer as a means to reduce the number of generated capsules into a few capsules. The CapsSum layer is inspired by the Convolutional Fully-Connected (CFC) [6]. As mentioned earlier, CapsNet includes a simple feature extractor including two convolutional layers. Originally, the extracted data is directly reshaped to capsules. Alternatively, the CFC layer was proposed for translating the low-level extracted features to fewer capsules, resulting in parameter reduction and network speed-up.

We customize and integrate the CFC layer in the proposed network. We summarize the generated capsules by the DeepCaps network to produce a new set of PCs.

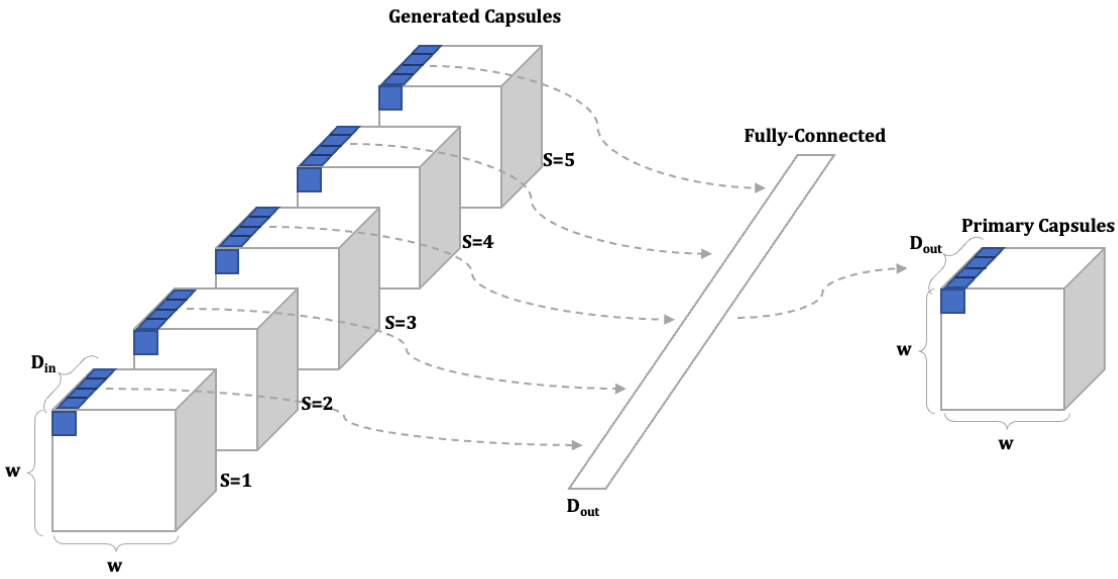


Figure 4.8: Capsule Summarization Module

To this end, we introduce a capsule summarization layer. Figure 4.8 shows how this layer works. As the figure shows, the nearby generated capsules are all flattened, and fed to a fully-connected layer to produce a single capsule. This procedure is repeated for all spatial locations in the generated capsules. There are a total of the $S \times w \times w$ generated capsules each with D_{in} elements. For each spatial location (i, j) where $i, j \in [1, w]$, a total of S capsules with D_{in} dimensionality are collected, flattened and fed to a single fully-connected layer to produce a single capsule with a different dimensionality D_{out}. It is noteworthy that the figure shows the procedure for the first spatial location (1,1). There are $w \times w$ fully-connected layers to summarize the whole generated capsules to the PCS. The proposed layer, reduces the number of capsules S times.

Intuitively, each output capsule corresponds to a set of nearby capsules. We reduce all those correlated nearby capsules to a single capsule using a fully-connected layer. The reduction process includes trainable parameters (contained in the fully-connected layer). However, the reduction in the number of capsules outnumbers the increase in the parameters.

4.4.3 Multi-Level Capsule Extractor (MLCE) module

We carefully designed the MLCE module with two goals. First, to create a rich and robust representation of the input images using the extracted capsules, and second, to ensure the network is small enough (in terms of the number of parameters).

To meet the first goal, we stack two 3DR CapsCells. 3DR algorithm is not as computationally expensive as the DR algorithm due to performing the routing in a localized manner. As a result, it is possible to stack two 3DR CapsCells to make a deep representation of data.

To meet the second goal, we use the CapsSum layer. The number of PCs used in

capsule networks is a key factor in determining the network's number of parameters and also the network training and testing speed. The primary capsules are multiplied by weight matrices each of which corresponds to one of the categories in the classification task. Therefore, reducing the number of PCs results in fewer weight matrices and as a result fewer overall number of parameters. In addition, the more primary capsules there are in the network, the more computationally expensive the DR algorithm would become. The more primary capsules there are, the more time it takes for the DR algorithm used to infer the output capsules from input capsules We use two instances of the CapsSum layer inside the MLCE module to reduce the number of generated capsules number of PCs.

The architecture of the MLCE module is depicted in figure 4.6. As shown in the figure, the MLCE module infers two sets (levels) of capsules from the input capsules, and concatenates them to generate a combination of low-level and high-level capsules. This module stacks two 3DR CapsCells. First, the input capsules are fed to a 3DR CapsCell. The first set of output capsules (low-level capsules) are formed by using a CapsSum layer on top of this 3DR CapsCell. Afterwards, there is another 3DR CapsCell, and another CapsSum layer is used to summarize the high-level capsules into a few capsules (high-level capsules).

The 3DR CapsCells create a robust part-to-whole representation of data by localizing the routing process. Stacking two of these layers, results in creating a deep representation of data. In addition, combining the output of the first and second CapsCells ensures that the created capsules include both low-level and high-level information which results in an increased generalization ability of the network. On the other hand, the CapsSum layer improves the representation, reduces the number of parameters, and enhances network speed. We use a CapsSum layer after each 3DR CapsCell. The combination of 3DR routing and the CapsSum layer in two levels,

enables the network to form a multi-level representation of the input image.

Chapter 5

Evaluation and Results

In this chapter, we provide and elaborate on the results of our works, and we compare the proposed architectures to the Base-CapsNet. The chapter begins with explaining the datasets that we use for evaluating networks. Afterwards, we describe the experiment settings, and finally we provide the results for the different networks. The network test accuracy and number of parameters are reported. To estimate the energy efficiency [1], the inference time of the networks is also reported. All codes are provided in Github [2345].

5.1 Datasets

In general, CapsNet does not perform well on large-scale datasets with a high number of categories such as ImageNet. This is due to the structure of capsules-based networks. These networks obtain a tremendously high number of parameters and also be-

[1]Complex operations such as Softmax which include exponentiation, make it difficult to estimate the total number of FLoating-Point Operations FLOPs in the network

[2]QCN: github.com/pouyashiri/Quick-CapsNet.git

[3]CFC-CapsNet: github.com/pouyashiri/CFC-CapsNet.git

[4]LE-CapsNet: github.com/pouyashiri/LECapsNet.git

[5]DL-CapsNet: github.com/pouyashiri/DL-CapsNet.git

come significantly slow as the number of categories and the image size increases. Here we test QCN, CFC-CapsNet and LE-CapsNet on certain target datasets, i.e., Fashion-MNIST (F-MNIST) [40], SVHN [41], CIFAR-10 and CIFAR-100 [42] datasets. We also test all the networks on the affNIST [6] dataset to check how robust our solutions are against applying affine transformations. Table 5.1 lists the datasets used for evaluating the performance of our proposed network.

Table 5.1: Dataset specifications

Name	**Image Size**	**#Channels**	**Training samples**	**Test Samples**
F-MNIST	28x28	1	60,000	10,000
SVHN	32x32	3	73,257	26,032
CIFAR-10	32x32	3	50,000	10,000
CIFAR-100	32x32	3	50,000	10,000
affNIST	40x40	1	60,000	320,000

5.1.1 Fashion-MNIST (F-MNIST)

Fashion-MNIST (F-MNIST) [40] is a dataset containing 28x28 grey-scale images of 10 different types of clothing. Some sample images of this dataset are shown in Figure 5.1. This dataset follows the same format as the MNIST dataset [7] [43], however, the images are more complex in the F-MNIST dataset. There are 50,000 and 10,000 images in the training and testing sets of this dataset, respectively.

[6] https://www.cs.toronto.edu/ tijmen/affNIST/

[7] Some networks based on CapsNet keep on reporting results for MNIST. Since Base-CapsNet obtains 99.61% test accuracy, improving further in this regards is trivial. For other metrics, F-MNIST and MNIST share the same numbers because they share the same data structure.

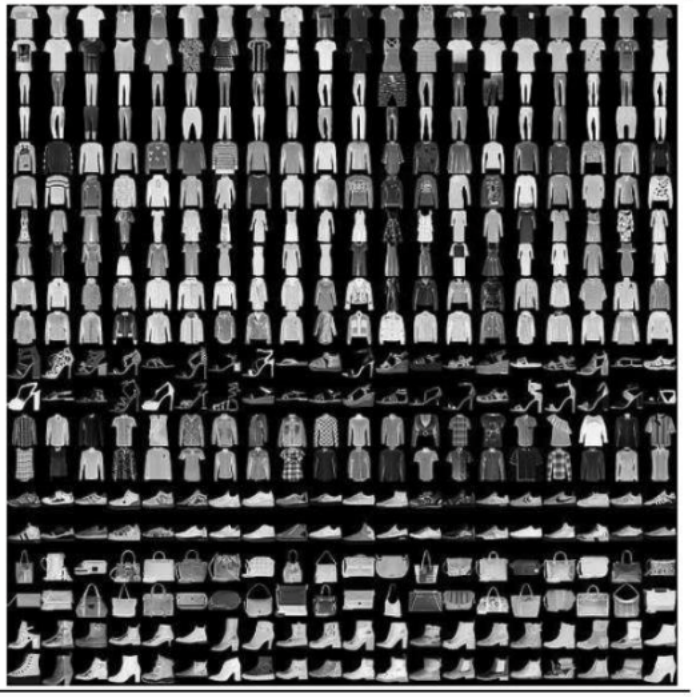
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 5.1: Fashion MNIST dataset [3]

5.1.2 SVHN

Street View House Numbers (SVHN) [41] is a dataset obtained from house numbers using the Google Street View images. The images are small cropped digits and the dataset includes more than 600,000 RGB images which are 32x32 in size. The most complex samples are separated as the training and test set as a total of 73,257 and 26,032 images respectively. This dataset is designed to help solve a real world problem i.e. to recognize the digits and numbers in natural scene images. Sample images from the dataset are shown in Figure 5.2.

Figure 5.2: SVHN dataset [8]

5.1.3 CIFAR-10 and CIFAR-100

CIFAR-10 dataset includes 32x32 coloured images of 10 different categories including airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The training and test sets include 50,000 and 10,000 images respectively. Figure 5.3 shows sample images from this dataset.

CIFAR-100 dataset is similar to CIFAR-10 in terms of number of images and image dimensions. However, there are 100 categories. These categories are grouped into 20 super-classes. Each image has two labels, one corresponding to the super-class

[8]http://ufldl.stanford.edu/housenumbers/
[8]https://www.cs.toronto.edu/ kriz/cifar.html

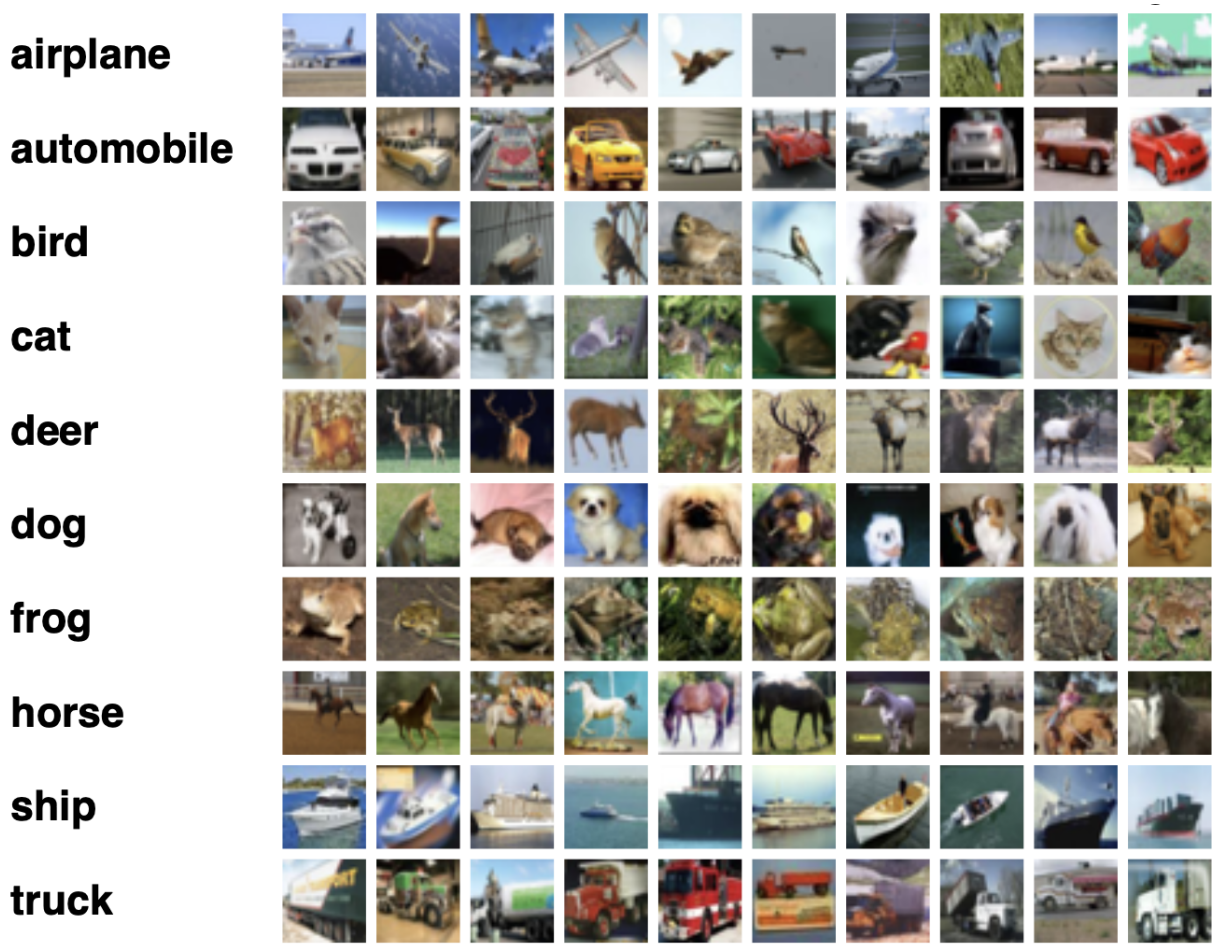


Figure 5.3: CIFAR-10 dataset [9]

and one corresponding to the actual class.

It is noteworthy that testing CIFAR-100 is only possible in DL-CapsNet since the application of MLCE module makes this network scalable. Following [39], for SVHN, CIFAR-100 and CIFAR-10 datasets, the input images are resized from $32 \times 32 \times 3$ to $64 \times 64 \times 3$ using bicubic re-sampling, and for the rest of datasets, the original images are used throughout the experiment. We do the resizing because the images in these datasets include richer features compared to the F-MNIST dataset.

5.1.4 affNIST

This dataset is created using the MNIST dataset, such that various reasonable affine transformations are applied to each sample of this dataset. Figure 5.4 shows some samples of this dataset. In this process, the 28x28 samples become 40x40 pixels. There are 32 different transformations applied to the test set of MNIST. Therefore, the test set contains 320,000 samples. The training set contains every training samples of MNIST randomly placed on a 40x40 grid.

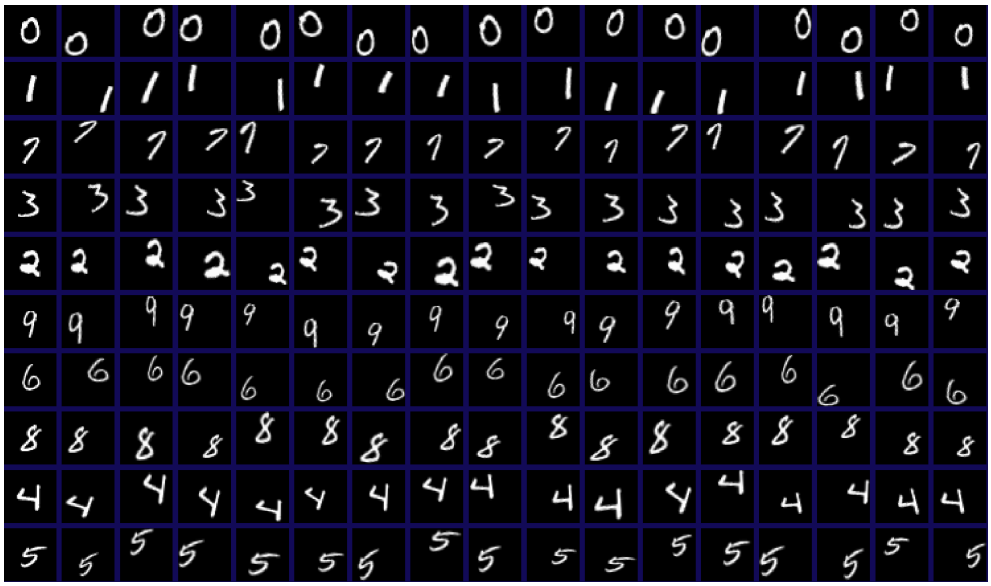


Figure 5.4: affNIST dataset [10]

5.2 Experiment Settings

We use a NVIDIA 2080Ti GPU to run the training sessions. The initial round of training is run for 100 epochs. All the experiments perform a hard training session after the initial round of training for another 100 epochs, unless stated otherwise. All network weights are initialized randomly. Experiments are repeated five times, and we report the average values. We perform a statistical analysis for the best results

of each network, to check the reliability of results in terms of the network accuracy. Choosing the learning rate properly is very important, since choosing a large value results in overshooting, and smaller values lead to slow convergence. We use the Adam optimizer with the default learning rate ($LR = 0.001$). Following [2], we also use an exponential decay of $\gamma = 0.96$ for the learning rate to gradually decrease the learning rate to avoid getting stuck. The batch size is set to 128.

5.3 General Results

In this section, we provide the network accuracy, network speed (training and test time), and the number of parameters of the networks we proposed.

5.3.1 Network Accuracy

The networks based on CapsNet could be divided into those with a shallow feature extractor and a deep one. Three of our solutions i.e. QCN, CFC-CapsNet and LE-CapsNet, all include a shallow feature extractor, whereas DL-CapsNet and networks such as DeepCaps includes a multi-layer deep feature extractor.

Table 5.2 shows the network inference accuracy for the networks with a shallow feature extractor. Our solutions and the best numbers are shown in bold, and the baseline is shown in the first row. LE-CapsNet obtains the best results on CIFAR-10 and FMNIST. QCN obtains slightly lower accuracy on all datasets, as the focus of this work has been solely to minimize the number of capsules.

Table 5.3 includes the network inference accuracy for the networks with a deep feature extractor. Our solutions and the best numbers are shown in bold, and the baseline is shown in the first row. DL-CapsNet obtains the best results on the CIFAR-100 dataset, and a very competitive accuracy on other datasets. It is noteworthy that

only a few networks based on capsules are capable of handling CIFAR-100.

Despite the significant improvements made to the networks based on capsules, there is still a gap in terms of network accuracy between these networks and the state-of-the-art CNNs, specially for highly complex datasets such as CIFAR-100. Table 5.4 shows the network accuracy for our VGG19 [15], ResNet [44], DenseNet [45], NAT-M3 [46] and MUXNet-m [47]. DL-CapsNet is shown in the first row as the baseline.

Table 5.2: Network Inference Accuracy, Capsule-Based, Shallow Extractors

Model	**CIFAR-10**	**SVHN**	**FMNIST**
CapsNet (PyTorch)	71.69%	92.70%	91.37%
HitNet [48]	73.30%	**94.50%**	92.30%
MS-CapsNet [2]	72.30%	92.68%	92.26%
MLCN [26]	75.18%	-	92.63%
QCN8+ [5]	68.17%	88.70%	89.36%
CFC-CapsNet (K=1, D=1) [6]	72.55%	93.19%	92.46%
LE-CapsNet [8]	**75.46%**	92.72%	**92.83%**

Table 5.3: Network Inference Accuracy, Capsule-Based, Deep Extractors

Model	**CIFAR-100**	**CIFAR-10**	**SVHN**	**FMNIST**
CapsNet (PyTorch)	-	71.69%	92.70%	91.37%
DA-CapsNet [35]	-	85.47%	94.82%	93.98%
DeepCaps (7-ens) [19]	-	**92.74%**	**97.56%**	94.73%
DCN-UN MDR [36]	60.56%	90.42%	-	93.33%
Gabor CapsNet [37]	68.17%	85.24%	-	**94.78%**
AC-CapsNet [38]	66.09%	92.02%	96.86%	-
DL-CapsNet (7-ens)	**68.36%**	91.29%	97.09%	94.72%

With 68.17% accuracy on the CIFAR-10 dataset, QCN with 8 PCs and the optimized decoder (QCN8+) obtains a marginally lower accuracy compare to other models, because it pushes the network in terms of the training and testing time. CFC-CapsNet obtains a competitive accuracy of 72.55% on the CIFAR-10 dataset and it

Table 5.4: Network Inference Accuracy, CNNs vs DL-CapsNet

Model	**CIFAR-100**	**CIFAR-10**
DL-CapsNet (7-ens)	68.36%	91.29%
NAT-M3 [46]	**87.7%**	**98.20%**
MUXNet-m [47]	86.1%	98.0%
ResNet-50 [44]	67.06%	93.05%
VGG-19 [15]	70.03%	93.70%
DenseNet-169 [45]	82.4%	96.40%

is significantly faster than Base-CapsNet (not as fast as QCN8+). With 75.46% test accuracy on the CIFAR-10 dataset, LE-CapsNet obtains the highest accuracy among the networks with shallow feature extractors.

With 68.36% accuracy for the CIFAR-100 dataset, DL-CapsNet obtains the highest accuracy among the deep variants of CapsNet. This network obtains highly competitive accuracy for other datasets as well.

There is a slight variation in the network accuracy between training sessions run on the same network and same configurations for all our proposed network. To demonstrate this, we perform the training session for the best configuration of each network 30 times on the CIFAR-10 dataset and we report the maximum, average and standard deviation of the final network accuracy in table 5.5.

Table 5.5: Statistical analysis for results on CIFAR-10 dataset.

Model	**Average**	**Maximum**	**SD**
QCN8+ [5]	68.38%	69.12%	0.42%
CFC-CapsNet (K=1, D=1) [6]	72.45%	73.41%	0.29%
LE-CapsNet [8]	75.34%	76.73%	0.46%
DL-CapsNet	89.82%	90.67%	0.31%

5.3.2 Capsule Dropout

The capsule dropout layer is tested in all four solutions. Following [2], we have used 0.1 and 0.4 dropout rates. Since there are only few PCs generated in our solutions, it turned out that the application of this layer is only marginally beneficial in only two scenarios. In CFC-CapsNet, using a dropout layer with 0.1 rate on the CIFAR-10 dataset, results in an additional boost the accuracy (73.85% compared to 73.15%).

In LE-CapsNet, the same dropout layer results in improving the test accuracy from 75.75% to 76.73%. For the rest of cases (all other datasets and solutions), the application of the dropout layer degrades the network generalization. This is because there are only few capsules in our solutions, and these capsules cannot be reduced any further. As a result, we do not recommend using the dropout layers for our networks.

5.3.3 Robustness to Affine Transformations

Being robust to applying affine transformation to the input images, is one of the advantages of CapsNet over conventional CNNs. Table 5.6 compares how our solutions perform, compared to the baseline CapsNet, and a traditional convolutional network with Max-Pooling and Dropout (as reported by Sabour et. al. [1]) [11]. Our solutions and the best numbers are in bold, and the baseline is shown in italic. QCN8+ obtains the lowest accuracy on this dataset. The motivation behind QCN has been to explore the possibility of minimizing the number of capsules. CFC-CapsNet improves the accuracy by nearly 4 percent (compared to 77.15% of CapsNet), and LE-CapsNet is slightly less robust to affine transformations, but still slightly higher than baseline. With 96.53%, DL-CapsNet obtains the highest accuracy. The high robustness of this solutions is due to the application of 3DR CapsCells twice.

[11] The numbers reported here are different with the ones in the submitted papers for the solutions. This is because here we have implemented the training set of affNIST dataset ourselves carefully, and our results for CapsNet are very close with that of Sabour et al. For more information, refer to the Github codes provided for the solutions.

Table 5.6: Results for affNIST dataset.

Model	**Accuracy**	**Parameters**
Conventional CNN [1]	66%	-
CapsNet (PyTorch)	77.15%	13.47M
QCN8+ [5]	59.48%	19.05M
CFC-CapsNet (K=1, D=1) [6]	81.03%	8.05M
LE-CapsNet [8]	78.58%	**6.54M**
DL-CapsNet	**96.53%**	9.3M

5.3.4 Inference Time

To compare the network inference times, the time it takes for each of the proposed network to perform the inference on the test set of the CIFAR-10 dataset is measured. Figure 5.5 shows the network inference times. These experiments were run using an RTX 3090 GPU. As the figure shows, QCN8+ obtains the highest speedup. CFC-CapsNet obtains the second place with slightly higher time required to perform the inference. LE-CapsNet requires twice more time compared to QCN8+ because it has more PCs resulting from a multi-stage feature extractor. Finally, with 13.52 seconds, DL-CapsNet takes more than twice longer than the base CapsNet and this is due to the deep sub-network used for extracting features.

The inference times for some state-of-the-art CNNs is also shown in Figure 5.5. It is noteworthy that these networks are designed for large-scale datasets such as ImageNet, and the performance on they are not optimized for low-scale datasets such as CIFAR-10. However, we have included these results to show how our solutions compete with CNNs.

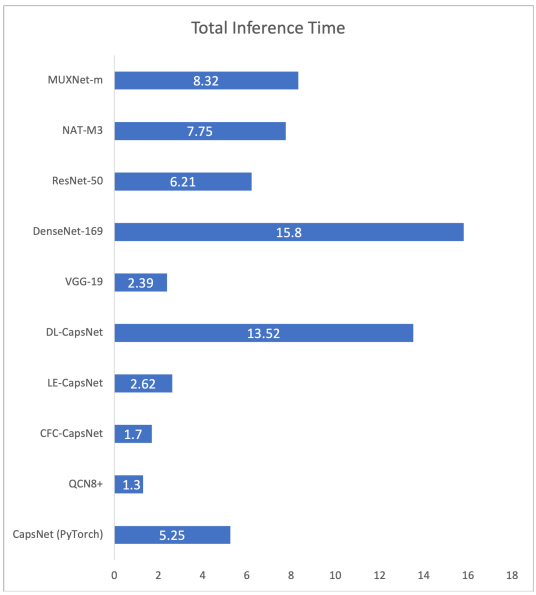


Figure 5.5: Network inference time (in seconds) for different networks on the CIFAR-10 dataset

5.3.5 Number of parameters

Tables 5.7 shows the number of parameters for capsule-based networks with a shallow feature extractor. LE-CapsNet obtains the fewest number of parameters. Table 5.8 compares the number of trainable parameters for capsule-based networks with a deep feature extractors, as well as some of the state-of-the-art CNNs. All the proposed networks reduce the number of parameters compared to the baseline (CapsNet). While there are some capsule-based networks with fewer parameters compared to our proposed solutions, it is noteworthy that our network obtains higher accuracy for those instances.

Table 5.7: Number of Parameters, Capsule-Based, Shallow Extractors

Model	**CIFAR-10**	**FMNIST**
CapsNet [1]	11.7M	8.2M
HitNet [48]	8.9M	8.9M
MS-CapsNet [2]	11.2M	10.8M
QCN8+	9.8M	6.9M
CFC-CapsNet	5.9M	5.7M
LE-CapsNet	**3.8M**	**3.4M**

Table 5.8: Number of Parameters, Capsule-Based with Deep Extractors (top) and CNNs (bottom)

Model	**CIFAR-100**	**CIFAR-10**
CapsNet [1]	11.7M	8.2M
DL-CapsNet	11.2M	6.8M
DeepCaps [19]	-	13.4M
DCN-UN MDR [36]	4.8M	**1.4M**
Gabor CapsNet [37]	22.6M	10.4M
AC-CapsNet [38]	**4.12M**	**1.26M**
MUXNet-m [47]	7.8M	2.08M
ResNet-50 [44]	23.6	23.6M
VGG-19 [15]	38.9M	38.9M
DenseNet-169 [45]	12.6M	12.6M

5.4 Network-Specific Results

In this section we provide the experiments related to analyzing the parameters of a single network and how they affect the performance.

5.4.1 Number of Primary Capsules

To investigate the effect of the number of PCs on different aspects of the network, we changed the number of kernels (the depth) of the second convolutional layer of Base-CapsNet. We performed these experiments without hard-training on the CIFAR-10 dataset. We choose this number as $N_k \in \{32, 64, 128, 192, 256\}$. Each output vector

is 8-dimensional and the size of the feature map is 8x8, so the number of capsules would be $N_{PC} \in \{256, 512, 1024, 1536, 2048\}$.

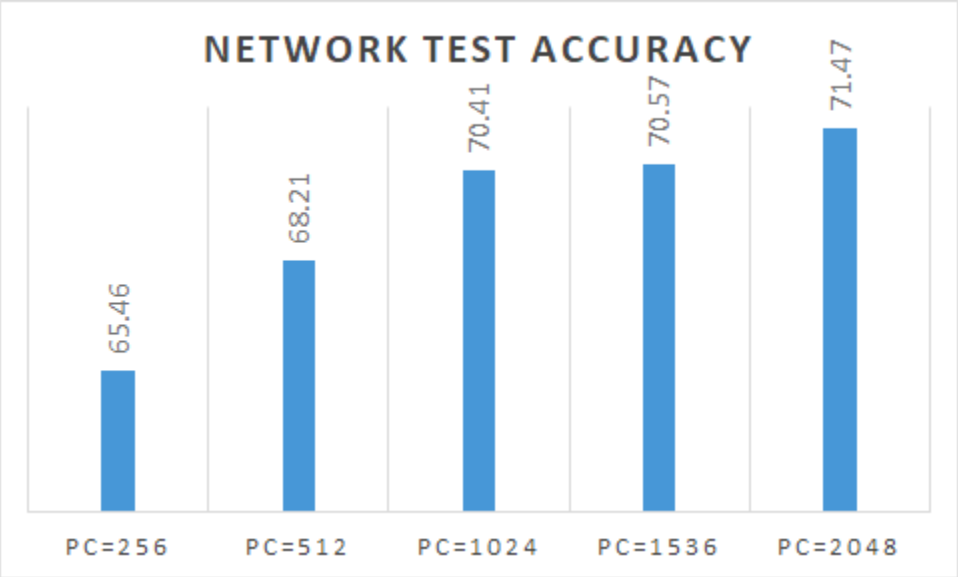


Figure 5.6: CapsNet accuracy for different PC

Figure 5.6 shows how the network test accuracy changes by changing the number of capsules. As the number of PCs gets higher, its effect on the network accuracy diminishes. Comparing the third and the last columns (PC=1024 and PC=2048) shows adding 1024 more capsules, only helps improve the accuracy very marginally.

As Figure 5.7 shows, the number of parameters changes significantly from 11.7M for 2048 PCs down to 4.8M for 256 PCs.

Finally, Figure 5.8 shows how the network training and testing times change by varying the number of primary capsules. As this figure shows, the network is six times faster using 256 PCs compared to the case of 2048 capsules.

Considering the results for the different metrics, it is clear that the number of capsules plays a significant role in CapsNet. While reducing the number of capsules makes the network faster and lighter (fewer number of parameters), it drops the network accuracy and therefore is costly.

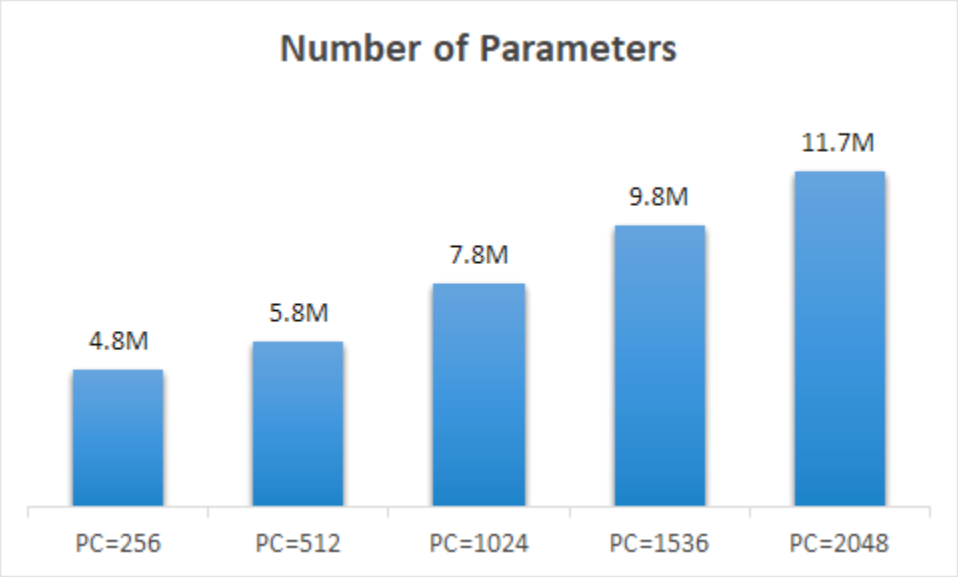


Figure 5.7: Network parameters for different PCs

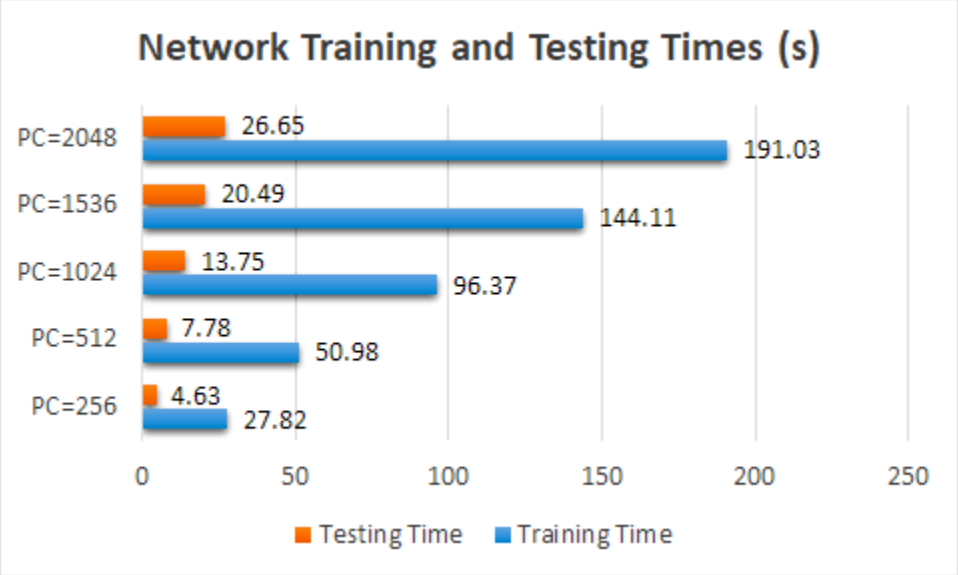


Figure 5.8: Network time for different PC

5.4.2 Number of Capsules in QCN

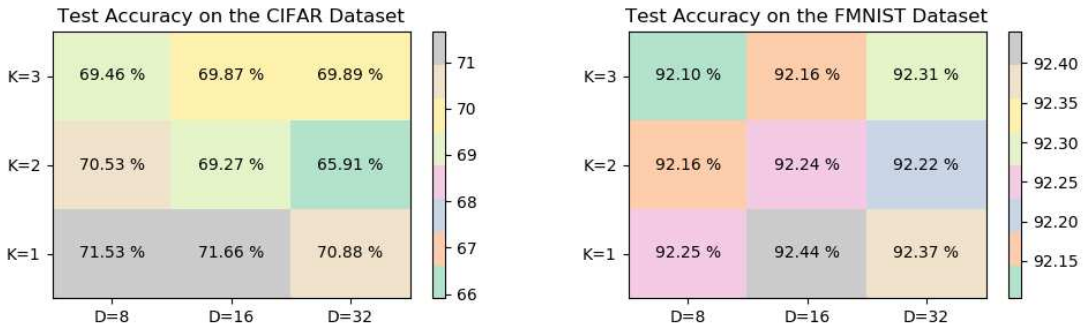
The number of PCs used in QCN affects the network accuracy as well the number of parameters. Tables 5.10 and 5.9 show the number of parameters and the change in the network accuracy for all four datasets. As table 5.10 shows, QCN and QCN+ are lighter networks compared to the baseline, as they come with a smaller number of parameters. Table 5.9 shows that there is loss of accuracy in all cases. This loss is marginal for some cases. With 8 PCs, QCN and QCN+ achieve the highest accuracy. In this case, QCN has 2.6% and 7.2% fewer parameters compared to the baseline for FMNIST and Cifar-10/SVHN, respectively. QCN+ has even fewer number of parameters as it shows 16.5% reduction for all datasets. QCN+ provides a marginally better accuracy compared to QCN for almost all cases. This is the result of employing a powerful decoder. With 8 PCs, QCN and QCN+ lose 1.17% and 1.13% for F-MNIST. The change of decoder impacts F-MNIST less than CIFAR-10 and SVHN. This is because the images in F-MNIST are very simple and employ only one channel. Consequently even the conventional decoder performs well on reconstructing images. This is different for more complex datasets such as Cifar-10 and SVHN. QCN and QCN+ show 4.16% and 1.16% accuracy loss on Cifar-10, and 6.45% and 5.22% loss on SVHN.

Table 5.9: Comparing the accuracy between QCN, QCN+ and the baseline CapsNet. QCN+ achieves higher accuracy.

Dataset	F-MNIST	Cifar-10	SVHN
CapsNet(PyTorch)	89.97	68.33	91.42
QCN (4 PCs)	88.63	63.12	84.55
QCN+ (4 PCs)	88.12	65.55	85.34
QCN (6 PCs)	88.99	64.28	85.51
QCN+ (6 PCs)	88.76	66.12	85.01
QCN 8 PCs)	88.80	64.18	84.97
QCN+ (8 PCs)	88.84	67.18	86.20

Table 5.10: Comparing the number of parameters between QCN, QCN+ and the baseline CapsNet. QCN+ includes fewer number of parameters.

Dataset	F-MNIST	Cifar-10 | SVHN
CapsNet (PyTorch)	8.21M	11.75M
QCN (4 PCs)	4.71M	8.54M
QCN+ (4 PCs)	3.59M	5.09M
QCN (6 PCs)	6.35M	13.26M
QCN+ (6 PCs)	5.23M	7.45M
QCN (8 PCs)	7.99M	10.90M
QCN+ (8 PCs)	6.87M	9.81M



(a) CIFAR-10 Accuracy (b) FMNIST Accuracy

Figure 5.9: CFC-CapsNet accuracy for different K and D

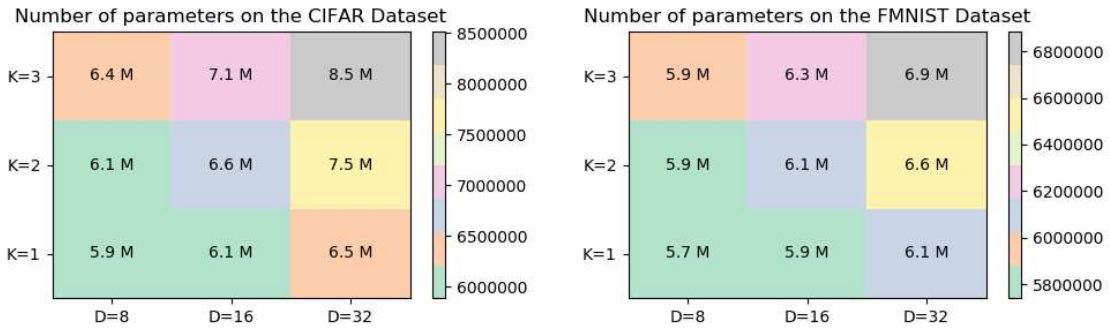
5.4.3 Parameter Exploration for CFC Layer

In this section we investigate using different values for the parameters of the CFC layer: D and K. We choose $D \in \{8, 16, 32\}$ and $K \in \{1, 2, 3\}$. We test the different parameters on the FMNIST and CIFAR-10 datasets and report network accuracy, the number of parameters and the inference time. We skip the SVHN dataset due to its similarity in structure and complexity to the CIFAR-10 dataset.

For the experiments of this section, we do not perform hard training. This is because for each parameter setting, the network is trained for 100 epochs in the normal training session and the training loss is converged. This makes it fair to compare how the different choice of parameters affect the network without using the hard training.

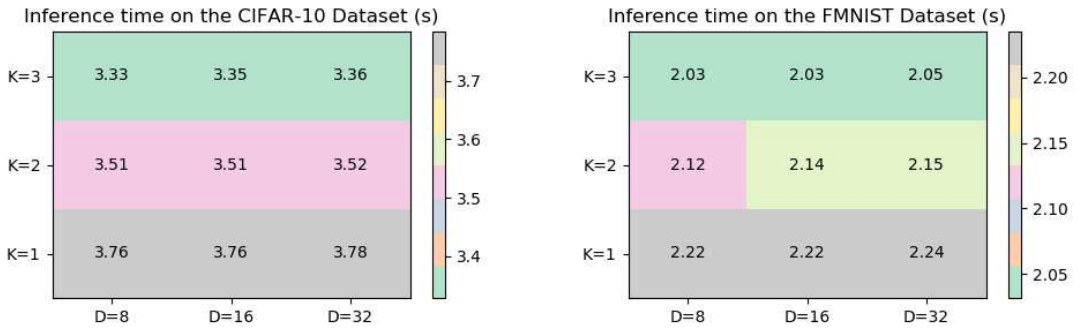
As Figure 5.9 shows, for the FMNIST dataset there is not much difference in accuracy for different choices of K and D. For the CIFAR-10 dataset, $K = 1$ is the best choice. For this choice, both $D = 8$ and $D = 16$ achieve competitive results.

The impact of K and D on the number of parameters is reported in Figure 5.10. For both datasets higher Ks result in a higher number of parameters. This is due to the enlargement of each FC layer as K grows. In addition as D increases, the number of parameters increases as well. The higher dimensionality the capsules have, the



(a) CIFAR-10 number of parameters (b) FMNIST number of parameters

Figure 5.10: CFC-CapsNet parameters for different K and D



(a) CIFAR-10 inference time (b) FMNIST inference time

Figure 5.11: CFC-CapsNet inference time for different K and D

heavier the matrix multiplication process would be.

There is a slight change in the network inference time resulting from changing the values of D and K. Increasing K, leads to a network with fewer number of capsules. Consequently, it takes less time for the DR algorithm and the network becomes faster. However, increasing D slows down the network due to an increase in the complexity of the matrix multiplication process.

Considering the influences of the parameters K and D on the different aspects of the network's performance, we select $K = 1$ and $D = 8$ as the default values for the CFC layer. This choice achieves the minimum number of parameters while providing a competitive accuracy at the cost of a marginally lower speed.

Chapter 6

Conclusion

Sabour et al. introduced Capsule Network (CapsNet) [1] as the new generation of classifiers with several advantages over traditional Convolutional Neural Networks (CNNs). CapsNet is more robust to applying affine transformations to the input images. It also detects images with overlapping categories easier than CNNs. CapsNet offers competitive accuracy showing promising results on small-scale datasets such as MNIST [43] and Fashion-MNIST [40]. On more complex datasets such as CIFAR-10 and CIFAR-100 [42], however, the results are not comparable to state-of-the-art CNN-based networks. Since CapsNet was introduced, there have been several works aiming at facilitating supporting datasets with a high number of categories.

In our first work, QCN, we improved the CapsNet's network speed by modifying the architecture. QCN spends significantly less time in training and testing. QCN comes with a slight drop in the testing accuracy. QCN is equipped with a more more robust decoder compared to Base-CapsNet which has fewer parameters.

Our second work, CFC-CapsNet, enhances CapsNet by integrating a new layer called the CFC layer which is responsible for translating the low-level extracted features into capsules. This approach results in fewer parameters, faster training and

inference and a slight increase in the network test accuracy. CFC-CapsNet achieves 50% fewer parameters, 4x faster inference and 2.16% increase in the network accuracy on the CIFAR-10 dataset.

Our third work, LE-CapsNet, serves as a novel and resource-aware variant of CapsNet. This network includes a specialized feature extractor consisting of multiple scales. It also integrates a capsule translation module to produce capsules out of the extracted features. This implementation achieves a higher accuracy using fewer capsules and significantly fewer weights. On the CIFAR-10 dataset, LE-CapsNet improves accuracy from 71.69% up to 77.21% while reducing the number of weights by 67%. The network is also 4 times faster during the inference.

Our last network DL-CapsNet achieves competitive performance compared to the state-of-the-art CapsNet-based networks. To address the issue of high number of parameters, the network introduces the CapsSum layer to reduce the number of generated capsules. Despite the deep structure of the network, DL-CapsNet is a light network while maintaining a competitive inference accuracy. This is achieved by the MLCE module, as it generates only a few robust capsules. Using a 7-ensemble model, DL-CapsNet achieves a competitive accuracy of 91.29% for the CIFAR-10 dataset using 6.79M parameters. With 68.36% test accuracy for CIFAR-100, DL-CapsNet is among the few variants of CapsNet that performs well on complex datasets with a high number of categories.

Bibliography

[1] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic Routing Between Capsules. (Nips), 2017.

[2] Canqun Xiang, Lu Zhang, Yi Tang, Wenbin Zou, and Chen Xu. MS-CapsNet: A Novel Multi-Scale Capsule Network. *IEEE Signal Processing Letters*, 25(12):1850–1854, dec 2018.

[3] Greeshma K V and J Gripsy. Image classification using hog and lbp feature descriptors with svm and cnn. 03 2020.

[4] Ramin Sharifi, Pouya Shiri, and Amirali Baniasadi. Zero-skipping in capsnet. is it worth it? In Gordon Lee and Ying Jin, editors, *Proceedings of 35th International Conference on Computers and Their Applications*, volume 69 of *EPiC Series in Computing*, pages 355–361. EasyChair, 2020.

[5] Pouya Shiri, Ramin Sharifi, and Amirali Baniasadi. Quick-capsnet (qcn): A fast alternative to capsule networks. In *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–7, 2020.

[6] Pouya Shiri and Amirali Baniasadi. Convolutional fully-connected capsule network (cfc-capsnet). In *Workshop on Design and Architectures for Signal and Image Processing (14th Edition)*, DASIP '21, page 19–25, New York, NY, USA, 2021. Association for Computing Machinery.

[7] Ramin Sharifi, Pouya Shiri, and Amirali Baniasadi. Prunedcaps: A case for primary capsules discrimination. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1437–1442, 2021.

[8] Pouya Shiri and Amirali Baniasadi. Le-capsnet: A light and enhanced capsule network. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1767–1772, 2021.

[9] Pouya Shiri and Amirali Baniasadi. Convolutional fully-connected capsule network (CFC-CapsNet): A novel and fast capsule network. *Journal of Signal Processing Systems*, 94(7):645–658, January 2022.

[10] Pouya Shiri and Amirali Baniasadi. DL-CapsNet: A deep and light capsule network. In *Design and Architecture for Signal and Image Processing*, pages 57–68. Springer International Publishing, 2022.

[11] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-second international joint conference on artificial intelligence*, 2011.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[14] Wei Zhao, Jianbo Ye, Min Yang, Zeyang Lei, Soufei Zhang, and Zhou Zhao. Text Classification. 2017.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1–9. IEEE Computer Society, oct 2015.

[18] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[19] Jathushan Rajasegaran, Vinoj Jayasundara, Sandaru Jayasekara, Hirunima Jayasekara, Suranga Seneviratne, and Ranga Rodrigo. DeepCaps: Going Deeper with Capsule Networks. 2019.

[20] Aryan Mobiny and Hien Van Nguyen. Fast CapsNet for lung cancer screening. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11071 LNCS, pages 741–749. Springer Verlag, 2018.

[21] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. ZeNA: Zero-Aware Neural Network Accelerator. *IEEE Design and Test*, 35(1):39–46, feb 2018.

[22] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6791 LNCS, pages 44–51, 2011.

[23] Aryan Mobiny and Hien Van Nguyen. Fast CapsNet for lung cancer screening. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11071 LNCS:741–749, 2018.

[24] Rodrigo Benenson. Classification datasets results, 2016.

[25] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. Technical report.

[26] Vanderson Martins Do Rosario, Edson Borin, and Mauricio Breternitz. The Multi-Lane Capsule Network. *IEEE Signal Processing Letters*, 26(7):1006–1010, 2019.

[27] XifengGuo. A Keras implementation of CapsNet in NIPS2017 paper "Dynamic Routing Between Capsules". Now test error 0.34%., 2017.

[28] Mohammed Amer and Tomás Maul. Path Capsule Networks. 2019.

[29] Xianli Zou, Shukai Duan, Lidan Wang, and Jin Zhang. Fast convergent capsule network with applications in MNIST. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10878 LNCS, pages 3–10. Springer Verlag, 2018.

[30] Jian-wei Liu, Feng Gao, Run-kun Lu, Yuan-feng Lian, Dian-zhong Wang, Xiong-lin Luo, and Chu-ran Wang. FSC-CapsNet : Fractionally-Strided Convolutional Capsule Network for complex data. *2019 International Joint Conference on Neural Networks (IJCNN)*, (July):1–7, 2019.

[31] Prem Nair, Rohan Doshi, and Stefan Keselj. Pushing the Limits of Capsule Networks. *[Technical Note]*, pages 1–16, 2018.

[32] Adrien Deliège, Anthony Cioppa, and Marc Van Droogenbroeck. Hitnet: a neural network with capsules embedded in a hit-or-miss layer, extended with hybrid data augmentation and ghost capsules. *CoRR*, abs/1806.06519, 2018.

[33] Ayush Jaiswal, Wael Abdalmageed, Yue Wu, and Premkumar Natarajan. CapsuleGAN: Generative Adversarial Capsule Network. (arXiv:1802.06167v5 [stat.ML] UPDATED). pages 1–10.

[34] Geoffrey Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[35] Wenkai Huang and Fobao Zhou. DA-CapsNet: dual attention mechanism capsule network. *Scientific Reports*, 10(1), July 2020.

[36] Junying Chen and Zhan Liu. Mask dynamic routing to combined model of deep capsule network and u-net. *IEEE Transactions on Neural Networks and Learning Systems*, 31(7):2653–2664, 2020.

[37] Mighty Abra Ayidzoe, Yongbin Yu, Patrick Kwabena Mensah, Jingye Cai, Kwabena Adu, and Yifan Tang. Gabor capsule network with preprocessing blocks for the recognition of complex images. *Machine Vision and Applications*, 32(4), June 2021.

[38] Jianwei Tao, Xiankun Zhang, Xuexiong Luo, Yuan Wang, Chen Song, and Yue Sun. Adaptive capsule network. *Computer Vision and Image Understanding*, 218:103405, April 2022.

[39] Jathushan Rajasegaran, Vinoj Jayasundara, Sandaru Jayasekara, Hirunima Jayasekara, Suranga Seneviratne, and Ranga Rodrigo. Deepcaps: Going deeper with capsule networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2019-June, pages 10717–10725, 2019.

[40] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. aug 2017.

[41] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. The Street View House Numbers (SVHN) Dataset, 2011.

[42] A Krizhevsky, V Nair, and G Hinton. CIFAR-10 and CIFAR-100 datasets, 2009.

[43] LECUN and Y. THE MNIST DATABASE of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[45] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[46] Zhichao Lu, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. Neural architecture transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2971–2989, sep 2021.

[47] Zhichao Lu, Kalyanmoy Deb, and Vishnu Naresh Boddeti. Muxconv: Information multiplexing in convolutional neural networks, 2020.

[48] Adrien Deli. HitNet : a neural network with capsules embedded in a Hit-or-Miss layer , extended with hybrid data augmentation and ghost capsules. pages 1–19, 2018.