Improving Capsule Networks using Zero-skipping and Pruning

by

Ramin Sharifi

B.Sc., Iran University of Science and Technology, 2018

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Improving Capsule Networks using Zero-skipping and Pruning

by

Ramin Sharifi

B.Sc., Iran University of Science and Technology, 2018

Supervisory Committee

Dr. Amirali Baniasadi , Co-Supervisor

(Department of Electrical and Computer Engineering)

Dr. T. Aaron Gulliver , Supervisor

(Department of Electrical and Computer Engineering)

ABSTRACT

Capsule Networks are the next generation of image classifiers. Although they have several advantages over conventional Convolutional Neural Networks (CNNs), they remain computationally heavy. Since inference on Capsule Networks is time-consuming, thier usage becomes limited to tasks in which latency is not essential. Approximation methods in Deep Learning help networks lose redundant parameters to increase speed and lower energy consumption.

In the first part of this work, we go through an algorithm called zero-skipping. More than 50% of trained CNNs consist of zeros or values small enough to be considered zero. Since multiplication by zero is a trivial operation, the zero-skipping algorithm can play a massive role in speed increase throughout the network. We investigate the eligibility of Capsule Networks for this algorithm on two different datasets. Our results suggest that Capsule Networks contain enough zeros in their Primary Capsules to benefit from this algorithm.

In the second part of this thesis, we investigate pruning as one of the most popular Neural Network approximation methods. Pruning is the act of finding and removing neurons which have low or no impact on the output. We run experiments on four different datasets. Pruning Capsule Networks results in the loss of redundant Primary Capsules. The results show a significant increase in speed with a minimal drop in accuracy. We also, discuss how dataset complexity affects the pruning strategy.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank Professor Baniasadi for his incredible advisement and support. Also, I would like to thank my parents for their encouragement. This accomplishment would not have been possible without them.

DEDICATION

To my Parents.

Chapter 1

Introduction

Machine Learning (ML) has changed our experience of everyday life. ML technology is the tool that powers Netflix to recommend which movie to watch next (recommender systems), Google to identify spam emails (Classification), Tesla to drive automatically (Reinforcement Learning and object detection) and Apple's Voice assistant Siri to answer our questions (Natural Language Processing).

Deep Learning is defined as "representation-learning methods in which features are processed in multiple stages that every stage is simple but non-linear" [15]. Representation-learning are algorithms that automatically find the representations necessary to detect or classify a pattern in the input data. Deep Learning is profoundly used in different areas because of it's capacity to discover features from complicated input data. There are many challenging tasks for deep learning such as image classification and object detection.

Convolutional Neural Networks (CNN) are a subclass of Deep Neural Networks (DNN). They made a breakthrough in image classification and object classification task when AlexNet was introduced in 2012. Although researchers have gained a lot of success using CNNs, it still has shortcomings. CNNs remove useful information in

pooling layers. The responsibility of this layer is to keep one element per region for better semantic information based on the input feature map. Also, CNNs only check for the presence of features and do not take into account the spatial relationship of features.

The next generation of Neural Networks are Capsule Networks (CapsNets). They have drawn the attention of researchers since the introduction of an algorithm called Dynamic Routing that makes the idea of CapsNets feasible. CapsNets overcome CNNs shortcomings. Spatial relationship of objects are encoded within each capsule. There is no lost information since there are no pooling layers. CapsNets have reached accuracies that have never been reached before on standard ML datasets such as handwritten digits Modified National Institute of Standards and Technology database (MNIST). [16] CapsNets have their own downsides. CapsNets are computationally expensive to train and test.

Since the need for ML applications has increased, and the tasks are more complex, the demand for custom hardware designed for Deep Learning has become a hot topic. Research on algorithms that reduce the computational complexity has become a necessity. These algorithms can speed up the process of training and inference on custom or general hardware. In addition to the speed gain, parameters like memory consumption, off chip access and energy consumption should be kept in mind when designing these algorithms.

One of the common approaches to improve speed and save energy is to skip trivial operations. Trivial operations are the type for which that the output could be known without doing any operation on the input. An example of trivial operations is multiplication by zero. If at least one of the inputs of a multiplication is zero, then the output is a zero.

A hardware implementation of the zero-skipping algorithm is proposed by J. Al-

bercio *et al.* on Convolutional Neural Networks [2]. CNNs check for the presence of features by applying image filters on the input image which generate zeros if a feature is absent. An average of 44% of a CNN layer output is zeros which makes the algorithm worth implementing on custom hardware.

But the question is: whether CapsNets can generate enough zeros so that the zero-skipping algorithm makes a difference. In this thesis, we have explored how CapsNets behave over training on standard Machine Learning datasets and how the number of zeros evolve through training epochs.

Neural Network pruning is another approach to gain speed both in training and inference. Pruning is defined as the process of removing redundant connections in a Deep Learning network. Removing redundant neurons can improve the energy efficiency of the hardware, since fewer computations are needed throughout the network. A pruned network is usually referred to as compressed. The process of identifying the redundant neurons remains a challenge. Parameter salience decides whether to prune a parameter. The success of pruning is relies on the salience estimation. Similarity evaluations and iterative pruning processes share the same idea of pruning at an element-wise granularity. These pruning techniques are also called fine-grain pruning. This pruning can result in uneven weight distributions which can degrade hardware throughput. On the other hand, coarse-grained pruning techniques can find the sweet spot in the compression-throughput trade-off. In this thesis, we use a pruning technique that relies on first-order gradient information for parameter saliency. This technique has been successful in CNNs and showed promising results in CapsNets by removing more than 95% of the parameters with a minuscule drop in accuracy. [27]

Chapter two of this thesis gives the background of this work. We explain Convolutional Neural Networks and how they are trained, activation functions, and Capsule Networks and how they compare to Convolutional Neural Networks. In the last sec-

tion of this chapter, we describe the datasets used in this work.

Chapter three of this work presents zero-skipping, an algorithm that will ignores zero multiplication operations. We investigate if zero-skipping would be beneficial in CapsNets. We fully train CapsNets on handwritten digits MNIST and Fashion MNIST. Histograms are presented to better visualize of Primary Capsules. The results of this chapter have been published in ACM CATA 2020.

In Chapter four, we investigate pruning in Capsule Networks. We provide the background of pruning and how it's application in our work. We compare CapsNets pruning for four different datasets, and in the last section of this chapter, we compare the datasets from a complexity perspective and how they affect Capsule Networks behaviour.

Chapter 2

Background

In this chapter, background information on CNNs and CapsNets is provided, and the pros and cons of each network will be explained. First, in this chapter, an introduction to CNNs and their state-of-the-art classification is be given. Then, the layers utilized in CNN architectures are explicated. Next, CapsNets are introduced. We also discuss how CapsNets differ in terms of information preservation through the network and their advantages over CNNs. This chapter ends with an introducing of the datasets used in the experiments.

2.1 Convolutional Neural Networks

Machine Learning has a pivotal role in many applications such as image classification. Deep learning is a branch of Machine Learning which is gaining lots of attention from researchers due to the complex patterns it can recognize and the advancements made to deploy Deep Learning on hardware [1]. Image classification is the task of assigning a proper label to an image that belongs in a category based on its patterns. This problem can become complex fairly quickly by adding different patterns into an image. Convolutional Neural Networks are the solution offered by deep learning researchers.

The use of CNNs has proven useful as they beat their predecessor classification methods in terms of accuracy. CNNs have achieved high accuracy on standard machine learning datasets such as handwritten digits MNIST, Fashion MNIST, SVHN and CIFAR-10.

2.1.1 Convolutional Neural Networks Training, Validation and Testing

The main task of a CNN is to find meaningful patterns which translate the input image to the ground truth. Datasets are split into three different sets. The largest part of the dataset is used for training. This is called the training set. The next set of images are used to tune hyperparameters, which is called the validation set. Lastly, we have the test set which can provides a vision of how well the model performs. It is important that the model's parameters are not tuned during the testing phase. Training a CNN is an optimization process in which the optimizer is trying to minimize the loss function defined for the model. The loss function is a mathematical function that outputs a score of how well the model performed its predictions. Several different loss functions have been used. A loss function usually consists of two terms. First, the main loss is calculated based on the predictions of the model and the ground truth. The second term in a loss function is the regularizer. The regularizer is used to prevent overfitting and its definition is dependent on the model.

2.1.2 Gradient Propagation

Each layer in a CNN has parameters that need to be set during the training phase which is called weights. Weights are building blocks of every neural network. An image goes through multiple layers in a CNN before a prediction is made. The weights are updated every time a set of images, which are called a batch, have their

labels predicted by the model. Updating the weights is done by gradient propagation usually called back-propagation. Gradients are calculated from the last layer going back to the first layer. These gradients contain the information needed to update the weights in the back-propagation process.

2.1.3 Convolutional Layer

The convolutional layer is the most important part of a CNN. This layer consists of different filters which are applied to its input to generate the output which is called a feature map. Each filter consists of weights that need to be trained in the training phase. Each convolutional layer has another part called bias which is applied independently to the feature map. The size of the output feature map is

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

where O is the size of the feature map, K is the kernel size, W being the size of the input, P is the padding and S is the stride. Padding is the process of adding elements on the edges of the original input to overcome the border effect. This effect is the reduction in the size of the input as a result of the convolution process. The number of weights in each convolutional layer is

$$P = W + B = K^2 \times C \times N + N \tag{2.2}$$

where P is the number of weights, W is the number of kernel weights, and B is the number of the bias. K is the kernel size, C is the number of channels and N is the number of filters.

2.1.4 Fully-Connected (FC) Layer

The fully-Connected(FC) layer will input all elements for each feature map and apply their weights to produce the desired outputs. The output of the FC layers in a CNN architecture is the classification results. Classification results are called model predictions. Number of parameters in an FC layer can be is $(N + 1) \times M$ where N is the number of input elements and M is the output size.

2.1.5 Activation Functions

Activation functions are non-linear mathematical functions which are used in neural networks. The goal of a deep learning model is to approximate the function which generated the inputs. Since these generator functions are non-linear, activation functions can give us the ability to estimate non-linear functions. Some of the most famous activation functions are Rectified Linear Unit (ReLU), Leaky ReLU, Exponential Linear Unit (ELU), Sigmoid and Softmax. In this part, some of these activation functions are explained.

Rectified Linear Unit (ReLU)

This activation is one of the most used activation function in Neural Networks. ReLU can be described as

$$ReLU(x) = \max(0, x) \tag{2.3}$$

Vanishing gradients is a problem which can happen during training of neural networks, when the back-propagated gradient is so small that it cannot update the weights. This problem interferes with the training and brings it to a halt. The ReLU activation function will prevent the vanishing gradient problem. The main

disadvantage of ReLU is not having an upper limit on its output. Leaky ReLU js similar to ReLU but instead of outputting zero for negative inputs, they output a small negative number which translate to a positive gradient. This small positive gives the neurons a chance to recover when not being able to update.

Sigmoid

The Sigmoid activation function outputs values between 0 and 1. It is a well-behaved function. and can be described as

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

Sigmoid has a limited output which is an advantage over ReLU. Still, when its input is small or large, it can result in the vanishing gradient problem. Also, since the function does not output zero for inputs which are zero, the gradient function will move in different directions which makes the optimization difficult.

SoftMax

Softmax is usually chosen for the prediction layer which is the final layer of the CNN. This is because softmax maps the inputs to probabilities so the output probabilities sum to one. Softmax is

$$Softmax(x) = \frac{\exp(x)}{1 + \exp(x)} \tag{2.5}$$

2.1.6 Pooling Layer

Pooling helps with managing the size of feature maps. Since feature maps are stacked for each layer, if pooling layers are not utilized, the system may run out of memory. A pooling layer outputs a single value for each square segment of the feature map. One

of the most popular pooling layers is called max-pooling. Max-pooling is the process taking the largest value for each square segment of the feature map and ignoring the rest of the values in that segment. The downside of pooling layers, in general, is the loss of information.

2.1.7 Regularizers and Dropout

There are several methods used in a network to avoid overfitting in the training phase. Two of these are regularization and dropout. Overfitting happens when the for on the training set differs largely from that of test set. If a model is overfit on the training set, it will perform poorly on the test set. Two of the most important regularizers used in CNNs are weight-loss and reconstruction losses. Regularizers are added to the loss function to prevent overfitting. Weight-loss regularizers will not tolerate large weights in a network, and the model will adapt to this restriction.

The reconstruction loss method compares the reconstructed image to the input. This will forces the network to produce meaningful interpretations of the input image. The dropout method randomly chooses neurons and removes them from the output. Dropout also helps the network to generalize over the dataset. If a neuron is omitted from the output, the model has to utilize other neurons to generalize over the dataset.

2.2 Capsule Network (CapsNet)

A capsule is a group of neurons that are in vector format. The Capsule Network was introduced by Sabour et al. [30], and it's shown in the figure 4.6. The architecture starts with two convolutional layers. These two convolutional layers are feature extractors. The extracted features are reshaped into vectors called Primary Capsules (PC). PCs are then multiplied by a weight matrix. This matrix is called the affine

transform matrix. The robustness of the capsules is due to this matrix. The output of this multiplication is the prediction of the first layer.

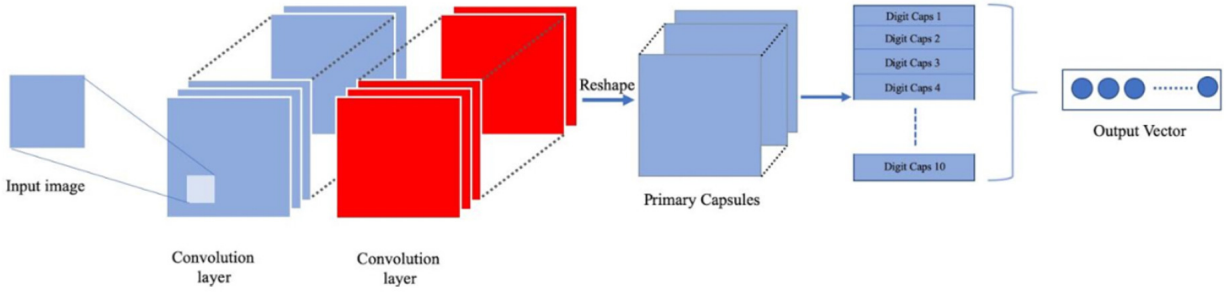


Figure 2.1: CapsNet Architecture

If u_i is the output of a capsule, then $\hat{u}_{j|i}$ will demonstrate the next layer predictions and is given by

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

where W_{ij} is the affine transformation matrix. The output of the current layer goes through the squash function

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

where the c_{ij} are the coefficients which are set by the dynamic routing. Squash is a non-linear activation function with maximum value one and minimum value zero.

A capsule Network outputs N vectors, where N is the number of classes in our classification problem. The length of each vector is the output probability of the network. Each element of the vector is an instantiating parameter of the image. The Squash function will detect small-length vectors and replace them with zero. This function also, helps the largest vector to maintain a length of one. Squash function

is represented below:

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

where s_j is the input and v_j is the output vector.

2.2.1 Dynamic Routing Algorithm

Dynamic routing is an algorithm which helps capsule networks go through a stable training [30]. Dynamic routing determines where the output vectors of the first layer of capsules should be placed in their next layer. Dynamic routing parameters are should be determined for each input. This algorithm checks the degree of agreement between the predictions of the first layer

Algorithm 1 Dynamic Routing Algorithm

1: **procedure** ROUTING($u_{j|i}, l, r$)
2: $b_{ij} \leftarrow 0$ ▷ for all capsules i and j in layers l and $l+1$
3: **for** r iterations **do**
4: $c_i \leftarrow SoftMax(b_i)$ ▷ for all capsule i in layer l
5: $v_j = Squash(\sum_i c_{ij} \hat{u}_{j|i})$ ▷ for all capsule j in layer l+1
6: $b_{ij} = b_{ij} + \hat{u}_{j|i}.v_j$ ▷ for all capsule i and j in layers l and l+1
return v_j

2.2.2 Margin Loss

The length of a capsule reflects the probability that an entity exists in an image. This is a factor that has been taken into consideration in CapsNet loss functions. Margin loss accounts for each capsule predicting the right entity in each image. Higher values of margin loss indicate that the capsules are predicting each category incorrectly. As a result, the margin loss is described as

$$L_k = T_k \max(0, m^+ - ||V_k||)^2 + \lambda(1 - T_k) \max(0, ||V_k|| - m^-)^2 \tag{2.9}$$

where L_k denotes the loss term for each capsule, T_k is determined based on the prediction of each capsule (one for correct predictions and zero otherwise), λ is the weight considered for penalizing wrong predictions, and m^+ and m^- are used to remove capsules with high or low probabilities from taking part in the margin loss.

2.2.3 Regularization by Reconstruction

Each input to the CapsNet is reconstructed using a decoder. Comparing the reconstructed images with the input images adds an extra term to the loss function. Input images are masked before being fed into the decoder section of a CapsNets. In the training phase, all outputs except for the ground truth are set to zero. In the testing phase, only the highest magnitude is kept and the rest of the vectors are masked to zero. The decoder utilized in the original CapsNet is a Fully-Connected layer to the decode the output vectors. Reconstruction loss is refrained from dominating the margin loss.

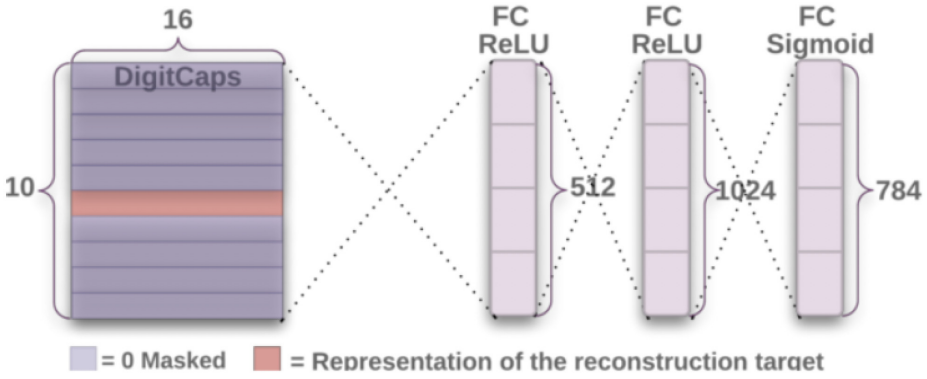


Figure 2.2: CapsNet Decoder [30]

2.2.4 Advantages of CapsNets over CNNs

Convolutional Neural Networks ignore the relationship between low-level and high-level features. Low-level features are edges and blobs. High-level features are whole objects. In the case of face detection, low-level features are lips, noses and eyes and their shapes. CNNs cannot distinguish between a normal face with features in their right spot and a shape that only includes all the features without any particular spatial formation. This means that if the lips and nose were misplaced, CNNs will not understand the difference.

CapsNet provides a vector for each category in which they encode the spatial information. The part-to-whole relationship between the different levels of features is reserved in Capsule Networks in the Dynamic Routing algorithm which determines how much low and high level features are in agreement.

CapsNets are more robust to affine transformations applied to input images. This means that CapsNets can perform better than CNNs with affine transformation even if they are not trained on a particular set of transformations. This was verified in [30] by testing CapsNet and CNN on an affined transformed MNIST dataset. In this experiment, CapsNet was trained on the raw handwritten digits MNIST dataset and tested on the transformed dataset. Capsule Networks also perform better on detecting overlapping images over CNNs classifying images. [30]. The dataset was generated by overlaying an input of one category on top of the other from a different category. The same datasplit was used for both testing and training the CNNs and Capsule Networks.

2.2.5 CapsNet Drawbacks

Capsule Networks are much slower to train than Convolutional Neural Networks. This is due to the multidimensional matrix multiplications and the Dynamic Routing

Algorithm which is an iterative process. Dynamic Routing is a computationally expensive operation which has to be repeated three times for each set of input images. Also, the number of capsules directly influences the computational complexity of the architecture.

Capsule Networks generate K output vectors for K classes which are being predicted. As a result, CapsNets contain too many learnable weights for the classification of datasets with high number of classes, e.g. ImageNet (1000 classes). CapsNets also to perform poorly on visually complex datasets.

2.3 Datasets

In this section, we discuss the datasets used in this work. The focus on the four datasets which are handwritten digits MNIST, Fashion MNIST, SVHN and CIFAR-10.

Handwritten Digits MNIST and Fashion MNIST

Handwritten Digits MNIST and Fashion MNIST datasets are 28×28 grey-scaled images in 10 different classes. The training and testing sets have 50,000 and 10,000 images, respectively. The classes for MNIST are handwritten digits between zero and nine. Fashion MNIST consists of different categories of clothes. Examples of both datasets are shown in Figs. 2.4 and 2.3.

SVHN and CIFAR-10

CIFAR-10 and SVHN consist of 32×32 pixel images in ten different categories. Both classes contain RGB (3 channels) images which are split into 50,000 and 10,000 images for CIFAR-10 and 73,257 and 26,032 samples for SVHN. SVHN contain house

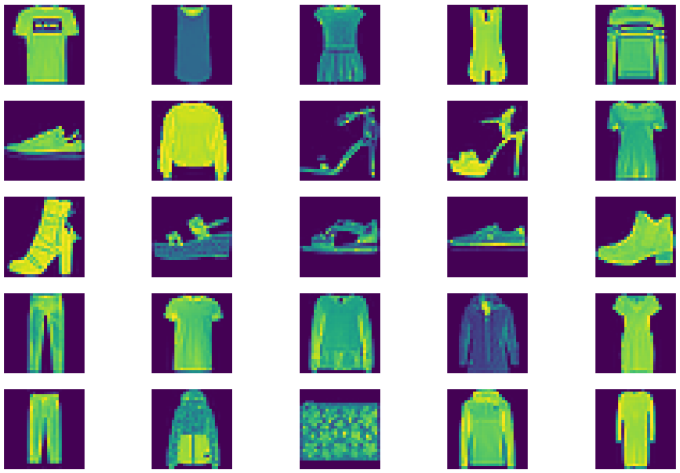


Figure 2.3: Fashion MNIST images

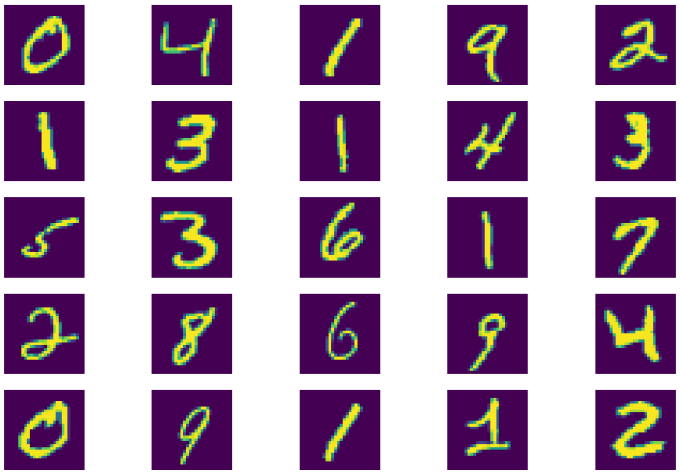


Figure 2.4: Handwritten digits MNIST images

numbers as samples. An example of this dataset is shown in Fig. 2.5 CIFAR-10 contains 10 non-related categories: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Examples of this dataset are shown in Fig. 2.6. The complexity of these datasets is explained in more detail in Chapter 3. The introduction on datasets

Table 2.1: Datasets used in this work and related information

Name	**Image Size**	**# Channels**	**# Training Samples**	**Test Samples**
Handwritten digits MNIST	28×28	1	50,000	10,000
Fashion MNIST	28×28	1	50,000	10,000
SVHN	32×32	3	73,257	26,032
CIFAR-10	32×32	3	50,000	10,000

are summarized into Table 2.1.

Figure 2.5: Samples of the SVHN dataset

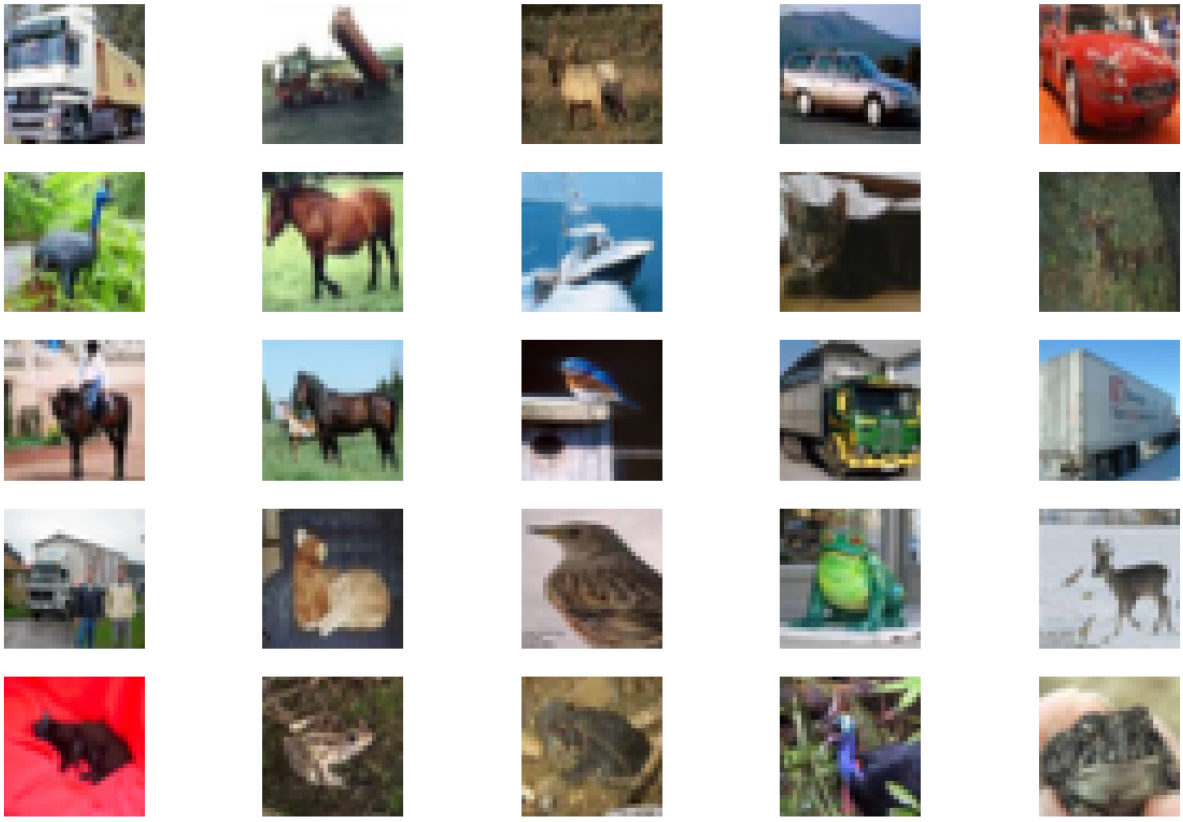


Figure 2.6: Samples of the CIFAR-10 dataset

Chapter 3

Zero-skipping in CapsNet. Is it worth it?

Machine learning methods have a wide range of applications such as Natural Language Processing (NLP), pattern recognition and computational learning. Deep learning structures were introduced in the late 20th century and originated from studying Artificial Neural Networks (ANNs) [21] and has attracted a lot of attention in recent years [8, 10, 37, 39, 40]. A Neural Network is constructed using a network of neurons that produce activations and have weights that needed to be adjusted for the network to learn a specific pattern in data [21]. Deep Neural Networks (DNNs) use a large number of layers to learn features of different depths from the input data.

Convolutional Neural Networks (CNNs) are DNNs that perform effectively when the input data is two-dimensional, e.g. images [21]. These networks include several types of layers. One of these layers is the Pooling layer which is used to create different receptive fields to obtain richer semantic information from the feature maps. This layer reduces the size from the previous layer and so results in some information being lost. Pooling is a convolutional layer and is responsible for keeping some values per

region of the feature map depending on its type.

Capsule Network (CapsNet) is the next-generation neural network and has drawn the attention of researchers in the past few years [13, 24, 29, 35]. The concept "Capsule" was first introduced in [9] but there was no practical implementation until Sabour et al. introduced Dynamic routing between Capsules [30] and presented a practical new network based on capsules called a Capsule Network. A capsule is a set of neurons that provide vectors as output (instead of the scalars used in CNNs), which represent different properties called the instantiation parameters for a type of entity. In contrast to CNNs, this network considers the relationship between low-level and high-level features while performing the classification. This is done by using vectors to represent features and applying an iterative algorithm referred to as Routing by Agreement or Dynamic Routing, to find the relationship between vectors of one layer and the output vectors of the next one.

Trivial operations are those operations where the output can be calculated without performing the operation. Examples include multiplication by or addition with zero. Skipping trivial operations results in networks requiring less memory and performing faster training and testing. This consequently results in saving time and energy. Trivial operations in neural networks were considered in [20, 12, 2]. Here we focus on zero-skipping and identifying and skipping redundant multiplications by zero. The CapsNet architecture includes two convolutional layers, a weight matrix multiplication and the dynamic routing algorithm to generate the output vectors whose length determines the type of an entity. These vectors are fed to 3 Fully-Connected (FC) layers to reconstruct the input image. This reconstruction is used as a regularizer for the training process.

In this work, we investigate the maximum savings achievable by removing trivial operations in CapsNet by analyzing the fraction of zeros present in the CapsNet

predictions. These are the zeros generated by the primary capsules on the validation set for different epochs. This investigation serves as a first step in estimating the potential benefits of applying zero-skipping techniques on CapsNet. All experiments are done on MNIST handwritten digits [16] and Fashion-MNIST [36] datasets.

3.1 Related Work

Zero-skipping and power aware neural networks have attracted significant research attention in recent years. In [2], Albercio et al showed that on average 44% of Convolutional Neural Network multiplications produce zero. Their architecture is a modified version of the DaDianNao architecture [23]. Albercio et al [2] introduced CNVLUTIN as a modified architecture to skip multiplications by zero in Convolutional Neural Networks. CNVLUTIN improved the performance on average by 34%. CNVLUTIN's best and worst results were on CNNS with 55% and GoogleNet with 24% performance improvement, respectively. They introduced a zero-free neuron array format which only contains non-zero value neurons to avoid zero multiplication. In order to keep all lanes busy, they use a dispatcher to store neural values to fill empty lanes. CNVLUTIN decouples the original architecture lanes so each lane can perform zero skipping.

In ZeNA [12], Kim et. al were the first to skip trivial computations (also referred to as ineffectual operations), caused by weights and activations. They proposed zero-aware kernel allocation and dynamic work group. They reported a 4.4 times speed up for AlexNet over fixed-point Iris which they used as a baseline. Kim et al [12] present ZeNA as a highly scalable solution. In their work, zero-aware kernel allocation sorts the kernel values first and then allocates them to processing elements. By this method, the load is distribute the load more uniformly. Dynamic work group deals

with inter-workgroup load imbalance.

In PredictiveNet [20], Lin et.al proposed a predictive convolutional neural network which can predict the output of convolutional layers and skip over trivial (ineffectual) multiplications. They showed that PredictiveNet can decrease the computational cost by a factor of 2.9x for the MNIST handwritten digits dataset. The proposed technique is supported by mean squared error perseverance. They reported a marginal accuracy loss with the reduced computational cost.

3.2 Background

A capsule is a set of neurons that provide vectors as outputs (instead of scalars in CNN) which represent different properties called the instantiation parameters for a type of entity [30]. The length of these vectors reflects the probability of an entity's existence. There are multiple layers of capsules (here two layers). Lower-level capsules send their predictions using transformation matrices to the higher-level capsules. This is done via the routing-by-agreement algorithm. The activation of a higher-level capsule depends on the agreement of predictions. The structure proposed in [30] achieves state-of-the-art accuracy on MNIST dataset. It also has several advantages over conventional CNNs. CapsNet is better than a CNN in recognizing highly overlapping datasets. It is also more robust to affine transformations applied to the input data.

The MNIST dataset of handwritten digits is one of the most popular datasets in machine learning applications. It has 50,000 images for training and 10000 images for testing. Each image is 28 by 28 pixels and depicts a digit between 0 and 9. In [30], other datasets including Fashion MNIST. Fashion MNIST is an image dataset which has the same dataset size, number of classes and image size as the MNIST

dataset, but it contains images of different types of clothing. Fashion-MNIST serves as a replacement for the original MNIST dataset for benchmarking.

The architecture of the CapsNet includes two convolutional layers. The output of the second convolutional layer is reshaped to 8-dimensional vectors which are called Primary Capsules. Next, a weight matrix is multiplied by these vectors to create a new set of vectors. The analysis on skipping zeros is performed on these vectors. The next layer consists of 10 vectors each with 16 dimensions (DigitCaps). Here, the Dynamic-Routing algorithm at find an agreement between the DigitCaps and the vectors from the previous layer.

The work in [2] includes investigations on pruning. Pruning relies on setting a threshold on values of neurons. In [2] the threshold values are determined for each layer of network separately. For simplicity, they have just used powers of two as threshold values. Any value below the threshold will be truncated to zero. CNVLUTIN architecture performs pruning dynamically. In CNVLUTIN, pruning has no effects on accuracy up to a certain threshold. After that, accuracy starts to decline. However, since many values are set to zero and consequently skipped, overall, we gain in speedup.

In this work we analyze the vector values during the training process. For training, the dataset is divided into training, validation and testing sets. The validation set is used to verify the progress of the network. Our analysis is done on the validation set. To optimize a neural network, a dataset is divided into fixed-size batches and these batches are fed to an optimizer. The procedure of applying the whole dataset to the neural network is called an epoch. We store all values over different iterations in each epoch and inspect them to determine the percentage of data that is zero.

3.3 Experiments and Results

We analyze the prediction values from the primary capsules while training the CapsNet on two different datasets, i.e. MNIST and Fashion-MNIST. The predictions of the primary capsules are in fact the output of the secondary feature map reshaped into 8-dimensional vectors and then multiplied by the weight matrix. We save all the tensor values over all validation iterations and then analyze and investigate the results from different perspectives. There is a total of $T \times B \times 1152 \times 10 \times 16$ values where T is number of iterations per epoch and B is batch size. B is 64 for both datasets and T is 78 and 156 for MNIST and F-MNIST, respectively. This gives a total of 920,125,440 and 1,840,250,880 values per epoch for MNIST and F-MNIST, respectively.

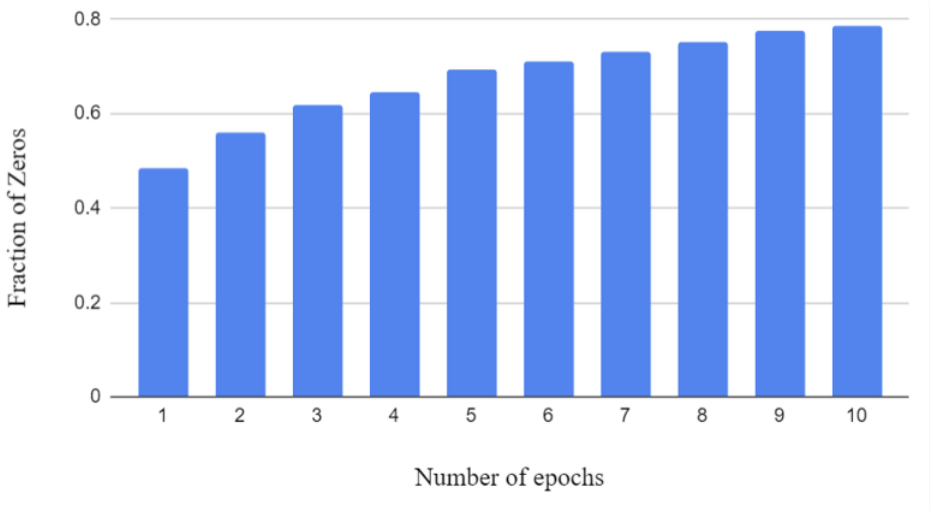


Figure 3.1: Fraction of zeros versus the number of epochs for MNIST

The range of tensor values is [-1.2312, 1.3018] and [-1.2766, 1.1403] for MNIST and F-MNIST, respectively. Both datasets have a significant number of zeros during the training processes. Our first observation is that the number of zeros increases as the training makes progress. Figs. 3.1 and 3.2 show the fraction of zero values over different epochs for both datasets. As the accuracy increases, the features get

more specific. As the accuracy of the model improves, the number of zeros increases. This is because during checking for specific features in images, many of them are found to be absent. This is the result of the fact that each feature exists in only a different set of images. For this reason, F-MNIST has fewer of absolute zeros over all epochs compared to MNIST. This can be justified by the higher accuracy the network achieves when trained on MNIST (99.42%) compared to F-MNIST (90.45%).

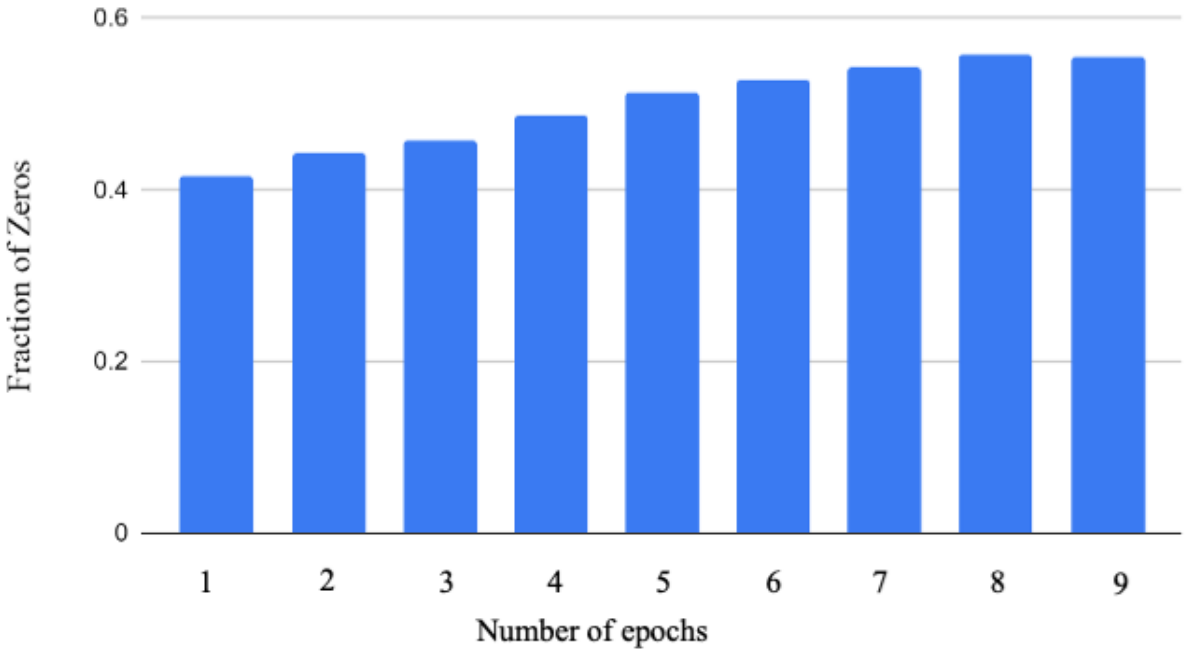
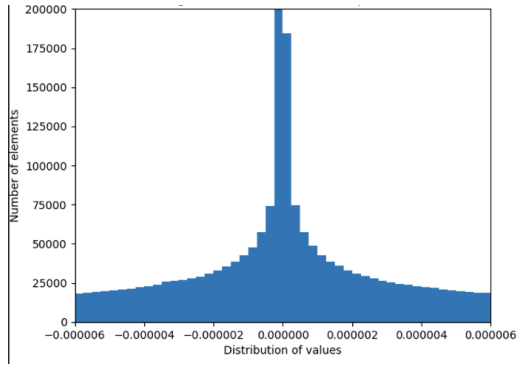


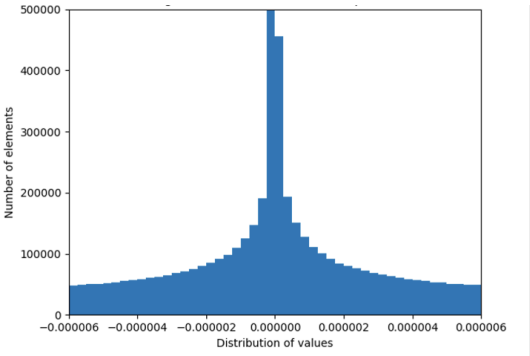
Figure 3.2: Fraction of zeros over epochs for F-MNIST

Both datasets achieve a high number of absolute zero values. This percentage ranges from 32% to 77% in the first and 10th epochs for MNIST and from 42% to 55% for FMNIST. Figs. 3.3b and 3.3a depicts histograms of the distribution of predictions generated by the primary capsules for the datasets. Both plots are cropped from the top to make the surrounding points more visible. In other words, the bin for the number of zeros is removed since it has a significantly higher number of values than the other bins. Even though there are many values close to zero, their numbers are far less that the number of absolute zeros available. As a result,

the reduction in computations for values near zero would be negligible compared to that for absolute zero values. Therefore, for our case pruning does not apply. Our investigation shows that more than 50% of primary capsule predictions consist of absolute zeros. Multiplications with these zero values can be skipped using methods like the one proposed in [2] and this will result in fewer computations.



(a) MNIST



(b) F-MNIST

Figure 3.3: Distribution of predictions by the primary capsules

3.4 Conclusion

CapsNet is considered as the next generation neural network since it not only achieves state-of-the-art results on MNIST handwritten digits dataset, but also has additional advantages over the conventional CNNs, e.g. detecting overlapping entities. However, CapsNet is computationally expensive. In this work, we investigated the potential of

zero skipping in CapsNet and showed that there are many zeros to be skipped when trained on MNIST and Fashion-MNIST. Further, this may lead to significant savings in energy in Capsule Networks.

Chapter 4

PrunedCaps: A Case for Primary Capsules Discrimination

4.1 Introduction

Artificial Neural Networks (ANNs) have recently become effective and popular models for the task of classification, pattern recognition, prediction and data clustering tasks in various fields [1]. ANNs are frequently used today for approximating functions due to properties such as adaptability, self-learning, non-linearity and advanced mapping of the inputs to the outputs [33].

ANNs continue to grow more complicated as architectures become more complex. Complex networks can achieve high accuracy but at the cost of slow training and inference phases [4]. Neural Network approximation methods redesign complex ANNs with the goal of saving memory and reducing computation power. These improvements usually come at the cost of marginal accuracy losses. Such approximations play a crucial role in reducing the latency and improving throughput in various applications. These methods can be categorised into two major groups: quantization

and weight reduction. Quantization algorithms focus on reducing the precision of the network parameters. While weight reduction solutions eliminate redundant neurons or structures [34].

Pruning is a well-known method for weight reduction methods. Pruning an ANN is the process of removing neurons from a network. This method aims to omit neurons that have zero to low impact on the output accuracy. ANN pruning was introduced by Lecun et al [17]. There are two main types of pruning for ANNs: fine-grain and coarse-grain. Similarity evaluations and iterative pruning processes both implement pruning at an element-wise granularity. These techniques are referred to as the fine-grain pruning. Fine-grain pruning can result in uneven weight distributions which can affect hardware's throughput. Coarse-grained pruning techniques, however, can find best in the compression-throughput tradeoff by considering the whole architecture instead of focusing on individual elements [14].

Sabour et al introduced Capsule Networks (CapsNets) as the next generation of ANNs [30]. The CapsNet computational unit consists of a vector of neurons referred to as a capsule. The final representation of this network is also in the form of capsules. CapsNet contains more information compared to other conventional ANNs such as Convolutional Neural Networks (CNNs) due to the absence of pooling layers. Pooling reduces the size of an feature map. Pooling layers in CNNs are the main cause of information loss and are not in CapsNets.

Primary Capsules (PCs) are reshaped vectors calculated through multiplication of a feature extractor output and a weight matrix. PCs contain the most trainable number of parameters in the entire CapsNet architecture. Another trainable part of CapsNets is the convolutional layers which build the feature extractor and translate images to the feature maps. The last section which has trainable parameters is the decoder layer which reconstructs the image and compares it to the input. The focus

of this chapter is on pruning PCs to speed-up inference.

Pooling layers are mainly used to reduce the size of feature maps by removing elements that are not significant to the output. Not using pooling layers may cause over-fitting in smaller networks. Choosing the maximum element or averaging certain elements are examples of pooling. Since CapNets do not utilize pooling layers, the accumulation of convolutional layer outputs results in a very large multi-dimensional matrix. Having a multi-dimensional matrix requires time consuming computations which affects the architecture's suitability for low latency applications such as self-driving cars. In this chapter, we use pruning technique that rely on first-order gradient information for parameter salience. This technique has been successful in CNNs and showed promising results in CapsNets removing more than 95% of the parameters with a minuscule drop in accuracy. [27]

In summary our contributions of this chapter are as follows:

- Pruning CapsNet Primary Capsules using Taylor expansion. Primary Capsules are a reshaped representations of features which are multiplied by a matrix. Taylor's expansion is used as a metric to select and remove PCs which are redundant to the architecture.

- A comprehensive analysis is given using popular datasets such as MNIST hand written digits, Fashion-MNIST, SVHN and CIFAR-10. Inference time, number of FLOPS and an architecture different numbers of capsules.

- Providing insight on how pruned CapsNets behave for different number of PCs. We also study how dataset complexity (i.e. size and features), impacts CapsNet behaviour. We show how Taylor expansion pruning removes PCs and how CapsNet is influenced by the reduction in the ability of the network to fitting the input function (imposed by pruning).

This chapter is organized as follows. Section 4.2 describes related works. Section 4.3 details the background and the proposed method. Section 4.4 reports the results and discussions. Section 4.5 offers some concluding remarks.

4.2 Related Work

There are several works results in the literature focusing on pruning ANNs. Lecun et al. and Hassibi et al. are the pioneers in removing unnecessary weights in ANNs [17, 7]. Pruning can be effective in different ways. Han et al. and Suzuki et al. show how pruning can sometimes increase accuracy by training longer compared to the baseline model, without overfitting [6, 32]. Residual Network (ResNet) are a kind of Deep Learning Network which use residual blocks. Residual blocks help when there are many layers to train without vanishing gradients by appending the input feature to the output of the block. Previous research has established that the error rate on the ResNet20 network can be decreased by setting sparse weights to the pruned network [18]. Kalchbrenner et al. [11] used pruning for efficient audio synthesis. They used a single Recurrent Neural Network (RNN) referred to as WaveNet. Their findings suggest that a sparse architecture can outperform a smaller dense network with the same number of parameters.

Random pruning is the process of selecting the target neurons and removing them randomly. Researchers have observed that method-based pruning beats random pruning. [4, 5, 38, 26]. Frankle et al. [4] argued that a large, dense, and randomly initialized network contains subnetworks. These subnetworks can be trained to perform competitively to their parent network. These subnetworks are initialized with the original weights of the network. Lottery ticket pruning is the process of finding these subnetworks, whose size is under 10-20% of the size of the original network. Lee et

al. [19] have identify irrelevant connections using a method referred to as Single-Shot Network Pruning (SNIP). They start pruning prior to training. This can lead to better results due to network sparsity at initialization.

Pruning can also be done by removing neurons from all layers of a network in a uniform fashion. To date, several studies have investigated this approach. Performing pruning uniformly is outperformed by a smart parameter allocation technique [5, 6]. Lou et al. [22] demonstrated a compressed and accelerated pruning method for CNNs. Their method is not uniform and achieves better accuracy.

A pruned network will lose accuracy if it is not fine-tuned. Fine-tuning is the process of continuing the training of an ANN after initial prunning. Recent studies [41, 38] suggest that if all weights are set to zero, training a pruned architecture falls behind fine-tuning.

4.3 Background

In this section, we first review CapsNet and its architecture. and then explain the pruning techniques.

4.3.1 Capsule Network (CapsNet)

The basic computational units in CapsNet are capsules (vectors of neurons). The architecture of CapsNet is shown in Figure 4.1. According to this figure, the network starts with extraction of low-level features using two convolutional layers. The extracted features are then reshaped to vectors. These vectors are then multiplied by a matrix, encoding the spatial relationship between them. The resulting vectors are referred to as Primary Capsules (PCs).

The next layer of capsules (the output capsules) are inferred from the PCs. There

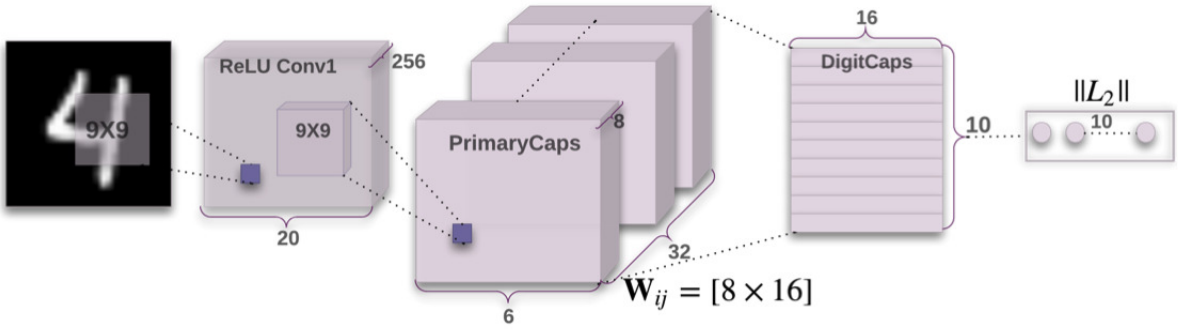


Figure 4.1: The original CapsNet Architecture [30]

is a fully-connected relationship between the PCs and the output capsules. However, the coefficients are not learned during the training process. Instead, they are determined in each iteration of the training process by an iterative algorithm referred to as Dynamic Routing (DR). This algorithm determines the contribution of each PC to each output capsule based on the level of agreement among PCs.

CapsNet is designed such that there are as many output capsules as the number of categories in the classification task. The output capsules hold two important pieces of information. First, the capsule with the highest magnitude is the capsule corresponding to the correct class. Second, the different dimensions of each output capsule hold information about the instantiation parameters of the input image such as pose and deformation [30].

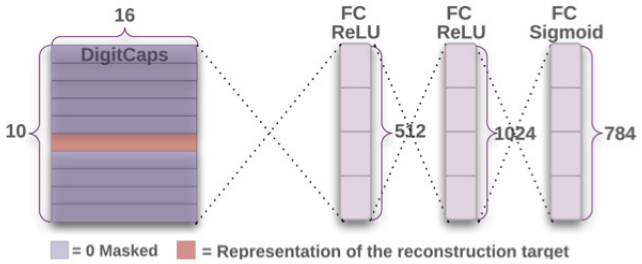


Figure 4.2: Decoder architecture of CapsNet [30]

CapsNet also includes a basic decoder to reconstruct the input images using the

output capsules as input. The decoder consists of fully-connected layers. The reconstructed image is compared to the input images and the difference, known as the reconstruction loss, is used to regularize the training process.

The loss function in CapsNet consists of two terms. The first term is the reconstruction loss explained earlier. The second term is called the margin loss and is based on the predictions made by the output capsules. The loss for each output capsule

$$L_k = T_k \max(0, m^+ - ||V_k||)^2 + \lambda(1 - T_k) \max(0, ||V_k|| - m^-)^2$$

where T_k is determined based on the prediction of each capsule (one for correct predictions and zero otherwise), λ is the weight considered for penalizing wrong predictions, and m^+ and m^- are used to remove capsules with high or low probabilities from taking part in the margin loss.

4.3.2 Pruning Methods

Pruning consists of removing the least important parameters. It is essential to minimize the difference in accuracy between the normal and pruned networks. Optimally, the impact of removing each parameter should be evaluated in the terms of the impact on network accuracy.

There are two criteria used to estimate the importance of the parameters. The first is based on the sign change in the loss $L_{D,W'} - L_{D,W}$, where $L_{D,W}$ is the value of loss for the network over dataset D using parameters W. W' is the new set of parameters with the redundant parameters removed. The second criterion is the absolute difference in the loss $|L_{D,W'} - L_{D,W}|$.

It is significantly expensive to consider the effect of each parameter individually on the dataset. Therefore there are various estimation methods including Minimum

Weight, Activation, and Taylor Expansion. Here we briefly explain each method.

Minimum Weight

The minimum weight method is based on the magnitude of the parameters in the kernel. The intuition behind this method that the lower the $l2$-norm of a kernel, the less important features it detects. Regularizing the network would be beneficial to this method since kernels corresponding to less important features will be pushed to have smaller values.

Activation

Rectified Linear Unit (ReLU) is a mathematical function defined as $\max(0, x)$ where x is the input of the function. The ReLU activation function is sparse and is used as a feature detector in the convolutional layers. This is due to the fact that these layers check for existing features in their input. Any feature element with zero or negative activation means that the associated feature does not exist.

Activation pruning works by removing small activation values. The issue with activation pruning is that it can only use the ReLU activation function. Other common activation functions used in CNNs do not clip the negative values like ReLU, and therefore cannot be used for pruning the feature extractor.

Taylor Expansion

This pruning method aims to minimize $|L_{D,W'} - L_{D,W}|$ by removing some parameters [25]. The Taylor expansion method can approximate the effect of removing a parameter on the loss function. Assuming that parameters are independent, for parameter i our goal is to minimize $|L_{D,W_i=0} - L_{D,W_i}|$. We can estimate the minimization target using the Taylor expansion. The Taylor expansion for function f at the point $x = a$

is

$$f(x) = \sum_{0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{4.1}$$

where $f(n)$ denotes the n-th order derivative of function $f(x)$.

The Taylor expansion for function $L_{D,W_i=0}$ at $W_i = 0$ is

$$L_{D,W_i=0} = L_{D,W_i} - \frac{\partial L}{\partial W_i} W_i + R \tag{4.2}$$

where R contains all the remaining higher order terms in the Taylor expansion. We neglect the remaining terms for two reason. First, they increase the computation complexity. Second, the value of the higher order terms in the Taylor expansion are often negligible compared to the first order term.

Based on 4.2, the minimization target now changes to

$$|L_{D,W_i=0} - L_{D,W_i}| = \left| \frac{\partial L}{\partial W_i} W_i \right| \tag{4.3}$$

In short, the Taylor method results in pruning parameters associated with small gradients in the loss function. Implementing this method is feasible as it needs the multiplication of the gradient of a parameter by the parameter itself. This information are all available during back-propagation. Considering the effect of all parameters, the pruning method using Taylor expansion can be reformulated as minimizing

$$F(W) = \left| \frac{1}{M} \sum_{m} \frac{\partial L}{\partial W_m} W_m \right| \tag{4.4}$$

where M is the number of all parameters.

4.3.3 PrunedCaps Method

In this section, we use the Taylor's expansion method. Primary Capsule is a reshaped representation of the feature extraction layer which is multiplied by a matrix that encodes the spatial relations into vectors. Since PCs are multi-dimensional, changing them requires high computational power.

Feature vectors which are ignored and not processed further in the network can be considered removed from the network. This is despite the fact that they are computed every time there is an input to the network.

Selecting which Primary Capsules to remove in each pruning epoch requires ranking capsules according to their activations and back-propagation gradients. Therefore, Primary Capsules are ranked according to the product of their back-propagation gradient and activations. The results of the activation times the gradients are then divided by the number of features present. If this results in a high number compared to the output of the same process for other PCs, we assume is significance to the the network for inference. In our case, the number of features which are present is the number of remaining Primary Capsules. This is the implementation presented in (4). $F(W)$ is a weighted average of activations and the weights are gradients which represent the significance of the activation.

Our method prunes the Primary Capsules that show little to no change with respect to their output. As iterations grow towards a complete epoch, this value is accumulated giving a result indicating how each Primary Capsule would behave over a dataset. After each pruning epoch ends, they are ranked and are the lowest values for the ranking criteria are selected. The number of PCs to be removed is a hyper-parameter which is set before training begins.

After each pruning phase, CapsNet goes through a training phase to restore its accuracy and adapt to the changes made to its architecture. This training phase,

which happens after the network has been pruned, is usually referred to as fine-tuning phase. Between pruning epochs, we fine-tune for several epochs so the network can reach the maximum possible accuracy. The number of fine-tuning epochs is set empirically. If we increase this number of epochs, there could be a minuscule accuracy gain whose effects can be neglected. If this number is reduced it is uncertain if the network has reached its maximum accuracy. In our case, we tested different numbers of epochs and decided on 50 which ensures a fully-trained network. The algorithm in Fig. 4.3.

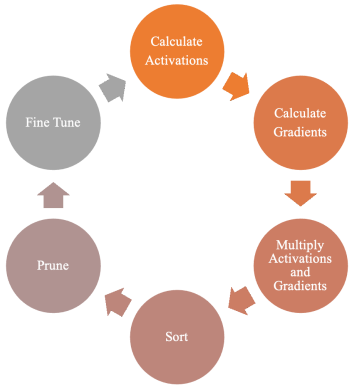


Figure 4.3: Summary of the Pruning Algorithm

4.4 Experiments and Results

In this section, we present the experiments and the results. Experiments were done with a machine equipped with an NVIDIA 2080 Ti GPU and 32 GB of memory and a machine equipped with an NVIDIA V100 GPU and 13 GB of memory.

We use the MNIST handwritten digits, Fashion-MNIST datasets, CIFAR-10 and SVHN datasets. The MNIST handwritten digits and Fashion MNIST datasets contain 28×28 single-channel images. MNIST handwritten digits contain images of handwritten digits from zero to nine. The Fashion MNIST dataset includes images of different pieces of clothing. They have the same number of classes for classification.

By default, MNIST handwritten digits and Fashion MNIST are divided into 50,000 and 10,000 images for training and testing. CIFAR-10 and SVHN datasets have the same 32×32 image size and ten classes of RGB images. CIFAR-10 has 50,000 images for the training set and 10,000 images for the testing set, whereas SVHN has 73,257 images for the training set and 26,032 images for the testing set. CIFAR-10 consists of ten very different classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks. SVHN dataset classes are the same as MNIST handwritten digits, but the digits are house numbers obtained from Google's Street View [28]. For 28×28 size images, we start with 1152 Primary Capsules, which is the baseline

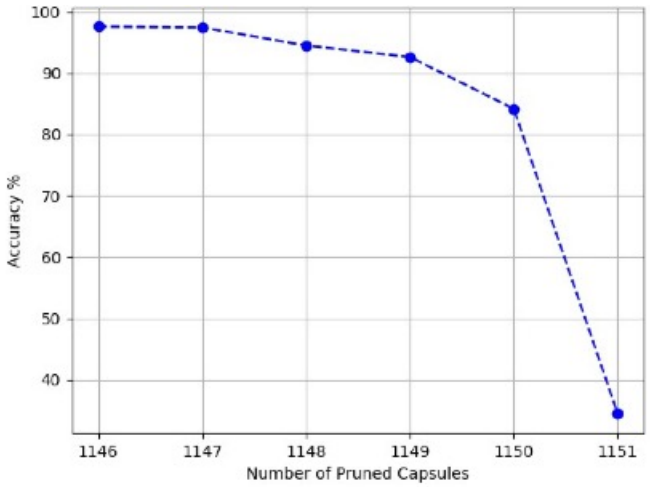


Figure 4.4: CapsNets accuracy drop on MNIST starts with only 5 PCs remaining.

number of capsules in the original architecture. We then train the network to reach its baseline accuracy on each dataset. The baseline architecture reached 99.47% accuracy on MNIST and 90.23% accuracy on Fashion MNIST testing sets samples. We save the weights at the end of the training phase to be used in the pruning phase.

Pruning weights starts with one epoch of training so that the backpropagation gradients can be calculated for PC rankings. Since more than half of the PCs are either zero or have values near zero, we chose to prune 100 PCs in each pruning epoch for the first 1100 PCs. After each pruning epoch, the network is fine-tuned

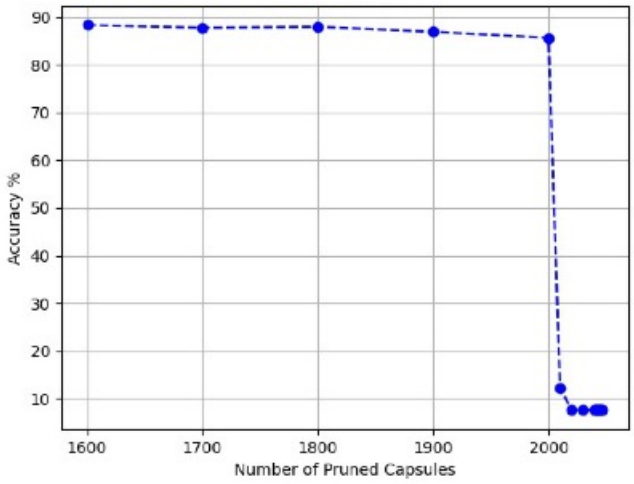


Figure 4.5: The drop in accuracy begins after 2000 PCs pruned on SVHN dataset.

on the dataset to reach maximum accuracy. After 1100 PCs are pruned, we lower the number of PCs to be pruned to 10.This change in the number of PCs pruned is because the network changes in accuracy starts at 1100 PCs pruned. Fig. 4.4 and 4.7, we show that the network cannot recover from the PCs removed from its architecture for MNIST and FMNIST datasets. These two figures show the maximum accuracy reached by the architecture at each number of capsules.

The same process is repeated for CIFAR-10 and SVHN datasets. The main difference is the starting number of Primary Capsules. The starting number of PCs is a function of the input size. For MNIST and Fashion-MNIST this number is 1152 which is $6 \times 6 \times 32$. This is the reshaped feature size of the input image which went through two Convolutional layers with stride one and stride two for the second layer. For 32×32 pixel images, 1152 changes to 2048 which is $8 \times 8 \times 32$. Sabour et al. discussed how input image shapes change in the CapsNet [30].

Since CIFAR-10 and SVHN have 32×32 images, the starting number of PCs is 2048. We present the performance measures for SVHN in Fig. 4.5. The starting accuracy for a fully-trained network is 92.65%. After pruning 1500 PCs out of 2048, we only lose 1.85% of performance. The results for CIFAR-10 are presented in Fig. 4.6.

The starting accuracy for CIFAR-10 is 71.37% for the original CapsNet architecture.

After the pruning process, 2000 PCs have been removed.

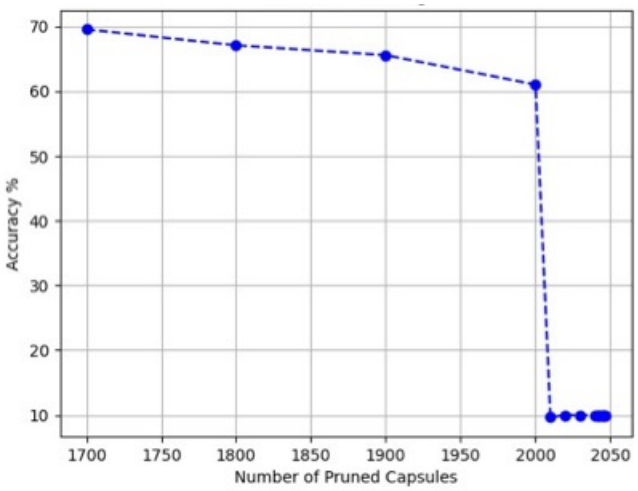


Figure 4.6: Sudden drop in accuracy with the CIFAR-10 dataset compared to MNIST and F-MNIST

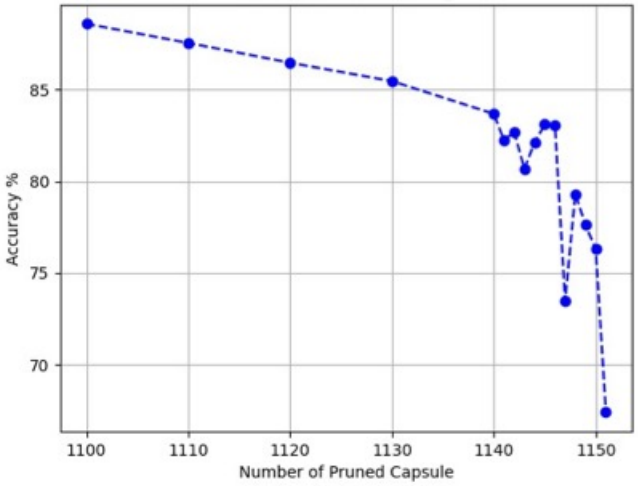


Figure 4.7: Accuracy drop starts when 52 PCs are remaining for the Fashion-MNIST dataset. F-MNIST is considered to be more complex than MNIST.

4.4.1 Number of FLOPS

Floating-point operations are multiplications or summations done by the hardware.

The CapsNet baseline architecture takes 276,480 FLOPS to calculate the matrix

multiplications, which produce the 1152 PCs for the MNIST datasets. This number drops to 12,480 FLOPS when the number of PCs remaining is 52. This 95.48% drop in the number of FLOPS is a major improvement over the baseline architecture. Also, dynamic routing achieves a 95.36% drop in the number FLOPS. The SVHN and CIFAR-10 datasets also have major drops in the number of FLOPS. The SVHN has a 73.24% drop FLOPS when network was operating on 1500 PCs removed. CIFAR-10 can be pruned until the reduction in FLOPS is 83.01% at the cost of 1.85%.

Since the only part of the network that is changing is the number of PCs (feature extraction and decoding are unchanged), there is no change in the total number of FLOPS an input image in other parts of the architecture.

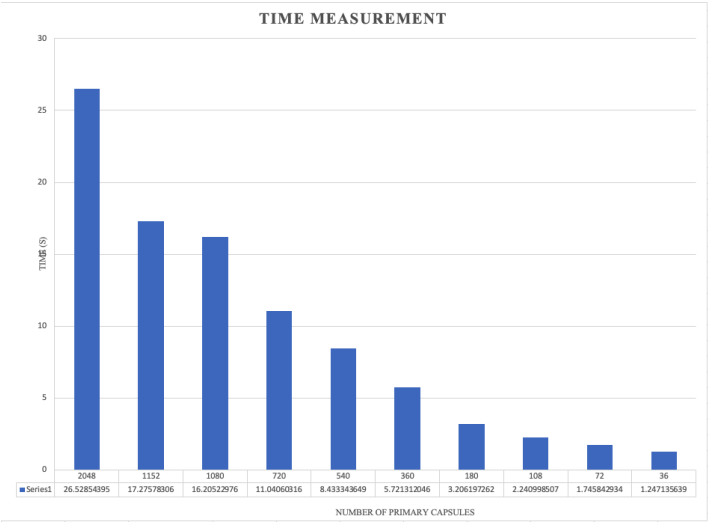


Figure 4.8: Architecture testing time in seconds for different numbers of PCs

4.4.2 Inference Time Reduction

As shown in the previous section, we save more than 95% in the number of FLOPS in the PCs and dynamic routing section of the architecture. This helps lower the inference time. Fig. 4.8 shows the time for CapsNet to output results for 10,000 test images. This is measured when running on the NVIDIA 2080 Ti equipped with 32

GB of memory. The pruned architecture produces results 9.90 times faster than the original architecture on MNIST datasets. The first two bars correspond to for the baseline architecture and the highest bar is for 2048 PCs. This is the baseline number of PCs for the SVHN and CIFAR-10 datasets. The second bar is for 1152 PCs which is the baseline number of PCs when the architecture is trained on MNIST datasets.

4.4.3 Discussion

In this section, we further discuss how pruning affects the accuracy, number of FLOPS, and number of PCs in CapsNet for different datasets. In [31], Sharifi et al. showed that more than 50% of PC weights are either zero or insignificant. With our method, pruning PCs that are trained on MNIST handwritten digits and Fashion-MNIST does not show significant change in accuracy. The small accuracy drop is due to the fact that most of the PCs are zeros for these two datasets. This is not the case for SVHN and CIFAR-10. These two datasets are visually and mathematically more complex compared to MNIST handwritten digits and Fashion MNIST [3]. Also, the SVHN and CIFAR-10 images are larger in size, 32×32 pixels, versus those in MNIST and Fashion MNIST, which are only 28×28 pixels. Our results show that a larger and more complex dataset needs more remaining PCs to maintain its performance. Therefore for SVHN and CIFAR-10, CapsNet is more sensitive to pruning and cannot recover after removing a certain number of PCs.

This is consistent with our understanding of CapsNet capacity. Capacity in a deep learning model is defined by the ability of the network to approximate different functions. A model's capacity consists of all trainable parameters in the network. Removing trainable parameters from a network changes its capacity to learn. Since PCs have the highest number of trainable parameters in the CapsNet architecture, they play a significant role in model capacity. As explained earlier SVHN and CIFAR-10

are considered complex datasets compared to MNIST datasets. Therefore maintaining network accuracy will require a higher number of PCs for SVHN and CIFAR-10 compared to the MNIST datasets.

4.5 Conclusion

This chapter investigated Primary Capsules pruning of CapsNets. Although, CapsNets have specific advantages over Convolutional Neural Networks as their training and inference phases are inefficient. We trained and fine-tuned the original CapsNet on the MNIST handwritten digits, Fashion-MNIST, SVHN and CIFAR-10 datasets. Our results show up to 9.90 times speedup and more than a 95% drop in the number of FLOPS over the baseline architecture with only a minuscule drop in accuracy. We also provided better insight on why CapsNet's behaves differently when pruned on more complex datasets such as SVHN and CIFAR-10 compared to the MNIST and Fashion-MNIST datasets.

Chapter 5

Conclusions

This thesis considered the problem of Capsule Network efficiency. Capsule Networks have several advantages over conventional Convolutional Neural Networks, but they are computationally expensive.

Chapter two provided an introduction to CapsNets. We discussed how the architecture is trained, loss functions and regularization in CapsNets and Convolutional Neural Networks. The CapsNets advantages and disadvantages over CNNs were presented. A brief summary of the datasets used in this work was also given.

Chapter three investigated if CapsNets produce enough zeros to be suitable for zero-skipping algorithms. We trained CapsNet on handwritten digits MNIST and Fashion MNIST. The results suggested that more than 50% of the capsules had zero or near zero values for a fully trained CapsNet.

Chapter four presented an algorithm to prune CapsNets. Pruning is the act of removing neurons that have low or no impact on the output. CapsNet was trained on four different datasets (training phase). The fully trained CapsNet was pruned and fine-tuned to reach attain maximum accuracy. The results obtained suggest a more than 95% drop in the number of FLOPS. A pruned CapsNet performed 9.90 times

faster than the original architecture. We also provided an insight into the behaviour of the network on different datasets with respect to their complexities.

Bibliography

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat Abd Elatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4, 2018.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In *ACM/IEEE International Symposium on Computer Architecture*, pages 1–13, Seoul, South Korea, June 2016.

[3] Frederic Branchaud-Charron, Andrew Achkar, and Pierre Marc Jodoin. Spectral metric for dataset complexity assessment. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 2019.

[4] Michael Carbin and Jonathan Frankle. the lottery ticket hypothesis. *International Conference on Learning Representations*, pages 1–42, 2019.

[5] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. 2019.

[6] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems*, 2015-January, 2015.

[7] Babak Hassibi, David G. Stork, and Gregory J. Wolff. Optimal brain surgeon and general network pruning. *IEEE International Conference on Neural Networks*, 1993.

[8] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, Vol. 313(5786):504–507, 2006.

[9] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 44–51. Springer Berlin Heidelberg, 2011.

[10] Nan Hou, Hongli Dong, Zidong Wang, Weijian Ren, and Fuad E. Alsaadi. Non-fragile state estimation for discrete Markovian jumping neural networks. *Neurocomputing*, 179:238–245, February 2016.

[11] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimber, Aäron Van Den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. volume 6, 2018.

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

[13] Youngjoo Kim, Peng Wang, Yifei Zhu, and Lyudmila Mihaylova. A Capsule Network for Traffic Speed Prediction in Complex Road Networks. In *2018 Sensor Data Fusion: Trends, Solutions, Applications*, pages 1–6, Bonn, October 2018. IEEE.

[14] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December, 2016.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[16] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[17] Yann LeCun, John S Denker, and Sara A. Solla. Optimal brain damage (pruning). *Advances in neural information processing systems*, 1990.

[18] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020.

[19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H.S. Torr. Snip: Single-shot network pruning based on connection sensitivity. 2019.

[20] Yingyan Lin, Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. PredictiveNet: An energy-efficient convolutional neural network via zero prediction. In *IEEE International Symposium on Circuits and Systems*, pages 1–4, Baltimore, MD, USA, May 2017.

[21] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, April 2017.

[22] Jian Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, October 2017.

[23] Tao Luo, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang, Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen. DaDianNao: A Neural Network Supercomputer. *IEEE Transactions on Computers*, 66(1):73–88, January 2017.

[24] Aryan Mobiny and Hien Van Nguyen. Fast CapsNet for Lung Cancer Screening. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention*, volume 11071, pages 741–749. Springer International Publishing, 2018.

[25] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2498–2507. JMLR.org, 2017.

[26] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11256–11264, 2019.

[27] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[28] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[29] Jathushan Rajasegaran, Vinoj Jayasundara, Sandaru Jayasekara, Hirunima Jayasekara, Suranga Seneviratne, and Ranga Rodrigo. DeepCaps: Going Deeper

With Capsule Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10717–10725, Long Beach, CA, USA, 2019.

[30] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *Advances in Neural Information Processing Systems*, 2017-December, 2017.

[31] Ramin Sharifi, Pouya Shiri, and Amirali Baniasadi. Zero-skipping in capsnet. is it worth it? In *International Conference on Computers and Their Applications*, volume 69, pages 355–361, 2020.

[32] Taiji Suzuki, Hiroshi Abe, Tomoya Murata, Shingo Horiuchi, Kotaro Ito, Tokuma Wachi, So Hirai, Masatoshi Yukishima, and Tomoaki Nishimura. Spectral pruning: Compressing deep neural networks via spectral analysis and its generalization error. In *International Joint Conferences on Artificial Intelligence Organization*, pages 2839–2846, 2020.

[33] Ding Wang, Haibo He, and Derong Liu. Intelligent optimal control with critic learning for a nonlinear overhead crane system. *IEEE Transactions on Industrial Informatics*, Volume 14, 2018.

[34] Erwei Wang, James J. Davis, Ruizhe Zhao, Ho Cheung Ng, Xinyu Niu, Wayne Luk, Peter Y.K. Cheung, and George A. Constantinides. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Computing Surveys*, Volume 52, 2019.

[35] 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, December 2018.

[36] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[37] Fan Yang, Hongli Dong, Zidong Wang, Weijian Ren, and Fuad E. Alsaadi. A new approach to non-fragile state estimation for continuous neural networks with time-delays. *Neurocomputing*, 197:205–211, July 2016.

[38] Ruichi Yu, Ang Li, Chun Fu Chen, Jui Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching Yung Lin, and Larry S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[39] Yajing Yu, Hongli Dong, Zidong Wang, Weijian Ren, and Fuad E. Alsaadi. Design of non-fragile state estimators for discrete time-delayed neural networks with parameter uncertainties. *Neurocomputing*, 182:18–24, March 2016.

[40] Jie Zhang, Lifeng Ma, and Yurong Liu. Passivity analysis for discrete-time neural networks with mixed time-delays and randomly occurring quantization effects. *Neurocomputing*, 216:657–665, December 2016.

[41] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 38, 2016.