Filter Pruning in Convolutional Neural Networks Using Structural Similarity Based K-Means

by

Ahmed Al Dallal

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

© Ahmed Al Dallal, 2021

Abstract

Convolutional Neural Networks (CNNs) have been recently seeing great success in various image classification fields and applications. However, this success has been accompanied by a significant increase in memory and computational demands, limiting their use in resource-limited devices, e.g., smartphones. In response, network pruning methods, in particular filter pruning, are seeing increased interest. The principal goal of the current pruning algorithms is to substantially reduce the resource demands for executing the forward pass of a trained CNN, while minimizing performance degradation.

In this thesis, we propose a new approach for filter pruning in CNNs. Our filter pruning method utilizes K-Means clustering based on the Structural Similarity Index Measurement to group similar filters together in each convolutional layer. A representative filter is selected from each cluster and the remaining filters are considered redundant and pruned from the CNN. We evaluated our filter pruning method on the VGG-16 architecture with the benchmark CIFAR-10 dataset. We were able to reduce the computational demands (floating-point operations) of VGG-16 by over 50%. Simultaneously, the network's performance remained significantly better than the one pruned by the HRank algorithm. The results of our experiments provide promising indications that our method can significantly outperform state-of-the-art filter pruning methods.

Acknowledgments

First and foremost, I would like to thank Allah (God) Almighty for his grace and mercy in making all this possible.

I would like to thank my supervisor Prof. Scott Dick for his support and guidance during my M.Sc. studies. His advice and supervision were instrumental for completing my thesis.

I would like to express my utmost gratitude for my parents for their endless support, encouragement, and prayers. I would also like to thank my brother for his motivational talks and moral support when I needed them the most.

Table of Contents

Chapter 1: Introduction 1

Chapter 2: Literature Review 5

2.1 Artificial Neural Networks 5

2.1.1 Convolutional Neural Networks 12

2.2 Filter Pruning 33

2.3 Clustering Algorithms 37

2.3.1 K-Means Clustering 40

2.4 Cluster Validation 43

2.4.1 Silhouette Index 45

2.5 Similarity Measures for Images 46

2.5.1 Structural Similarity Index Measurement 47

Chapter 3: Methodology 49

3.1 Overview 49

3.2 Implementation 51

3.2.1 Dataset 51

3.2.1.1 CIFAR-10 51

3.2.2 VGG-16 52

3.2.3 SSIM Based K-Means Clustering 55

3.2.4 SSIM Based Silhouette Index 59
3.3 Filter Clustering Algorithm........ 62
3.4 Filter Pruning Algorithm........ 63
3.5 Evaluation 66
Chapter 4: Results and Discussion........ 70
4.1 Silhouette Plots 70
4.2 Compression Scenarios........ 84
4.2.1 Compression Scenario A........ 87
4.2.2 Compression Scenario B 88
4.2.3 Compression Scenario C 89
4.3 Accuracy Results 90
4.3.1 Compression Scenario A........ 91
4.3.2 Compression Scenario B 93
4.3.3 Compression Scenario C 94
Chapter 5: Conclusion and Future Work 97
References........ 99

List of Figures

Figure 1: Artificial Neuron 6

Figure 2: Details of the signal flow in output neuron j [26] 9

Figure 3: Details of the signal flow in output neuron k connected to hidden neuron j [26] 11

Figure 4: The relationship between weights, kernels, convolutional filters, kernel matrices, and feature maps 15

Figure 5: Max-Pooling operation 18

Figure 6: Original VGG-16 architecture 21

Figure 7: First iteration of the convolutional operation in convolutional layer 1 23

Figure 8: A neuron in convolutional layer 1 in VGG-16 24

Figure 9: Neurons needed to generate 1 feature map in convolutional layer 1 in VGG-16 25

Figure 10: First iteration of the convolutional operation in convolutional layer 2 27

Figure 11: A neuron in convolutional layer 2 in VGG-16 28

Figure 12: Neurons needed to generate 1 feature map in convolutional layer 2 in VGG-16 29

Figure 13: High level view of feature maps in convolutional layers 1 and 2 in VGG-16 30

Figure 14: The difference between weight pruning and filter pruning 36

Figure 15: Our proposed approach 50

Figure 16: Some sample images of the CIFAR-10 dataset 52

Figure 17: VGG-16 architecture 54

Figure 18: K-Means initialization 57

Figure 19: K-Means cluster assignment and centroid re-estimation 58

Figure 20: Mean of three filters 59

Figure 21: Silhouette Index 61

Figure 22: Silhouette plot for convolutional layer 1 72
Figure 23: Silhouette plot for convolutional layer 2 73
Figure 24: Silhouette plot for convolutional layer 3 74
Figure 25: Silhouette plot for convolutional layer 4 75
Figure 26: Silhouette plot for convolutional layer 5 76
Figure 27: Silhouette plot for convolutional layer 6 77
Figure 28: Silhouette plot for convolutional layer 7 78
Figure 29: Silhouette plot for convolutional layer 8 80
Figure 30: Silhouette plot for convolutional layer 9 81
Figure 31: Silhouette plot for convolutional layer 10 82
Figure 32: Silhouette plot for convolutional layer 11 83
Figure 33: Silhouette plot for convolutional layer 12 84

List of Tables

Table 1: The descriptions of the convolutional layers before and after pruning using compression scenario A 88

Table 2: The descriptions of the convolutional layers before and after pruning using compression scenario B... 89

Table 3: The descriptions of the convolutional layers before and after pruning using compression scenario C... 90

Table 4: Filter pruning resulting accuracy using compression scenario A 92

Table 5: Filter pruning resulting accuracy using compression scenario B 94

Table 6: Filter pruning resulting accuracy using compression scenario C 96

List of Abbreviations

CNN	-Convolutional Neural Network
DNN	-Deep Neural Network
AI	-Artificial Intelligence
FLOP	-Floating-Point Operation
ANN	-Artificial Neural Network
SGD	-Stochastic Gradient Descent
MBSGD	-Mini-Batch Stochastic Gradient Descent
ILSVRC	-ImageNet Large Scale Visual Recognition Challenge
ReLU	-Rectified Linear Unit
SSIM	-Structural Similarity Index Measurement
CIFAR	-Canadian Institute for Advanced Research
ASS	-Average Silhouette Score
LMSS	-Local Maximum Silhouette Score

Chapter 1

Introduction

Over the last ten years, Deep Neural Networks (DNNs) have become a very important field in Artificial Intelligence (AI). Giant tech companies such as Google, Amazon, Apple, Facebook, Microsoft, and Tesla use DNNs to power many of their services [1-6]. Researchers have found that DNNs significantly outperform all current competitors in many fields including self-driving cars, facial recognition, detecting land mines, detecting oil spills, photo editing, augmented reality, robotics, understanding handwritten text, language modeling, machine translation, text summarization, speech recognition, and textual entailment [7-16]. DNNs are also used in medical applications such as psychology, aiding visually challenged users to read, and diagnostic support for cancer, bone fractures, and skin conditions [17-19]. DNNs are also starting to see an increasing presence in the film, games, music, and fashion industries [20-23]. These industries combined bring in trillions of dollars every year [24, 25], and so the economic potential of DNNs is enormous.

However, DNNs are made up of thousands or even millions of neurons and millions of connections between them [26]. This in turn implies a massive number of computations; for example, a VGG-16 network (a form of convolutional neural network with only feed-forward connections) trained on the CIFAR-10 dataset (having only 60,000 training images) is a relatively small DNN, with only 4,224 filters. Yet even this network requires over 300 million Floating-Point Operations (FLOPs) to execute a single forward computational pass. There are millions of

computations taking place which demand tremendous hardware capabilities. While this is not a tremendous load for a workstation, a number of useful DNN applications would need to be deployed on resource-constrained devices such as smartphones, or low-power wearable or internet of things devices. For these devices, such computationally intensive applications can consume an unacceptable amount of battery power or require too much processing time on the relatively low-performance CPUs available [27-30]. A solution is needed that greatly reduces the resource consumption of DNNs, while not substantially degrading their performance.

One approach for reducing DNN resource consumption is pruning the DNN. Pruning is a well-known strategy that has been widely applied in shallow learning algorithms (e.g., decision trees, rule induction algorithms, neural networks, etc.). Classically, pruning has served both to reduce the resource consumption of executing a model, and also to improve its generalization (by reducing overfitting). However, given the sheer size of large DNNs, current research into pruning focuses only on the former goal; to the point that some degradation of performance (defined as minimizing a loss function) is expected and accepted [31]. Several pruning approaches have been investigated, including network pruning [32], parameter quantization [33], knowledge distillation [34], and filter compression [35]. Among these options, network pruning has shown great promise [31]. There are two kinds of network pruning methods: weight pruning and filter pruning. Weight pruning is the process of removing certain weights within filters from the neural network [36]. Filter pruning on the other hand, is the process of removing entire filters, which are deemed redundant, from the neural network [31]. The problem with weight pruning is that it leads to sparse weight matrices across the network and often requires using specialized software and hardware [31, 37, 38]. In comparison, filter pruning does not introduce sparsity [37]. Hence, using specialized software and hardware is not required.

CNNs are a subset of DNNs originally designed for image processing and computer vision applications [39]. As the name implies, the signature operation in these networks is to compute the convolution of the input image and a function represented by a convolution mask; these are the filters in a CNN [39]. Filter pruning can be applied to CNNs to compress the network, with minimal impact on the CNN's performance. Several filter pruning approaches for CNNs have been proposed in the literature [31, 37, 40, 41]. Studies in this field aim at reducing the size of CNNs in terms of the number of filters, parameters, and/or required computations with very minimal compromise to the classification accuracy.

In this thesis, we propose and evaluate a novel filter pruning approach for CNNs. Our approach is based on clustering similar filters in each considered convolutional layer, selecting a representative filter from each cluster, and pruning all other filters. To evaluate our approach, we performed several experiments on the VGG-16 CNN [42] with the benchmark CIFAR-10 dataset [43]. Our proposed filter pruning approach was evaluated in terms of the achieved model compression, model acceleration, and the classification accuracy of the filter pruned model after retraining. Model compression was measured by the number of reduced parameters. Model acceleration was measured by the number of reduced computations in the form of FLOPs. We compared our results with those of a state-of-the-art filter pruning method [31].

The primary contributions of this thesis are:

1) We introduce a new method for filter pruning, which utilizes clustering to determine redundant filters in CNNs.

2) Our experiments demonstrated the effectiveness and efficiency of our new method in model compression, acceleration, and accuracy. In addition, our method outperforms the current state-of-the-art filter pruning algorithm.

The remainder of this thesis is organized as follows. In Chapter 2, we review essential background and the relevant literature for our proposal. In Chapter 3, we discuss our proposed approach and our experimental methodology for evaluating it. In Chapter 4, we present and discuss our results. In Chapter 5, we provide a summary and discuss future work.

Chapter 2

Literature Review

2.1 Artificial Neural Networks

The rich history of neural networks research reaches back to the early 1940s, when W. McCulloch and W. Pitts [44] attempted to mimic animal nerve cell activity using mathematical models. Today, multiple layers of connected artificial neurons are known as Artificial Neural Networks (ANN) [26]. There is a huge number of ANN architectures in the literature, which see practical use in a vast number of problem domains [45]. CNNs, which are the focus of this thesis, belong to the subclass of ANNs known as layered feedforward networks. The defining features of this subclass are that neurons are arranged in a layered graph structure (with each layer typically having a homogenous transfer function), with no intra-layer connections, and no cycles within the graph.

Layered feedforward networks implement a complex functional mapping with a large (even huge) number of parameters; this is what makes them useful for modeling and decision-making in such a wide variety of applications. This mapping is realized through a combination of many individual neurons, each of which has a fairly simple transfer function. Computing the value of this mapping with the current parameter vector is commonly called a "forward pass" through the network. In the forward pass, each artificial neuron in the first layer receives a set of inputs and computes its transfer function. The outputs from the first-layer neurons then pass along the

graph connections to the second layer. This process repeats at each successive layer until the last one; the outputs of that final layer are returned as the output of the entire neural network.

Like many other ANN architectures, CNNs make use of a modified version of the McCulloch-Pitts neuron, as shown in Figure 1. Each connection for an incoming input 'x_i' of an artificial neuron has its own synaptic weight 'w_i'. The inputs are multiplied by the weights and the summation of the products 'v_i' along with a bias 'b' then goes through an activation function 'φ'. This operation can be expressed mathematically as follows:

$$v_i = \sum_{i=1}^{m} w_i x_i + b \,,$$

where 'm' is the number of inputs.

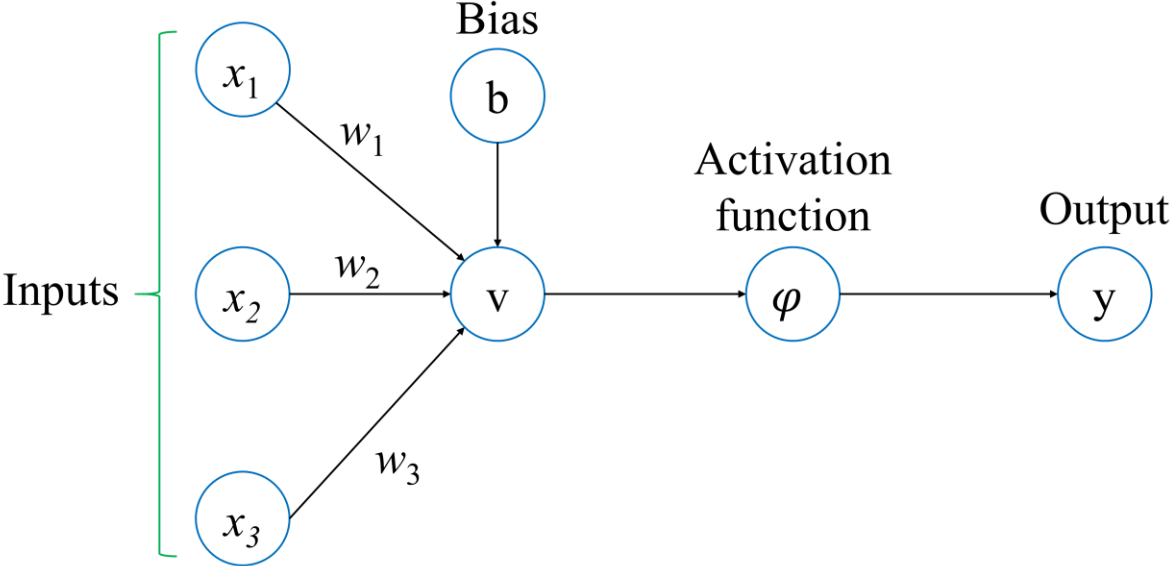

Figure 1: Artificial Neuron

CNNs are supervised algorithms, meaning that for each example input in a dataset, the correct output is known and used for adjusting the network's parameter values. Once the ANN has gone through an iteration of the forward pass, a loss function quantifies how closely the computed actual outputs match the ground-truth outputs recorded in the dataset. The neuron weights are then adjusted to hopefully reduce the loss on the next forward pass. How to make that adjustment was one of the early stumbling blocks in ANN research; it is necessary to decide what the contribution of one weight value out of millions of weights, in our case, contributes to the total loss of the network. This is the credit assignment problem that each multi-layered ANN must resolve. CNNs, again in common with many others, employ the technique of ordered derivatives, which have been independently discovered several times but which were first applied to neural networks by D. Rumelhart and J. McClelland [46] in the Back-propagation algorithm.

At its core, back-propagation is a gradient descent algorithm, formulated as follows:

$$W_{new} = W_{old} - \eta \nabla J(w) ,$$

where W_{new} is the new weight value, W_{old} is the old weight value, η is the learning rate, and $\nabla J(w)$ is the gradient of the loss function. We will discuss the classic back-propagation algorithm for Rumelhart's Multi-Layer Perceptron architecture below [46].

The details of output neuron j along with its incoming signals can be seen in Figure 2 [26]. The back-propagation algorithm for an output neuron j can be computed as follows [26]:

$$e_j(n) = d_j(n) - y_j(n)$$

$$\varepsilon_j(n) = \frac{1}{2} e_j^2(n) = \frac{1}{2}\left(d_j(n) - y_j(n)\right)^2$$

where $\varepsilon_j(n)$ is the squared prediction error between the desired output $d_j(n)$ of the neuron and the actual output $y_j(n)$ of the neuron for the nth data pattern.

The total error energy is the sum of the squared errors for all output neurons:

$$\varepsilon(n) = \sum_{j=1}^{m} \varepsilon_j(n)$$

The chain rule is used to compute the gradient of $\varepsilon(n)$ with respect to $w_{ji}(n)$:

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)},$$

where

$$\frac{\partial \varepsilon(n)}{\partial e_j(n)} = e_j(n)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)}\left(d_j(n) - y_j(n)\right) = -1$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \varphi\left(v_j(n)\right)}{\partial v_j(n)} = \varphi'\left(v_j(n)\right)$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \frac{\partial}{\partial w_{ji}(n)} \sum_{i=1}^{m} w_{ji}(n) y_j(n) = y_i(n)$$

This leads us to:

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = e_j(n) \cdot (-1) \cdot \varphi'\left(v_j(n)\right) \cdot y_i(n)$$

As such:

$$\Delta w_{ji}(n) = -\eta \cdot \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \eta \cdot e_j(n) \cdot \varphi'\left(v_j(n)\right) \cdot y_i(n)$$

The local gradient $\delta_j(n)$ at neuron j is defined as:

$$\delta_j(n) = \frac{\partial \varepsilon(n)}{\partial v_j(n)} = (-1) \cdot e_j(n) \cdot \varphi'\left(v_j(n)\right)$$

$$\delta_j(n) = \left[d_j(n) - y_j(n)\right] \cdot \varphi'\left(v_j(n)\right)$$

The change $\Delta w_{ji}(n)$ to the ith weight for output neuron j becomes:

$$\Delta w_{ji}(n) = -\eta \cdot \delta_j(n) \cdot y_i(n)$$

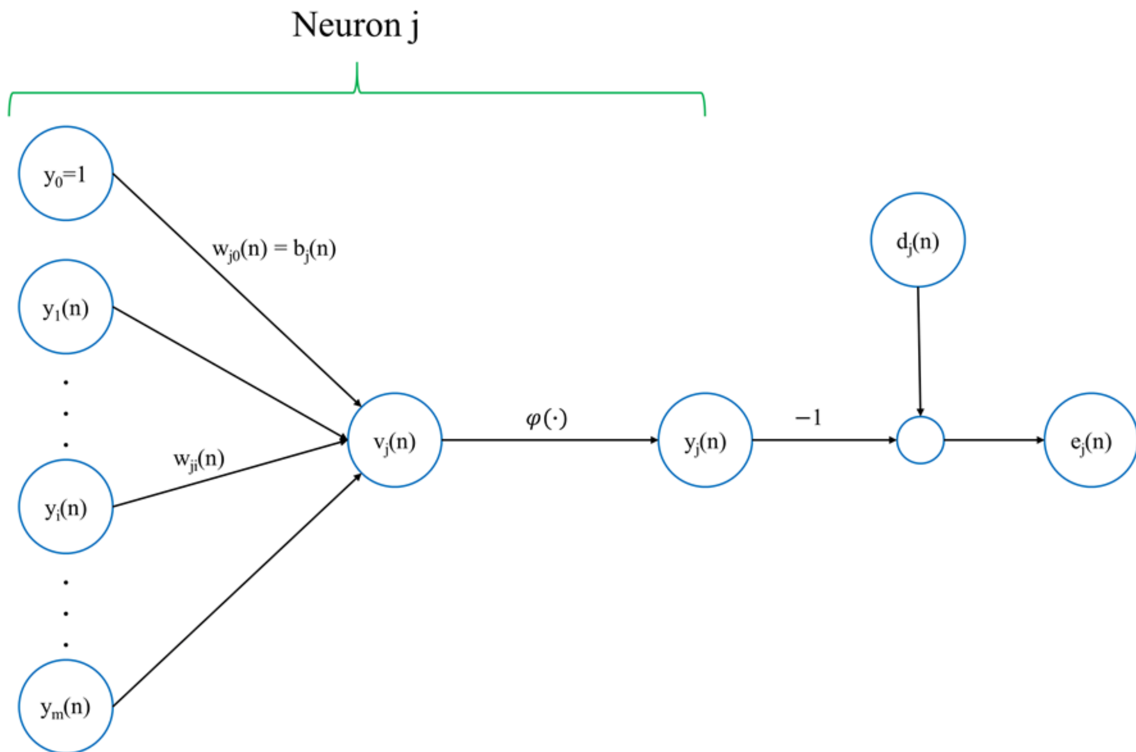

Figure 2: Details of the signal flow in output neuron j [26]

In the case of hidden neuron j being connected to output neuron k, as shown in Figure 3, the back-propagation algorithm for hidden neuron j can be computed as follows [26]:

$$\delta_j(n) = \frac{\partial \varepsilon(n)}{\partial v_j(n)} = \frac{\partial \varepsilon(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \varepsilon(n)}{\partial y_j(n)} \cdot \varphi'\left(v_j(n)\right)$$

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)}\left(\frac{1}{2}\sum_k e_k^2(n)\right) = \frac{\partial \varepsilon(n)}{\partial e_k(n)} \cdot \frac{\partial e_k(n)}{\partial y_j(n)}$$

$$=\sum_k \left[e_k(n) \cdot \frac{\partial e_k(n)}{\partial y_j(n)}\right] = \sum_k \left[e_k(n) \cdot \frac{\partial e_k(n)}{\partial v_k(n)} \cdot \frac{\partial v_k(n)}{\partial y_j(n)}\right]$$

$$=\sum_k \left[e_k(n) \cdot (-1) \cdot \varphi'(v_k(n)) \cdot \frac{\partial v_k(n)}{\partial y_j(n)}\right],$$

where the induced local field due to neuron k is:

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_i(n)$$

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

This leads us to:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k e_k(n) \cdot \varphi'\left(v_k(n)\right) \cdot w_{kj}(n)$$

$$= -\sum_k \delta_k(n) \cdot w_{kj}(n)$$

So, the back-propagation formula for the local gradient for hidden neuron j becomes:

$$\delta_j(n) = \varphi'\left(v_j(n)\right)\sum_k \delta_k(n)\cdot w_{kj}(n)$$

The change $\Delta w_{ji}(n)$ to the ith weight for hidden neuron j becomes:

$$\Delta w_{ji}(n) = \eta\cdot\delta_j(n)\cdot y_i(n)$$

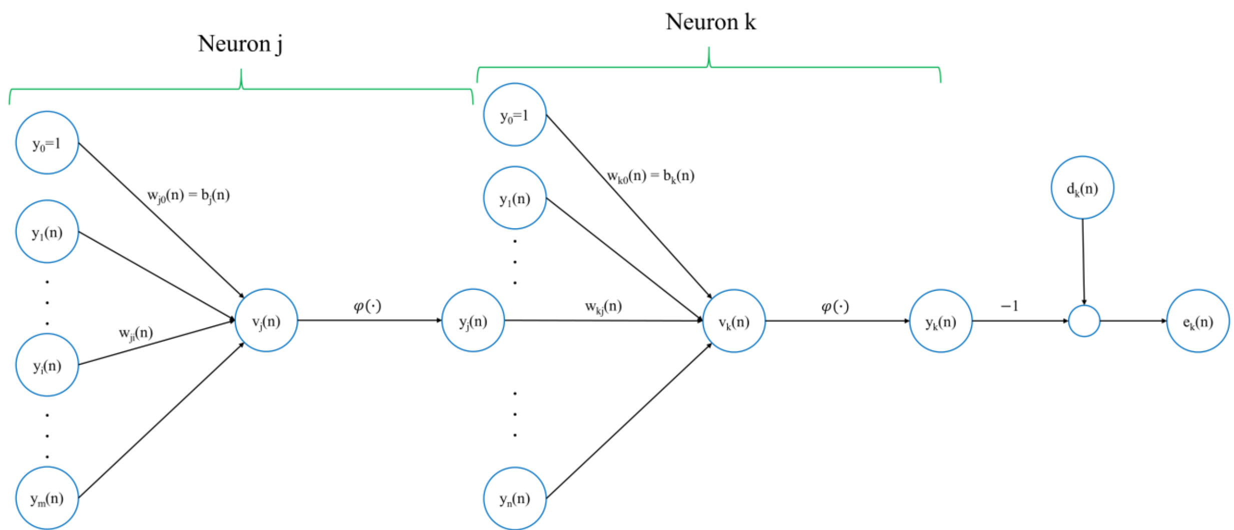

Figure 3: Details of the signal flow in output neuron k connected to hidden neuron j [26]

The optimization of the loss function can sometimes take many iterations until convergence. There are three main optimization approaches including Gradient Descent, Stochastic Gradient Descent (SGD), and Mini-Batch Stochastic Gradient Descent (MBSGD). The difference between Gradient Descent, SGD, and MBSGD is in the number of training samples of a dataset that need to pass through an ANN in each iteration before computing the gradient [47]. In each iteration of Gradient Descent, the gradient is computed once all training samples of a dataset pass through the ANN in the forward pass before any parameters in the ANN can be

updated. On the other hand, in each iteration of SGD, the gradient is computed after only one randomly chosen training sample from the dataset passes through the ANN. Empirically, SGD is faster but does not guarantee reaching the best optimization solution. MBSGD is considered the middle ground between both approaches. The gradient is computed after a subset (mini-batch) of the training samples passes through the ANN.

2.1.1 Convolutional Neural Networks

CNNs are the most commonly used neural networks for image classification. Even though CNNs could be used to tackle other problems, they were originally created for image classification [48]. The origins of CNNs date back to the 1960s with the research of neurophysiologists D. Hubel and T. Wiesel. Their description of simple and complex cells in the human visual cortex [49] later on inspired K. Fukushima to propose the Neocognitron, which is an early neural network model and the seed for the CNN architecture [50]. K. Fukushima's work inspired Y. LeCun et al. in the late 1990s to develop the modern-day CNN [48]. Research in CNNs was stagnant for a while afterwards. However, new advances in computing hardware and the availability of large public image repositories led to revived interest in CNNs. Along with this, a breakthrough in CNNs appeared around ten years ago by A. Krizhevsky et al. [51], who had developed a CNN now popularly known as AlexNet. AlexNet took first place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 [51]. It had showcased revolutionary results for image classification and recognition tasks by reducing the error rate from 25.8 to 16.4 as compared to previous traditional computer vision techniques [52]. The breakthrough came in the form of stacking convolutional layers and increasing the depth of a CNN, forming the first Deep CNN [52]. Since then, research in CNNs has rapidly increased [53].

Training a CNN for image classification requires a database of images annotated with their assigned classes [54]. A CNN architecture has a structure which goes through the processes of feature extraction, feature mapping, subsampling, and classification in the forward pass. Learning in a CNN is accomplished via back-propagation, most commonly in the mini-batching version of SGD. However, the CNN uses a different pattern of interconnections than a Multi-Layer Perceptron. Unlike densely connected layers, convolutional layers learn from local patterns of images instead of global patterns. Local patterns are in the form of an image broken down into small 2D windows making use of features such as textures and edges in an image. Once a pattern is learned by a convolutional layer, it can be recognized in other parts of an image. Dense layers on the other hand have to relearn patterns again if they appear somewhere else in an image. Spatial hierarchies of patterns can also be learned by CNNs.

The deeper the CNN, the more layers are present, and the more convolutional operations take place in the hidden layers. A CNN could consist of thousands or even millions of neurons with also thousands or millions of connections between multiple hidden layers [26]. Feature extraction is performed by convolutional filters, implemented via neurons whose inputs come exclusively from a convolution window. The convolutional filters of the first convolutional layer extract the features of the input image. The output of this convolutional operation is known as a feature map. The resulting feature maps of the first convolutional layer are then considered the input for the next convolutional layer. The relationship between weights, kernels, convolutional filters, kernel matrices, and feature maps is illustrated in Figure 4. In layer i, there are n_i input feature maps fm_i. Each feature map is of height h_i and width w_i. The 4D kernel matrix M_i consists of n_{i+1} 3D filters F_i. The filters F_i consist of n_i 2D kernels k_i. Each kernel k_i consists of s^2 weights $W_{i,j}$ arranged in an $s \times s$ matrix, forming a convolutional mask. Thus, a filter F_i consists of $n_i \times s^2$

weights $W_{i,j}$. The filters F_i are applied to the input feature maps fm_i and give us n_{i+1} feature maps fm_{i+1} as an output.

Just like digital images, filters can be visualized and interpreted as small images including colors and edges [55, 56]. In this case, the values of the pixels are based on the filter weights. As the name suggests, the convolutional filters convolve over the inputs and extract their features. As a convolutional filter is applied to an input image or feature map, it moves across the input in left to right and top to bottom directions. A convolutional filter starts by default from the top left edge of the input where the top left weight of the filter is aligned with the top left pixel of the input. The convolutional filter stops when it reaches the bottom right edge of the input where the bottom right weight of the filter is aligned with the bottom right pixel of the input. The number of pixels it moves at a time horizontally and vertically is called convolutional stride [54]. The default convolutional stride is (1,1), which means that the convolutional filter moves one pixel at a time from left to right of the input and then one pixel at a time from top to bottom of the input. It is important to note that in a convolutional layer, a kernel does not actually move. Rather, each possible location of the input is associated with a separate neuron, and all neurons for a specific convolutional filter in a convolutional layer share the same weights. The convolutional operation is shown in the following formula [47]:

$$C[i,j] = (I * K)[i,j] = \sum_{m}\sum_{n} I[i+m, j+n]K[m,n] \ ,$$

where $'I'$ is the input, $`K'$ is the convolutional kernel, and the indices of the rows and columns of the result, which is usually in the form of a feature map, are represented with i and j, respectively.

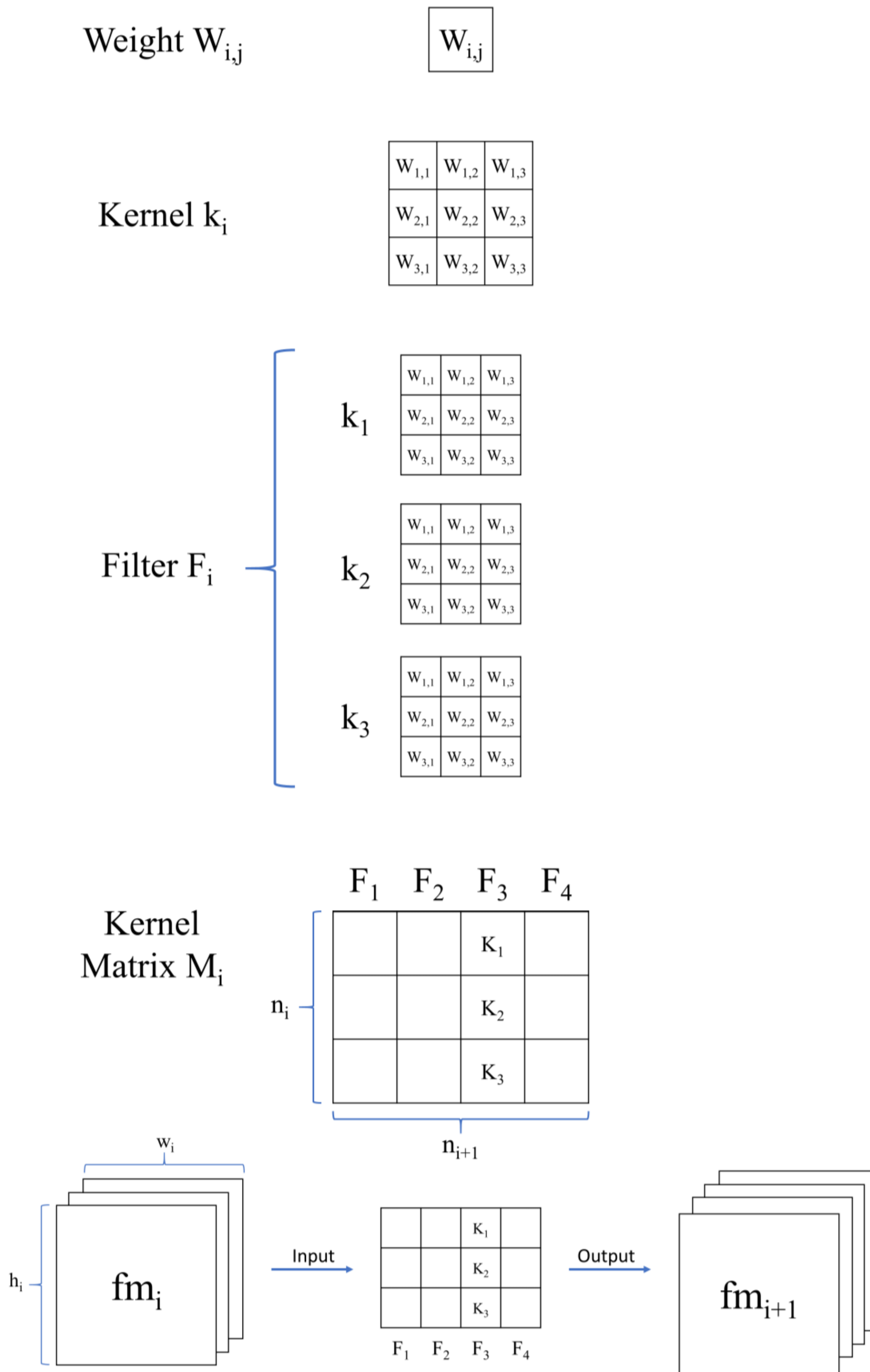

Figure 4: The relationship between weights, kernels, convolutional filters, kernel matrices, and feature maps

The problem with this convolutional operation is that the size of the output is smaller than the input. Therefore, padding is used to counteract the effect of convolutional stride, which decreases the size of the output of a convolutional filter. Padding is adding empty pixels, which consist of zeros, to the sides or frame of an input image or feature map [54]. For example, if we have a 3x3 filter with stride (1,1) applied to an 8x8 input image, the resulting output feature map is of size 6x6. Thus, adding padding is needed if we want the output feature map to have the same size as the input image. In this case, we need padding of 1, which would result in an input of size 10x10, but the extra pixels will not have any effect on the output because the padding only consists of empty pixels. Applying the same 3x3 filter to the 10x10 input image after padding, will result in an output feature map of size 8x8, which is the same size of the original input image.

An important step to consider is that a nonlinear activation function is applied to the output of the previous convolutional operation before they form feature maps. This nonlinearity is essential to the generalizability of neural networks; without it, the network would just be a superposition of linear functions, which is itself linear. However, certain superpositions of nonlinear functions can be universal approximators. The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU) function, given by [47]:

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

An important step that is added to CNNs is normalization. This is to counteract the boundless nature of some activation functions such as ReLU, where the output of some layers is not bounded by a specific range of values. Normalization is usually applied to the output of hidden neurons either before or after entering the activation function. In particular, batch normalization is used because it also addresses another problem that we face in training CNNs, which is called

Covariate Shift. Covariate Shift occurs during the training process of CNNs when the distribution of the inputs of each layer changes while parameters change in previous layers [57]. Batch normalization usually consists of taking the Z-score of a mini-batch relative to its mean; the Z-score has zero mean and unit variance. Therefore, the introduction of batch normalization has led, in some cases, to eliminating the need to use Dropout in CNNs [57], which is a technique where some neurons are deactivated or dropped out randomly from the CNN to avoid any possible overfitting problems [58].

Once features are extracted in convolutional layers and the output is in the form of feature maps, a pooling operation is performed. The purpose of pooling layers is to down sample the feature maps. Sub-sampling helps in making the outputs less sensitive to shifts and distortions of inputs [48]. One of the most commonly used pooling approaches is called Max-Pooling. In max-pooling, a window with a specified size moves across a feature map and extracts the highest value in the portion of the feature map where the max-pooling window is applied [47], and just like convolutional filters, max-pooling also has a stride of its own. This way, the most important and activated features are highlighted, extracted, and passed to the next stage in the CNN. The first convolutional layer would learn a small local pattern and then the second convolutional layer could build on that and learn larger and more complex patterns made up of the smaller local patterns of the first layers. For example, if a max-pooling window of size 2x2 with stride of (2,2) is applied on a feature map of size 4x4, the resulting output will be of size 2x2. The max-pooling operation can be seen in Figure 5.

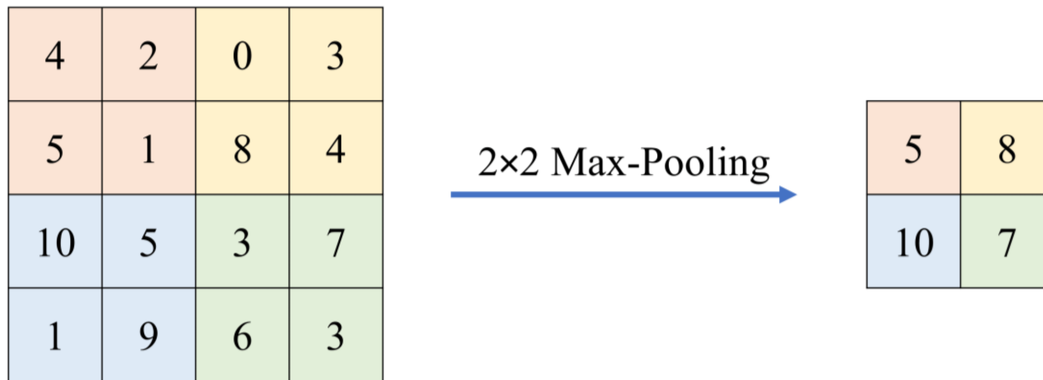

Figure 5: Max-Pooling operation

For classification, a Softmax layer is used. The softmax layer is usually the very last layer in a CNN with a multi-class dataset. The softmax layer is based on the softmax function, which is applied to the outputs of the last fully connected layer. The softmax function is defined as follows [47]:

$$S(y)_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}},$$

where $S(y)_i$ is the ith probability output corresponding to output y_i, $'n'$ is the number of classes in the multi-class dataset, and ‘e’ is the base of the natural logarithm.

Once the CNN has gone through an iteration of the forward pass and the softmax layer provides the resulting output probabilities, it is important to compute the error of the CNN’s accuracy. One of the most commonly used loss functions in CNNs is the Categorical Cross Entropy Loss function [54]. This is specifically used for multi-class datasets. The categorical cross entropy loss function is defined as follows [59]:

$$\text{Loss} = -\sum_{i=1}^{n} y_{i,\text{target}} \cdot \log(y_i)\ ,$$

where $'n'$ is the number of outputs, y_i is the ith output, and $y_{i,target}$ is the ith target output for the input of the CNN. The number of outputs $'n'$ is the same as the number of classes of a dataset. If y_i is the desired output, then $y_{i,target}$ is set to 1. On the other hand, if y_i is not the desired output, then $y_{i,target}$ is set to zero.

For example, if the dataset we are using has 3 classes, the number of outputs $'n'$ will be 3. Let us assume that an input that has passed through a neural network belongs to the second class, the output probabilities $'y'$ for the input are [0.15, 0.8, 0.05], and the input belongs to the second class, which means that the target outputs y_{target} are [0, 1, 0]. This will result in a very small loss, because the output with the highest probability corresponds to the desired target output. This is shown in the following equation:

$$\text{Loss} = -[(0 \times \log{(0.15)}) + (1 \times \log{(0.8)}) + (0 \times \log{(0.05)})] = 0.097$$

Using back-propagation, the CNN's weights are then tweaked to improve the accuracy of the CNN by reducing the error.

Today, there are various architectures of CNNs such as LeNets [48], AlexNets [51], VGGNets [42], GoogLeNets [60], ResNets [61], and DenseNets [62] with varying advantages and disadvantages to all of them. Our study focuses on the VGG-16 network, a popular deep CNN by K. Simonyan and A. Zisserman [42]. It is part of a series of VGGNets, CNNs with varying layer depth. Their work won the first and second places in the localization and classification tasks in ILSVRC 2014 [42], respectively. It is considered one of the most popular CNN models. The idea

behind their work was to see the effects of increasing the depth of a CNN architecture. This was possible because of their use of small 3x3 convolutional kernels in all convolutional layers. VGGNets' sizes range from 11 weight layers in the smallest architecture to 19 weight layers in the largest architecture. With this increase in depth, there is an increase in the number of weights, also sometimes called parameters. As such, they are considered very time consuming and computationally intensive architectures to train and test. Yet, it has been observed that the classification error of the architectures decreases with the increased depth. VGG-16 consists of 22 different layers. These include: 16 weight layers, 5 spatial pooling layers, and one softmax layer. Of the 16 weight layers, 13 are convolutional layers and 3 are fully connected layers. All 5 spatial pooling layers use max-pooling. The architecture for the original VGG-16 CNN is shown in Figure 6. The values next to the name of each layer represent the height, width, and number of feature maps/channels in each layer.

The convolutional and fully connected layers use ReLU activation functions. The number of filters in the convolutional layers ranges from 64 to 512. The number of filters increase by a factor of two after every max-pooling layer, until they reach 512 filters. The first two layers of VGG-16 are convolutional layers with 64 filters each. Convolutional layers 3 and 4 have 128 filters each. Convolutional layers 5, 6, and 7 have 256 filters each. The remaining 6 convolutional layers, 7 until 13, have 512 filters each. The first two fully connected layers have 4,096 channels, and the last fully connected layer has 1,000 channels. A convolutional stride of size (1,1) pixel along with padding of size 1 pixel were used. Max-pooling was done with a window size of 2x2 pixels along with stride of size (2,2). No padding was used for the max-pooling layers.

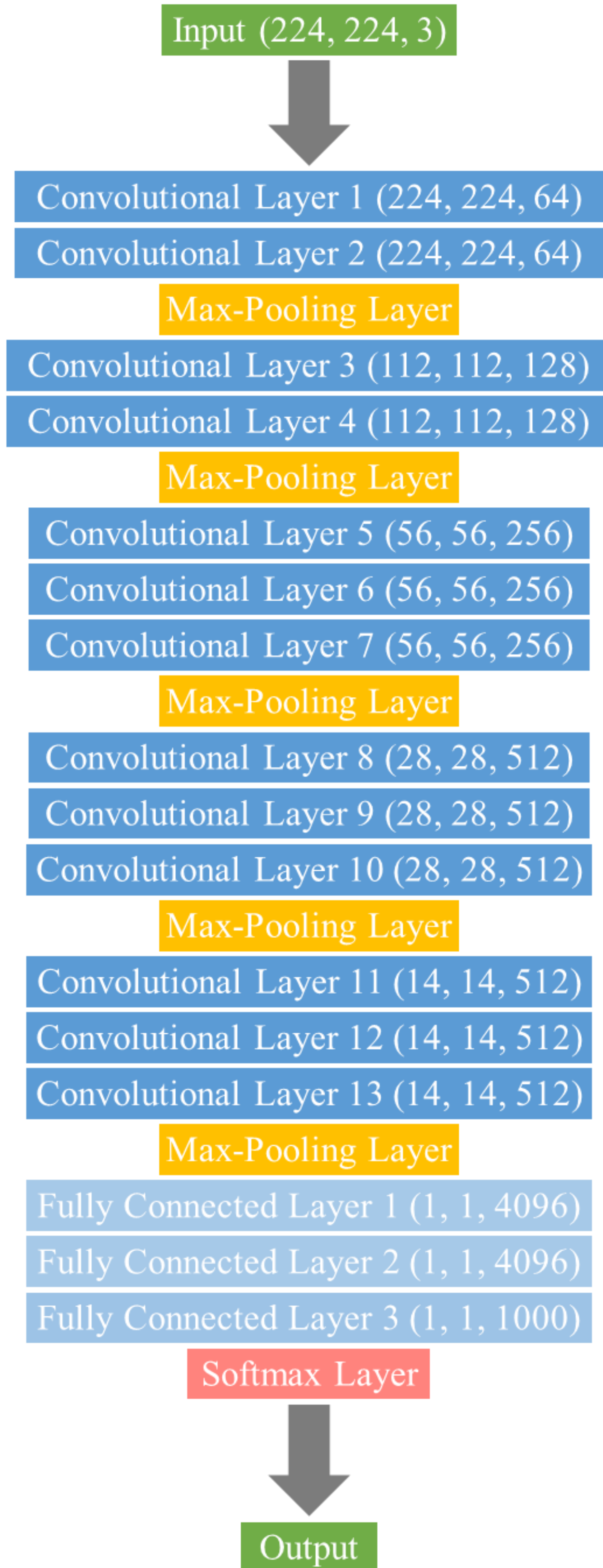

Figure 6: Original VGG-16 architecture

The difference between CNNs and regular neural networks is that convolutional operations take place between the inputs and weights instead of matrix multiplications [47]. As seen in Figure 7, the first iteration of the convolutional operation is computed. Filter F_1, which consists of three kernels, is applied to the input image. Each weight is given an index corresponding to its position, the kernel it belongs to, and the filter it belongs to. For example, the first weight in the top left corner of the first kernel in filter 1 is labeled as $w_{1,1,1,1}$. The first two numbers in the subscript tell us that the weight is at the first row and first column of the kernel. The third number tells us that the weight belongs to kernel '1', and the fourth number tells us that the weight belongs to filter '1'. Each kernel is applied to only one channel of the input image. The input image is of size (224, 224, 3). The first two dimensions are the height and width, and the third dimension is the number of channels and since it is an RGB image, there are three channels. There is one channel for each color (red, green, and blue). A padding of 1 is added to each channel of the input image. Each pixel value is given an index corresponding to its position and the channel it belongs to. For example, the first pixel in the top left corner of the red channel is labeled as $x_{1,1,1}$. The first two numbers in the subscript refer to the row and column indices of the input pixel, and the third number tells us that the input pixel belongs to channel '1'. One filter applied to one input image results in one feature map. Each pixel is multiplied by its corresponding weight, then all multiplication results are summed together, in addition to a bias value, to give us a single output in a feature map. An output feature map in the first convolutional layer in VGG-16 is of size (224, 224). These correspond to the height and width. Each output is given an index corresponding to its position. For example, the first output in the top left corner of the feature map is labeled as $y_{1,1,1}$. The first two numbers in the subscript refer to the row and column indices of the output, and the third number tells us that the output belongs to feature map '1'.

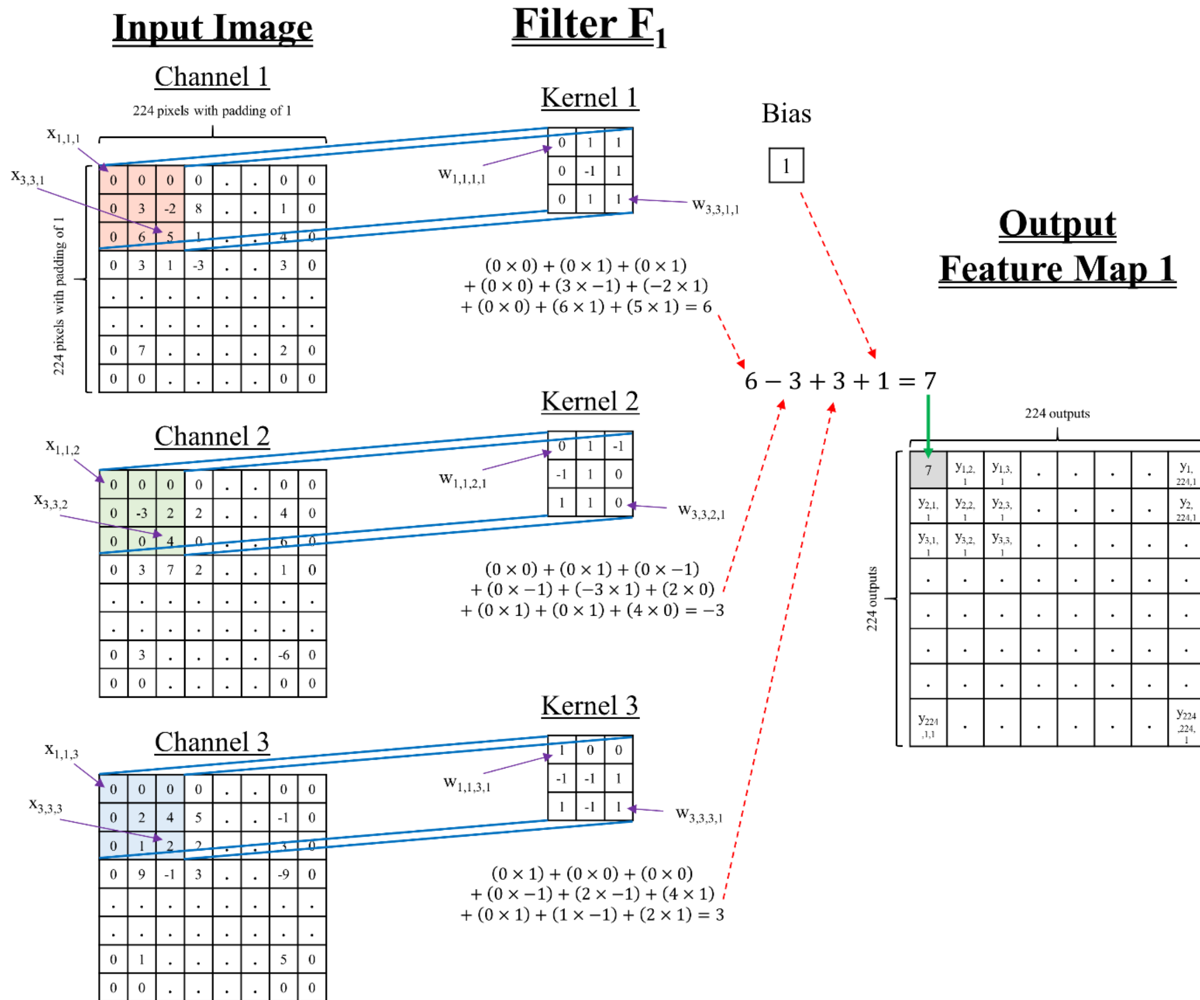

Figure 7: First iteration of the convolutional operation in convolutional layer 1

The operation seen in Figure 7 is done using only one neuron in the first convolutional layer in VGG-16. This is further illustrated in more detail in Figure 8, where there is a total of 27 inputs and 27 corresponding weights. In this case, 9 pixels in each of the three channels of the input image are multiplied by 9 weights in each of the three kernels of the filter. The summation result then passes through the activation function, which in VGG-16 is the ReLU activation function, and then gives us a single output.

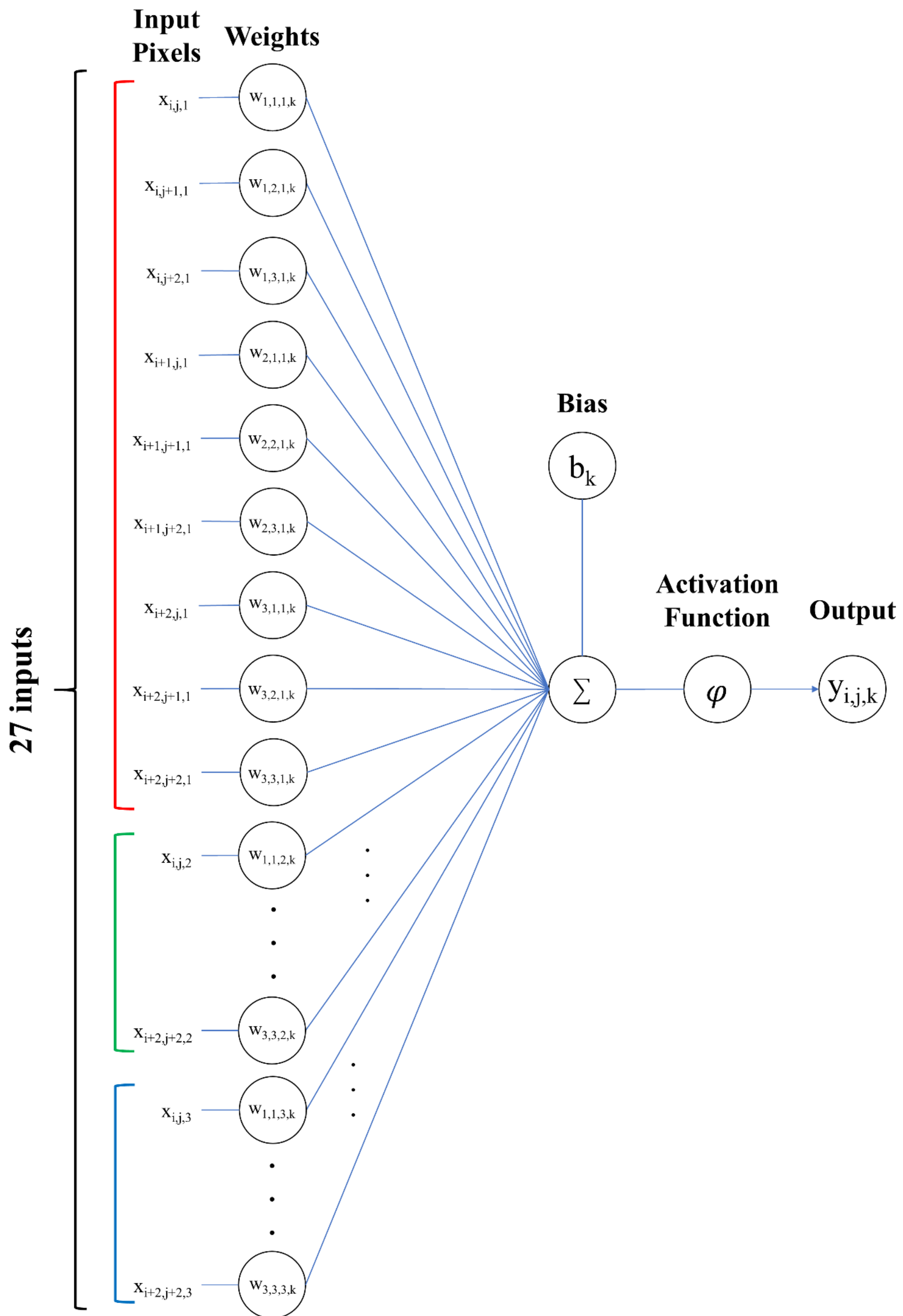

Figure 8: A neuron in convolutional layer 1 in VGG-16

The kernels then convolve across the input image channels to get us the remaining outputs in a feature map. Since there are $224 \times 224 = 50{,}176$ outputs in a single feature map in the first convolutional layer in VGG-16, this also means that $50{,}176$ neurons are needed to generate a single feature map, where all neurons contain the same weights. This is shown in Figure 9.

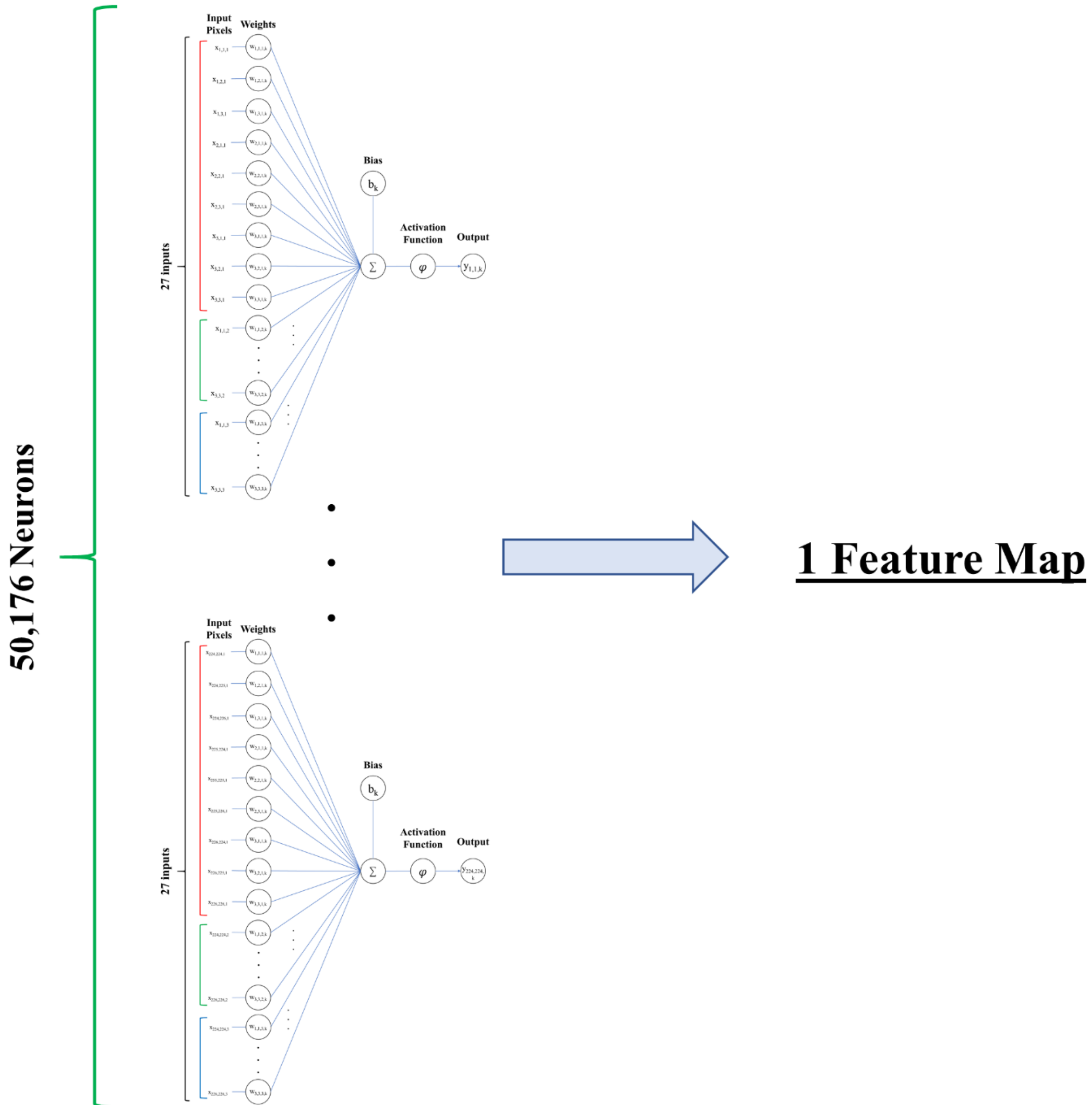

Figure 9: Neurons needed to generate 1 feature map in convolutional layer 1 in VGG-16

The convolution operation is then repeated for each of the 64 convolutional filters in convolutional layer 1 in VGG-16. This means that there is a total of $64 \times 50{,}176 = 3{,}211{,}264$ neurons in convolutional layer 1.

In the case of convolutional layer 2, the first iteration of the convolutional operation is computed. This is illustrated in Figure 10. In convolutional layer 2, each filter consists of 64 kernels corresponding to the 64 feature maps of convolutional layer 1, which are considered the input for convolutional layer 2. Just like in convolutional layer 1, each weight in convolutional layer 2 is given an index corresponding to its position, the kernel it belongs to, and the filter it belongs to. Each kernel is applied to only one input feature map. A padding of 1 is added to each input feature map of convolutional layer 1. Each input value is given an index corresponding to its position and the input feature map it belongs to. In convolutional layer 2, one filter applied to all 64 input feature maps of convolutional layer 1 results in only one new output feature map. An output feature map in convolutional layer 2 in VGG-16 is also of size (224, 224).

The operation seen in Figure 10 is done using only one neuron in convolutional layer 2 in VGG-16. This is further illustrated in more detail in Figure 11, where there is a total of 576 inputs and 576 corresponding weights. In this case, 9 pixels in each of the 64 input feature maps are multiplied by 9 weights in each of the 64 kernels of the filter. The summation result then passes through the activation function and then gives us a single output. The kernels then convolve across the 64 input feature maps to get us the remaining outputs in an output feature map. Since there are $224 \times 224 = 50{,}176$ outputs in a single feature map in convolutional layer 2 in VGG-16, this also means that $50{,}176$ neurons are needed to generate a single feature map, where all neurons contain the same weights. This is shown in Figure 12.

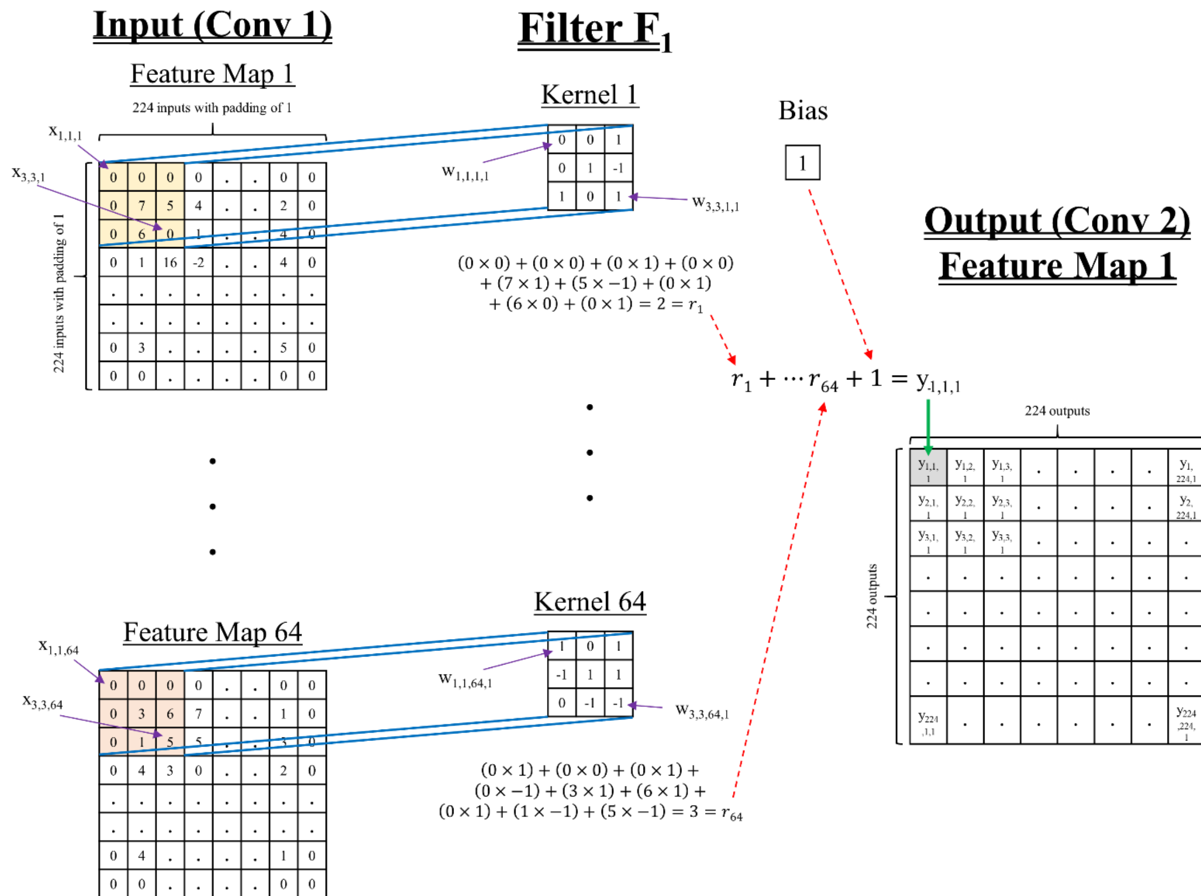

Figure 10: First iteration of the convolutional operation in convolutional layer 2

The convolution operation is then repeated for each of the 64 convolutional filters in convolutional layer 2 in VGG-16. This means that there is a total of $64 \times 50{,}176 = 3{,}211{,}264$ neurons in convolutional layer 2.

Figure 13 shows the number of channels for the input image and the number of feature maps in both convolutional layers 1 and 2.

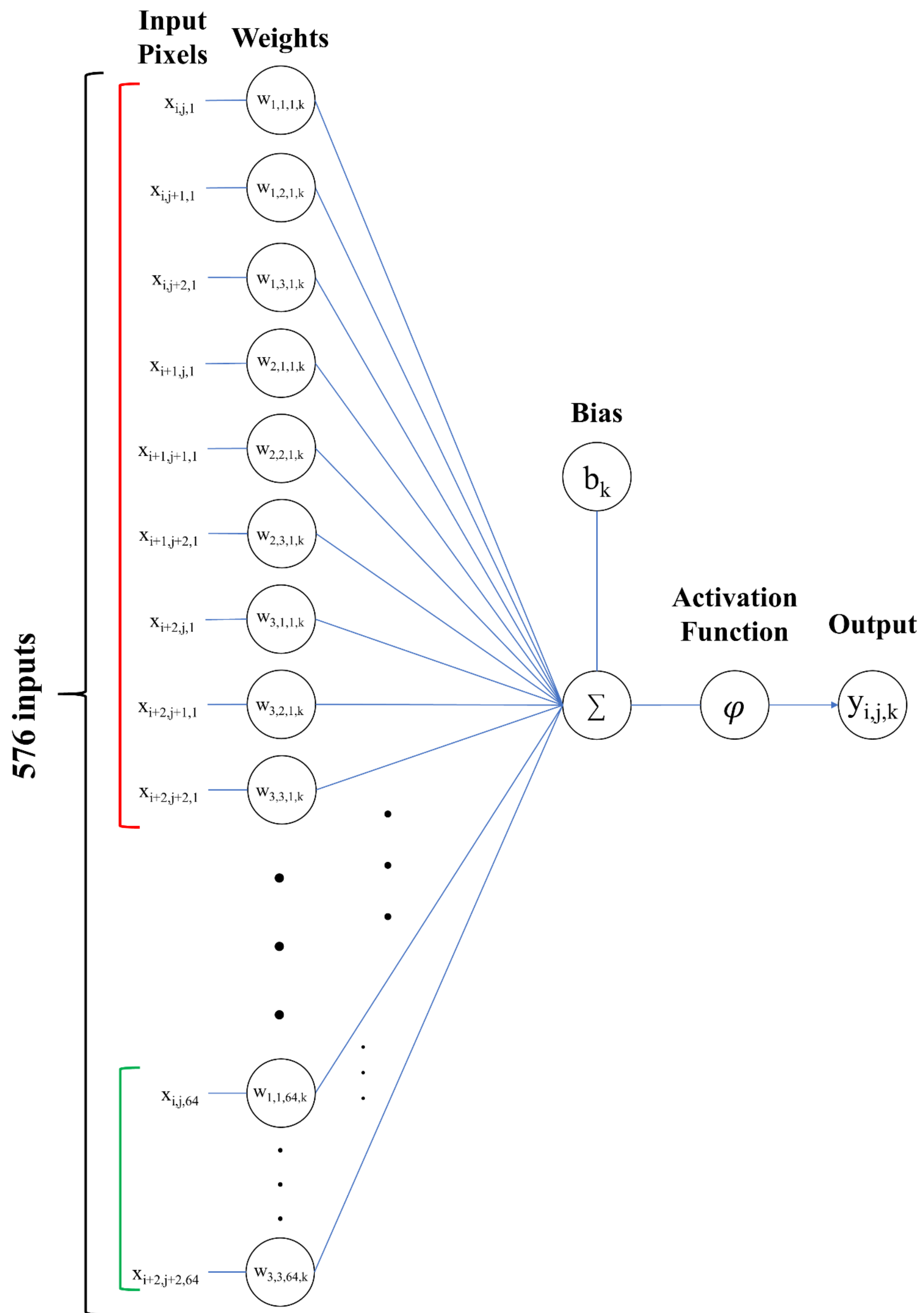

Figure 11: A neuron in convolutional layer 2 in VGG-16

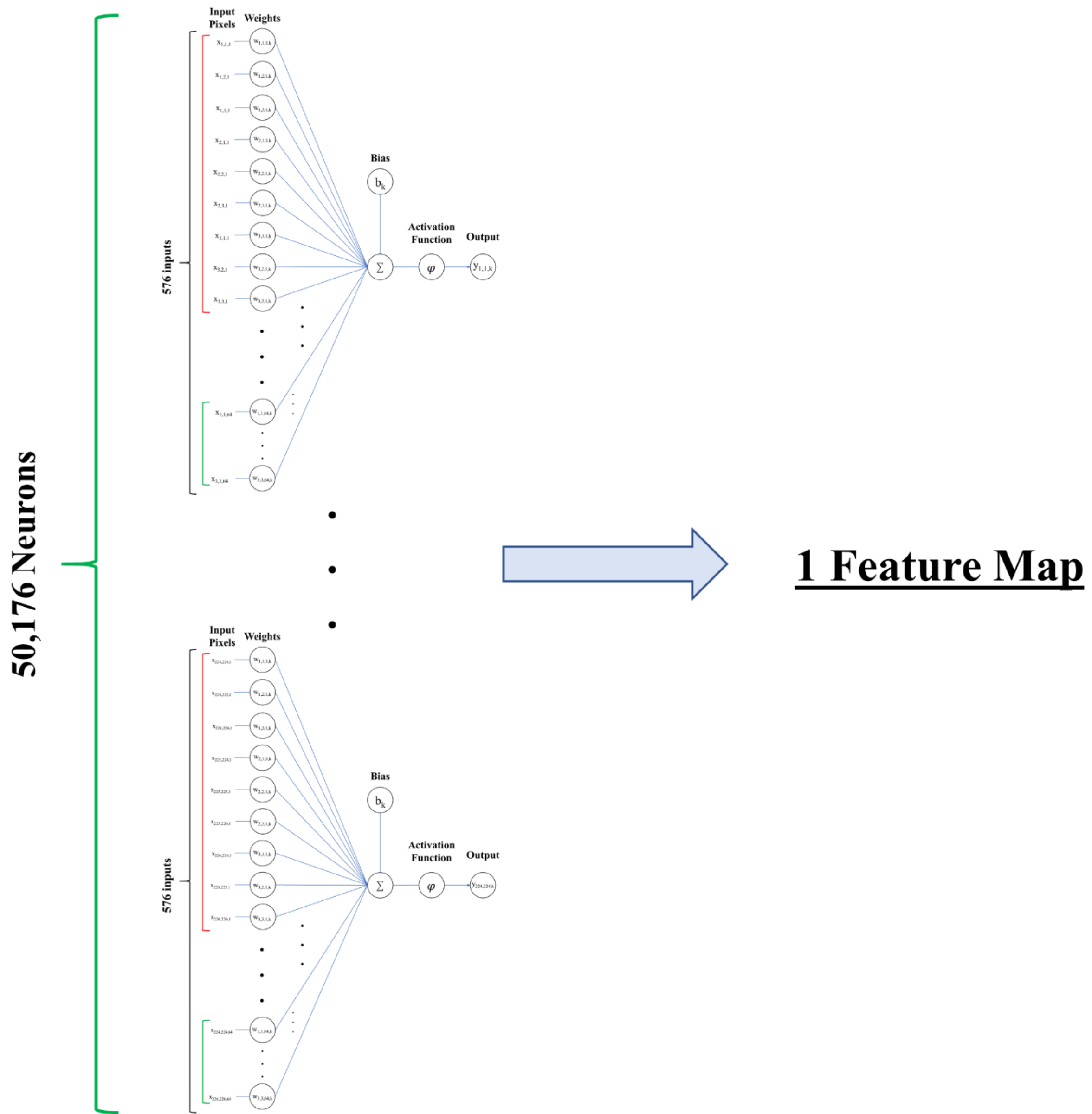

Figure 12: Neurons needed to generate 1 feature map in convolutional layer 2 in VGG-16

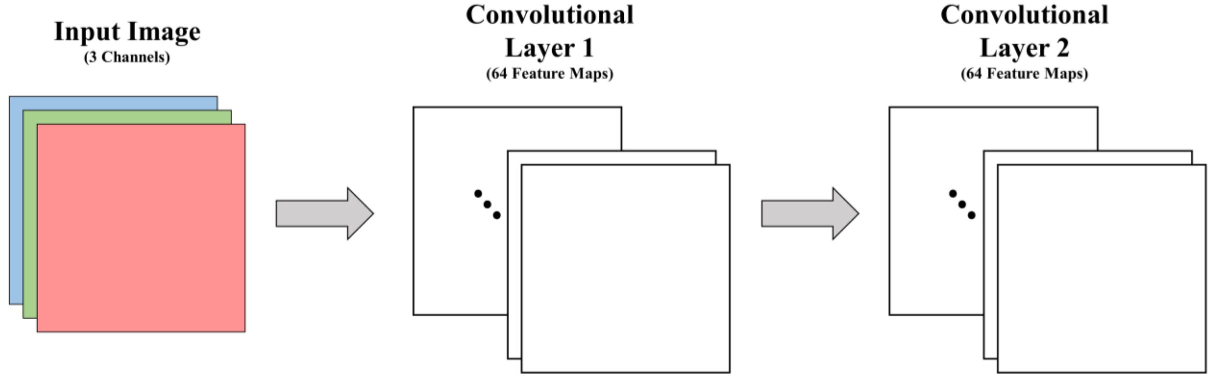

Figure 13: High level view of feature maps in convolutional layers 1 and 2 in VGG-16

The convolutional operation can be expressed mathematically as follows [63]:

$$v^{[p]}_{(i,j,k)} = \left(x * w^{[p]}\right)(i,j,k) + b^{[p]}_{(k,1)} = \sum_{l=1}^{3}\sum_{m=1}^{3}\sum_{n=1}^{3} x_{(i+l-1,j+m-1,n)} w^{[p]}_{(l,m,n,k)} + b^{[p]}_{(k,1)} ,$$

where 'i' and 'j' are the indices of the output 'v' of the convolutional operation in feature map $'k'$. The variables 'l', 'm', and $'n'$ are the row number, column number, and channel number for 'x' in the input image or feature map. In the case of weight 'w', the variables 'l', 'm', $'n'$, and $'k'$ are the row number, column number, kernel number, and filter it belongs to. The superscript $'[p]'$ is the number of the layer.

Once we have the convolutional operation outputs, the activation function is applied on them as follows:

$$a^{[p]} = \varphi(v^{[p]}) ,$$

where the activation $'\varphi'$ is applied element-wise to every element in the outputs 'v'.

This convolutional operation repeats itself in each convolutional layer in the VGG-16 architecture and the forward pass of VGG-16 goes through the next layers, such as the max-pooling layers and softmax layer, of the architecture with each layer operating the same way as explained earlier. Once the forward pass is over, it is time to compute the error of the VGG-16 CNN and perform back-propagation to tweak the parameters to improve the accuracy of the VGG-16 CNN by reducing the error. For the sake of simplicity, we will assume that the input image has one channel and that the input image or input feature map is of size H×W, which results in an output feature map of size (H-2) × (W-2). The back-propagation algorithm can be performed by making use of the chain rule to compute the gradient with respect to weight $w_{l',m'}$ [64]:

$$\frac{\partial \varepsilon}{\partial w_{l',m'}} = \sum_{i=1}^{H-2} \sum_{j=1}^{W-2} \frac{\partial \varepsilon}{\partial v_{i,j}} \frac{\partial v_{i,j}}{\partial w_{l',m'}}$$

$$= \sum_{i=1}^{H-2} \sum_{j=1}^{W-2} \delta_{i,j} \frac{\partial v_{i,j}}{\partial w_{l',m'}}$$

To compute $\frac{\partial v_{i,j}}{\partial w_{l',m'}}$, we substitute $v_{i,j}$ with the appropriate convolution formula. The equation becomes [64]:

$$\frac{\partial v_{i,j}}{\partial w_{l',m'}} = \frac{\partial}{\partial w_{l',m'}} \left(\sum_{l=1}^{3} \sum_{m=1}^{3} x_{(i+l-1,j+m-1)} w_{l,m} + b \right)$$

If we expand the summations and take the partial derivatives for all components with respect to $w_{l',m'}$, this will result in zero values except when $l = l'$ and $m = m'$ [64]. The equation becomes [64]:

$$\frac{\partial v_{i,j}}{\partial w_{l',m'}} = \frac{\partial}{\partial w_{l',m'}}\left(x_{(i,j)}w_{1,1} + \cdots x_{(i+l',j+m')}w_{l',m'} + \cdots + b\right)$$

$$= \frac{\partial}{\partial w_{l',m'}}\left(x_{(i+l',j+m')}w_{l',m'}\right) = x_{(i+l',j+m')}$$

This leads to [64]:

$$\frac{\partial \varepsilon}{\partial w_{l',m'}} = \sum_{i=1}^{H-2}\sum_{j=1}^{W-2} \delta_{i,j} x_{(i+l',j+m')}$$

The change $\Delta w_{l',m'}$ becomes:

$$\Delta w_{l',m'} = -\eta \cdot \left(\sum_{i=1}^{H-2}\sum_{j=1}^{W-2} \delta_{i,j} x_{(i+l',j+m')}\right)$$

The two summations in the equation are the result of weight sharing in the CNN, i.e., the same weights are convolved over the entire input image or input feature map [64].

2.2 Filter Pruning

ANN compression is the process of reducing the size of a network with minimal compromise to the accuracy. The aim is to improve the generalization of neural networks and to make running them faster and possible on resource-limited hardware devices. Earlier work done on compressing neural networks in the late 1980s and early 1990s were mainly based on two approaches, either employing weight-decay terms or saliency criteria [65, 66]. The first approach was utilized by Y. Chauvin [67] and A. Weigend et al. [68]. They adjusted custom error functions used for back-propagation during training. Theses error functions contained weight-decay terms such that if the magnitudes of some weights went below a certain threshold, they were pruned. The second approach uses saliency criteria to measure the sensitivity of error functions with respect to either weights or neurons. This gives an indication of the importance of weights or neurons. M. Mozer and P. Smolensky [69] measured the sensitivity of an error function with respect to neurons. Y. LeCun et al. [70] and B. Hassibi et al. [71] measured the Hessian of an error function with respect to weights. The work done on compressing neural networks during the early 1990s inspired a lot of the research done in this field during the past decade. It was apparent that with the usefulness of deeper CNNs, there had to be a way to deploy them in smaller devices. Research in this field has expanded. Currently, the four most popular techniques utilized for compressing CNNs are:

1) Knowledge Distillation

   Knowledge distillation is the process of training a large CNN model, sometimes known as a teacher, and transferring the knowledge of the large CNN model learned by using its predictions to train a smaller CNN model, sometimes known as a student. The idea behind this approach is that large CNNs have a higher knowledge capacity than small CNNs, but there

may be the possibility that this capacity is not fully utilized and there are many redundancies [34].

2) Parameter Quantization

Parameter quantization works by converting weights inside filters from floating-point numbers to fixed-point numbers. Fixed-point operations require less computational resources, as they are easier and faster to operate on. In addition, they occupy a smaller memory footprint, which means larger models are suitable to use in case of limited memory capacities and bandwidth requirements [33].

3) Filter Compression

Filter compression finds approximations for convolutional filters that are computationally more efficient while preserving the CNN's accuracy. In other words, the number of parameters is reduced. The new approximated convolutional filters are essentially cleaned up versions (free of redundancies) of the original convolutional filters [35].

4) Network Pruning

Network pruning is the process of removing parameters to reduce the size of a CNN [31]. There are two kinds of network pruning methods: weight pruning and filter pruning.

Weight pruning is the process of removing weights, which do not contribute much to the accuracy of a CNN, within filters [36]. The pruned weights usually have small magnitudes below a certain threshold value and are deemed unimportant [72]. The pruned CNN ends up preserving its original architecture, but as a result of the pruning process, becomes more sparse. The weight pruning process is usually done with a binary mask, which has the same size of the convolutional filter, consisting of zeros and ones. If a weight is to be pruned, its equivalent index in the mask

matrix will have a value of zero. This way, the weight's value in the convolutional filter is overwritten and set to zero. If a weight is important, its equivalent index in the mask matrix will have a value of 1. This way, the weight keeps its original value without change. The problem we are faced with after weight pruning is that it leads to sparse weight matrices across the network and often requires using specialized software and hardware [31, 37, 38].

Filter pruning on the other hand, is the process of removing entire filters, which are deemed redundant and unimportant, without reducing the accuracy of a CNN [31]. The three main steps in any filter pruning algorithm are: 1) determining which filters are important and which are redundant, 2) pruning the redundant filters, and 3) retraining the neural network. Retraining is important because it compensates for any possible drop in accuracy due to removing filters. There are various methods through which a filter may be deemed redundant. The filter pruning process is done with a binary mask, which has the same size of the convolutional filter, consisting entirely of zeros. Unlike in weight pruning, filter pruning does not introduce sparsity into a CNN's architecture because zero values are not scattered across the weight matrices that make up filters. Hence, using specialized software and hardware is not required. Therefore, implementing filter pruning is a better option. The difference between weight pruning and filter pruning can be illustrated with the following simple example shown in Figure 14. In the case of a 3x3 convolutional filter, we would have 9 weights. In weight pruning, we would set certain weights to zero based on some defined criteria. In filter pruning, the entire filter (all 9 weights) will be set to zero because the filter was deemed redundant based on some defined criteria.

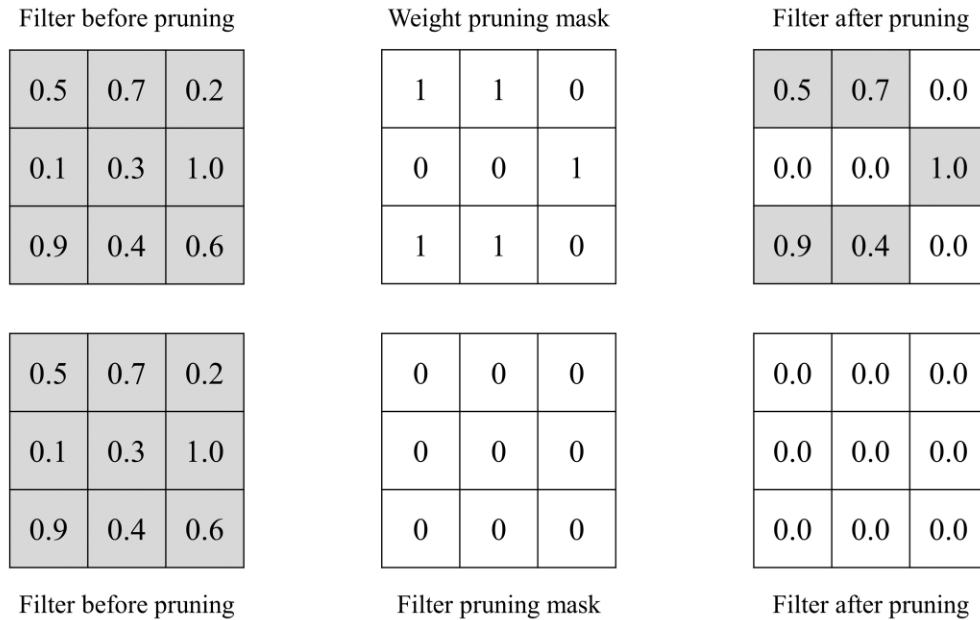

Figure 14: The difference between weight pruning and filter pruning

Recent work on filter pruning methods has produced some very promising ideas. P. Molchanov et al. [40] determined the importance of filters based on first order gradient information. Based on this, the least important filters were pruned. H. Hu et al. [41] took advantage of the sparsity of outputs in a CNN to prune filters which have a large portion of their activations as zero. Y. He et al. [73] proposed an iterative two-step algorithm to prune redundant filters. The algorithm was based on using least absolute shrinkage, selection operator regression, and least square reconstruction of feature maps. H. Li. et al. [37] based their filter pruning algorithm on computing the sum of the absolute weights for filters, referred to as the L_1-norm. They proposed that filters with small L_1-norm values are less important and can be pruned. J. Luo et al. [74] proposed tackling filter pruning as an optimization problem. Filters were pruned based on statistical information from the following layer, not the layer the filter was in. R. Yu et al. [75] proposed propagating importance scores from the second-to-last layer before classification to every filter in a CNN. The

least important filters were pruned based on these scores. Y. He et al. [76] calculated the geometric median of filters in each convolutional layer. The filters nearest to the geometric median were pruned. M. Lin et al. [31] proposed a filter pruning algorithm called HRank. They defined an information measurement to rank feature maps. The information measurement determines the information richness of a feature map. The feature maps are ranked based on their information richness, where a feature map with important information would have a high ranking, while a feature map with little information would have a low ranking. If a feature map is low ranked, then the filter it was produced by is not important to the CNN. This means that the filter can be pruned without affecting the accuracy of the CNN. On the other hand, if a feature map is high ranked, then the filter it was produced by is important to the CNN. This means that the filter should not be pruned. HRank outperformed other state-of-the-art filter pruning algorithms [31].

2.3 Clustering Algorithms

Clustering refers to a class of unsupervised learning that partitions datasets into subgroups, known as clusters. The aim of using clustering algorithms is to group the most similar data points into one cluster while being dissimilar to all other clusters [77]. This will result in identifying possible natural structures in certain datasets. Clustering is used in a wide range of fields such as image segmentation [78], data mining [79], pattern recognition [80], economics [81], biology [82], social sciences [83], and so much more [84, 85].

It is important to note that a perfect clustering algorithm does not exist. Furthermore, clustering is often applied to datasets without any ground truth labels; in this case, even determining what the optimal division of the dataset might be likely impossible. This extends to

when the same clustering algorithm is used but with different parameters and configurations, or different clustering algorithms are used.

Clustering algorithms can be generally categorized into the following main types [77, 86, 87]:

1) Partitional clustering

   Partitional clustering partitions the dataset into a predetermined number of clusters where data points are clustered based on a distance metric. They are highly dependent on setting an initial parameter for the number of centroids. One of the most popular partitional clustering algorithms is K-Means clustering.

2) Hierarchical clustering

   Hierarchical clustering decomposes the dataset based on a certain hierarchy. The clusters created are of tree like partitions called a dendrogram. There are two approaches to hierarchical clustering: Agglomerative and Divisive. The agglomerative approach follows a bottom-up flow for creating the clusters. The number of clusters decreases after each step because two or more clusters are merged into one new cluster. On the other hand, the divisive approach follows a top-down flow for creating the clusters. The number of clusters increases after each step because a cluster is split into two or more new clusters. A predetermined number of clusters is not required in hierarchical clustering.

3) Density-based clustering

   Density-based clustering works by discovering areas of concentration or density for data points and areas of emptiness where the concentrated data points are separated. Usually, data points that are not part of clusters, in other words concentrations, are considered noise.

4) Grid-based clustering

    In this type of clustering, the data space is partitioned into a finite number of cells, thus creating a grid like structure. The cells are then sorted based on their densities and consequentially clustered. These are efficient to use in clustering large multidimensional datasets.

Another categorization of clustering algorithms depends on the cluster overlapping. These include [77]:

1) Crisp clustering

    Sometimes also known as hard clustering. Crisp clustering results in non-overlapping partitions where a data point belongs to only one cluster. The result of most clustering algorithms is crisp clusters.

2) Fuzzy clustering

    Sometimes also known as soft clustering. Fuzzy clustering makes use of fuzzy techniques to cluster datasets where a data point may belong to more than one cluster. The resulting clustering schemes are usually compatible with daily life experiences where the uncertainty of real data is present. The most popular fuzzy clustering algorithm is Fuzzy C-Means clustering [88].

Clustering algorithms can be further subcategorized depending on the type of datasets used. These include [77]:

1) Statistical

    Similarity measures are used to partition numeric data points that are centered around statistical analysis concepts.

2) Conceptual

These cluster categorical datasets based on the concepts they hold.

Of the many different clustering algorithms available, we chose to use K-Means clustering. K-means is considered one of the simplest, most scalable, and most well-known and widely used partitional clustering algorithms. It is thus a reasonable first candidate for our filter pruning algorithm.

2.3.1 K-Means Clustering

The basic idea of K-Means clustering was first proposed by S. Lloyd [89], then it was further adapted and enhanced by E. Forgy [90] and J. Macqueen [91]. K-Means clustering is considered part of the partitional clustering family and is specifically used in crisp clustering. The objective of K-Means clustering, just like all clustering algorithms, is to maximize the inter-cluster distances and minimize the intra-cluster distances. The K-Means clustering algorithm goes through three main steps: First, initialization of centroids. Second, each data point is assigned a label depending on which cluster it belongs to. This is usually based on calculating the Euclidean distance [92] for each data point to each centroid and assigning the data point to the cluster with the nearest centroid. This is done as follows [93]:

$$A(x_i) \leftarrow \underset{j \in 1 \ldots K}{\operatorname{argmin}} \left\| x_i - C_j \right\|^2 ,$$

where $A(x_i)$ is the assignment function for each data point x_i, and the distance between each data point x_i and each centroid C_j, where $j \in 1 \ldots K$, is computed using Euclidean distance, and the data point is assigned to the cluster that its centroid has the minimum distance among all centroids.

Third, once all data points are assigned to clusters, centroids are re-evaluated by taking the mean position for all data points in a certain cluster. The formula used for centroid re-estimation is as follows [93]:

$$C_j^{(t+1)} \leftarrow \frac{1}{\left|P_j^{(t)}\right|} \sum_{x_i \in P_j^{(t)}} x_i \,,$$

where $C_j^{(t+1)}$ is the new centroid for cluster partition $P_j^{(t)}$, and x_i is a data point within cluster partition $P_j^{(t)}$. If the new centroid position is different from the old position, we repeat the second and third steps until convergence, i.e., the centroids do not change positions. The K-Means clustering algorithm is given below:

K-Means Clustering Algorithm

---

1) Initialization: Choose 'K' centroids C_i randomly

Repeat until convergence:

2) Cluster assignment:

for i = 1… n:

Set $A(x_i)$ using $A(x_i) \leftarrow \underset{j \in 1 \ldots K}{\operatorname{argmin}} \left\| x_i - C_j \right\|^2$

3) Centroid re-estimation:

For j = 1… K:

Set C_j using $C_j^{(t+1)} \leftarrow \frac{1}{\left|P_j^{(t)}\right|} \sum_{x_i \in P_j^{(t)}} x_i$

The repetition of steps 2 and 3 is known as alternating optimization [94]. Alternating optimization is an iterative technique for dealing with optimization functions with many variables. This is used in case no clear solution exists to optimize the desired function for all its variables simultaneously. The optimization problem is tackled by solving for a subset of a function's variables, while other subsets of variables are constant. We then alternate between solving these subproblems. This makes optimization problems easier to tackle by splitting the problem into multiple smaller subproblems, where one subset of variables is fixed, while the other subsets of variables are changing, and vice versa. In K-Means clustering, we alternate between the two steps of assigning labels to data points and re-estimating centroids.

In order to initialize centroids, the number of clusters 'K' needs to be determined. Determining the most appropriate number for 'K' is usually tricky, which is why a process called cluster validation has to be performed to aid in this dilemma. Determining how to initialize the centroids from our dataset is also very important. Initializing centroids in a suboptimal way often leads to poor clustering results. The most basic idea for initializing centroids is by selecting them randomly from the dataset [90]. This means that not only two successive runs of the K-Means clustering algorithm on the same dataset will most likely result in different clustering outcomes, but also that there is no guarantee that the randomly selected centroids are of properly spaced-out clusters. There is also the possibility that an outlier in the dataset may be selected as a centroid.

There have been many proposed ideas to tackle the pitfalls that accompany the initialization step over the years, all of which have brought their fair share of advantages and disadvantages. One of the most notable proposed ideas was K-Means++ by D. Arthur and S. Vassilvitskii [95]. The initialization step in K-Means++ only selects the first centroid at random. The next centroid is selected based on a probability directly proportional to the distance between the initial centroid

and all remaining data points. In other words, the data point farthest away from the initial centroid has the highest probability of being selected as the next centroid. The major problem resulting from this idea is that the probability of selecting an outlier, if it exists, as a centroid, increases significantly. To increase the possibility of avoiding these issues, we chose to use an initialization step inspired by what was proposed by S. Hussain and M. Haris [93]. Instead of using a single data point, the proposed idea uses multiple data points to represent a centroid.

2.4 Cluster Validation

Once we are done running a clustering algorithm and a dataset has been processed and data points partitioned into clusters, there must be a way to validate the "goodness" of the resulting clusters and tackle the shortcomings present in clustering algorithms. A process called cluster validation is introduced for this task. This process estimates how well data points fit into the clusters [77]. Although this is not an easy task, there are several benefits for cluster validation that cannot be ignored such as [96]:

1) Determining the number of clusters for the dataset used. In our case, this would be the value of 'K' in K-Means clustering.

2) Determining whether a structure to the dataset exists and the data points have natural partitions and not random patterns.

3) Comparing the results of various clustering algorithm configurations or even completely different clustering algorithms for the dataset used.

Cluster validation techniques are categorized into three main classes: internal, external, and relative cluster validation [97]. Internal cluster validation depends on internal information found in the created clusters to evaluate how well data points fit into the clusters when the use of external information as a reference is not possible. It is also useful for determining the number of clusters needed. External cluster validation depends on externally available information to compare the created clusters with the externally known results. This technique is useful for comparing different clustering algorithms for specific datasets. Relative cluster validation compares the results obtained by changing the parameters and configurations of a specific clustering algorithm. Ultimately, this aids in finding the optimal parameters and configurations to use.

The cluster validation technique which interests us is the internal cluster validation. This is because we have no prior knowledge on the optimal number of clusters or whether there is a natural structure to the data we have. As such, we have to depend on two commonly used criteria for evaluating the quality of clustering, which are the compactness and separation of the clusters. Measuring these two criteria is the basis for providing insight into the performance of clustering algorithms. Compactness describes how close data points are within the same cluster. Separation describes how widely spaced clusters are from each other.

Since there is a plethora of cluster validity indices, we resorted to the work presented by O. Arbelaitz et al. [96], where they extensively compared 30 different cluster validity indices in various environments with various characteristics. The results of their work demonstrated the performance effectiveness, or lack thereof, of these various cluster validity indices. These can be used as guidelines for selecting appropriate cluster validity indices for many potential applications. Most of the indices combined the measured compactness and separation to compute a comparable quality measure. The writers in [96] focused on cluster validity indices that could be easily

evaluated and did not need any subjective decision making, which is why the Modified Hubert Index [98] and the Elbow Method [99] were left out for example. Fuzzy indices were also left out, as the focus was primarily on crisp clustering. Nonetheless, many well-known cluster validity indices were included in the study such as the Silhouette index [100], Dunn index [101], Calinski-Harabasz index [102], and Davies–Bouldin index [103]. The results showed that no single cluster validity index was a standout winner and better in everything, but the Silhouette index did obtain the best results in many of the experiments. As such, the cluster validity index we chose to use was the Silhouette Index.

2.4.1 Silhouette Index

The Silhouette index was first introduced by P. Rousseeu in [100]. The Silhouette index measures how compact data points are within a cluster and how well separated a cluster is to its nearest neighboring cluster. The resulting value of this index is known as Silhouette coefficient or Silhouette score. The Silhouette score is computed using the following formula [96, 104]:

$$S(C) = \frac{1}{N} \sum_{c_k \in C} \sum_{x_i \in c_k} \frac{b(x_i, c_k) - a(x_i, c_k)}{\max\{a(x_i, c_k), b(x_i, c_k)\}},$$

where

$$a(x_i, c_k) = \frac{1}{|C_k| - 1} \sum_{x_i \in C_k} d(x_i, x_j),$$

$$b(x_i, c_k) = \min_{c_l \in C \setminus c_k} \{\frac{1}{|c_l|} \sum_{x_j \in c_l} d(x_i, x_j)\}.$$

To explain how the silhouette score is computed, let us assume we have performed K-Means clustering on dataset ‘X’, which contains a set of ‘N’ data points, $X = \{x_1, x_2, x_3, \ldots, x_N\}$. There are ‘K’ clusters ‘C’, $C = \{c_1, c_2, c_3, \ldots, c_K\}$ where $\bigcup_{c_k \in C} c_k = X,\ c_k \cap c_l = \emptyset\ \forall k \neq l$. The distance ‘d’ between two data points ‘x_i’ and ‘x_j’ is $d(x_i, x_j)$. Usually, Euclidean distance is used. Compactness ‘a’ of data point ‘x_i’ to all other data points in the same cluster ‘c_k’ is $a(x_i, c_k)$. Separation ‘b’ of data point ‘x_i’ to all other data points in the nearest neighboring cluster ‘c_k’ is $b(x_i, c_k)$.

The values of the silhouette score range from -1 to 1. A value close to 1 means that the data points are well clustered. A value of zero means that the data points could be in any cluster. A value close to -1 means that the data points are poorly clustered. So, in order to decide on the best number of clusters ‘K’ for K-Means clustering, we compute the silhouette score for the different values of ‘K’. The value of ‘K’ which gives us the largest silhouette score is the best one because it provides the highest cluster quality. The key takeaway is that we want to insure the maximization of the inter-cluster distances and the minimization of the intra-cluster distances.

2.5 Similarity Measures for Images

Similarity measures are methods that compute the resemblance or likeness between two objects [105]. There are many similarity measures based on image quality assessment techniques. These are techniques that assess the quality of an image [106]. If a complete reference image is available, it is considered full-reference image quality assessment. If a reference image is not available, it is considered no-reference or blind image quality assessment. Of the many image quality assessment techniques, we chose to use Structural Similarity Index Measurement (SSIM)

[106], which is considered a full-reference image quality metric. SSIM was compared to various methods such as Peak-Signal-to-Noise-Ratio [107], Sarnoff's Just Noticeable Difference model [108], Universal Quality Index [109], and Mean Square Error [110]. SSIM performed better than all mentioned methods [106].

2.5.1 Structural Similarity Index Measurement

Images are usually structured in nature. Pixels in an image show strong dependencies, especially when they are adjacent. These dependencies hold valuable information regarding the structures of various objects in an image. SSIM was introduced by Z. Wang et al. [106] as an improvement to the Universal Quality Index. It has been widely used as an image quality assessment technique and similarity measure. The idea from developing SSIM is to mimic the human visual perception system, where it is possible to identify structural information from images and ultimately identifying the differences between them. It is important to note that measuring the similarity between two images using SSIM must be done to images of the same size.

Three components are taken into account when measuring the similarity between two images: luminance, contrast, and structure.

The formula for luminance is [106]: $l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$,

where $C_1 = (K_1L)^2$, L is the dynamic range of the pixel values, and K_1 is a small constant set to 0.01. μ_x and μ_y are the mean values of images x and y, respectively.

The formula for contrast is [106]: $c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$,

where $C_2 = (K_2L)^2$, L is the dynamic range of the pixel values, and K_2 is a small constant set to 0.03. σ_x and σ_y are the standard deviation of images x and y, respectively.

The formula for structure is [106]: $s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3}$,

where $C_3 = C_2/2$ and σ_{xy} is the covariance of images x and y.

The formula for SSIM, which is the combined formula for all three components together, is as follows [106]:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x{}^2 + \mu_y{}^2 + C_1)(\sigma_x{}^2 + \sigma_y{}^2 + C_2)}$$

The range of values for SSIM is (-1, 1]. The more similar two images are, the closer the value is to 1. If two images are identical, the value of SSIM will be exactly 1. The more dissimilar two images are, the closer the value is to -1.

Chapter 3

Methodology

3.1 Overview

Filter pruning is an important new frontier when it comes to developing improved CNN models. As discussed in section 2.2, the three main steps in any filter pruning algorithm are: 1) determining which filters are important and which are redundant, 2) pruning the redundant filters, and 3) retraining the neural network. In our thesis, we are interested in introducing and evaluating a new approach for the first step. Our approach determines which filters are redundant and can be pruned with very minimal compromise to the accuracy of the chosen CNN compared to the current state-of-the-art filter pruning method.

In our approach, we used the K-Means clustering algorithm to cluster similar filters together. A cluster contains similar filters which accomplish a similar job inside a convolutional layer. The K-Means clustering algorithm we implemented was adapted to use SSIM instead of Euclidean distance. We ran the K-Means clustering algorithm using various values of 'K' for each considered convolutional layer individually. We computed the silhouette score over a range of values for 'K' for each considered convolutional layer, which was used to determine the best value for the number of clusters 'K' for each considered convolutional layer. The Silhouette index we implemented was also adapted to use SSIM instead of Euclidean distance. We then used the clustering outcome with the highest silhouette score for each considered convolutional layer to

continue our experiments. The value of 'K' we selected for each considered convolutional layer was used to determine the pruning rate of that specific layer. The pruning rate represented the percentage of filters to be pruned. Then, we selected a representative filter from each cluster. We considered this filter to be important and all remaining filters in the cluster were considered redundant. Our proposed approach is illustrated in Figure 15. As for the second and third steps of filter pruning, we kept only one filter from each cluster and pruned the rest. The CNN was retrained after pruning filters in each layer. This is done to compensate for any possible drop in accuracy due to filter pruning. To test our new approach, we ran experiments on the VGG-16 architecture with the CIFAR-10 dataset.

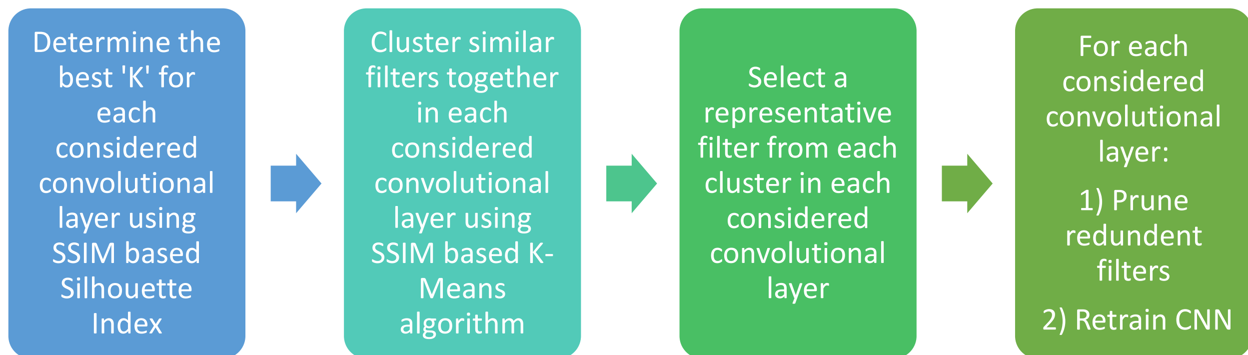

Figure 15: Our proposed approach

3.2 Implementation

3.2.1 Dataset

Of the most commonly used datasets for training CNNs, such as MNIST [111], Fashion MNIST [112], and CIFAR-10, the dataset used in the experiments reported in this thesis is the CIFAR-10 dataset by A. Krizhevsky [43]. It is a more challenging training dataset in comparison to the MNIST and Fashion MNIST datasets [113, 114].

3.2.1.1 CIFAR-10

The CIFAR-10 dataset, which stands for Canadian Institute for Advanced Research, is a subset of the 80 million tiny images dataset by A. Torralba et al. [115]. It consists of 60,0000 RGB images. Each image is 32x32 pixels in size. The CIFAR-10 dataset is divided into 10 mutually exclusive classes in total with 6,000 images per class. The dataset is further divided into 5 training batches and one testing batch. Each batch consists of 10,000 images, which means that there are 50,000 images in total used for training and 10,000 images used for testing CNNs. The testing batch has exactly 1,000 randomly selected images from each class, while the training batches contain the remaining images in random order. This results in the possibility of some training batches containing more images from certain classes than others. The ten classes are: trucks, ships, horses, frogs, dogs, deer, cats, birds, automobiles, and airplanes. The Python PyTorch library was used to load the dataset. The size of the dataset is 163 MB. Some sample images of the CIFAR-10 dataset are shown in Figure 16.

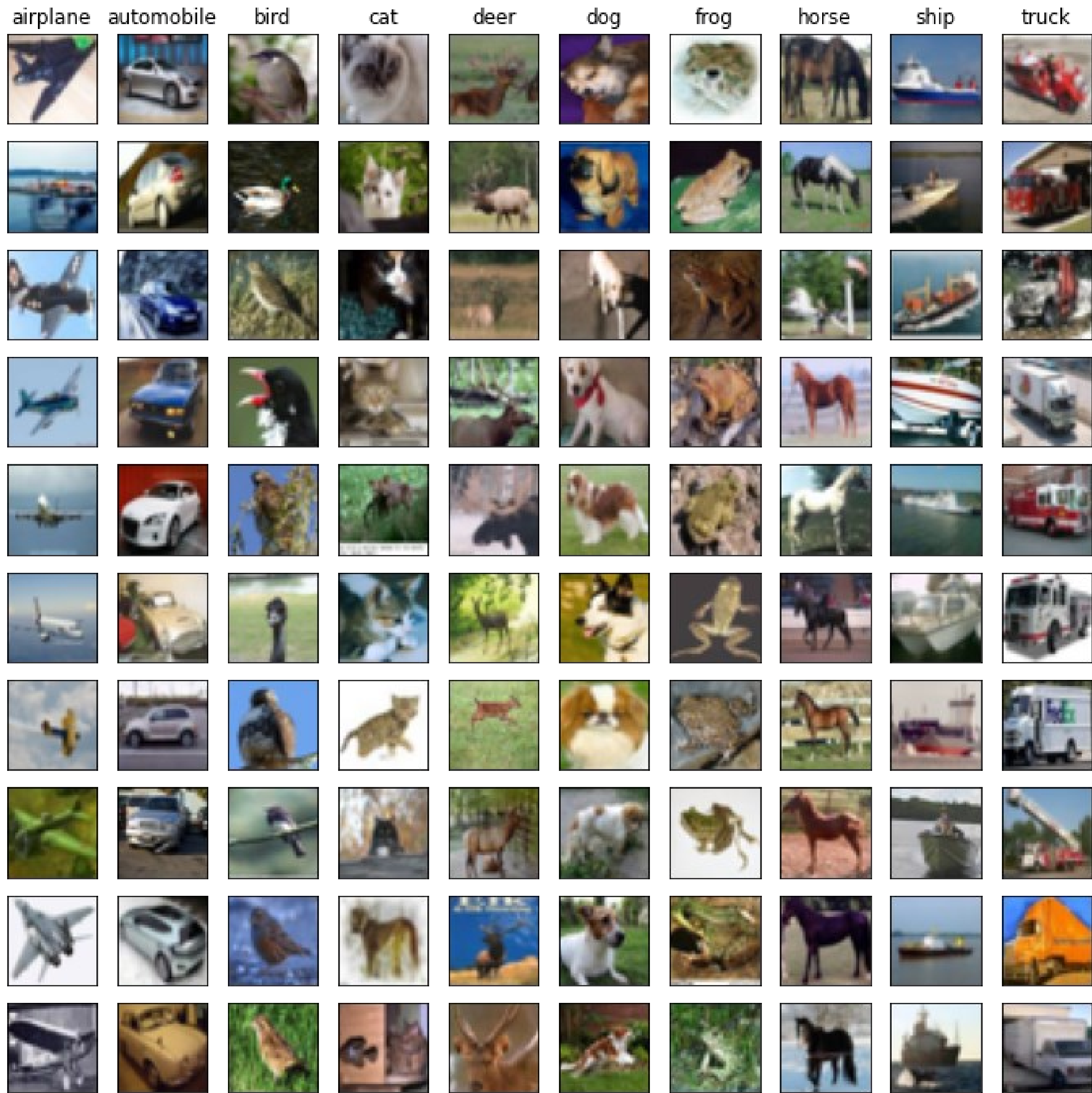

Figure 16: Some sample images of the CIFAR-10 dataset

3.2.2 VGG-16

Because we are using VGG-16 on CIFAR-10 and not ImageNet as it was originally created by K. Simonyan and A. Zisserman [42], we are using a variation similar to what was introduced by S. Zagoruyko [116] and H. Li et al. [37] to accommodate the smaller images of the CIFAR-10

dataset. This variation of VGG-16 consists of 21 different layers, instead of 22 layers. These include: 15 weight layers, 5 spatial pooling layers, and one softmax layer. Of the 15 weight layers, 13 are convolutional layers and 2 are fully connected layers. All 5 spatial pooling layers are max-pooling layers. The architecture for the VGG-16 CNN we are using is shown in Figure 17.

All convolutional layers and the first fully connected layer use ReLU activation functions. In addition, Batch Normalization was added to all convolutional layers and the first fully connected layer in this VGG-16 architecture. Batch Normalization was used instead of Dropout. All convolutional layers use 3x3 convolutional kernels. Just like the original VGG-16 architecture, the number of filters in the convolutional layers ranges from 64 to 512. The number of filters increases by a factor of two after every max-pooling layer, until they reach 512 filters. The first two layers of VGG-16 are convolutional layers with 64 filters each. Convolutional layers 3 and 4 have 128 filters each. Convolutional layers 5, 6, and 7 have 256 filters each. The remaining 6 convolutional layers, 7 until 13, have 512 filters each. A convolutional stride of size (1,1) pixel along with padding of size 1 pixel were used. Max-pooling was done with a window size of 2x2 pixels along with stride of size (2,2). No padding was used for the max-pooling layers. One of the key differences between the VGG-16 architecture we used, and the original VGG-16 architecture is that the former architecture has two fully connected layers instead of three and the number of channels in the fully connected layers was decreased. The first fully connected layer has 512 channels instead of 4,096. This is because after the last max-pooling layer we are left with 512 weights. The second, also final, fully connected layer has only 10 channels instead of 1,000. This is to correspond to the number of classes of the dataset we used, such that we can have 1 output for each of the 10 classes in CIFAR-10.

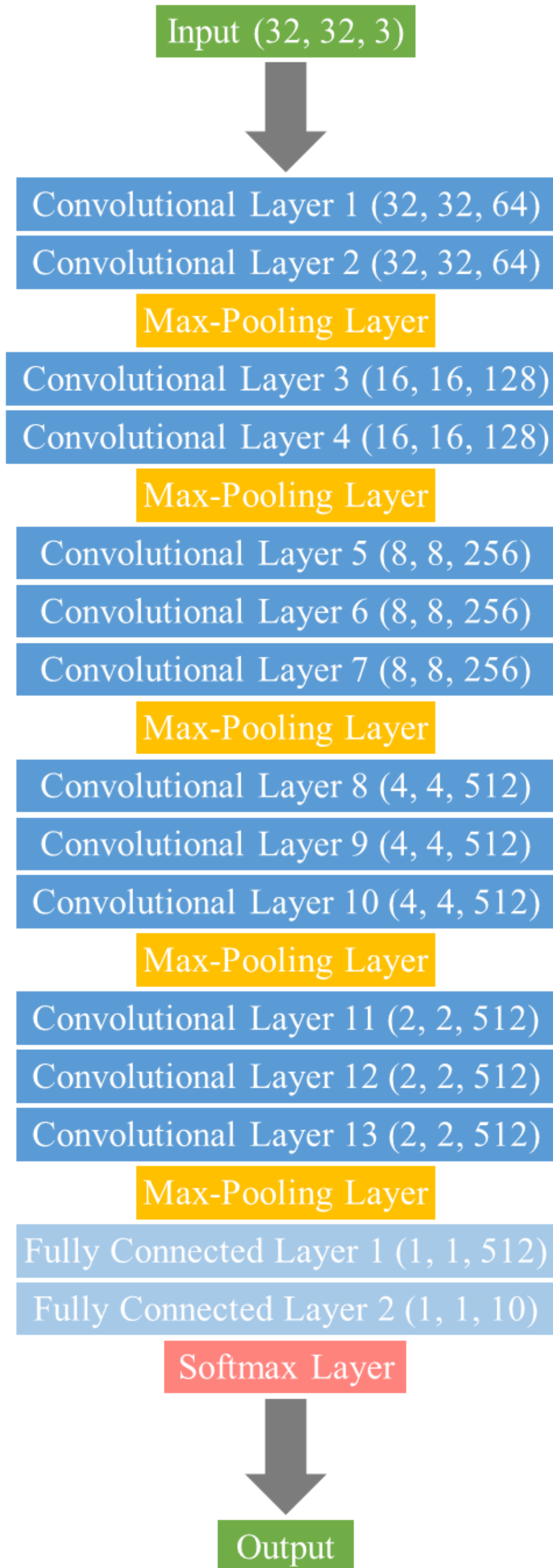

Figure 17: VGG-16 architecture

The pre-trained VGG-16 model we used was downloaded from [117]. The Python PyTorch library was used to load, prune, and retrain the pretrained network. The MBSGD optimizer was used for retraining the network. The Categorical Cross Entropy loss function was used. We set an initial learning rate at a value of 0.001. The learning rate was divided by 10 at epochs 5 and 10. The momentum was set at 0.9, and the weight decay was set at 0.0005. The training batch size was set to 128. We retrained the network for 30 epochs. The parameters we used were based on the same settings experimented by [31]. The pre-trained VGG-16 model on the CIFAR-10 dataset we used had a baseline accuracy of 93.96%.

3.2.3 SSIM Based K-Means Clustering

What we are trying to perform is clustering convolutional filters. As explained in Section 2.1.1, convolutional filters in a CNN can be thought of as image-like objects with a certain dimension size of pixels. The filters consist of weights which resemble pixel values of an image. Our aim from clustering is to group similar convolutional filters together. As such, using SSIM, which is suitable for comparing images, instead of a distance metric was a more logical and suitable choice. Therefore, we compare filters using SSIM. In our approach, the input used for our K-Means clustering algorithm are the convolutional filters from individual convolutional layers. The output are clusters of similar convolutional filters such that the intra-cluster similarities are maximized, and the inter-cluster similarities are minimized. We will explain our proposed approach for the three main steps in K-Means clustering: initialization, cluster assignment, and centroid re-estimation.

1) Initialization

Determining how to initialize centroids from a dataset, which are the convolutional filters in our case, is a very important step. Initializing centroids in a suboptimal way might lead to poor clustering results. To increase the possibility of avoiding this issue, we chose to use an initialization step inspired by what was proposed by S. Hussain and M. Haris [93]. Instead of using a single data point, the proposed idea uses multiple data points to represent a centroid. This implies that in addition to determining the number of clusters 'K' before starting, we also need to determine a number of initial centroids 'ic' to use in the initialization process of clustering. There are three main steps in our initialization process. The first step is choosing 'K' points at random from our dataset. The second step is finding the most similar 'ic-1' data points to a specific centroid. This is done using SSIM. The function we used to compute SSIM was from the Python Scikit-Image library [118]. Finally, we take the mean value of the 'ic' data points to get our new centroid.

Let us assume we set the value of 'K' to 5 and the value of 'ic' to 3. Let $C_{1,1}$, $C_{2,1}$, $C_{3,1}$, $C_{4,1}$, and $C_{5,1}$ be our K = 5 randomly chosen centroids from our considered filters. We chose the 2 $(3 - 1 = 2)$ most similar filters to $C_{1,1}$. They will be known as $C_{1,2}$ and $C_{1,3}$. The same step is repeated for the remaining centroids $C_{2,1}$ to $C_{5,1}$. In total, we will have 'K' × 'ic', 5 × 3 = 15 filters selected. We will then take the mean value for $C_{1,1}$, $C_{1,2}$, and $C_{1,3}$ to get our new centroid C_1. The same step is repeated for the remaining centroids. In the end, we will have 5 new centroids. We illustrate our initialization method with an example based on 2D space shown in Figure 18.

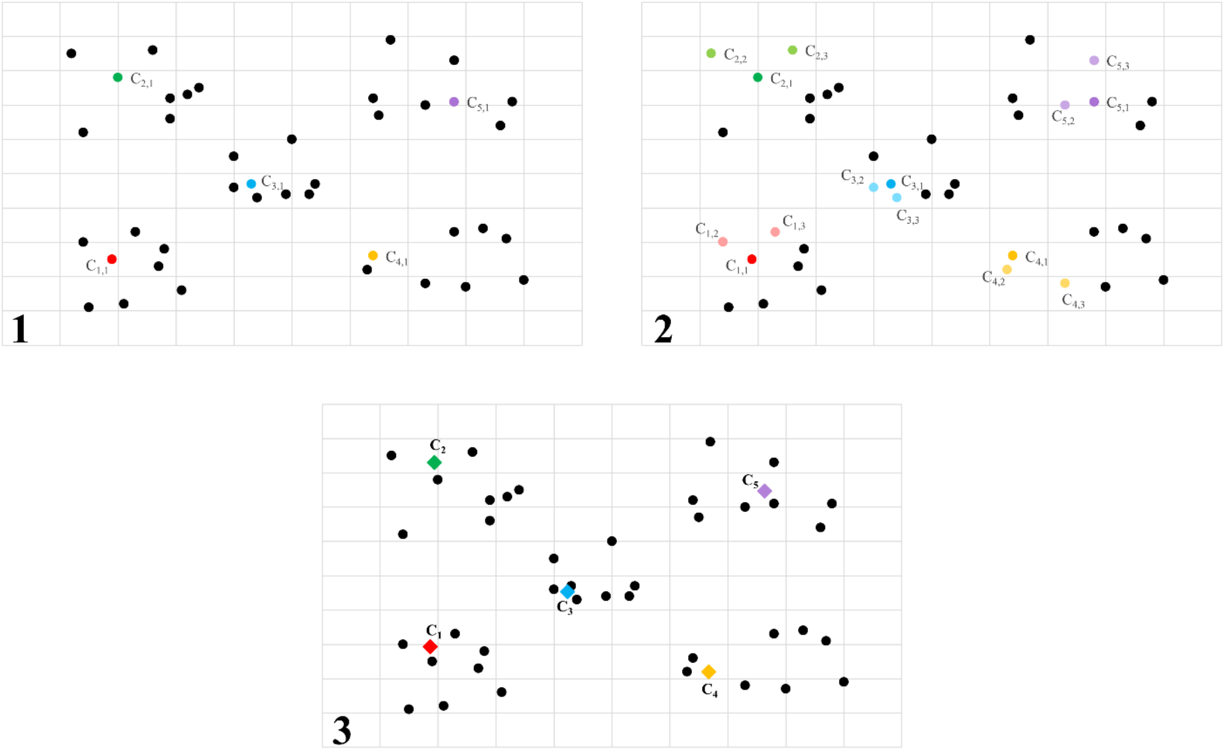

Figure 18: K-Means initialization

2) Cluster Assignment

Once centroid initialization is done, the second step in our clustering algorithm is similar to standard K-Means clustering. We assign filters to clusters based on the most similar centroid. This is done using SSIM. This step is illustrated in Figure 19A. The formula used for assigning filters is as follows:

$$A(F_i) \leftarrow \underset{j \in 1 \dots K}{\operatorname{argmax}}\left(SSIM\left(F_i, C_j\right) \right),$$

where $A(F_i)$ is the assignment function for each filter F_i. The similarity between each filter F_i and each centroid C_j, where $j \in 1 \dots K$, is computed using SSIM, and the filter is assigned to the cluster that its centroid has the maximum similarity to among all centroids.

3) <u>Centroid Re-estimation</u>

Once all filters are assigned to a cluster, we want to re-estimate or compute new centroids. This is done in the third step by computing the mean of the filters in each cluster. This step is illustrated in Figure 19B.

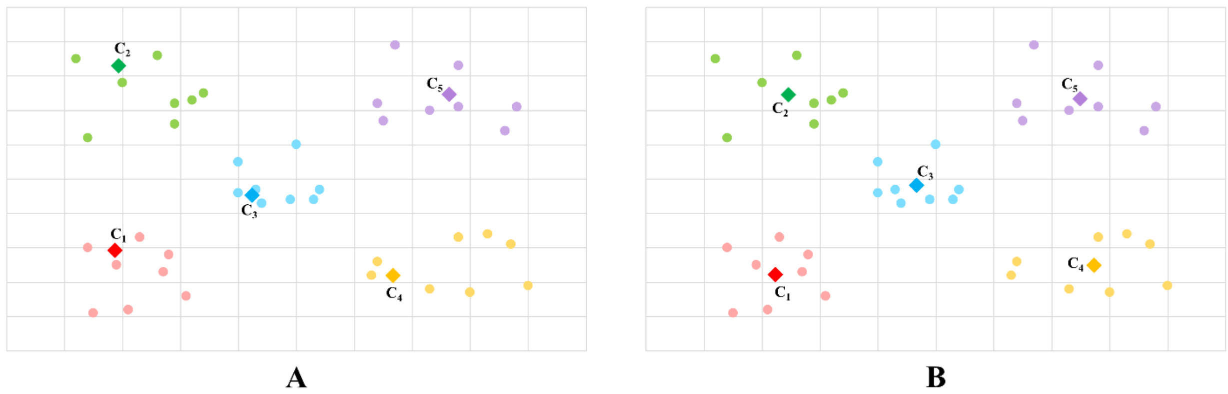

Figure 19: K-Means cluster assignment and centroid re-estimation

The second and third steps (cluster assignment and centroid re-estimation) are repeated until convergence, i.e., the centroids do not change.

When we say that we are taking the mean of multiple convolutional filters, this means that we are taking the mean of the corresponding weights from each filter. That is, the mean is actually a matrix of weights. The size of the matrix is the same size of a filter. The value of the mean $M_{i,j}$ at row i and column j is formally computed as:

$$M_{i,j} = \frac{\sum_{k=1}^{n} W_k[i,j]}{n},$$

where n is the number of filters for which the mean is computed, $W_k[i,j]$ is the weight at row i and column j of filter F_k. Figure 20 shows an example of three filters and their corresponding mean. In this example, the mean value of $M_{1,1}$ is computed as follows:

$$M_{1,1} = \frac{W_1[1,1] + W_2[1,1] + W_3[1,1]}{3} = \frac{0.5 + 0.3 + 0.7}{3} = 0.5$$

The rest of the weights are computed in a similar way.

Filter 1

0.5	0.7	0.2
0.1	0.3	1.0
0.9	0.4	0.6

Filter 2

0.3	0.7	0.8
0.5	0.2	0.9
0.1	0.7	0.1

Filter 3

0.7	0.6	0.7
0.1	0.4	0.6
0.4	0.4	0.3

Mean(F_1, F_2, F_3)

0.5	0.7	0.6
0.2	0.3	0.8
0.5	0.5	0.3

Figure 20: Mean of three filters

3.2.4 SSIM Based Silhouette Index

Cluster validation acts as a failsafe for us. Even if our clustering algorithm's initialization step does not provide good centroids, by using the silhouette index we can know that the clusters are poorly created. The steps for computing the silhouette score are straightforward. We compute

the similarity between a specific filter and each of the remaining filters in the same cluster. Then we compute the similarity between the same filter and each filter in the most similar cluster to the original cluster. This is repeated for all filters in the dataset.

When implementing the silhouette index, we used SSIM instead of Euclidean distance. To accommodate the use of SSIM, we shifted the range of values we get using SSIM from (-1,1] to (0,2] and switched the variables $a(x_i, c_k)$ and $b(x_i, c_k)$ in the silhouette index formula. This is because similarity is the inverse of distance. The formula for computing the silhouette score becomes:

$$S(C) = \frac{1}{N} \sum_{c_k \in C} \sum_{x_i \in c_k} \frac{a(x_i, c_k) - b(x_i, c_k)}{\max\{a(x_i, c_k), b(x_i, c_k)\}},$$

where

$$\mathrm{a}(x_i, c_k) = \frac{1}{|C_k| - 1} \sum_{x_i \in C_k} SSIM(x_i, x_j),$$

$$\mathrm{b}(x_i, c_k) = \min_{c_l \in C \setminus c_k} \{\frac{1}{|c_l|} \sum_{x_j \in c_l} SSIM(x_i, x_j)\}.$$

This means that the range of values for our silhouette index is (-1,1). A value close to 1 means that the filters are well clustered. A value of zero means that the filters could be in any cluster. A value close to -1 means that the data points are poorly clustered. We illustrate our silhouette index with an example based on 2D space shown in Figure 21.

For example, if filter F_i is exactly similar to all other filters in the same cluster C_k, then the value of $a(F_i, C_k)$ will be equal to 2. In addition, if filter F_i is very dissimilar to all other filters in the most similar cluster c_l to the original cluster c_k, then the value of $b(F_i, C_k)$ will be very close to

zero. The minimum function used in $b(F_i, C_k)$ selects the most similar cluster to the original cluster. The average similarity between filter F_i and each filter in other clusters is computed and the minimum average similarity determines the most similar cluster to the original cluster which filter F_i belongs to. So, if the average similarity between filter F_i in cluster C_k and each filter in cluster C_l is 0.1, cluster C_2 is 0.5, and cluster C_3 is 0.3, we choose cluster C_l. Therefore, in our example, the value of $b(F_i, C_k)$ will be 0.1. The silhouette score for the single filter F_i becomes:

$$S(x_i) = \frac{2-0.1}{\max\{2,0 .1\}} = \frac{1.9}{2} = 0.95$$

This step is then repeated for all filters and the average value for the individual filter silhouette scores is the silhouette score we use to determine how good the clustering outcome is for this run of the algorithm. The silhouette index was implemented with the help of the Python Scikit-Learn library [119, 120], while performing the changes indicated earlier to suit our needs of using SSIM.

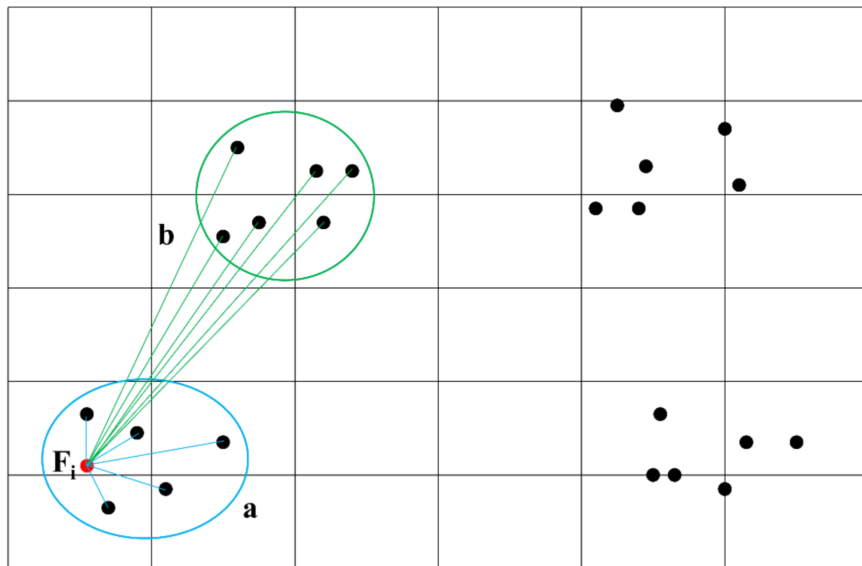

Figure 21: Silhouette Index

3.3 Filter Clustering Algorithm

As explained in section 2.1.1 and with the help of Figure 4, a convolutional filter in the VGG-16 model we used consisted of multiple kernels. We had to flatten the kernels together to transform a filter into one image-like object instead of multiple objects. For example, each convolutional filter in the first convolutional layer of VGG-16 consisted of 3 kernels. Each kernel is of size 3x3, which means a total of 9 weights. This means that each filter is of size 3x3x3, consisting of a total of 27 weights. We flattened the 3 kernels together to make each filter to be of size 9x3, which still maintains a total of 27 weights in each filter. Once this was done to all convolutional filters, we ran our K-Means clustering algorithm for each considered convolutional layer individually. This is because convolutional filters in different layers may have different sizes and perform different tasks. By definition, SSIM can only be applied on objects with the same size.

We ran the code 10 times for each value of 'K'. We computed the silhouette score for each of the 10 runs for a specific value of 'K'. We then computed the average of the 10 silhouette scores we have for the same specific value of 'K'. This was done for all values of 'K' we ran. The average silhouette score at a certain value of 'K' is referred to as ASS_K, and the maximum ASS_K is referred to as ASS_{max}. The individual maximum silhouette score out of the 10 silhouette scores for each value of 'K' is called the local maximum silhouette score ($LMSS_K$). At the value of 'K' corresponding to ASS_{max}, the individual maximum silhouette score out of the 10 silhouette scores is called the global maximum silhouette score. The value of 'K' corresponding to ASS_{max} was considered the best one. The value for 'ic' gradually decreased as the value of 'K' increased. In

our experiments, we selected the value of 'ic' to range from 5 to 1. This was done to ensure that 'ic' × 'K' was not greater than the number of filters in a specific convolutional layer.

For example, if we ran our K-Means clustering algorithm on the first convolutional layer in VGG-16, which has 64 filters, the value for 'K' would range from 2 to 63. This means that we tested 62 values for 'K' and for each one we computed 10 clustering outcomes with each one giving us a different silhouette score. In total, we obtained 62×10 = 620 clustering outcomes for the first convolutional layer alone. We computed the average of the 10 silhouette scores we got for each 'K' we tested. This results in 62 average silhouette scores corresponding to the 62 values of 'K' we tested. We then select the 'K' with the highest average silhouette score and selected the clustering outcome with the highest silhouette score out of the 10 runs we did for that specific value of 'K'. If the highest average silhouette score, for example, corresponds to K=20, we would select the clustering outcome with the highest silhouette score, which we have out of the 10 clustering outcomes we produced for K=20, to use as the input for our filter pruning algorithm.

3.4 Filter Pruning Algorithm

Once the clustering algorithm has been applied and all filters are allocated to a cluster, the filters inside a cluster are similar and hence accomplish a similar job inside a convolutional layer. This means that if more than one filter extracts the same or similar features, it could be assumed that the filters are redundant and not necessary for the overall accuracy and process of a CNN. We then select a representative point from each cluster, which we consider the most similar filter to the centroid. That is, we calculate the similarity between each filter in the cluster and the centroid. The filter that has the highest similarity value with the centroid is selected to be the representative

filter of the cluster. All remaining filters in the clusters are considered redundant and will be pruned. The pruning rate for each considered convolutional layer depends on the number of clusters ‘K’ which we have selected.

For example, in the first convolutional layer, we have 64 filters. If the value of ‘K’ we are using is 16, the pruning rate for the first convolutional layer is 1-16/64 = 0.75. We will keep only 16 filters out of 64. This means that 48 filters will be pruned, which represents 75% of the filters in the first convolutional layer. The value of ‘K’, which we selected for each considered convolutional layer, was used to determine the pruning rate of that specific layer.

We performed three experiments. In the first experiment, we selected ‘K’ in each convolutional layer corresponding to ASS_{max}. In the second and third experiments, we tested higher pruning rates. Therefore, we tested using smaller values for ‘K’ corresponding to other $LMSS_K$. The criteria we considered for selecting the ‘K’ values in the second and third experiments are as follows. In the second experiment, we experimented using almost the same pruning rate for all layers. The pruning rate, and consequently the value of ‘K’, is selected such that it is the highest possible rate that does not cause the $LMSS_K$ in each layer to be less than the ASS_{max}. After this selection of ‘K’ values, if the selected value of ‘K’ in any layer is almost the same as the value of ‘K’ corresponding to ASS_{max}, we search for a value of ‘K’ for that specific layer that results in the highest possible pruning rate without causing the value of the $LMSS_K$ to be less than ASS_{max}. Using these criteria in the second experiment, we ensured that none of the selected values of ‘K’ had $LMSS_K$ less than the ASS_{max}. In the third experiment, we experimented relaxing the selection criteria for the $LMSS_K$ for the last 25% of the selected layers to explore the impact of increasing the pruning rate for the last layers compared to the former ones on the classification accuracy. In this case, we experimented using the same pruning rate for the last three considered layers such

that this rate is the highest possible rate that does not cause the $LMSS_K$ in each of these layers to be less than 85% of ASS_{max}.

The main filter pruning code we used was from [117]. We adapted the code to our needs. Originally, the filters were given values from 0 to 100 indicating their importance. The higher the value, the more important a filter is, and it is higher ranked. What we did instead was assigning only a value of 100 or 0. There were no values in between. The filters we wanted to keep were given a value of 100, and the redundant filters were given a value of zero. Filters were then ranked based on descending order. This way, the important filters were at the top of the rank and the redundant filters were at the bottom. The percentage of filters that were in the bottom of the rank were pruned. For example, in the first convolutional layer, if we want to prune 48 filters out of 64 filters, we need to make sure that the pruning rate is set to 0.75. If the pruning rate is less than 0.75, the filter pruning code will not prune all 48 filters which were given a value of zero. If the pruning rate is higher than 0.75, the filter pruning code will prune some of the 16 filters which were given a value of 100. A mask was then applied to the percentage of redundant filters corresponding the pruning rate we had set. The weights for all redundant filters were set to zero, effectively pruning them from the CNN. We applied a layer-wise iterative process for filter pruning and retraining. We would filter prune one layer at a time and then retrain the whole CNN to compensate for any possible drop in accuracy due to removing filters from that specific convolutional layer. Therefore, we prune the first convolutional layer and retrain for 30 epochs, then we prune the second convolutional layer and retrain for 30 epochs and so on.

3.5 Evaluation

Our proposed filter pruning algorithm was evaluated in terms of the achieved model compression, model acceleration, and model accuracy. We used the same pretrained VGG-16 model [117] provided by HRank. We also made use of the filter rankings they generated based on the same pretrained VGG-16 model. We filter pruned the pretrained VGG-16 model with the same pruning rate for both our method and the state-of-the-art method of HRank. This provided a fair and level starting point for comparison. The pruning rate we used depends on the value of 'K' with the highest silhouette score. HRank on the other hand ranked the filters of each convolutional layer based on their importance. The filters with the high values were more important than filters with low values. Their pruning rates were not chosen based on predetermined criteria. So, if the pruning rate was set to 0.6, 60% of the filters with the lowest ranks will be pruned.

To calculate the number of parameters and FLOPs in our VGG-16 model, we recall the terminology used in section 2.1.1 to describe Figure 4. In layer i, there are n_i input feature maps fm_i. Each feature map is of height h_i and width w_i. The 4D kernel matrix M_i consists of n_{i+1} 3D filters F_i. The filters F_i consist of n_i 2D kernels k_i. Each kernel k_i consists of s^2 weights $W_{i,j}$. Basically, a filter F_i consist of $n_i \times s^2$ weights $W_{i,j}$. The filters F_i are applied to the input feature maps fm_i and give us n_{i+1} feature maps fm_{i+1} as an output.

The number of parameters in each convolutional layer can be calculated using the following formula:

$$Number\ of\ Parameters = (s^2 \times n_i + 1) \times n_{i+1}$$

Where the added 1 is due to the bias term. For example, the first convolutional layer in our VGG-16 model has an input of $n_i = 3$ channels, which could also be considered feature maps. Each kernel consists of $s^2 = 3^2 = 9$ weights. There are 64 filters, which results in $n_{i+1} = 64$ feature maps as an output. As such, there are:

$$(3^2 \times 3 + 1) \times 64 = 1{,}792 \; parameters$$

The second convolutional layer has an input of $n_{i+1} = 64$ feature maps. Each kernel also consists of $s^2 = 3^2 = 9$ weights. There are 64 filters, which results in $n_{i+2} = 64$ feature maps as an output. As such, there are:

$$(3^2 \times 64 + 1) \times 64 = 36{,}928 \; parameters$$

The number of FLOPs in each convolutional layer can be calculated using the following formula:

$$Number\ of\ FLOPS = n_i \times n_{i+1} \times s^2 \times h_{i+1} \times w_{i+1}$$

For example, the first convolutional layer in our VGG-16 model has an input of $n_i = 3$ channels. Each kernel consists of $s^2 = 3^2 = 9$ weights. There are 64 filters, which results in $n_{i+1} = 64$ feature maps as an output. The height and width of the feature maps is 32. As such, there are:

$$3 \times 64 \times 3^2 \times 32 \times 32 = 1{,}769{,}472 \text{ FLOPs}$$

The second convolutional layer has an input of $n_{i+1} = 64$ feature maps. Each kernel also consists of $s^2 = 3^2 = 9$ weights. There are 64 filters, which results in $n_{i+2} = 64$ feature maps as an output. The height and width of each feature map is 32. As such, there are:

$$64 \times 64 \times 3^2 \times 32 \times 32 = 37{,}748{,}736 \text{ FLOPs.}$$

Therefore, when a filter is pruned, the number of reduced parameters is:

$$Number\ of\ reduced\ parameters = s^2 \times n_i + 1$$

When a filter is pruned, the number of reduced FLOPs is:

$$Number\ of\ reduced\ FLOPS = n_i \times s^2 \times h_{i+1} \times w_{i+1}$$

For example, if one filter is pruned from the first convolutional layer, the number of reduced parameters is:

$$3^2 \times 3 + 1 = 28\ parameters$$

In addition, if one filter is pruned from the first convolutional layer, the number of reduced FLOPs is:

$$3 \times 3^2 \times 32 \times 32 = 27{,}648\ FLOPs$$

The code used for calculating the number of reduced parameters and reduced FLOPs was provided by [117].

We compared the resulting average accuracy, after filter pruning and retraining the pretrained VGG-16 model, of both our method and HRank's method. To get the accuracy of the pretrained VGG-16 model after filter pruning, we did 10 experiments for each filter pruning method. We did 10 experiment runs based on the filters we deemed redundant and then calculated the average value for the 10 accuracy values we got from the 10 experiment runs. We did the same thing based on the filters HRank deemed least important. We compared the average accuracy values obtained using our method and HRank's method to check whether the difference between them is statistically significant. Before we could compare the average accuracy values, we had to check whether each set of data followed the normal distribution using the Shapiro-Wilk test [121].

The type of test needed to compare our sets of data differed if our data did not follow the normal distribution. Once our sets of data were confirmed to follow normal distribution, we ran a T-test [122] to confirm whether there was a statistically significant difference in the accuracy. The T-test is a statistical test that determines whether there is a significant difference between the mean values of two groups.

Chapter 4

Results and Discussion

4.1 Silhouette Plots

There is one silhouette plot for each of the 12 convolutional layers we filter pruned. The plots show the change of the average silhouette score over 'K'. In addition, for each convolutional layer, two more curves are plotted for the average silhouette score ±1 standard deviation (σ). In all cases, the two plotted curves showed that the amount of variability from the individual silhouette scores to the average silhouette score at each 'K' is relatively small. The average and maximum ratios of sigma to the average silhouette scores for each plot are reported. The first and last 10% of the average silhouette scores were excluded from calculating the average and maximum ratios of sigma. This is because the excluded values of the average silhouette scores are very small which, in some cases, causes the ratios of sigma to the average silhouette scores to be relatively high, which gives a misleading impression when reporting the maximum ratio. Excluding these values does not affect the interpretations of the results as these excluded values are for 'K' values that are far from the 'K' value that corresponds to the global silhouette score.

In each plot, the x-axis is for the number of clusters 'K'. The y-axis is for the silhouette scores. The global maximum average silhouette score and any local maximum silhouette scores are marked in each plot, and the 'K' values corresponding to them are also indicated. The selection of the global maximum average silhouette score and any local maximum silhouette scores is based

on the criteria explained in section 3.4. Hence, using a clustering outcome at the 'K' values corresponding to local maximum silhouette scores provides the opportunity to test higher pruning rates.

Figure 22 is for convolutional layer 1 which has 64 filters. The number of clusters 'K' ranges from 2 to 63. This means that we have tested 62 values for 'K' and for each one we computed 10 clustering outcomes with each one giving us a different silhouette score. In total, we have obtained 62×10 = 620 clustering outcomes for the first convolutional layer alone. The value of 'ic' ranges from 5 to 1 and it gradually decreases as the value of 'K' increases. The value of 'ic' is set to 5 for the number of clusters 'K' ranging from 2 to 10 and is set to 4 for the number of clusters 'K' ranging from 11 to 14. In addition, the value of 'ic' is set to 3 for the number of clusters 'K' ranging from 15 to 20 and is set to 2 for the number of clusters 'K' ranging from 21 to 31. The value of 'ic' is set to 1 for the number of clusters 'K' ranging from 32 to 63. This was done to ensure that 'ic' × 'K', i.e., the total number initial centroids, was not greater than the number of filters in a specific convolutional layer. For example, if the value of 'K' was 18 and the value of 'ic' was 4, then the total number of initial centroids would be 4×20=72, which is greater than the number of filters in convolutional layer 1. On the other hand, if the value of 'ic' was 3, then the total number of initial centroids would be 3×18=54, which is less than the number of filters in convolutional layer 1. This means that we cannot set the value of 'ic' to be 4 at K=18, but it is possible to set the value of 'ic' to be 3.

As shown in Figure 22, the average silhouette scores widely range from 0.02 to 0.60. The global maximum average silhouette score was found to correspond to the value of K = 24. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 24 was 0.63. The selected local maximum silhouette score corresponds to the value of K = 20. The highest

individual silhouette score out of the 10 silhouette scores we computed at K = 20 was 0.61. The average ratio of sigma to the average silhouette scores is 0.041. The maximum ratio of sigma to the average silhouette scores is 0.079.

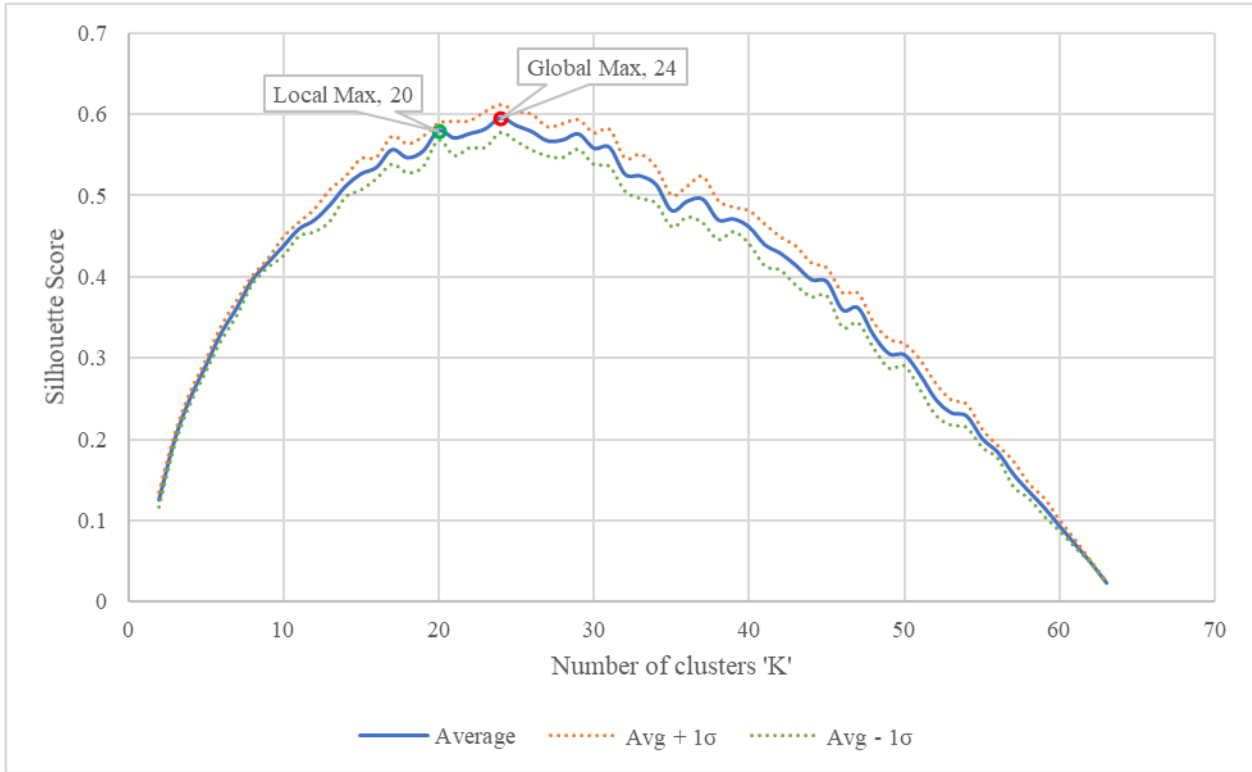

Figure 22: Silhouette plot for convolutional layer 1

Figure 23 is for convolutional layer 2 which has 64 filters, as in convolutional layer 1. Therefore, we have also obtained 620 clustering outcomes for convolutional layer 2. The value of ‘ic’, which ranges from 5 to 1, changes at the same rate as in convolutional layer 1 because both convolutional layers 1 and 2 have the same number of filters. As shown in Figure 23, the average silhouette scores widely range from 0.02 to 0.48. The global maximum average silhouette score was found to correspond to the value of K = 29. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 29 was 0.51. The selected local maximum silhouette score corresponds to the value of K = 23. The highest individual silhouette score out of the 10

silhouette scores we computed at K = 23 was 0.48. The average ratio of sigma to the average silhouette scores is 0.064. The maximum ratio of sigma to the average silhouette scores is 0.115.

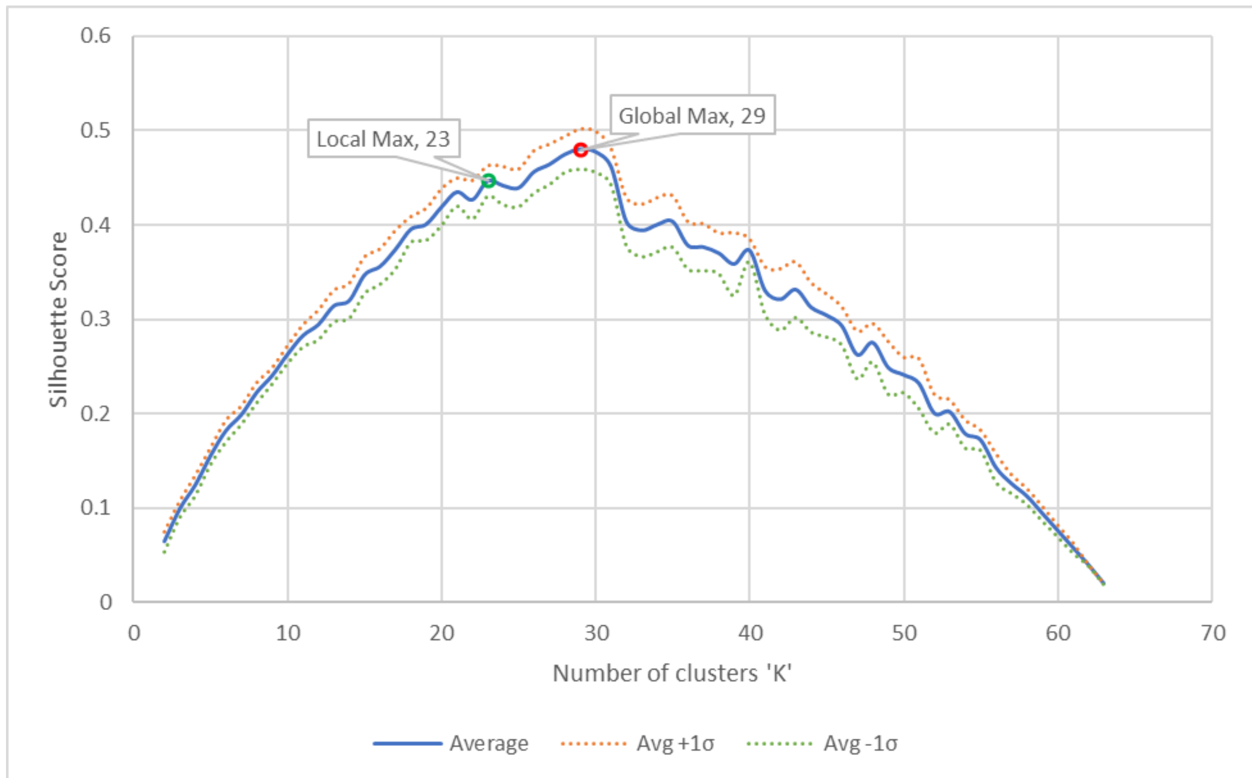

Figure 23: Silhouette plot for convolutional layer 2

Figure 24 is for convolutional layer 3 which has 128 filters. The number of clusters ‘K’ ranges from 2 to 127. This means that we have tested 126 values for ‘K’ and for each one we computed 10 clustering outcomes with each one giving us a different silhouette score. In total, we have obtained 126×10 = 1,260 clustering outcomes for convolutional layer 3. The value of ‘ic’ ranges from 5 to 1 and it gradually decreases as the value of ‘K’ increases. The value of ‘ic’ is set to 5 for the number of clusters ‘K’ ranging from 2 to 20 and is set to 4 for the number of clusters ‘K’ ranging from 21 to 28. In addition, the value of ‘ic’ is set to 3 for the number of clusters ‘K’ ranging from 29 to 40 and is set to 2 for the number of clusters ‘K’ ranging from 41 to 62. The

value of ‘ic’ is set to 1 for the number of clusters ‘K’ ranging from 63 to 127. As shown in Figure 24, the average silhouette scores widely range from 0.02 to 0.47. The global maximum average silhouette score was found to correspond to the value of K = 53. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 53 was 0.48. The selected local maximum silhouette score corresponds to the value of K = 45. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 45 was 0.47. The average ratio of sigma to the average silhouette scores is 0.044. The maximum ratio of sigma to the average silhouette scores is 0.084.

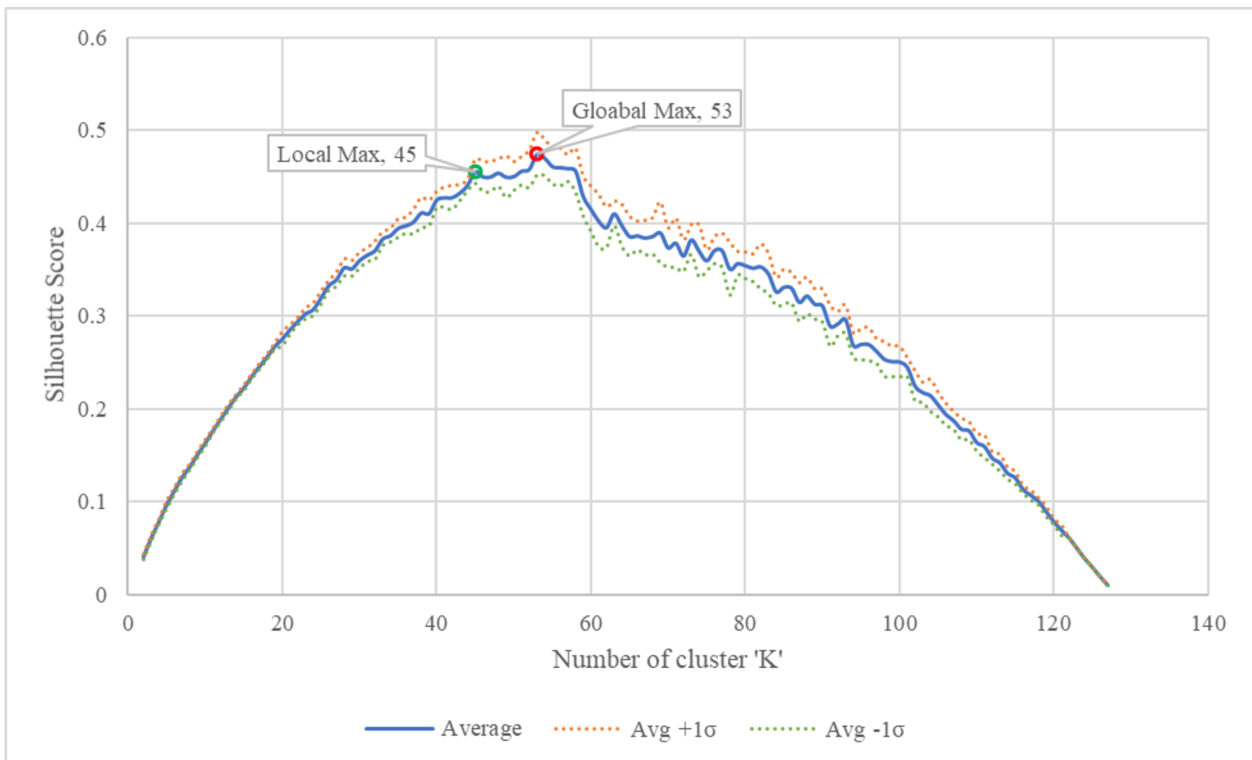

Figure 24: Silhouette plot for convolutional layer 3

Figure 25 is for convolutional layer 4 which has 128 filters, as in convolutional layer 3. Therefore, we have also obtained 1,260 clustering outcomes for convolutional layer 4. The value of ‘ic’, which ranges from 5 to 1, changes at the same rate as in convolutional layer 3 because both convolutional layers 3 and 4 have the same number of filters. As shown in Figure 25, the average

silhouette scores widely range from 0.01 to 0.43. The global maximum average silhouette score was found to correspond to the value of K = 56. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 56 was 0.47. The selected local maximum silhouette score corresponds to the value of K = 45. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 45 was 0.44. The average ratio of sigma to the average silhouette scores is 0.043. The maximum ratio of sigma to the average silhouette scores is 0.083.

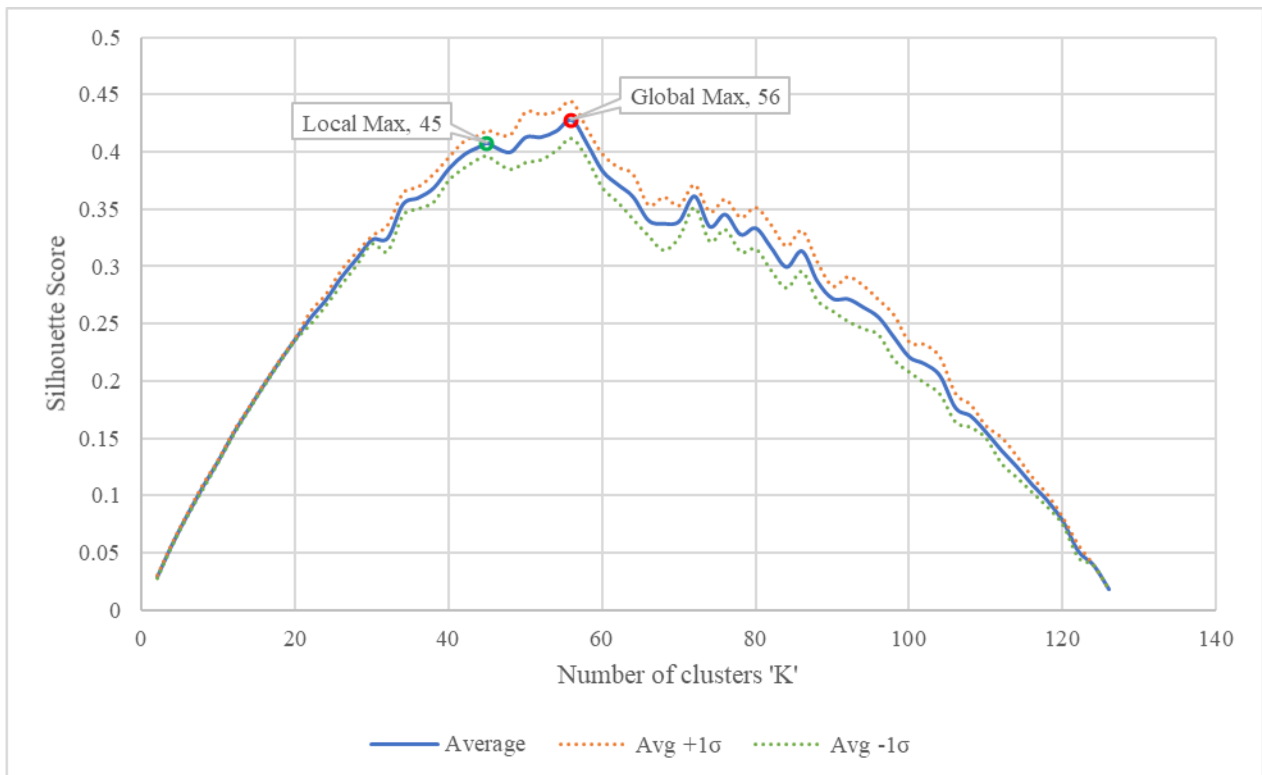

Figure 25: Silhouette plot for convolutional layer 4

Figure 26 is for convolutional layer 5 which has 256 filters. The number of clusters ‘K’ ranges from 2 to 255. This means that we have tested 254 values for ‘K’ and for each one we computed 10 clustering outcomes with each one giving us a different silhouette score. In total, we have obtained 254×10 = 2,540 clustering outcomes for convolutional layer 5. The value of ‘ic’ ranges from 5 to 1 and it gradually decreases as the value of ‘K’ increases. The value of ‘ic’ is set

to 5 for the number of clusters ‘K’ ranging from 2 to 40 and is set to 4 for the number of clusters ‘K’ ranging from 41 to 56. In addition, the value of ‘ic’ is set to 3 for the number of clusters ‘K’ ranging from 57 to 80 and is set to 2 for the number of clusters ‘K’ ranging from 81 to 112. The value of ‘ic’ is set to 1 for the number of clusters ‘K’ ranging from 113 to 255. As shown in Figure 26, the average silhouette scores widely range from 0.01 to 0.40. The global maximum average silhouette score was found to correspond to the value of K = 102. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 102 was 0.44. The selected local maximum silhouette score corresponds to the value of K = 90. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 90 was 0.40. The average ratio of sigma to the average silhouette scores is 0.051. The maximum ratio of sigma to the average silhouette scores is 0.189.

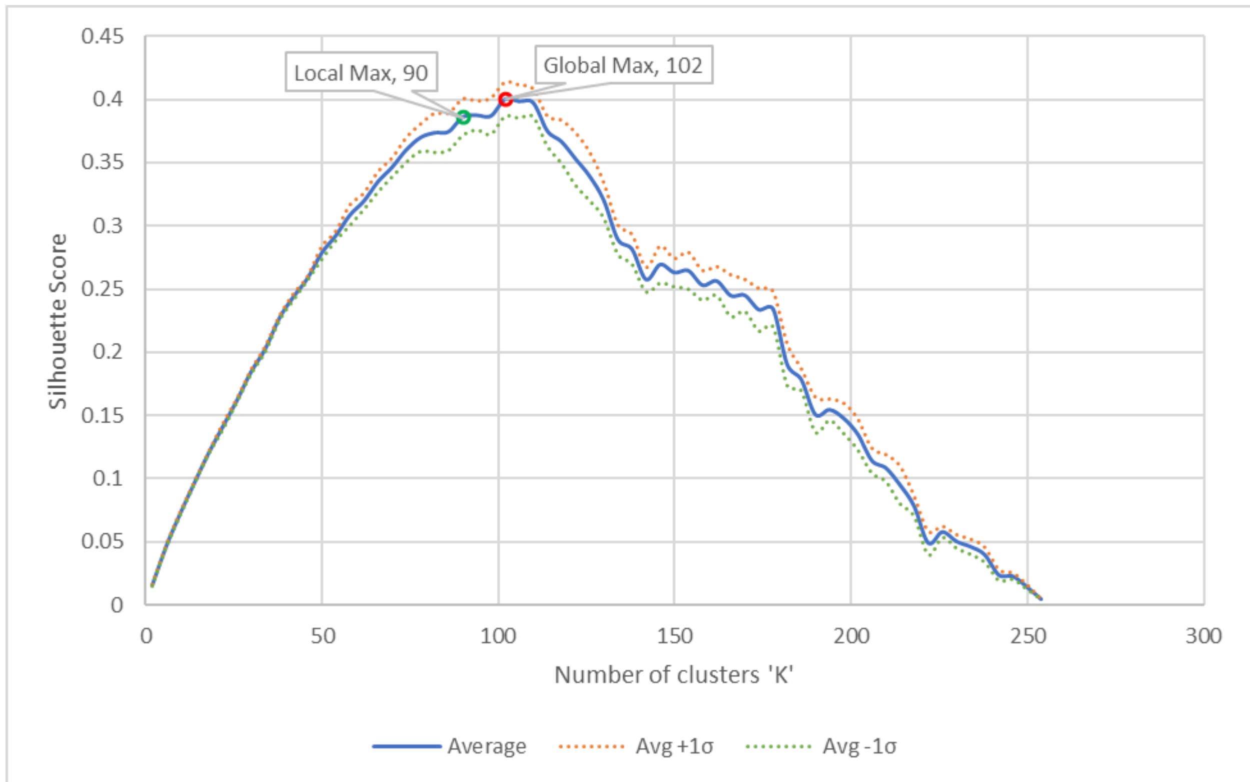

Figure 26: Silhouette plot for convolutional layer 5

Figure 27 is for convolutional layer 6 which has 256 filters, as in convolutional layer 5. Therefore, we have also obtained 2,540 clustering outcomes for convolutional layer 6. The value of 'ic', which ranges from 5 to 1, changes at the same rate as in convolutional layer 5 because both convolutional layers 5 and 6 have the same number of filters. As shown in Figure 27, the average silhouette scores widely range from 0.01 to 0.37. The global maximum average silhouette score was found to correspond to the value of K = 100. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 100 was 0.40. The selected local maximum silhouette score corresponds to the value of K = 90. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 90 was 0.38. The average ratio of sigma to the average silhouette scores is 0.073. The maximum ratio of sigma to the average silhouette scores is 0.213.

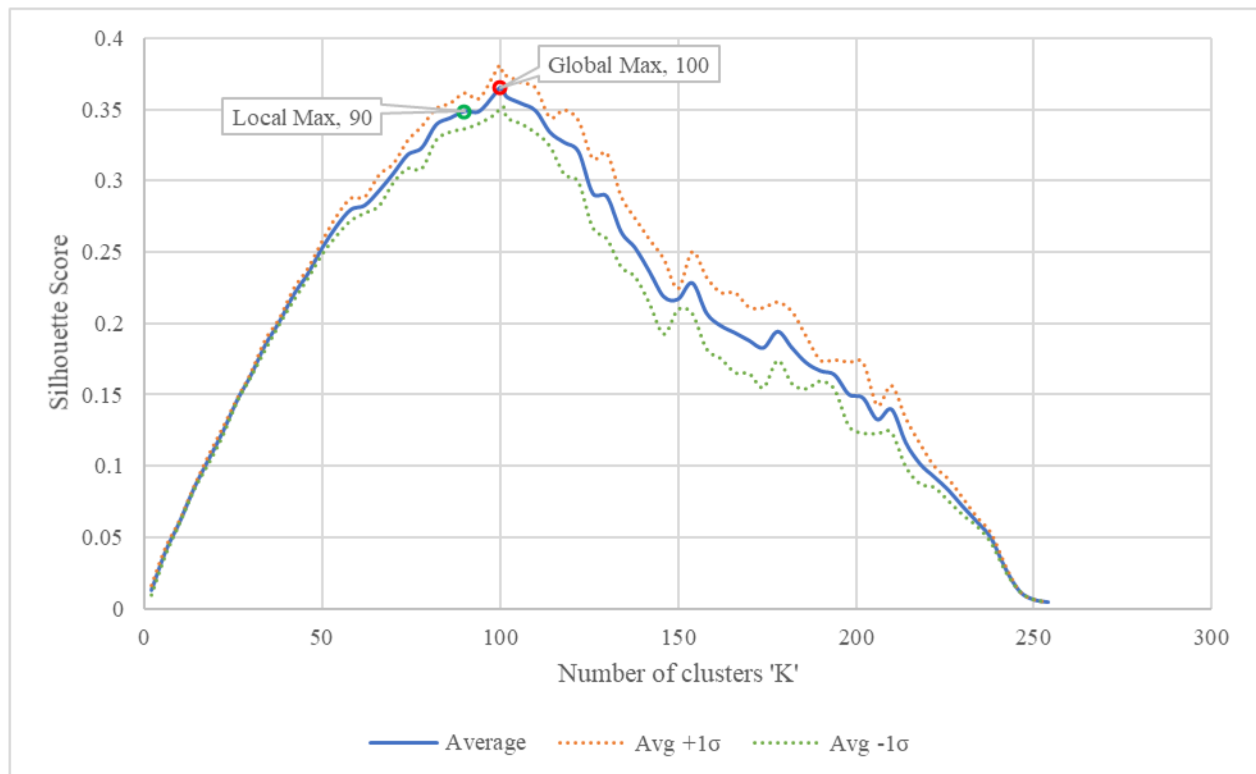

Figure 27: Silhouette plot for convolutional layer 6

Figure 28 is for convolutional layer 7 which has 256 filters, as in convolutional layer 5. Therefore, we have also obtained 2,540 clustering outcomes for convolutional layer 7. The value of ‘ic’, which ranges from 5 to 1, changes at the same rate as in convolutional layer 5 because both convolutional layers 5 and 7 have the same number of filters. As shown in Figure 28, the average silhouette scores widely range from 0.01 to 0.36. The global maximum average silhouette score was found to correspond to the value of K = 110. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 110 was 0.39. The selected local maximum silhouette score corresponds to the value of K = 90. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 90 was 0.37. The average ratio of sigma to the average silhouette scores is 0.051. The maximum ratio of sigma to the average silhouette scores is 0.162.

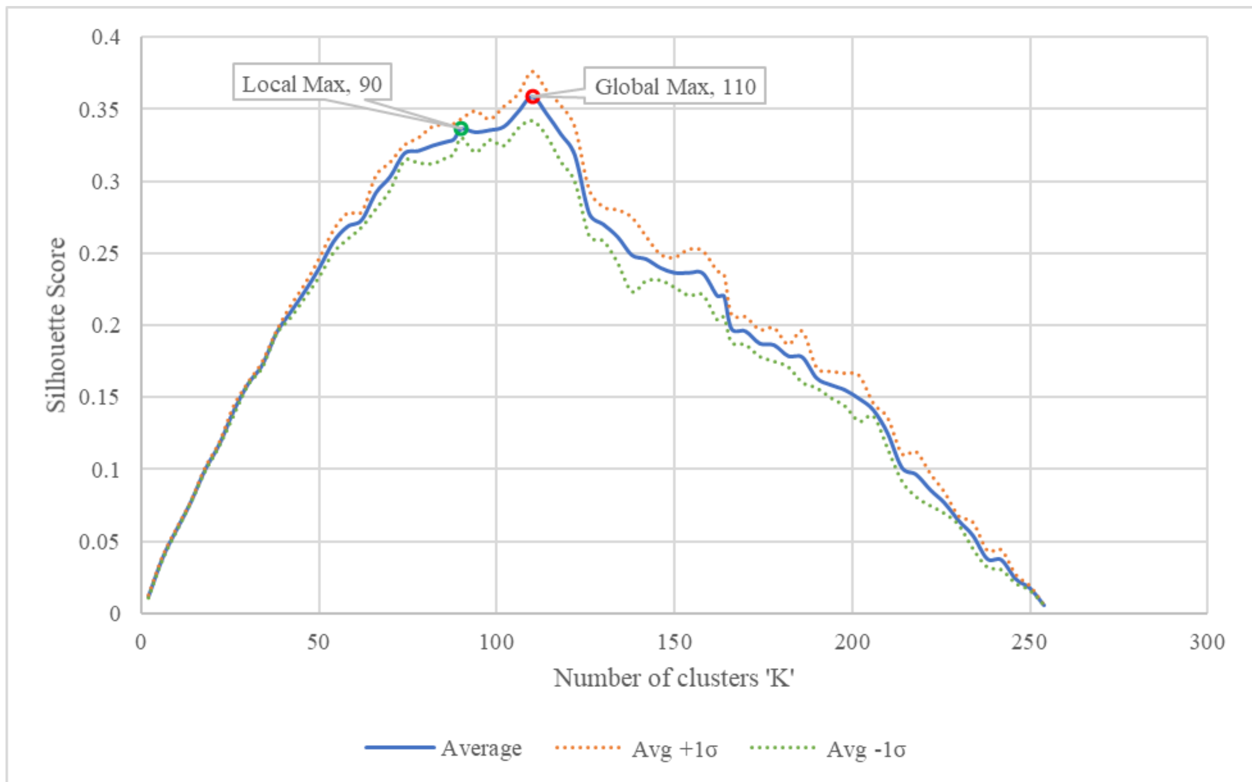

Figure 28: Silhouette plot for convolutional layer 7

Figure 29 is for convolutional layer 8 which has 512 filters. The number of clusters ‘K’ ranges from 2 to 511. This means that we have tested 510 values for ‘K’ and for each one we computed 10 clustering outcomes with each one giving us a different silhouette score. In total, we have obtained $510 \times 10 = 5{,}100$ clustering outcomes for convolutional layer 8. The value of ‘ic’ ranges from 5 to 1 and it gradually decreases as the value of ‘K’ increases. The value of ‘ic’ is set to 5 for the number of clusters ‘K’ ranging from 2 to 80 and is set to 4 for the number of clusters ‘K’ ranging from 81 to 112. In addition, the value of ‘ic’ is set to 3 for the number of clusters ‘K’ ranging from 113 to 160 and is set to 2 for the number of clusters ‘K’ ranging from 161 to 224. The value of ‘ic’ is set to 1 for the number of clusters ‘K’ ranging from 225 to 511. As shown in Figure 29, the average silhouette scores range from 0.01 to 0.30. The global maximum average silhouette score was found to correspond to the value of $K = 217$. The highest individual silhouette score out of the 10 silhouette scores we computed at $K = 217$ was 0.34. The selected local maximum silhouette score corresponds to the value of $K = 180$. The highest individual silhouette score out of the 10 silhouette scores we computed at $K = 180$ was 0.30. The average ratio of sigma to the average silhouette scores is 0.062. The maximum ratio of sigma to the average silhouette scores is 0.186.

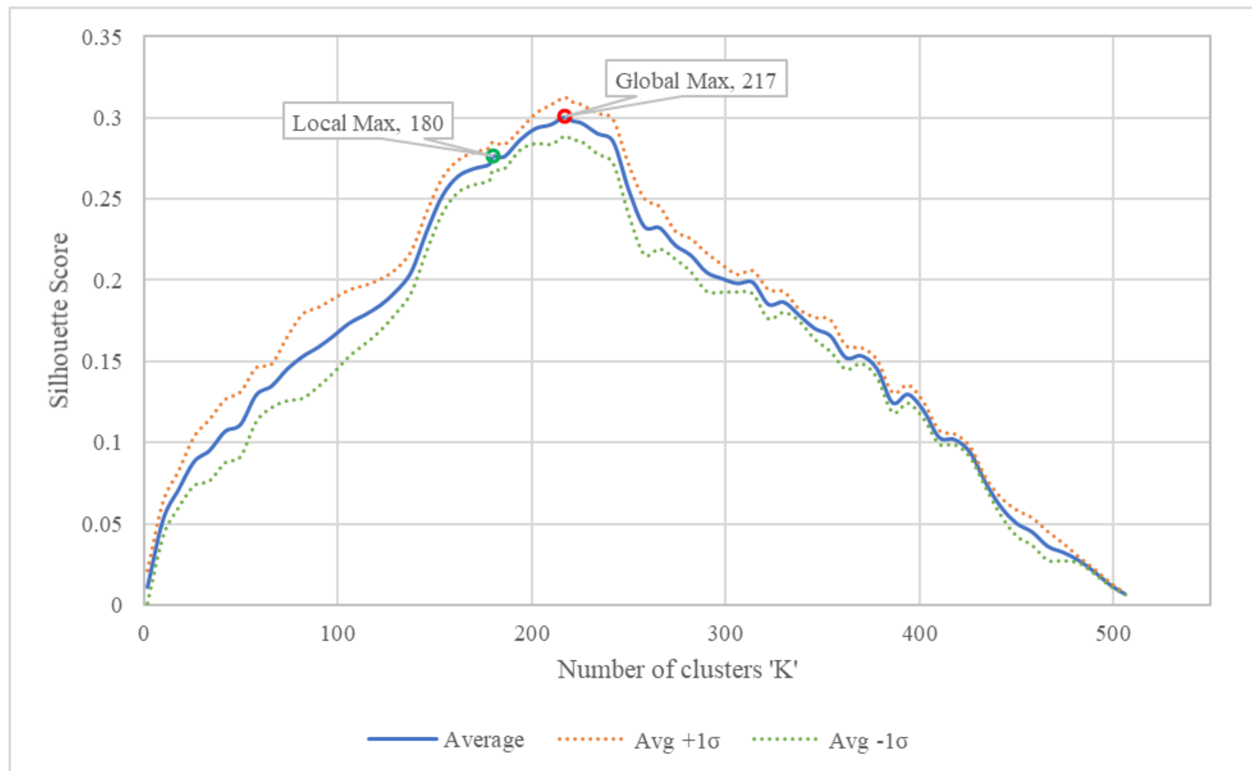

Figure 29: Silhouette plot for convolutional layer 8

Figure 30 is for convolutional layer 9 which has 512 filters, as in convolutional layer 8. Therefore, we have also obtained 5,100 clustering outcomes for convolutional layer 9. The value of 'ic', which ranges from 5 to 1, changes at the same rate as in convolutional layer 8 because both convolutional layers 8 and 9 have the same number of filters. As shown in Figure 30, the average silhouette scores range from 0.01 to 0.28. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 226 was 0.32. The selected local maximum silhouette score corresponds to the value of K = 180. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 180 was 0.31. The average ratio of sigma to the average silhouette scores is 0.042. The maximum ratio of sigma to the average silhouette scores is 0.091.

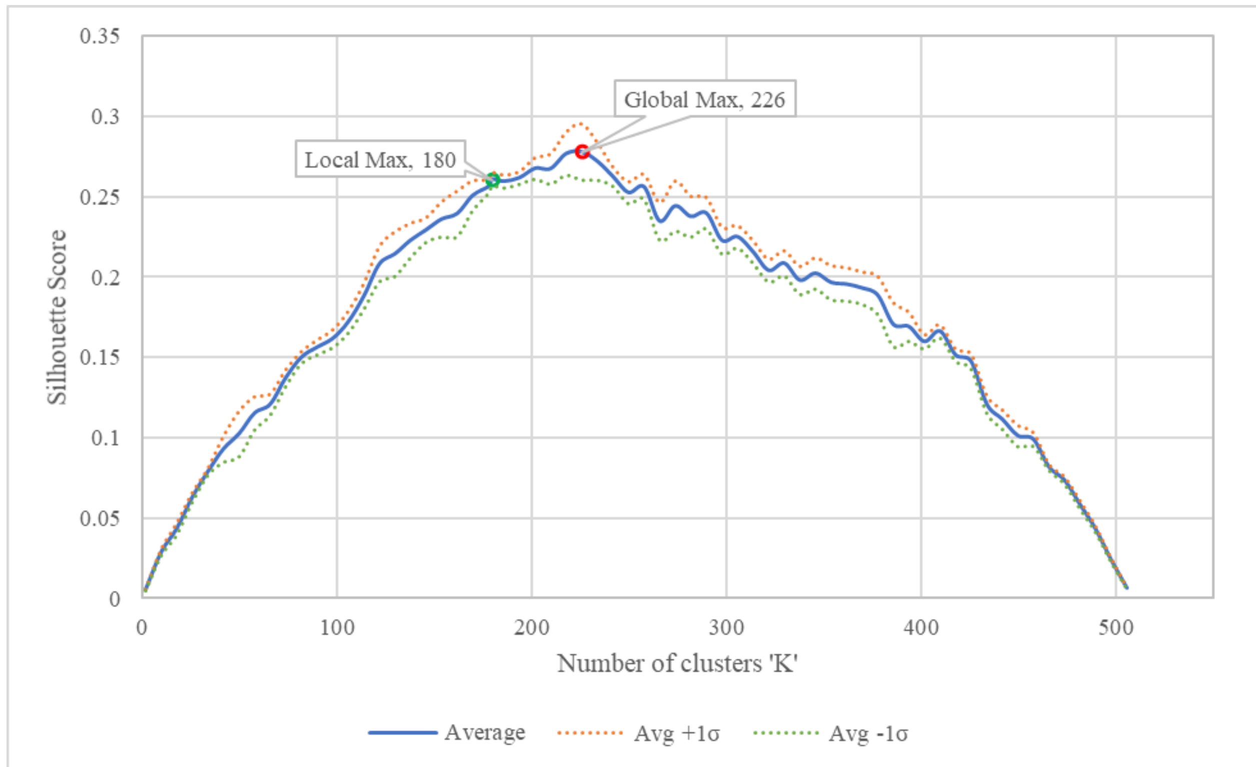

Figure 30: Silhouette plot for convolutional layer 9

Figure 31 is for convolutional layer 10 which has 512 filters, as in convolutional layer 8. Therefore, we have also obtained 5,100 clustering outcomes for convolutional layer 10. The value of 'ic', which ranges from 5 to 1, changes at the same rate as in convolutional layer 8 because both convolutional layers 8 and 10 have the same number of filters. As shown in Figure 31, the average silhouette scores range from 0.01 to 0.26. The global maximum average silhouette score was found to correspond to the value of K = 110. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 235 was 0.30. The two selected local maximum silhouette scores correspond to the values of K = 128 and K = 180. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 128 was 0.23. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 180 was 0.29. The average ratio of sigma

to the average silhouette scores is 0.045. The maximum ratio of sigma to the average silhouette scores is 0.125.

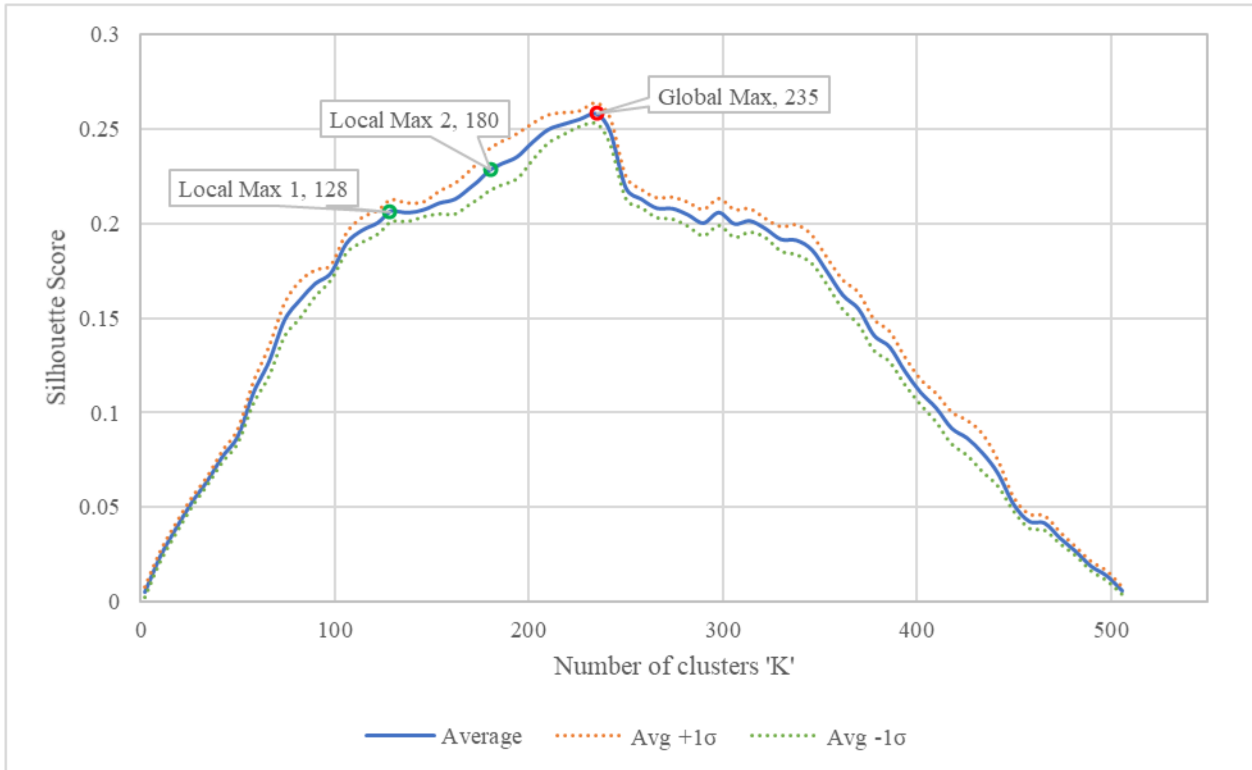

Figure 31: Silhouette plot for convolutional layer 10

Figure 32 is for convolutional layer 11 which has 512 filters, as in convolutional layer 8. Therefore, we have also obtained 5,100 clustering outcomes for convolutional layer 11. The value of 'ic', which ranges from 5 to 1, changes at the same rate as in convolutional layer 8 because both convolutional layers 8 and 11 have the same number of filters. As shown in Figure 32, the average silhouette scores range from 0.01 to 0.26. The global maximum average silhouette score was found to correspond to the value of K = 229. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 229 was 0.30. The two selected local maximum silhouette scores correspond to the values of K = 128 and K = 180. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 128 was 0.24. The highest individual silhouette

score out of the 10 silhouette scores we computed at K = 180 was 0.29. The average ratio of sigma to the average silhouette scores is 0.042. The maximum ratio of sigma to the average silhouette scores is 0.086.

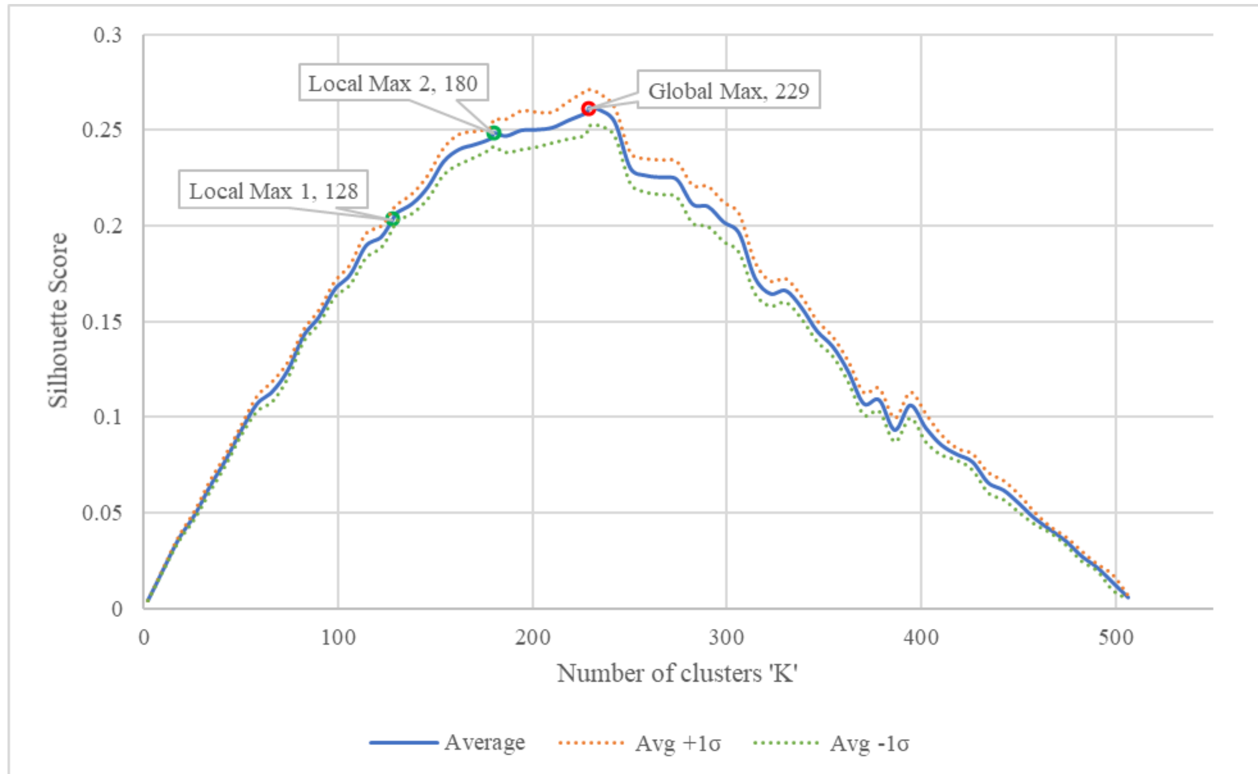

Figure 32: Silhouette plot for convolutional layer 11

Figure 33 is for convolutional layer 12 which has 512 filters, as in convolutional layer 8. Therefore, we have also obtained 5,100 clustering outcomes for convolutional layer 12. The value of 'ic', which ranges from 5 to 1, changes at the same rate as in convolutional layer 8 because both convolutional layers 8 and 12 have the same number of filters. As shown in Figure 33, the average silhouette scores range from 0.01 to 0.26. The global maximum average silhouette score was found to correspond to the value of K = 218. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 218 was 0.28. The two selected local maximum silhouette

scores correspond to the values of K = 128 and K = 180. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 128 was 0.23. The highest individual silhouette score out of the 10 silhouette scores we computed at K = 180 was 0.26. The average ratio of sigma to the average silhouette scores is 0.040. The maximum ratio of sigma to the average silhouette scores is 0.097.

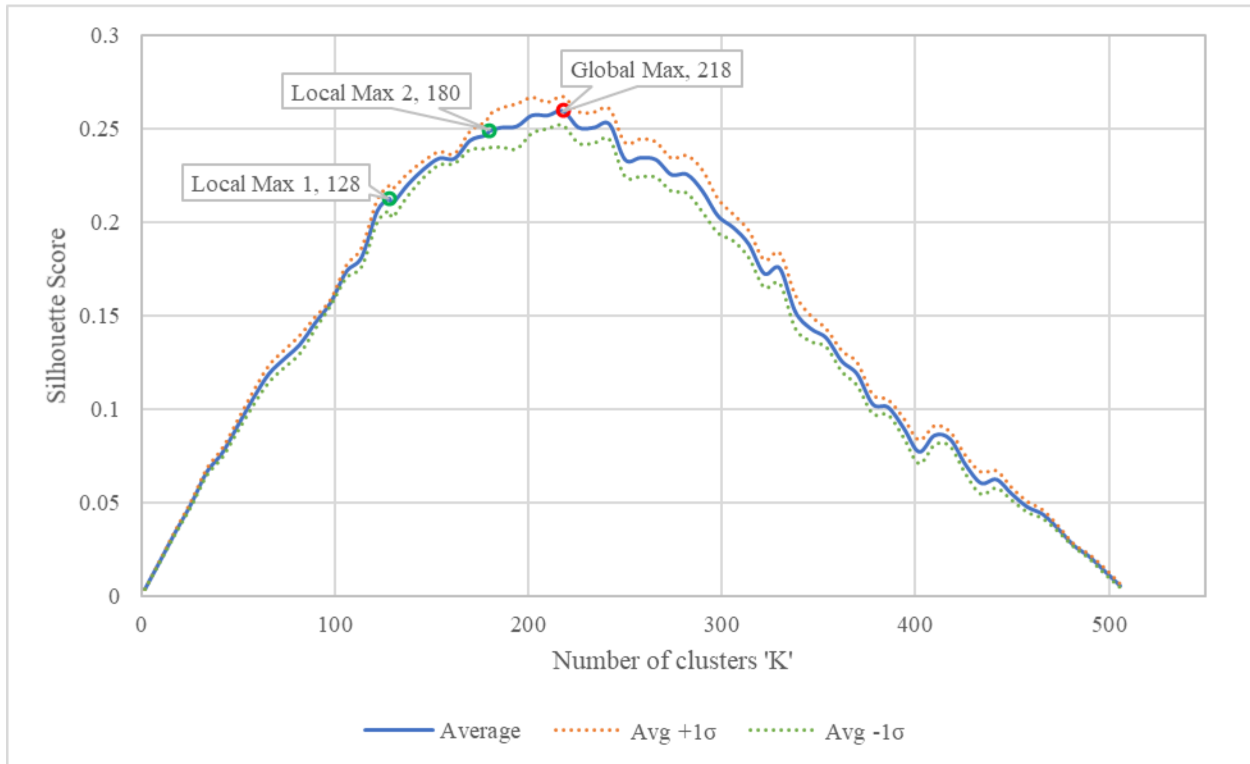

Figure 33: Silhouette plot for convolutional layer 12

4.2 Compression Scenarios

Tables 1, 2, and 3 provide the descriptions of the convolutional layers before and after pruning using 3 different compression scenarios. The first scenario A uses the ‘K’ values which have the global maximum silhouette scores found in section 4.1. The second and third scenarios B and C use ‘K’ values which have local maximum silhouette scores found in section 4.1. The tables

for the 3 compression scenarios are listed in ascending order in terms of the pruning rates for the total number of filters. The left side of each of these tables shows the size of the feature maps and the original number of filters, parameters, and FLOPs for each of the 12 convolutional layers before pruning. The right side of each of these tables shows the number of filters, parameters, and FLOPs after filter pruning. The numbers of parameters and FLOPs were calculated using the equations mentioned in Section 3.5. The second to last row for each of these tables reports the total number of filters, parameters, and FLOPs before and after pruning. The pruning rate for each convolutional layer is reported in the last column of each of these tables. The pruning rate represents the percentage of removed filters, which is also the same percentage of the removed parameters and FLOPs in each convolutional layer. In addition, the percentages of removed filters, parameters, and FLOPs from their respective totals are calculated and shown in the last row in each of these tables.

The number of remaining filters in each convolutional layer after filter pruning depends on the value of 'K' we selected in section 4.1. For example, the results for the first convolutional layer, in Table 3, show that we have removed 44 filters out of 64 filters, which means we have a pruning rate of 68.75% for each of the filters, parameters, and FLOPs. This leaves us with only 20 filters, 560 parameters, and 552,960 FLOPs after filter pruning. The number of remaining filters is the same value as 'K' we selected.

Some convolutional layers have the same number of filters but different number of parameters and FLOPs. For example, as shown in Table 3, convolutional layers 8 and 9 have the same number of filters, i.e., 512 filters, but convolutional layer 9 has a larger number of parameters and FLOPs than convolutional layer 8. When the same pruning rate is applied to different layers that have the same number of filters but different number of parameters and FLOPs, the resulting

number of parameters and FLOPs among the layers will be different. This is because when a pruning rate is applied to a layer, the numbers of filters, parameters, and FLOPs are reduced by the same rate. For example, Table 3 shows that the number of filters, parameters, and FLOPs, in convolutional layer 8, is reduced by the same percentage, which is equal to the pruning rate 64.84%. The same pruning rate is applied to convolutional layer 9, which results in reducing the number of filters, parameters, and FLOPs by 64.84%. However, the resulting number of parameters and FLOPs in convolutional layers 8 and 9 after pruning are different because the two layers originally had a different number of parameters and FLOPs before pruning. In this case, the number of parameters is reduced by 765,260 (from 1,180,160 to 414,900 as shown in Table 3) and 1,530,188 (from 2,359,808 to 829,620 as shown in Table 3), in layers 8 and 9, respectively. In addition, the number of FLOPs is reduced by 12,238,848 (from 18,874,368 to 6,635,520 as shown in Table 3) and 24,477,696 (from 37,748,736 to 13,271,040 as shown in Table 3), in layers 8 and 9, respectively. Therefore, the amount of reduction in the number of filters is the same in both layers, but the amount of reduction in the number of parameters and FLOPs is different. As a result, using the same pruning rate for different convolutional layers may result in a different number of reduced parameters and FLOPs, even though the layers have the same number of filters. This means that pruning certain convolutional layers might be more beneficial in terms of reducing more parameters and FLOPs than other convolutional layers.

A convolutional layer can have a larger number of parameters than another convolutional layer but a smaller number of FLOPs, and vice versa. For example, as shown in Table 3, convolutional layer 7 has a smaller number of parameters and larger number of FLOPs than convolutional layer 8. This is because the size of the feature maps in convolutional layer 7 is larger than that in convolutional layer 8. As such, if the aim from filter pruning a CNN is reducing the

number of parameters, then it would be beneficial to have a higher pruning rate for the layers that have a higher number of parameters, such as the last 5 convolutional layers described in Tables 1, 2, and 3. On the other hand, if the aim from filter pruning is reducing the number of FLOPs, then it would be beneficial to have a higher pruning rate for convolutional layers that have a higher number of FLOPs, such as convolutional layers 2, 4, 6, 7, 9, and 10 described in Tables 1, 2, and 3. Reducing the number of parameters results in a smaller CNN, while reducing the number of FLOPs results in a faster CNN.

In order to test higher pruning rates from the totals of filters, parameters, and FLOPs for scenarios B and C than those of scenario A, we based scenarios B and C on using local maximum silhouette scores instead of using global maximum silhouette scores as in scenario A. We will explore the effect of considering these different compression scenarios on the network's accuracy in the following section.

4.2.1 Compression Scenario A

In Table 1, the pruning rates for the convolutional layers range from 54.10% to 62.5%. The number of remaining filters for the convolutional layers ranges from 24 to 235. The last row in the table shows that we have removed 56.92%, 56.30%, and 56.94% of the total number of filters, parameters, and FLOPs, respectively.

	Original				After filter pruning			
Layer	Feature map size	Number of filters	Number of parameters	Number of FLOPs	Number of filters	Number of parameters	Number of FLOPs	Pruning rate
Conv 1	32x32	64	1,792	1,769,472	24	672	663,552	62.5%
Conv 2	32x32	64	36,928	37,748,736	29	16,733	17,104,896	54.69%
Conv 3	16x16	128	73,856	18,874,368	53	30,581	7,815,168	58.59%
Conv 4	16x16	128	147,584	37,748,736	56	64,568	16,515,072	56.25%
Conv 5	8x8	256	295,168	18,874,368	102	117,606	7,520,256	60.16%
Conv 6	8x8	256	590,080	37,748,736	100	230,500	14,745,600	60.94%
Conv 7	8x8	256	590,080	37,748,736	110	253,550	16,220,160	57.03%
Conv 8	4x4	512	1,180,160	18,874,368	217	500,185	7,999,488	57.62%
Conv 9	4x4	512	2,359,808	37,748,736	226	1,041,634	16,662,528	55.86%
Conv 10	4x4	512	2,359,808	37,748,736	235	1,083,115	17,326,080	54.10%
Conv 11	2x2	512	2,359,808	9,437,184	229	1,055,461	4,220,928	55.27%
Conv 12	2x2	512	2,359,808	9,437,184	218	1,004,762	4,018,176	57.42%
Total		3,712	12,354,880	303,759,360	1,599	5,399,367	130,811,904	-
Pruning rate from totals					56.92%	56.30%	56.94%	-

Table 1: The descriptions of the convolutional layers before and after pruning using compression scenario A

4.2.2 Compression Scenario B

In Table 2, the pruning rates for the convolutional layers range from 64.06% to 68.75%. The number of remaining filters for the convolutional layers ranges from 20 to 180. The last row in the table shows that we have removed 64.90%, 64.84%, and 64.77% of the total number of filters, parameters, and FLOPs, respectively.

	Original				After filter pruning			
Layer	Feature map size	Number of filters	Number of parameters	Number of FLOPs	Number of filters	Number of parameters	Number of FLOPs	Pruning rate
Conv 1	32x32	64	1,792	1,769,472	20	560	552,960	68.75%
Conv 2	32x32	64	36,928	37,748,736	23	13,271	13,565,952	64.06%
Conv 3	16x16	128	73,856	18,874,368	45	25,965	6,635,520	64.84%
Conv 4	16x16	128	147,584	37,748,736	45	51,885	13,271,040	64.84%
Conv 5	8x8	256	295,168	18,874,368	90	103,770	6,635,520	64.84%
Conv 6	8x8	256	590,080	37,748,736	90	207,450	13,271,040	64.84%
Conv 7	8x8	256	590,080	37,748,736	90	207,450	13,271,040	64.84%
Conv 8	4x4	512	1,180,160	18,874,368	180	414,900	6,635,520	64.84%
Conv 9	4x4	512	2,359,808	37,748,736	180	829,620	13,271,040	64.84%
Conv 10	4x4	512	2,359,808	37,748,736	180	829,620	13,271,040	64.84%
Conv 11	2x2	512	2,359,808	9,437,184	180	829,620	3,317,760	64.84%
Conv 12	2x2	512	2,359,808	9,437,184	180	829,620	3,317,760	64.84%
Total		3,712	12,354,880	303,759,360	1,303	4,343,731	107,016,192	-
Pruning rate from totals					64.90%	64.84%	64.77%	-

Table 2: The descriptions of the convolutional layers before and after pruning using compression scenario B

4.2.3 Compression Scenario C

In Table 3, the pruning rates for the convolutional layers range from 64.06% to 75%. The number of remaining filters for the convolutional layers ranges from 20 to 180. The last row in the table shows that we have removed 69.10%, 70.66%, and 66.66% of the total number of filters, parameters, and FLOPs, respectively.

	Original				After filter pruning			
Layer	Feature map size	Number of filters	Number of parameters	Number of FLOPs	Number of filters	Number of parameters	Number of FLOPs	Pruning rate
Conv 1	32x32	64	1,792	1,769,472	20	560	552,960	68.75%
Conv 2	32x32	64	36,928	37,748,736	23	13,271	13,565,952	64.06%
Conv 3	16x16	128	73,856	18,874,368	45	25,965	6,635,520	64.84%
Conv 4	16x16	128	147,584	37,748,736	45	51,885	13,271,040	64.84%
Conv 5	8x8	256	295,168	18,874,368	90	103,770	6,635,520	64.84%
Conv 6	8x8	256	590,080	37,748,736	90	207,450	13,271,040	64.84%
Conv 7	8x8	256	590,080	37,748,736	90	207,450	13,271,040	64.84%
Conv 8	4x4	512	1,180,160	18,874,368	180	414,900	6,635,520	64.84%
Conv 9	4x4	512	2,359,808	37,748,736	180	829,620	13,271,040	64.84%
Conv 10	4x4	512	2,359,808	37,748,736	128	589,952	9,437,184	75%
Conv 11	2x2	512	2,359,808	9,437,184	128	589,952	2,359,296	75%
Conv 12	2x2	512	2,359,808	9,437,184	128	589,952	2,359,296	75%
Total		3,712	12,354,880	303,759,360	1,147	3,624,727	101,265,408	-
Pruning rate from totals					69.10%	70.66%	66.66%	-

Table 3: The descriptions of the convolutional layers before and after pruning using compression scenario C

4.3 Accuracy Results

Tables 4, 5, and 6 show the resulting accuracy of the VGG-16 network on the CIFAR-10 dataset after filter pruning and retraining the network. The tables compare our SSIM based clustering filter pruning method and the state-of-the-art filter pruning method of HRank. Each table shows the accuracy for 10 experiments for each filter pruning method using the same compression scenario. This ensures fair comparison. The last 4 rows of each table show the average, standard

deviation, maximum, and minimum values of the 10 experiments for each filter pruning method. Tables 4, 5, and 6 show the results after filter pruning using compression scenarios A, B, and C, respectively. The baseline accuracy of the VGG-16 network before filter pruning is 93.96%.

4.3.1 Compression Scenario A

Table 4 shows that our filter pruning method outperforms the state-of-the-art filter pruning method of HRank using compression scenario A. Our average accuracy is higher than HRanks's average accuracy by 0.267 percentage points. Our maximum accuracy is higher than HRank's maximum accuracy by 0.23 percentage points. Our minimum accuracy is higher than HRank's minimum accuracy by 0.31 percentage points. Our standard deviation is lower than HRank's standard deviation. Our minimum accuracy is less than HRank's maximum accuracy by only 0.04 percentage points. After implementing our filter pruning method and retraining the network, there is on average a 2.46 percentage point decrease in accuracy compared to baseline accuracy. This means that we have preserved 97.38% of the original accuracy of the VGG-16 network even after pruning 56.92%, 56.30%, and 56.94% of the total number of considered filters, parameters, and FLOPs, respectively. As such we have achieved our goal of compressing the network with very little compromise to accuracy.

After performing the Shapiro-Wilk test on the 10 experiment accuracy values resulting from each filter pruning method found in Table 4, we confirmed that they indeed followed normal distribution. The resulting p-value of the test for the 10 accuracies of HRank was 0.450. The resulting p-value of the test for the 10 accuracies of our method was 0.411. As such, since the computed p-values are greater than the significance level alpha=0.05, we cannot reject the null

hypothesis H0, which states that the variable from which the sample was extracted follows a normal distribution.

We then performed the T-test to confirm whether there was a statistically significant difference between resulting accuracies of both filter pruning methods. The resulting p-value of the test was less than 0.0001. As such, since the computed p-value is lower than the significance level alpha=0.05, we should reject the null hypothesis H0, and accept the alternative hypothesis Ha, which states that the difference between the means is different from 0. This shows that there is a statistically significant difference between our average accuracy and HRank's average accuracy.

Filter Pruning Method	**HRank**	**Ours (SSIM based clustering)**
	Accuracy (%)	Accuracy (%)
Experiment 1	91.13 %	91.52 %
Experiment 2	91.34 %	91.65 %
Experiment 3	91.07 %	91.47 %
Experiment 4	91.10 %	91.49 %
Experiment 5	91.42 %	91.40 %
Experiment 6	91.17 %	91.64 %
Experiment 7	91.39 %	91.43 %
Experiment 8	91.24 %	91.49 %
Experiment 9	91.16 %	91.53 %
Experiment 10	91.31 %	91.38 %
Average	91.233 %	91.500 %
Standard Deviation	0.119	0.086
Maximum	91.42 %	91.65 %
Minimum	91.07 %	91.38 %

Table 4: Filter pruning resulting accuracy using compression scenario A

4.3.2 Compression Scenario B

Table 5 shows that our filter pruning method outperforms the state-of-the-art filter pruning method of HRank using compression scenario B. Our average accuracy is higher than HRanks's average accuracy by 0.454 percentage points. Our maximum accuracy is higher than HRank's maximum accuracy by 0.45 percentage points. Our minimum accuracy is higher than HRank's minimum accuracy by 0.43 percentage points. Although our standard deviation is slightly higher than HRank's standard deviation, our minimum accuracy is still higher than HRank's maximum accuracy by 0.12 percentage points. After implementing our filter pruning method and retraining the network, there is on average a 2.978 percentage point decrease in accuracy compared to baseline accuracy. This means that we have preserved 96.83% of the original accuracy of the VGG-16 network even after pruning 64.90%, 64.84%, and 64.77% of the total number of considered filters, parameters, and FLOPs, respectively. As such we have achieved our goal of compressing the network with very little compromise to accuracy.

After performing the Shapiro-Wilk test on the 10 experiment accuracy values resulting from each filter pruning method found in Table 5, we confirmed that they indeed followed normal distribution. The resulting p-value of the test for the 10 accuracies of HRank was 0.977. The resulting p-value of the test for the 10 accuracies of our method was 0.292. As such, since the computed p-values are greater than the significance level alpha=0.05, we cannot reject the null hypothesis H0, which states that the variable from which the sample was extracted follows a normal distribution.

We then performed the T-test to confirm whether there was a statistically significant difference between resulting accuracies of both filter pruning methods. The resulting p-value of

the test was less than 0.0001. As such, since the computed p-value is lower than the significance level alpha=0.05, we should reject the null hypothesis H0, and accept the alternative hypothesis Ha, which states that the difference between the means is different from 0. This shows that there is a statistically significant difference between our average accuracy and HRank's average accuracy.

Filter Pruning Method	**HRank**	**Ours (SSIM based clustering)**
	Accuracy (%)	Accuracy (%)
Experiment 1	90.49 %	90.87 %
Experiment 2	90.60 %	90.81 %
Experiment 3	90.58 %	91.10 %
Experiment 4	90.52 %	90.86 %
Experiment 5	90.62 %	91.01 %
Experiment 6	90.54 %	91.14 %
Experiment 7	90.41 %	91.00 %
Experiment 8	90.69 %	91.05 %
Experiment 9	90.38 %	91.10 %
Experiment 10	90.45 %	90.88 %
Average	90.528 %	90.982 %
Standard Deviation	0.093	0.112
Maximum	90.69 %	91.14 %
Minimum	90.38 %	90.81 %

Table 5: Filter pruning resulting accuracy using compression scenario B

4.3.3 Compression Scenario C

Table 6 shows that our filter pruning method outperforms the state-of-the-art filter pruning method of HRank using compression scenario C. Our average accuracy is higher than HRanks's average accuracy by 0.475 percentage points. Our maximum accuracy is higher than HRank's maximum accuracy by 0.65 percentage points. Our minimum accuracy is higher than HRank's minimum accuracy by 0.47 percentage points. Although our standard deviation is slightly higher

than HRank's standard deviation, our minimum accuracy is still higher than HRank's maximum accuracy by 0.05 percentage points. After implementing our filter pruning method and retraining the network, there is on average a 2.959 percentage point decrease in accuracy compared to baseline accuracy. This means that we have preserved 96.85% of the original accuracy of the VGG-16 network even after pruning 69.10%, 70.66%, and 66.66% of the total number of considered filters, parameters, and FLOPs, respectively. As such we have achieved our goal of compressing the network even further with very little compromise to accuracy. Although we have slightly higher pruning rates in compression scenario C compared to compression scenario B, we notice unexpectedly that our average accuracy in Table 6 is slightly higher than our average accuracy in Table 5. Yet, our standard deviation in Table 6 is also slightly higher than our standard deviation in Table 5.

After performing the Shapiro-Wilk test on the 10 experiment accuracy values resulting from each filter pruning method found in Table 6, we confirmed that they indeed followed normal distribution. The resulting p-value of the test for the 10 accuracies of HRank was 0.953. The resulting p-value of the test for the 10 accuracies of our method was 0.239. As such, since the computed p-values are greater than the significance level alpha=0.05, we cannot reject the null hypothesis H0, which states that the variable from which the sample was extracted follows a normal distribution.

We then performed the T-test to confirm whether there was a statistically significant difference between resulting accuracies of both filter pruning methods. The resulting p-value of the test was less than 0.0001. As such, since the computed p-value is lower than the significance level alpha=0.05, we should reject the null hypothesis H0, and accept the alternative hypothesis Ha, which states that the difference between the means is different from 0. This shows that there

is a statistically significant difference between our average accuracy and HRank's average accuracy.

Filter Pruning Method	HRank	Ours (SSIM based clustering)
	Accuracy (%)	Accuracy (%)
Experiment 1	90.47 %	90.80 %
Experiment 2	90.46 %	91.40 %
Experiment 3	90.75 %	90.97 %
Experiment 4	90.55 %	91.13 %
Experiment 5	90.68 %	91.09 %
Experiment 6	90.58 %	90.82 %
Experiment 7	90.33 %	91.12 %
Experiment 8	90.56 %	90.98 %
Experiment 9	90.38 %	90.81 %
Experiment 10	90.50 %	90.89 %
Average	90.526 %	91.001 %
Standard Deviation	0.121	0.179
Maximum	90.75 %	91.40 %
Minimum	90.33 %	90.80 %

Table 6: Filter pruning resulting accuracy using compression scenario C

Chapter 5

Conclusion and Future Work

In this thesis, we have proposed a new approach for filter pruning CNNs. Our filter pruning method utilizes K-Means clustering based on SSIM to group similar filters together and determine redundant filters in CNNs. The silhouette index is used for cluster validation and in our case is utilized to determine the best number of clusters. In our approach, the number of clusters is the same as the number of important filters in a CNN. The idea is that filters in the same cluster perform similar tasks in a convolutional layer. We have shown that it is possible for a single representative filter chosen from each cluster to compensate for the remaining filters in the same cluster. All remaining filters are considered redundant and hence can be pruned with very little consequence. We evaluated our filter pruning method on the VGG-16 architecture with the benchmark CIFAR-10 dataset and experimented with three different compression scenarios. Our new filter pruning method has demonstrated its experimental effectiveness and efficiency in model compression, acceleration, and accuracy. Our experimental results have shown that it is possible to prune substantial parts of a CNN, which makes them smaller and faster, with very minimal compromise to the accuracy of the network. The results provide promising indications that our method can consistently and significantly outperform the current state-of-the-art filter pruning method. We hope that our filter pruning method will aid in making CNNs more generalized and bring research a step closer to the possibility of running CNNs on devices with limited hardware capabilities.

Future work in this field of research includes substituting our K-Means clustering algorithm and implementing different clustering algorithms such as Fuzzy C-Means clustering, Gustafson-Kessel fuzzy clustering, or Gaussian Mixture Models clustering. This could lead to better clustering possibilities. Another lane to explore is in the area of cluster validation. Numerous cluster validation indices could be implemented in future work. In addition, changing the optimizer or related parameters such as the learning rate may improve the accuracy of the CNN in the retraining step after filter pruning.

Our experiments could be expanded to include several datasets and CNN architectures. We want to evaluate our filter pruning algorithm on VGG-16 with larger datasets such as CIFAR-100 and ImageNet. In addition, we want to evaluate our work on different CNN architectures such as DenseNets, ResNets, and GoogLeNet with various datasets such as CIFAR-10, CIFAR-100, and ImageNet.

References

[1] “How Google Translate Squeezes Deep Learning onto a Phone.” Google AI Blog, July 29, 2015. https://ai.googleblog.com/2015/07/how-google-translate-squeezes-deep.html.

[2] Ananthakrishnan, Shankar. “Amazon Scientists Applying Deep Neural Networks to Custom Skills.” Amazon Science. Amazon Science, July 23, 2020. https://www.amazon.science/blog/amazon-scientists-applying-deep-neural-networks-to-custom-skills.

[3] “An On-Device Deep Neural Network for Face Detection.” Apple Machine Learning Research. Accessed April 1, 2021. https://machinelearning.apple.com/research/face-detection.

[4] Marr, Bernard. “4 Mind-Blowing Ways Facebook Uses Artificial Intelligence.” Forbes. Forbes Magazine, December 12, 2018. https://www.forbes.com/sites/bernardmarr/2016/12/29/4-amazing-ways-facebook-uses-deep-learning-to-learn-everything-about-you/?sh=225e7ccac.

[5] Kris Zentner. “Microsoft Research: How We Operate Deep Neural Network with Log Analytics.” Azure Blog and Updates | Microsoft Azure. Accessed April 1, 2021. https://azure.microsoft.com/en-us/blog/microsoft-research-how-we-operate-deep-neural-network-with-log-analytics/.

[6] “Autopilot AI.” Tesla. Accessed April 1, 2021. https://www.tesla.com/en_CA/autopilotAI.

[7] Sharma, Neha, Reecha Sharma, and Neeru Jindal. "Machine Learning and Deep Learning Applications-A Vision." Global Transitions Proceedings 2, no. 1 (2021): 24-28.

[8] Wang, Jie, and Zihao Li. "Research on face recognition based on CNN." In IOP Conference Series: Earth and Environmental Science, vol. 170, no. 3, p. 032110. IOP Publishing, 2018.

[9] Lameri, Silvia, Federico Lombardi, Paolo Bestagini, Maurizio Lualdi, and Stefano Tubaro. "Landmine detection from GPR data using convolutional neural networks." In 2017 25th European Signal Processing Conference (EUSIPCO), pp. 508-512. IEEE, 2017.

[10] Shaban, Mohamed, Reem Salim, Hadil Abu Khalifeh, Adel Khelifi, Ahmed Shalaby, Shady El-Mashad, Ali Mahmoud, Mohammed Ghazal, and Ayman El-Baz. "A Deep-Learning Framework for the Detection of Oil Spills from SAR Data." Sensors 21, no. 7 (2021): 2351.

[11] Park, Jongchan, Joon-Young Lee, Donggeun Yoo, and In So Kweon. "Distort-and-recover: Color enhancement using deep reinforcement learning." In Proceedings of the IEEE Conference on computer vision and pattern recognition, pp. 5928-5936. 2018.

[12] Zhang, Jinwei, Zhe Liu, Shun Zhang, Hang Zhang, Pascal Spincemaille, Thanh D. Nguyen, Mert R. Sabuncu, and Yi Wang. "Fidelity imposed network edit (FINE) for solving ill-posed image reconstruction." Neuroimage 211 (2020): 116579.

[13] Lalonde, Jean-François. "Deep learning for augmented reality." In 2018 17th Workshop on Information Optics (WIO), pp. 1-3. IEEE, 2018.

[14] Thomas, Ajith, and John Hedley. "FumeBot: A Deep Convolutional Neural Network Controlled Robot." Robotics 8, no. 3 (2019): 62.

[15] Bluche, Théodore. "Deep neural networks for large vocabulary handwritten text recognition." PhD diss., Paris 11, 2015.

[16] Bayer, Ali Orkan, and Giuseppe Riccardi. "Semantic language models with deep neural networks." Computer Speech & Language 40 (2016): 1-22.

[17] Miotto, Riccardo, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T. Dudley. "Deep learning for healthcare: review, opportunities and challenges." Briefings in bioinformatics 19, no. 6 (2018): 1236-1246.

[18] Litjens, Geert, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I. Sánchez. "A survey on deep learning in medical image analysis." Medical image analysis 42 (2017): 60-88.

[19] Wieslander, Hakan, Gustav Forslid, Ewert Bengtsson, Carolina Wahlby, Jan-Michael Hirsch, Christina Runow Stark, and Sajith Kecheril Sadanandan. "Deep convolutional neural networks for detecting cellular changes due to malignancy." In Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 82-89. 2017.

[20] Zhou, Yao, and Gary G. Yen. "Evolving deep neural networks for movie box-office revenues prediction." In 2018 IEEE Congress on Evolutionary Computation (CEC), pp. 1-8. IEEE, 2018.

[21] Justesen, Niels, Philip Bontrager, Julian Togelius, and Sebastian Risi. "Deep learning for video game playing." IEEE Transactions on Games 12, no. 1 (2019): 1-20.

[22] Zhang, Pengjing, Xiaoqing Zheng, Wenqiang Zhang, Siyan Li, Sheng Qian, Wenqi He, Shangtong Zhang, and Ziyuan Wang. "A deep neural network for modeling music." In

Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, pp. 379-386. 2015.

[23] Lomov, Ildar, and Ilya Makarov. "Generative Models for Fashion Industry using Deep Neural Networks." In 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), pp. 1-6. IEEE, 2019.

[24] FinancialNewsMedia.com. Global Gaming, Media & Entertainment Market Could Exceed $2.1 Trillion In 2021, April 13, 2021. https://www.prnewswire.com/news-releases/global-gaming-media--entertainment-market-could-exceed-2-1-trillion-in-2021--301267299.html.

[25] Global Fashion Industry Statistics. Accessed April 18, 2021. https://fashionunited.com/global-fashion-industry-statistics/.

[26] Haykin, Simon. Neural networks and learning machines, 3/E. Pearson Education India, 2010.

[27] Luo, Jian-Hao, 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, pp. 5058-5066. 2017.

[28] Lin, Shaohui, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. "Towards optimal structured cnn pruning via generative adversarial learning." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2790-2799. 2019.

[29] Zhao, Chenglong, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. "Variational convolutional neural network pruning." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2780-2789. 2019.

[30] Lin, Shaohui, Rongrong Ji, Yuchao Li, Cheng Deng, and Xuelong Li. "Toward compact convnets via structure-sparsity regularized filter pruning." IEEE transactions on neural networks and learning systems 31, no. 2 (2019): 574-588.

[31] Lin, Mingbao, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. "Hrank: Filter pruning using high-rank feature map." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1529-1538. 2020.

[32] Han, Song, Jeff Pool, John Tran, and William J. Dally. "Learning both weights and connections for efficient neural networks." arXiv preprint arXiv:1506.02626 (2015).

[33] Gupta, Suyog, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep learning with limited numerical precision." In International conference on machine learning, pp. 1737-1746. PMLR, 2015.

[34] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).

[35] Denton, Emily L., Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. "Exploiting linear structure within convolutional networks for efficient evaluation." In Advances in neural information processing systems, pp. 1269-1277. 2014.

[36] Carreira-Perpinán, Miguel A., and Yerlan Idelbayev. "“learning-compression” algorithms for neural net pruning." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8532-8541. 2018.

[37] Li, Hao, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. "Pruning filters for efficient convnets." arXiv preprint arXiv:1608.08710 (2016).

[38] Han, Song, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. "EIE: Efficient inference engine on compressed deep neural network." ACM SIGARCH Computer Architecture News 44, no. 3 (2016): 243-254.

[39] Yamashita, Rikiya, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. "Convolutional neural networks: an overview and application in radiology." Insights into imaging 9, no. 4 (2018): 611-629.

[40] Molchanov, Pavlo, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. "Pruning convolutional neural networks for resource efficient inference." arXiv preprint arXiv:1611.06440 (2016).

[41] Hu, Hengyuan, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures." arXiv preprint arXiv:1607.03250 (2016).

[42] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[43] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

[44] McCulloch, Warren S., and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics 5, no. 4 (1943): 115-133.

[45] Abiodun, Oludare Isaac, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. "State-of-the-art in artificial neural network applications: A survey." Heliyon 4, no. 11 (2018): e00938.

[46] Rumelhart, David E., James L. McClelland, and PDP Research Group. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations." (1986).

[47] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[48] LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.

[49] Hubel, David H., and Torsten N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." The Journal of physiology 160, no. 1 (1962): 106-154.

[50] Fukushima, Kunihiko, and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition." In Competition and cooperation in neural nets, pp. 267-285. Springer, Berlin, Heidelberg, 1982.

[51] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012): 1097-1105.

[52] Khan, Asifullah, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. "A survey of the recent architectures of deep convolutional neural networks." Artificial Intelligence Review 53, no. 8 (2020): 5455-5516.

[53] Alom, Md Zahangir, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. "The history began from alexnet: A comprehensive survey on deep learning approaches." arXiv preprint arXiv:1803.01164 (2018).

[54] Chollet, Francois. Deep learning with Python. Simon and Schuster, 2017.

[55] Liu, Mengchen, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. "Towards better analysis of deep convolutional neural networks." IEEE transactions on visualization and computer graphics 23, no. 1 (2016): 91-100.

[56] Choo, Jaegul, and Shixia Liu. "Visual analytics for explainable deep learning." IEEE computer graphics and applications 38, no. 4 (2018): 84-92.

[57] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In International conference on machine learning, pp. 448-456. PMLR, 2015.

[58] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15, no. 1 (2014): 1929-1958.

[59] Zhang, Zhilu, and Mert R. Sabuncu. "Generalized cross entropy loss for training deep neural networks with noisy labels." In 32nd Conference on Neural Information Processing Systems (NeurIPS). 2018.

[60] Szegedy, Christian, 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 conference on computer vision and pattern recognition, pp. 1-9. 2015.

[61] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[62] Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely connected convolutional networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700-4708. 2017.

[63] Osajima, Jason. “Convolutional Networks - VGG16.” Convolutional Networks - VGG16 | Jason {osa-jima}, August 18, 2018. https://www.jasonosajima.com/convnets_vgg.html.

[64] Jefkine. “Backpropagation In Convolutional Neural Networks.” DeepGrid, September 5, 2016. https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/.

[65] Reed, Russell. "Pruning algorithms-a survey." IEEE transactions on Neural Networks 4, no. 5 (1993): 740-747.

[66] Lee, Namhoon, Thalaiyasingam Ajanthan, and Philip HS Torr. "Snip: Single-shot network pruning based on connection sensitivity." arXiv preprint arXiv:1810.02340 (2018).

[67] Chauvin, Yves. "A Back-Propagation Algorithm with Optimal Use of Hidden Units." In NIPS, vol. 1, pp. 519-526. 1988.

[68] Weigend, Andreas S., David E. Rumelhart, and Bernardo A. Huberman. "Generalization by weight-elimination with application to forecasting." In Advances in neural information processing systems, pp. 875-882. 1991.

[69] Mozer, Michael C., and Paul Smolensky. "Skeletonization: A technique for trimming the fat from a network via relevance assessment." In Advances in neural information processing systems, pp. 107-115. 1989.

[70] LeCun, Yann, John S. Denker, and Sara A. Solla. "Optimal brain damage." In Advances in neural information processing systems, pp. 598-605. 1990.

[71] Hassibi, Babak, David G. Stork, and Gregory J. Wolff. "Optimal brain surgeon and general network pruning." In IEEE international conference on neural networks, pp. 293-299. IEEE, 1993.

[72] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).

[73] He, Yihui, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks." In Proceedings of the IEEE international conference on computer vision, pp. 1389-1397. 2017.

[74] Luo, Jian-Hao, 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, pp. 5058-5066. 2017.

[75] Yu, Ruichi, 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 Conference on Computer Vision and Pattern Recognition, pp. 9194-9203. 2018.

[76] He, Yang, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. "Filter pruning via geometric median for deep convolutional neural networks acceleration." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4340-4349. 2019.

[77] Halkidi, Maria, Yannis Batistakis, and Michalis Vazirgiannis. "On clustering validation techniques." Journal of intelligent information systems 17, no. 2 (2001): 107-145.

[78] Dhanachandra, Nameirakpam, Khumanthem Manglem, and Yambem Jina Chanu. "Image segmentation using K-means clustering algorithm and subtractive clustering algorithm." Procedia Computer Science 54 (2015): 764-771.

[79] Berkhin, Pavel. "A survey of clustering data mining techniques." In Grouping multidimensional data, pp. 25-71. Springer, Berlin, Heidelberg, 2006.

[80] Baraldi, Andrea, and Palma Blonda. "A survey of fuzzy clustering algorithms for pattern recognition. I." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 29, no. 6 (1999): 778-785.

[81] Wang, Xi. "Application of Weighted Fuzzy Clustering Algorithm in Urban Economics Development." In International conference on Big Data Analytics for Cyber-Physical-Systems, pp. 1698-1702. Springer, Singapore, 2020.

[82] Nugent, Rebecca, and Marina Meila. "An overview of clustering applied to molecular biology." Statistical methods in molecular biology (2010): 369-404.

[83] Verleysen, Frederik T., and Arie Weeren. "Clustering by publication patterns of senior authors in the social sciences and humanities." Journal of Informetrics 10, no. 1 (2016): 254-272.

[84] Xu, Rui, and Don Wunsch. Clustering. Vol. 10. John Wiley & Sons, 2008.

[85] Anderberg, Michael R. Cluster analysis for applications: probability and mathematical statistics: a series of monographs and textbooks. Vol. 19. Academic press, 2014.

[86] Jain, Anil K., M. Narasimha Murty, and Patrick J. Flynn. "Data clustering: a review." ACM computing surveys (CSUR) 31, no. 3 (1999): 264-323.

[87] Aggarwal, Charu C., and Chandan K. Reddy. "Data clustering." Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra (2014).

[88] Dunn, Joseph C. "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters." (1973): 32-57.

[89] Lloyd, Stuart. "Least squares quantization in PCM." IEEE transactions on information theory 28, no. 2 (1982): 129-137.

[90] Forgy, Edward W. "Cluster analysis of multivariate data: efficiency versus interpretability of classifications." biometrics 21 (1965): 768-769.

[91] MacQueen, James. "Some methods for classification and analysis of multivariate observations." In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol. 1, no. 14, pp. 281-297. 1967.

[92] Singh, Archana, Avantika Yadav, and Ajay Rana. "K-means with Three different Distance Metrics." International Journal of Computer Applications 67, no. 10 (2013).

[93] Hussain, Syed Fawad, and Muhammad Haris. "A k-means based co-clustering (kCC) algorithm for sparse, high dimensional data." Expert Systems with Applications 118 (2019): 20-34.

[94] Bezdek, James C., and Richard J. Hathaway. "Some notes on alternating optimization." In AFSS international conference on fuzzy systems, pp. 288-300. Springer, Berlin, Heidelberg, 2002.

[95] Arthur, David, and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Stanford, 2006.

[96] Arbelaitz, Olatz, Ibai Gurrutxaga, Javier Muguerza, Jesús M. Pérez, and Iñigo Perona. "An extensive comparative study of cluster validity indices." Pattern Recognition 46, no. 1 (2013): 243-256.

[97] Theodoridis, Sergios, and Konstantinos Koutroumbas. Pattern Recognition, 4th Edition. Academic Press, 2009.

[98] Hubert, Lawrence, and Phipps Arabie. "Comparing partitions." Journal of classification 2, no. 1 (1985): 193-218.

[99] Thorndike, Robert L. "Who belongs in the family?." Psychometrika 18, no. 4 (1953): 267-276.

[100] Rousseeuw, Peter J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." Journal of computational and applied mathematics 20 (1987): 53-65.

[101] Dunn, Joseph C. "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters." (1973): 32-57.

[102] Caliński, Tadeusz, and Jerzy Harabasz. "A dendrite method for cluster analysis." Communications in Statistics-theory and Methods 3, no. 1 (1974): 1-27.

[103] Davies, David L., and Donald W. Bouldin. "A cluster separation measure." IEEE transactions on pattern analysis and machine intelligence 2 (1979): 224-227.

[104] Petrovic, Slobodan. "A comparison between the silhouette index and the davies-bouldin index in labelling ids clusters." In Proceedings of the 11th Nordic Workshop of Secure IT Systems, vol. 2006, pp. 53-64. sn, 2006.

[105] Chang, Mark. "Artificial intelligence for drug development, precision medicine, and healthcare." (2020).

[106] Wang, Zhou, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. "Image quality assessment: from error visibility to structural similarity." IEEE transactions on image processing 13, no. 4 (2004): 600-612.

[107] Yuanji, Wang, Li Jianhua, Lu Yi, Fu Yao, and Jiang Qinzhong. "Image quality evaluation based on image weighted separating block peak signal to noise ratio." In International

Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003, vol. 2, pp. 994-997. IEEE, 2003.

[108] Lubin, Jeffrey, and David Fibush. "Sarnoff JND vision model." (1997): 97.

[109] Wang, Zhou, and Alan C. Bovik. "A universal image quality index." IEEE signal processing letters 9, no. 3 (2002): 81-84.

[110] Wang, Zhou, and Alan C. Bovik. "Mean squared error: Love it or leave it? A new look at signal fidelity measures." IEEE signal processing magazine 26, no. 1 (2009): 98-117.

[111] LeCun, Yann. "The MNIST database of handwritten digits." http://yann. lecun. com/exdb/mnist/ (1998).

[112] Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747 (2017).

[113] Del Bimbo, Alberto, Rita Cucchiara, Stan Sclaroff, Giovanni Maria Farinella, Tao Mei, Marco Bertini, Hugo Jair Escalante, and Roberto Vezzani, eds. Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10-15, 2021, Proceedings, Part III. Vol. 12663. Springer Nature, 2021.

[114] Li, Hongmin, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. "Cifar10-dvs: an event-stream dataset for object classification." Frontiers in neuroscience 11 (2017): 309.

[115] Torralba, Antonio, Rob Fergus, and William T. Freeman. "80 million tiny images: A large data set for nonparametric object and scene recognition." IEEE transactions on pattern analysis and machine intelligence 30, no. 11 (2008): 1958-1970.

[116] Zagoruyko, Sergey. "92.45 on cifar-10 in torch, 2015." URL http://torch.ch/blog/ 2015/07/30/cifar. html.

[117] Lin, Mingbao, and Ethan Zhang. “Lmbxmu/HRank: Pytorch Implementation of Our Paper Accepted by CVPR 2020 (Oral) -- HRank: Filter Pruning Using High-Rank Feature Map.” GitHub. https://github.com/lmbxmu/HRank.

[118] Van der Walt, Stefan, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. "scikit-image: image processing in Python." PeerJ 2 (2014): e453.

[119] “Sklearn.metrics.silhouette_score.” scikit, n.d. https://scikit-learn.org/stable/modules/ generated/ sklearn.metrics.silhouette_score.html.

[120] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[121] Shapiro, Samuel Sanford, and Martin B. Wilk. "An analysis of variance test for normality (complete samples)." Biometrika 52, no. 3/4 (1965): 591-611.

[122] Hines, William W., Douglas C. Montgomery, and David M. Goldman Connie M. Borror. Probability and statistics in engineering. John Wiley & Sons, 2008.