Data Science and AI

Custom pooling methods for Convolutional Neural Networks

Yesmina El Arkoubi

Supervisor:
Prof. Dr. Michael S. Lew & Dr. Erwin M. Bakker

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 07-08-2024

Abstract

Convolutional Neural Networks(CNNs) have been successfully used in image classification tasks, with architectures like AlexNet, VGG, and ResNet setting performance benchmarks on various tasks. The pooling layer is one of the building blocks of CNNs, crucial in reducing spatial dimensions and improving computational efficiency. The most common pooling methods are maximum and average pooling. However, research has indicated that these pooling methods are not qualified to be optimal. In this study, a new pooling method named AVG-Mixed is proposed. This pooling method aims to improve the overall performance of CNNs on image classification by combining the values from both max and average pooling. Furthermore, this study further investigates the effectiveness of AVG-TopK, a recently proposed pooling method previously evaluated in LeNet, and transfer learning models like ResNet50. The AVG-Mixed and AVG-TopK pooling methods aim to combine the strengths of traditional pooling methods while mitigating their weaknesses by providing a more balanced approach.

The main objective of this study is to investigate the effectiveness of the AVG-Mixed and AVG-TopK pooling methods in CNNs for image classification tasks. The effectiveness of AVG-Mixed and AVG-TopK is assessed by integrating them into AlexNet, VGG16, and ResNet34 networks, and evaluating their performance on the MNIST, CIFAR-10, and CIFAR-100 datasets using accuracy and confusion matrices. A comparative assessment is conducted to determine the impact of the custom pooling methods on the performance of CNNs, comparing them with standard max pooling.

The experimental results demonstrate that the AVG-Mixed pooling method improved the performance of VGG16 and ResNet34 on the CIFAR-10 and CIFAR-100 datasets, compared to their baseline models with max pooling. The AVG-TopK pooling method slightly improved the performance of AlexNet on CIFAR-10, compared to the baseline model with max pooling. Additionally, the AVG-TopK pooling method consistently improved the performance of ResNet34 on the MNIST, CIFAR-10, and CIFAR-100 datasets, compared to the baseline ResNet34 model with max pooling.

Keywords: Convolutional Neural Networks, Pooling Methods, Image Classification, Downsampling.

Contents

1 Introduction

1.1 Background and motivation

Deep learning, a branch of machine learning, has made breakthroughs in fields such as computer vision, natural language processing, and speech recognition. This approach has surpassed traditional machine learning approaches in many tasks by its ability to automatically learn feature representations from large volumes of data and handle non-linear relationships [11]. One of the most prominent deep learning networks for computer vision tasks is Convolutional Neural Networks(CNNs) [48, 28]. CNNs can learn hierarchical feature representations directly from the data and identify patterns in images that would be difficult for traditional machine learning algorithms to achieve [11].

The neocognitron, which was proposed by Fukushima in 1980, is recognized as the origin of deep CNNs [12]. Its architecture was inspired by the neurophysiological findings, from the research by Hubel and Wiesel, on the visual cortex of mammals [12]. In 1998, LeNet-5 [26] was proposed which is one of the first successful applications of CNNs for image classification tasks. Despite its success, this early attempt was limited by the small size of available datasets and the low computational power at the time. CNNs have become state-of-the-art image classifiers since the ImageNet Large Scale Visual Recognition Competition(ILSVRC) in 2012 [24] [38]. This competition has been crucial in demonstrating the effectiveness of deep CNNs in image classification, object detection, and other computer vision tasks.

CNNs consist of three main types of layers: convolutional layers, pooling layers, and fully connected layers. In the convolutional layers, different kernels are applied to convolve the input image to create feature maps. This layer is often followed by a pooling layer that reduces the size of the feature maps and network parameters. The two main purposes of the pooling layer are to reduce the computational costs by reducing the number of parameters and to prevent the model from overfitting [14]. The pooling layer is often followed by a flattening layer and fully connected layers. In the flatten layer, the multi-dimensional feature maps are converted to 1D vectors that are used in the fully connected layers for the classification of the images.

Despite their success, CNNs are a relatively new technology, and there is a lot of research being done on enhancing their performance and capabilities. A significant focus is on improving pooling methods to better capture important features while reducing the spatial dimensions of feature maps [7]. The pooling layer greatly influences the performance of CNNs, which makes it essential to pay more attention to the choice of the pooling function. The effectiveness of the pooling method is influenced by, among others, the dataset and the specific computer vision task [7].

Max pooling and average pooling are the most commonly utilized pooling methods in CNNs due to their simplicity and usability [50]. Max pooling selects the largest value within each pooling window, while average pooling computes the average value within each window. Moreover, max pooling assumes that the most discriminative feature should be of the maximum activation value while average pooling assumes the local equality of features. In tasks such as image classification, where the presence of certain features is more important than the precise location, max pooling is often superior in performance compared to average pooling [8]. In tasks such as semantic segmentation [30], where the precise spatial locations of features are important, average pooling is often more suitable [33]. Each of the two pooling methods has advantages and disadvantages due to the inherent quantization errors in the pooling process [29]. These widely used pooling methods may prevent discriminative details from being retained, which is crucial for image recognition and

classification tasks [13]. Therefore, it has been argued that the optimal pooling method for a given classification problem may be neither max nor average pooling, but something in between [7].
New pooling methods have been proposed to overcome the limitations of max and average pooling. Mixed pooling, a method that randomly chooses between max pooling or average pooling, was proposed to promote the generalization ability of pooling [49]. This method successfully improved the overall performance of the pooling results but failed to reflect the advantages of both traditional pooling methods at the same time. In a later study, mixed max-avg pooling was proposed, which replaces the previous random choice between max or average pooling with a mixing proportion that is learned during training [27]. The features of both max and average pooling could be reflected in the pooling process by this mixing proportion mechanism. Another recent study proposed the AVG-TopK pooling method, a method that selects the K highest activation values and computes the average [33]. The experimental results of this study demonstrated that the AVG-TopK pooling method achieved improved performance compared to traditional pooling methods. However, the AVG-TopK pooling method was only evaluated on the LeNet architectures and transfer learning models, such as DenseNet. The LeNet model was trained from scratch but is a relatively simple model and the pre-trained models provide limited transparency in the training process.
Despite the advancements in the novel pooling methods, several potential gaps remain in the field. The new pooling methods are often evaluated using either basic CNN architectures or transfer learning models. Therefore, there is little insight into the generalizability and effectiveness of the new pooling methods in improving the performance of popular CNN architectures, such as VGG16 and ResNet34. The design of the CNN architecture could influence the effectiveness of the pooling method [7], making it important to evaluate novel pooling methods across different CNN architectures that are trained from scratch to understand their impact. Therefore, this study investigates the impact of the recently proposed AVG-TopK pooling method on the performance of CNNs by integrating it into three benchmark CNNs: AlexNet [24], VGG16 [42], and ResNet34 [19]. Moreover, this study aims to address the research gaps in the earlier study [33] by providing potential valuable insights on the effectiveness of the AVG-TopK pooling method in different CNN architectures than those earlier investigated.
Additionally, this study aims to contribute a new pooling method, called AVG-mixed, for minimizing information loss in the pooling layer. This pooling method is proposed to address the limitations of max and average pooling, by providing a more balanced approach that combines the strengths of both pooling methods The proposed AVG-Mixed pooling method could be used as a replacement for standard pooling methods in different CNN architectures. Additionally, it could improve existing pooling techniques in the field by providing valuable insights into the effectiveness of the AVG-Mixed pooling method in different CNN architectures.

1.2 Research objectives

The main objective of this study is to investigate the effectiveness of the AVG-Mixed and AVG-TopK pooling methods in CNNs for image classification tasks. Additionally, this study explores whether these custom pooling methods can improve the performance of CNNS, compared to the standard max pooling method. Both custom pooling methods aim to address the limitations of traditional pooling methods and are like traditional pooling methods, simple and user-friendly.

The AVG-Mixed pooling method has not been proposed in earlier studies. This pooling method computes the average of the average value and the maximum value within each pooling window of feature maps. The aim is that this method integrates the benefits of the two traditional pooling methods: max pooling and average pooling. This could lead to improved performance in terms of accuracy by maintaining the balance between generalization and specificity.
The AVG-TopK pooling method was proposed in another study where it was trained on a different CNN architecture [33]. The main aim of this pooling method is to prevent the loss of high representative values, which are discarded by the traditional pooling methods. This is achieved by selecting the K highest values and computing the average of those values, within each pooling window.
The new pooling methods will be evaluated by integrating them into popular CNN architectures: AlexNet [24], VGG16 [42], and ResNet34 [19]. The performance of the CNNs, which will be trained from scratch, will be assessed on benchmark datasets, including MNIST [25], CIFAR-10, and CIFAR-100. The performance of the baseline CNN models, with standard max pooling, on image classification tasks is compared with the performance of the models with the custom pooling methods by utilizing established evaluation metrics: accuracy and confusion matrix. The comparative assessment aims to provide valuable insights into how the custom pooling methods impact the performance of CNNs in image classification tasks.

1.3 Thesis overview

The remainder of this thesis is organized as follows. In Section 2, an overview of the literature on CNN architectures, and more specifically on AlexNet, VGG, and ResNet is provided. Traditional pooling methods, their functionalities, and limitations are also discussed in this section. In Section 3, an overview of related work on custom pooling methods for CNNs is provided. Section 4 describes the methodology used in the experiments. It starts with a comprehensive description of the software and hardware that are used in this study. Furthermore, the two custom pooling methods, Avg-TopK and Avg-Mixed, that are implemented in this study, are described. This is followed by a detailed description of the architectures of the baseline CNN models and the datasets that were used. This chapter also outlines the data preprocessing steps and the evaluation metrics that were used in the experiment. Section 5 describes the experimental setup and presents the results from the experiments in tables and figures. This section includes data on the accuracy of AlexNet, VGG16, and ResNet34 using both standard and custom pooling methods on the three datasets. Analysis of the performance data is also provided through the confusion matrices and plots. Section 6 discusses the experimental results and provides insights into their implications for the performance of the CNNs. Also, it addresses the limitations of this study and offers suggestions for future research. Finally, in Section 7, the thesis is concluded by summarizing the findings.

This thesis is part of the bachelor program Data Science and Artificial Intelligence at the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University and was written under the supervision of Prof. Dr. Michael Lew and Dr. Erwin Bakker.

2 Theory

This study evaluates the effectiveness of two custom pooling methods in different CNN architectures for image classification tasks. This section aims to provide an overview of the theoretical concepts necessary to comprehend the methodology that follows.
First, in Section 2.1, the building blocks of CNNs will be described, followed by Section 2.2 which provides a detailed explanation of the two traditional pooling methods. Section 2.3 will provide theory on the three popular CNN architectures, that are used in this study: AlexNet, VGG16, and ResNet34. More specifically, it includes the key features that make the CNNs successful in computer vision tasks.

2.1 Convolutional Neural Networks

A Convolutional Neural Network(CNN) is a type of Deep Learning network that performs exceptionally well on vision tasks, such as image classification [47] and object recognition [32], in comparison to other Artificial Neural Networks(ANNs) [2, 28]. The difference between CNNs and other ANNs is that the architecture of CNNs is inspired by the connectivity pattern of the visual cortex of humans, which plays an important role in processing visual stimuli [2] ĊNNs are able to automatically and adaptively learn what features in the filters are most important for a given task, while engineers had to manually design filters based on heuristics to handle image features in traditional image processing algorithms [26]. Furthermore, the weight-sharing property of CNNs has been a revolutionary concept in the field of deep learning and differentiates CNNs from other types of neural nets [2, 28]. Weight sharing reduces the number of parameters in the network by sharing the same set of weights, also known as filters or kernels, across different parts of the input, making the network less complex and easier to train. Also, weight sharing contributes to translation invariance, allowing CNNs to recognize patterns in images regardless of their position [26].
CNNs typically consist of three main layers: convolutional layers, pooling layers, and fully-connected(FC) layers. First, the convolutional layers convolve the image to detect features that could be used to classify the image. Then, the pooling layer will reduce the number of parameters in the input data. Finally, the fully connected layer performs the classification task.

2.1.1 Convolutional layers

Convolutional layers are one of the building blocks of CNN architectures. The process starts by sliding a kernel, also called filter, on the input image to detect important features. Each convolutional kernel is, by default, initialized with a random weight matrix which is then adjusted during the training process through backpropagation [2]. This allows the network to automatically learn the best set of kernels for a given task. The convolutional layer computes the dot product between the kernel and the receptive region of the input to generate a feature map that shows the presence of the detected features in the image. The same kernel is applied to all regions in the input image, the shared weights allow the model to detect the same features regardless of their position. In a single convolutional layer, a series of convolutional kernels can be applied to produce a set of feature maps. Each kernel corresponds to a specific feature the layer will seek to identify. The feature maps will be stacked together and serve as input for the next layer in the architecture, enabling CNNs to gradually build a hierarchical representation of the image. Usually, the kernels in the first layers detect basic features, such as lines, and the subsequent layers are more complex, combining the simpler features that were identified earlier to recognize more complex patterns, such as shapes [1].

2.1.2 Pooling layers

The convolutional layer is typically followed by a pooling layer. Pooling layers reduce the dimensionality of the feature maps generated by the convolutional layers, which makes the network more computationally efficient and less prone to overfitting [7]. Dimensionality reduction is conducted by sliding a fixed-shape window, known as a pooling window, across the input feature maps. Within each pooling window, a pooling function is applied and the output value generated within each window replaces the original values within that window. Figure 1 illustrates a feature map and a pooling window, demonstrating how the pooling process operates. Similar to a convolutional operation, this process is repeated across the entire feature map until all regions have been processed. Unlike convolutional operation, the pooling window does not have any trainable weights associated with it [10].
The pooling layer serves two primary purposes. Firstly, it reduces the dimensionality of feature maps, thereby reducing computational complexity and memory usage, and speeding up the training time. Secondly. it extracts the relevant features while minimizing redundant information, thereby limiting the risk of overfitting and improving the model's generalization ability. The pooling layer, along with the convolutional layer, contributes to translation invariance, making the model more robust to small variations in the position of the features [7].
The most common type of pooling is max pooling, which surveys all of the pixels contained within its receptive field as the pooling window sweeps across the input feature maps and selects the pixel with the maximum value. Another common pooling method, known as average pooling, takes a similar approach but calculates the average value within each window instead of the maximum value [10].

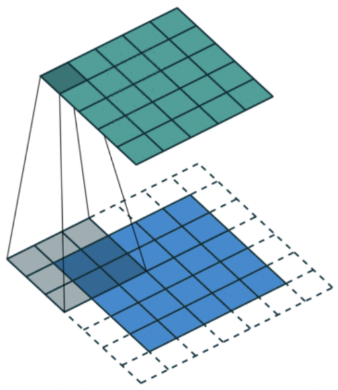


Figure 1: Illustration of a pooling process: the feature map(bottom) and the downsampled feature map(top)

2.1.3 Fully-connected layers

A Fully Connected(FC) layer is a type of layer that is typically found towards the end of a CNN architecture. In image classification tasks, the FC layer plays a critical role in classifying images based on the features extracted by the preceding layers.
The output tensors from the preceding layers are flattened before they can be fed to the FC layer. In this process, known as flattening, the multi-dimensional tensors from the pooled feature maps are converted to a 1D linear vector. Furthermore, all the neurons in the FC layer are connected to neurons of the adjacent layers. In each neuron, the weighted sum of the input is calculated and an activation function is applied to the output of the neuron. This allows the model to learn the relationships in the data and to make predictions based on a combination of different features.
In the CNNs that are used for image classification tasks, the number of neurons in the final FC layer matches the number of classes. The activation function in the final FC layer normalizes the output of the network to a probability distribution over predicted output classes.

2.2 Popular pooling methods

2.2.1 Max pooling

Max pooling [35] is a technique that selects the highest value within each pooling window, producing a downsampled feature map that retains the most prominent features from the previous layer. The max pooling method is especially effective in images with dark backgrounds and lighter regions of interest, as the max pooling method retains the brighter pixels, which represent the dominant features of the image [34]. For example: it is effective for the MNIST dataset because the digits are represented in white color and the background is black [4].
The benefit of this pooling method is that it effectively captures the higher activation values which is useful in tasks such as object recognition [32] where the higher activations are essential. Also, it is robust to noise as small variations in other values will be ignored [10].
However, the max pooling method also has drawbacks: it ignores the information of other values which may lead to the loss of valuable, often small-sized, features [52]. Also, high noise in a single activation value could result in the selection of an outlier. This could affect the result significantly, causing degradation of the performance of the model [3].

The max pooling function is defined as:

$$O_{\max}(X) = \left(\max_{i=1}^{N} X_i\right) \tag{1}$$

where N represents the total number of elements in the pooling window X, and X_i denotes the i-th element within the window.

2.2.2 Average pooling

Average pooling [26] computes the average value of the pixels within each pooling window and uses it to create a downsampled feature map. This pooling method is especially successful in images

with minimal color variation [33] or when the exact location of features is not crucial.
The benefit of this pooling method is that it captures the central tendency of the activation values by retaining information from all values in the window. This method is more robust to outliers relative to max pooling because it averages all values, preventing a single extreme value from significantly altering the results. However, there are also drawbacks, such as its inability to capture the dominant features from the feature maps. All the values in the pooling window are treated equally and we could lose the distinctions between certain features in images, which can be problematic for some tasks such as object detection and image classification. Moreover, if the number of high-energy pixels is low or the number of low-energy inputs is high, then the average pooling method is disadvantageous [33]. Hence, average pooling is more suitable when the overall texture or pattern is more important than specific details in image classification tasks.

The average pooling function is defined as:

$$O_{\text{avg}}(X) = \left(\frac{1}{N} \sum_{i=1}^{N} X_i \right) \tag{2}$$

where N represents the total number of elements in the pooling window$X$ and X_i denotes the i-th element within the window.

2.3 Benchmark CNN models

2.3.1 AlexNet

The AlexNet architecture was introduced in 2012 in the ImageNet Large Scale Visual Recognition(ILSVR) challenge. This CNN is one of the first deep convolutional networks to achieve significant accuracy on the challenge with an accuracy of 74.7%. The main reason this model achieved significant high performance in the competition is its depth which was expensive computationally but was made feasible due to GPUs during training [24]. AlexNet was one of the first CNNs that used GPU to increase performance. Furthermore, it introduced several new concepts that have inspired the research of future models [36]. One of the key features of this model is data augmentation, which includes mirroring and cropping the images, to increase the variation in the training dataset and thereby reduce overfitting. The CNN also distinguishes itself from other popular CNNs by using overlapped max pooling layers which are max pooling layers with strides less than the window size [28]. Furthermore, AlexNet uses the ReLU activation function, which allows the model to train much faster than the commonly used saturating activation functions, like tanh or sigmoid. The original paper showed that AlexNet achieved a 25% training error rate six times faster than an equivalent network with tanh [24]. Due to the unbounded nature of ReLU, AlexNet introduces Local Response Normalization(LRN) to prevent the learned variables from becoming unnecessarily high. Furthermore, AlexNet also addresses the overfitting problem by including drop-out layers in the architecture, where a connection is dropped during training with a probability of 0.5. Some of the introduced concepts in this model are still the standard for current CNNs.

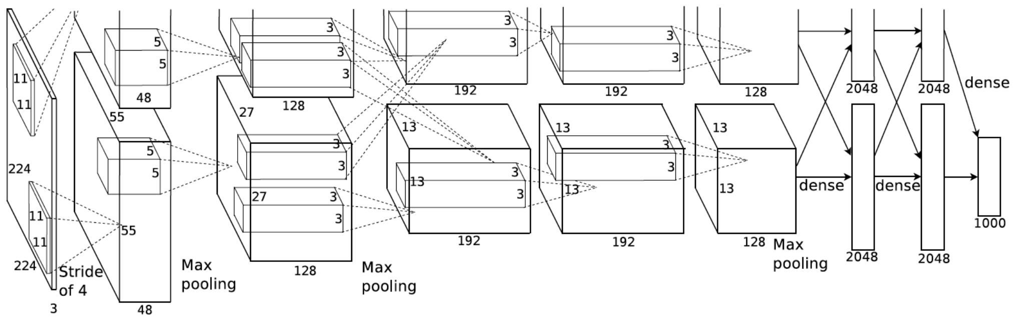


Figure 2: AlexNet architecture [24])

2.3.2 VGG

The Visual geometric group(VGG) was introduced in the ILSVR-2014 where it was one of the best-performing architectures [42]. This CNN model is characterized by its simplicity and uniform architecture, which makes it easy to understand and implement. VGG comes in different depths, including VGG16 and VGG19.
The VGG16 model significantly outperformed AlexNet, achieving 92.7% test accuracy on the ImageNet dataset. This is achieved by introducing the concept of grouping multiple convolutional layers with smaller, 3x3 kernel-sized, filters instead of one convolutional layer with a larger kernel size. The first captures the same receptive field as the latter kernels while using fewer parameters. This reduced number of parameters leads to faster learning and more robustness to overfitting. However, there are also limitations to the VGG models. The two main limitations are the training time and memory consumption. The original VGG model was trained on the Nvidia Titan GPU for 2-3 weeks and the model requires a lot of memory as the trained weights are large. Additionally, as the VGG models became deeper, the model was not able to converge to the minimum error rate. This problem is known as the vanishing gradient problem and is later resolved in an architecture known as ResNet [19] by introducing the concept of residual learning with skip connections.

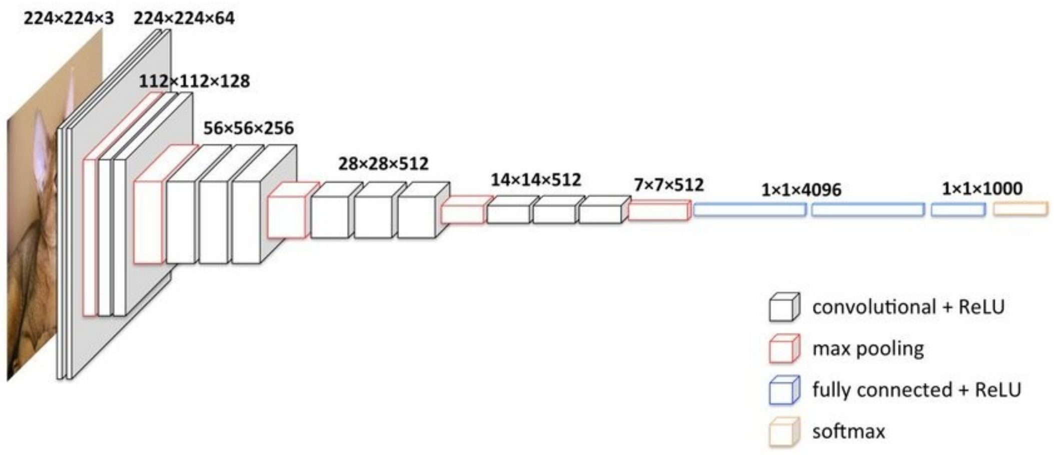


Figure 3: VGG16 architecture [42]

2.3.3 ResNet

Residual Networks, or ResNets, were introduced in the ILSVR-2015 [19]. One of the well-known ResNet architectures, ResNet50, achieved state-of-the-art performance in this competition. The main reason for the design of the model was to address the issue of the vanishing gradient. The vanishing gradient arises when increasing the number of layers in the CNN. The idea of increasing the number of layers is that the model will be able to more complex features from images. After AlexNet won the ILSVR-2012, the consecutive CNN architectures increased the number of layers to improve the model but experienced difficulties in learning due to the vanishing gradient [6].
ResNet provides an innovative solution, known as "skip connections", to the vanishing gradient problem. The skip connection allows the gradient signal to flow more easily through the network by bypassing one or more layers. The connections allow the network to learn the residual function instead of learning the complex mapping between the inputs and outputs. The connections also allow the model to make small updates to the parameters which enables the model to achieve better performance and converge faster [19]. Therefore, the skip connections allow an increased depth in CNNs without degrading the performance.
Despite the numerous benefits, ResNet also has a few drawbacks, such as its complexity and vulnerability to overfitting. Resnet is a complex architecture that requires more computational resources and memory than shallower networks. The interpretability of the network can also be challenging, as it learns complex and abstract representations. Also, ResNet is prone to overfitting, especially in cases where the number of layers in the ResNet network is high and the dataset size is limited [5].

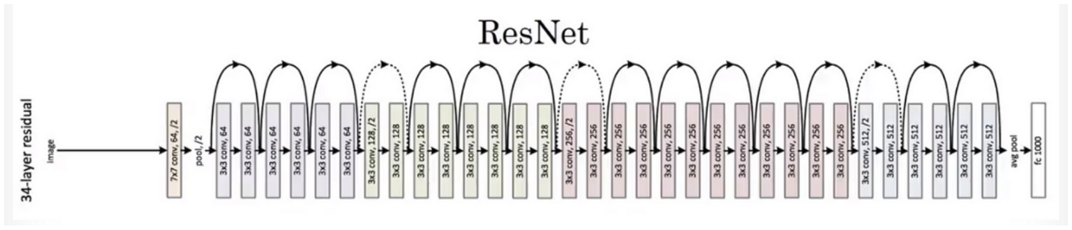


Figure 4: ResNet34 architecture [31]

3 Related Work

Different new pooling methods [34, 27, 40, 41, 20, 13, 39, 37] have been proposed to overcome the limitations of traditional pooling methods. A few of the proposed pooling methods will be described below.
Mixed max-average pooling was proposed with the idea that there are regimes in datasets in which either max pooling or average pooling performs better than the other [27]. This pooling method computes the weighted sum of the maximum and average values by learning specific mixing proportion parameters from the data. This pooling method is evaluated on the five standard benchmark datasets: MNIST, CIFAR-10, CIFAR-100, SVHN, and ImageNet. The AlexNet architecture is one of the CNN architectures used in the experiments. The results show that by replacing the max pooling layers in AlexNet on the ImageNet dataset with the proposed mixed

max-average pooling method, the test error(top 5, single-view) reduces by 6%.
The mixed max-average pooling function [27] is defined as:

$$O_{\text{mix}}(X) = \alpha \left(\max_{i=1}^{N} X_i \right) + (1 - \alpha) \left(\frac{1}{N} \sum_{i=1}^{N} X_i \right) \tag{3}$$

where N represents the total number of elements in the pooling window X, X_i denotes the i-th element within the window, and α is a weighting factor with $0 \leq \alpha \leq 1$.

L_p pooling is a successful pooling method, integrated into a CNN that is specifically trained on the Street View House Numbers dataset to recognize house numbers [40]. This pooling method is another alternative to the average and max pooling methods that alternates between one or the other depending on the value of p(i.e. a predefined parameter). The authors argue that the proposed pooling method achieved state-of-the-art performance in CNNs compared to max and average pooling. This pooling method is a sort of weighted function that assigns higher weights for more important features and lower for lesser ones. The value for p ranges from 1 to infinity.

The weighted L_p pooling function [40] is defined as:

$$O_{L_p}(X) = \left(\sum_{i=1}^{N} (X_i^p \cdot G_i) \right)^{\frac{1}{p}} \tag{4}$$

where N represents the total number of elements in the pooling window X, X_i denotes the i-th activation value within the window, G_i denotes the i-th Gaussian kernel value within the window, and p is a parameter that determines the type of pooling. For $p = 1$, L_p pooling corresponds to simple Gaussian average pooling, and $p \to \infty$, corresponds to max pooling.

Rank-based pooling is proposed to prevent the problem of relevant information loss encountered by max and average pooling [41]. In rank-based pooling, the values within each pooling window are sorted after assigning ranks to each activation value. Then, the desired pooling operation(e.g. max pooling) is applied to the sorted ranks. This pooling method can be regarded as an instance of weight pooling where the weighted sum of activation values is used to generate the output. The benefits of utilizing this method are that important activations can be distinguished by their ranks, the ranking list is invariant under changes in activation values, and the usage of rank can avoid the scale problem experienced by value-based methods [41]. Three forms of rank-based pooling are proposed in the paper: rank-based average pooling, rank-based weighted pooling, and rank-based stochastic pooling. The three methods are evaluated on four benchmark datasets: MNIST, CIFAR-10, CIFAR-100, and NORB. The architecture of CNN used in this study is simple, consisting of three convolutional layers, each followed by a pooling layer [41].
Rank-based average pooling alleviates some of the problems of the traditional pooling methods by using an average of the top t highest activation values. The weights of the top t highest activation are set to 1/t, and the other values to 0 [41]. The rank-based average pooling method is identical to the AVG-TopK pooling method that is being investigated in this study. This equivalence and its implications will be further discussed in Section 4.3.

The rank-based average pooling function [41] is defined as:

$$O_{\text{rank}}(X) = \frac{1}{N} \sum_{i=1}^{N} a_i \tag{5}$$

where X represents the pooling window, a_i denotes the value of the i-th activation within the window, and N is the number of activations in the pooling window X that meet the rank threshold t. The rank threshold t determines which activations are included in the computation.

4 Methodology

4.1 Software and hardware

The experiments were established on the ALICE server platform of Leiden University, and the models were trained on NVIDIA TESLA T4 GPUs. The most important packages for the implementation of the CNNs in this study are TensorFlow and Keras.

4.1.1 CUDA and cuDNN

CUDA, version 12.3.2, and cuDNN, version 8.9.7.29, were used in this study to allow for efficient usage of GPUs. These specific versions of CUDA and cuDNN are the default versions that are supported by the ALICE server, hence the choice to use those versions. CUDA is a framework, with cuDNN built on top of it, that provides a general-purpose programming interface for NVIDIA GPUs. This framework allows applications to use the GPU for different kinds of computing. CuDNN provides highly optimized GPU implementations for neural networks. It is specifically designed for deep learning functionalities and offers users GPU acceleration for common operations, like convolution and pooling. Moreover, CuDNN and TensorFlow are closely connected, which allows users to efficiently run TensorFlow models on the NVIDIA GPUs.

4.1.2 TensorFlow

TensorFlow, version 2.16.1, was used in this study for the implementation of the neural network models as this version is compatible with the above CUDA version. TensorFlow is an open-source framework that provides comprehensive tools for machine learning and deep learning applications. The framework has a high-level API built on top of it, known as Keras, that simplifies the construction of machine learning models, making it a suitable choice for research. Furthermore, TensorFlow offers GPU support, allowing to use NVIDIA CUDA: a parallel computing platform. The usage of GPUs accelerates the training process of the neural networks.

4.1.3 Keras

Tf-keras, version 2.16.0, was used in this study as this is compatible with the above TensorFlow version. Keras is a high-level neural networks API, integrated within TensorFlow, that is used to facilitate the design and training of deep learning models. Keras provides a simple and user-friendly interface, which allows for building and training deep learning models. Its integration with TensorFlow allows it to use the functionalities that TensorFlow provides, i.e. the heavy lifting of the computations, while maintaining the simplicity in the model development.

4.2 AVG-Mixed

In this study, a new pooling method called AVG-Mixed is proposed. This method involves computing both the average and maximum value within each pooling window of the feature map, and then taking the average of these two values. The AVG-Mixed pooling method aims to balance the two traditional pooling methods by averaging the results of both methods. Moreover, it aims to combine the smoothing effect of average pooling and the feature selection capability of max pooling, resulting in a more balanced feature representation. This pooling method could improve the generalization ability of the popular CNNs compared to their original pooling method: max pooling.

The AVG-Mixed pooling method was implemented by using TensorFlow and Keras. Custom layers were created to perform the AVG-Mixed pooling method and were integrated into the AlexNet, VGG16, and ResNet34 networks, replacing the max pooling layers. In the implemented AVG-Mixed pooling method, the AveragePooling2D() and MaxPooling2D() from TensorFlow's Keras API are used to compute the average and max pooling value for each patch of the input tensor. In each patch, the computed values are averaged, producing an output that captures both average and maximum values, as demonstrated in Figure 5.

The AVG-Mixed pooling function is defined as:

$$O_{\text{AVG-Mixed}} = \frac{1}{2}(\max(X) + \text{avg}(X)) \tag{6}$$

where X represents the pooling window of the input feature map.

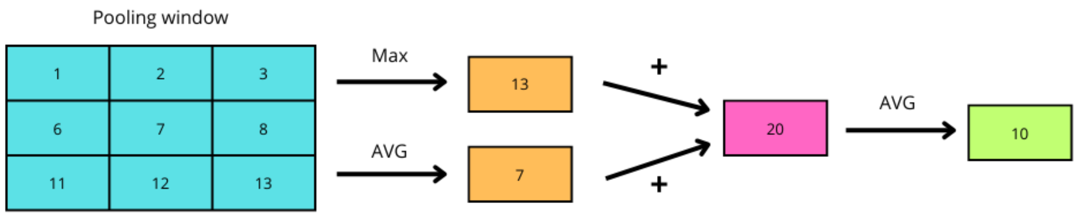


Figure 5: Sample AVG-Mixed pooling

4.3 AVG-TopK

The AVG-TopK method was proposed by Cuneyt Ozdemir in 2023 [33]. The proposed pooling method, AVG-TopK, aims to prevent the loss of high representative values by addressing the drawbacks of the max and average pooling methods. In max pooling, only the highest value is selected and all the other values are discarded. In average pooling, the highly representative values and least representative values are treated equally, often diluting the significance of prominent features. The proposed AVG-TopK method solves these problems by selecting more than one representative high value, and averaging these values. Moreover, in the AVG-TopK pooling method, the K highest activation values are selected from the pooling window of the incoming feature map, and the average of these values is computed.
In the seminal study [33], the AVG-TopK pooling method was integrated into the LeNet architecture and different transfer learning models. The effectiveness of the pooling method was evaluated by conducting a comparative analysis of the AVG-TopK pooling method with different values for K and different pool sizes. Additionally, the performance of the LeNet models with the AVG-TopK pooling method was compared to the LeNet models with both max and average pooling. All the models are trained on MNIST, CIFAR-10, and CIFAR-100. The experimental results show that this new method is superior to the traditional pooling methods [33].
In this study, the effectiveness of the AVG-TopK pooling method will be further investigated by integrating the pooling method into different CNN architectures that will be trained from scratch: AlexNet, VGG16, and ResNet34. This could provide valuable insights into the effectiveness and generalizability of the AVG-TopK pooling method, as the design of the CNN can greatly influence the effectiveness of the pooling method. However, after the experiments in this study were conducted, further investigation revealed that the AVG-TopK pooling method [33] is identical to the rank-based average pooling method [41], years earlier proposed. Despite this overlap, the study proceeds with the evaluation of the AVG-TopK pooling method due to its potential for providing valuable insights into its effectiveness, as the rank-based average was also only evaluated on a simple CNN architecture [41].

In this study, the Avg-TopK pooling method was implemented by using TensorFlow (TF) and Keras. Custom layers were created to perform the selection and averaging of top K values. These custom layers were integrated into the AlexNet, VGG16, and ResNet34 networks, replacing the max pooling layers. The implementation details of the AVG-TopK pooling method in TF are as follows: patches(corresponding to the pooling windows) from the input tensor are extracted, using tf.image.extract_patches, and reshaped accordingly. The K values are extracted from each patch, by using tf.nn.top_k with sorted=False to avoid sorting operations as this would have increased computational load unnecessarily. The K values are then averaged to produce the final output for each patch, as demonstrated in Figure 6. Additionally, in this study, the value of K in the AVG-TopK pooling method is set to 3, based on findings from the study by Ozdemir [33] where the AVG-TopK method with a pooling size set to 3 and K set to 3 demonstrated improved performance on the LeNet architecture. Additionally, a higher value for K would not be possible for VGG16, which uses 2x2 pooling windows. For example, setting K to 4 in VGG16 would effectively convert the AVG-TopK into AVG pooling.

The AVG-TopK pooling function is defined as:

$$O_{\text{AVG-TopK}}(X, K) = \frac{1}{K} \sum_{i=1}^{K} X_i \tag{7}$$

where X_i are the top K values selected from the pooling window X of the input feature map.

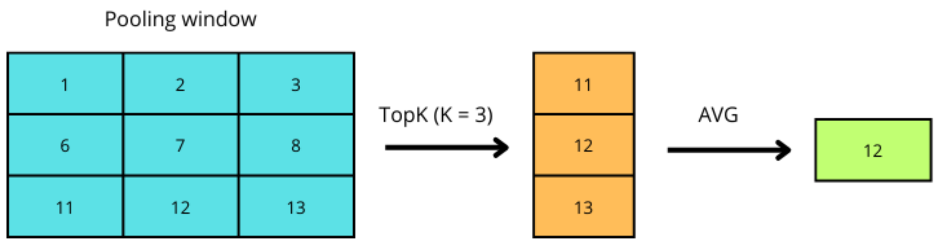


Figure 6: Sample AVG-TopK pooling

4.4 Models

The implementation of the baseline models, AlexNet, VGG16, and ResNet34, is described below. The original architectures of AlexNet, VGG16, and Resnet34 are visualized in Figures 2, 3, and 4, respectively. These models were selected in this study due to their simple structure and powerful classification capabilities. The three CNN architectures each significantly contributed to the development of CNNs. AlexNet was the first CNN that achieved significant performance on ImageNet at the time. The study on VGG is the first study that provided undeniable evidence that adding more layers increases performance, which was true up to a certain point. Lastly, ResNet introduced the concept of residual learning with shortcut connections, which solved the vanishing gradient problem that deep CNNs, like VGG, experienced.

In this study, the CNNs will be trained on MNIST, CIFAR-10, and CIFAR-100, while originally designed to classify the ImageNet dataset. The architectures and training process of the models are replicated as described in their original papers, with the only exception being the weight and bias initialization in each of the three CNNs and the learning rate in ResNet. Additionally, the data augmentation steps are not entirely identical to the original augmentation on all three CNNs. This will be addressed in the below sections.

4.4.1 AlexNet architecture

The exact AlexNet network architecture as implemented in this study is visualized in Figure 22, highlighting the pooling layers that will be replaced by custom pooling layers in the experiments. The AlexNet architecture [23] consists of 5 convolutional layers, 3 max-pooling layers, 2 local response normalization layers, 2 fully connected layers, and 1 SoftMax layer. Out of these layers, Only the five convolutional layers and the 3 fully connected layers are weighted. The input to the CNN is fixed-size 227x227 RGB images. The convolutional and fully connected layers are regularized with L2 regularization, also known as weight decay, set to 0.0005. The non-linear activation function ReLU is applied to the output of each convolutional and fully connected layer. Furthermore, after the first and second convolutional layers, there is a local response normalization layer. Max pooling layers follow the response-normalization layers and the fifth convolutional layer. These pooling layers use overlapping pooling windows, sized 3x3 with a stride of 2 between the adjacent pooling windows. The overlapping regions result in less loss of surrounding spatial information. After the last convolutional layer, there are two fully connected layers with 4096 outputs. Also, a dropout rate of 0,5 is applied before the first and second fully connected layers. The final fully connected layer is the output layer which uses a softmax activation function and produces a distribution that depends on the number of classes in the dataset. The network cannot be described in detail due to space constraints but is specified in Figure 22.

The original AlexNet model [23] had weights initialized from a zero-mean Gaussian distribution with a standard deviation of 0.01 and set biases to 1 in specific layers [24]. However, this specific initialization hindered learning in the implemented AlexNet model in this study. Therefore, the default weight initialization method of Keras, Glorot uniform initialization [15], is used.

4.4.2 VGG16 architecture

VGG comes in several depths, but this study focuses on the model that supports 16 layers, known as VGG16 [42]. The exact network architecture as implemented in this study is visualized in Figure 23, highlighting the pooling layers that will be replaced by custom pooling layers in the experiments. VGG16 is a 16-layer Artificial Neural Network that consists of thirteen convolutional layers and three fully connected layers. The input to the CNN is fixed-size 224x224 RGB images. The configuration of VGG16 has block structures where each block consists of a sequence of convolutional layers that are followed by a max-pooling layer. The first two blocks consist of two convolutional layers, each followed by a max pooling layer. The last three blocks consist of three convolutional layers, each also followed by a max pooling layer. The convolutional and fully connected layers are regularized with L2 regularization set to 0.0005. Furthermore, all the hidden layers use ReLU which is an innovation from AlexNet that reduces training time. The biases of all the convolutional layers and fully connected layers were initialized with zero. Unlike AlexNet, VGG16 does not use Local Response Normalization, because this increases training time without particularly increasing accuracy on the ILSVRC dataset [42].
The number of filters in the convolutional layers of the first block is 64 and doubles in the later blocks until it reaches 512. All the convolutional layers in all blocks use the same padding and apply the same kernel size(3x3). All the pooling layers in VGG16 have size 2x2 with strides of 2. As mentioned, the VGG architecture consists of three fully connected layers: two fully connected hidden layers and one fully connected output layer. The first two fully connected layers have 4096 channels each. The output layer consists of several channels that correspond to the number of categories of the dataset. The final layer in this architecture, that performs the classification, is the softmax layer.

In the original VGG16 architecture, the weights of the first four convolutional layers and the last three FC layers were pre-trained and initialized with specific values, while the intermediate layers were initialized randomly from a normal distribution with zero mean and 0.01 variance. However, the authors described in the original paper [42] that it is possible to initialize the weights without pre-training by using the Glorot uniform initialization. This implies that a similar effect is achieved with both initialization techniques. Therefore, for consistency, simplicity, and to address the learning issues encountered with AlexNet, Glorot uniform initialization [15] was also used in the experiments with the VGG16 models.

4.4.3 ResNet34 architecture

ResNet comes in several depths, but this study focuses on the model that supports 34 layers, known as ResNet34 [19]. The reason this ResNet architecture is chosen in this study, rather than a deeper architecture like ResNet50, is due to simplicity reasons. The exact network architecture as implemented in this study is, partially, visualized in Figure 24, highlighting the pooling layer that will be replaced by custom pooling layers in the experiments.
ResNet34 is a 34-layer CNN that consists of thirty-three convolutional layers and one fully connected layer. The input to the CNN is fixed-size 224x224 RGB images. The convolutional and fully connected layers in this network are regularized with weight decay set to 0.0001. Furthermore, the biases of all the convolutional layers and fully connected layers were initialized with zero. This CNN has one initial convolutional layer that consists of 64 filters with a kernel size of 7x7, followed by batch normalization and ReLU activation. This layer is followed by a max pooling layer which downsamples the spatial dimensions by 3x3 with a stride of 2. The rest of the architecture consists of four sets of residual blocks with similar structures that consist of 2 convolutional layers with filters of size 3x3, where each convolutional layer is followed by Batch Normalization and ReLU. In each block, the shortcut connections are included that add the input of the residual block to the output. The first set consists of 3 residual blocks, where each layer contains 64 filters. In the other three sets, the structure is similar but with an increasing number of filters in each set, and applies downsampling by using a stride of 2 in the first convolutional layer of each set. In the other blocks, a stride of size 1 is used. The second set consists of 4 residual blocks, where each convolutional layer contains 128 filters. The third set consists of 6 residual blocks, where each convolutional layer contains 256 filters. The fourth set consists of 3 residual blocks, where each convolutional layer contains 512 filters. Following these blocks, the feature maps are passed through the global average layer and then through one fully connected layer to classify the images.

In the original ResNet34 architecture, a zero-mean Gaussian distribution with a standard deviation of $\sqrt{\frac{2}{n_l}}$ is used to initialize the weights [19] [18]. This weight initialization method, called He initialization, is particularly well-suited for the ReLU activation and is proposed to allow extremely deep models to converge, while the Glorot method [15] cannot. However, in this study, for consistency and simplicity, the Glorot initialization with uniform distribution is used in the ResNet34 models as well. The Glorot initialization is not the most optimal choice for the ReLU activation function but worked well enough in the experiments. Moreover, the inventor of Glorot initialization argues that there is no vanishing gradient problem in the ReLU [16]. Therefore, the choice to use Glorot initialization for consistency in the experiments should not hinder the learning process of the models.

4.5 Datasets

The below datasets are classic datasets widely used for training and evaluating deep learning models on image classification tasks.

4.5.1 MNIST

The MNIST dataset contains a total of 70000 images which are divided into 60000 training images and 10000 testing images [25]. The images are divided into 10 different classes, each representing a digit from range 0 to 9. The images in this dataset are grayscale and have a size of 28x28. The samples are handwritten by different individuals, covering variations. Therefore, it's a valuable dataset for assessing the generalization ability of models in digit recognition tasks.
The MNIST dataset originally consists of three dimensions but is transformed by adding a fourth dimension to represent the number of channels. Since images in MNIST are grayscale, the number of channels is 1. This four-dimensional input shape is required by Keras, hence the transformation.

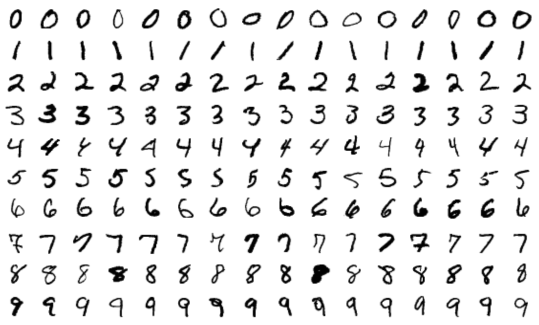


Figure 7: Samples from the MNIST dataset [33]

4.5.2 CIFAR-10

The CIFAR-10 dataset contains a total of 60000 images which are divided into 50000 training images and 10000 testing images [23]. The images are divided into 10 different classes, with each class having 6000 images. These classes include, among others, airplanes, cars, birds, cats, deer, and dogs. Each image in this dataset consists of RGB channels and has a size of 32x32. CIFAR-10 has become one of the benchmark datasets, extensively used, due to its relatively small size and diverse categories.

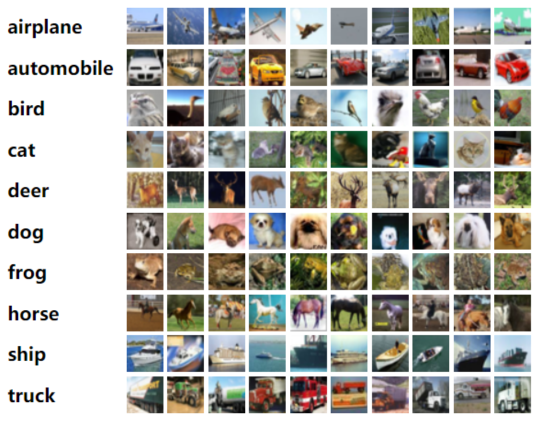


Figure 8: Samples from the CIFAR-10 dataset [33]

4.5.3 CIFAR-100

The CIFAR-100 dataset contains a total of 60000 images which are divided into 50000 training images and 10000 testing images [23]. The images are equally divided into 100 different fine-grained classes, with each class having 600 images. These classes include a wide range of objects, animals, and items. Each image in CIFAR-100 consists of RGB channels and has a size of 32x32 pixels. This dataset is an extended version of the CIFAR-10 dataset.

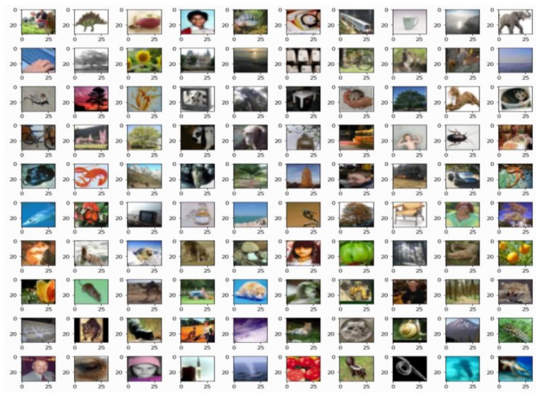


Figure 9: Samples from the CIFAR-100 dataset [33]

4.6 Data preprocessing

In the VGG and ResNet papers [42] [19], the authors described they used identical preprocessing methods on the training data as described in the paper on AlexNet [24]. The preprocessing steps regard the centering and augmentation of the data. Centering can help improve stability during learning. Centering is applied on the datasets which means the mean of the training data is subtracted from the data. After centering, from each dataset, 10% of the training data is allocated for the validation set in this study. Data augmentation is applied to the training data as this can improve the generalization ability of CNNs by reducing overfitting. The augmentation steps on the training data of the datasets in this study are extracting random 224x224 patches(and randomly their horizontal reflections) from the resized 256x256 images. Furthermore, the brightness of the images is randomly adjusted by a maximum delta of 0.1 and the contrast is randomly adjusted between a lower bound of 0.8 and an upper bound of 1.2. Random Gaussian noise is also added to the images with a mean of 0.0 and a standard deviation of 0.01, which is subtle noise. The testing and validation data are not augmented, merely resized to 224x224 images.

In this study, the input to the AlexNet model is changed to 227x227, even though the paper [24] mentions 224x224 as the math only makes sense if they are 227x227 [9]. Therefore, the augmentation steps on the AlexNet network differ from the VGG16 and ResNet as it extracts crops of size 227x227 from the 256x256 images.

The data augmentation of the training and testing sets in this study differ from the augmentation as described in the original papers of the AlexNet [24], VGG [42], and ResNet [19] networks. More specifically, the standard color augmentation techniques used in the original implementations were not applied to the training data on the three networks in this study due to implementation issues. Instead, the brightness and contrast of the training dataset are randomly adjusted to achieve a similar effect. Furthermore, the augmentation that is applied during testing in the original implementations is also not applied on the testing data on all three networks in this study, which is also due to implementation issues. The original AlexNet and ResNet implementation applied 10-crop testing [24] [42] and the original VGG applied a similar form of crop testing [19].

4.7 Evaluation metrics

The performance of the CNNs on image classification will be compared by using two evaluation metrics: accuracy and confusion matrix.

4.7.1 Accuracy

Accuracy measures how often the classifier correctly predicts the classes. It is the ratio of the number of correct predictions and the number of predictions. Accuracy is considered a useful evaluation metric if the model is trained on data with well-balanced classes.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{8}$$

4.7.2 Confusion matrix

Confusion matrix is an evaluation tool for evaluating the performance of machine learning models. The matrix presents combinations of the predicted and actual classes, displaying the number of true positives (correctly predicted positive class), true negatives (correctly predicted negative class), false positives (incorrectly predicted positive class), and false negatives (incorrectly predicted negative class) for each class. The matrix aids in analyzing the performance of the models, contributing to the understanding of the performance of the model in the classes where the model performs exceptionally well or needs improvement.

5 Experiments and results

5.1 Experimental Setup

In this study, the effectiveness of the AVG-Mixed and AVG-TopK(K=3) pooling methods is evaluated through experiments conducted using the same architecture, datasets, and training process as the baseline models, with the only difference being the pooling layer. The baseline models refer to the original architectures of AlexNet, VGG16, and ResNet34 as described in Section 4. The standard pooling layers in these models are as follows:

- In AlexNet [23] and VGG16 [42], the pooling layers of the baseline architecture consist solely of max pooling layers. Therefore, in the experiments, all the pooling layers in those architectures are replaced by the custom pooling layers, as visualized in Figure 22 and 24
- In ResNet34 [19], the architecture consists of a max pooling layer and a global average pooling layer. In the experiments, the global average pooling layer remains unchanged, while the max pooling layer is replaced by the custom pooling layers, as visualized in Figure 24.

All other experimental conditions, including data pre-processing, augmentation, and training hyperparameters, remain consistent with those used for the baseline models. This ensures that any observed differences in performance can be attributed to the pooling method used.

5.1.1 Training

In the experiments, each type of CNN model was trained using the specific hyperparameters and training process as described in their original papers [23] [42] [19], with a few exceptions as mentioned in Section 4. The settings were consistently applied across the AlexNet, VGG16, and ResNet34 models on all the experiments across the three datasets. Additionally, for consistency across comparisons, the number of epochs in all models is set to 50. However, all the models utilize learning rate schedulers, as described below, which might cause the models to terminate earlier.
The AlexNet models, in this study, are trained using Stochastic Gradient Descent(SGD) with a batch size of 128 and momentum of 0.9. The learning rate was initialized at 0.01 for all layers and then adjusted manually throughout training. The learning rate is decreased by a factor of 10 when the validation error rate stops improving with the current learning rate. The learning rate can be reduced three times before termination.
The VGG16 models are trained using mini-batch gradient descent with a batch size of 256 and momentum of 0.9. Identical to AlexNet, the learning rate was initialized at 0.01 and then decreased by a factor of 10 when the validation error rate stopped improving. The learning rate can be reduced three times before termination.
The ResNet models are trained using SGD with a mini-batch size of 256 and momentum of 0.9. The learning rate is initialized at 0.01 and then decreased by a factor of 10 when the validation error rate stops improving. The learning rate can be reduced three times before termination.

The learning rate of the original ResNet model is set to 0.1 [19], however, this learning rate hindered the learning process in the implemented ResNet34 model as it consistently hovered around 10% accuracy. Therefore, the learning rate is set to 0.01 in the ResNet34 models in this study.

5.2 MNIST

This section presents the experimental results evaluating the performance of AlexNet, VGG16, and ResNet34 using different pooling methods on the MNIST dataset. The standard architectures of these CNNs were modified by replacing the standard max pooling layers with the AVG-Mixed and AVG-TopK pooling layers. The performance of each architecture with these custom pooling methods is compared to the baseline architecture with standard max pooling.

Model	Accuracy(val)	Accuracy(test)
AlexNet		
Baseline	99.4%	**99.2%**
AVG-Mixed	98.9%	98.5%
AVG-TopK	98.6%	98.5%
VGG16		
Baseline	98.8%	**99.0%**
AVG-Mixed	98.9%	98.4%
AVG-TopK	98.8%	98.9%
ResNet34		
Baseline	99.0%	98.9%
AVG-Mixed	99.2%	**99.0%**
AVG-TopK	99.3%	**99.0%**

Table 1: Validation and test accuracies of the different models and pooling methods on MNIST

The results in Table 1 show that:

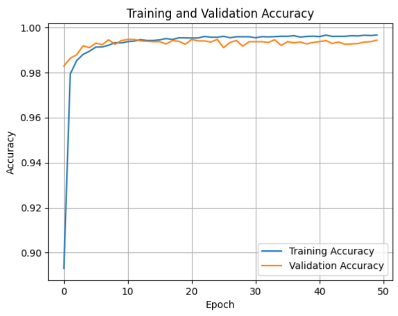
- **AlexNet:** The baseline model achieved the highest validation and test accuracy, of 99.4% and 99.2%. The AVG-Mixed and AVG-TopK pooling methods performed slightly worse than the baseline model, with AVG-Mixed achieving 98.8% validation accuracy and 98.5% test accuracy, and AVG-TopK achieving 98.6% validation accuracy and 98.5% test accuracy.

- **VGG16:** The baseline model achieved a validation accuracy of 98.8% and a test accuracy of 99.0%. The AVG-Mixed model achieved a slightly higher validation accuracy of 98.9%, but a lower test accuracy of 98.4%, compared to the baseline model. The AVG-TopK model achieved the same validation accuracy of 98.8% and a slightly higher lower test accuracy, decreased it by 0.01%, of 98.9%, compared to the baseline model.

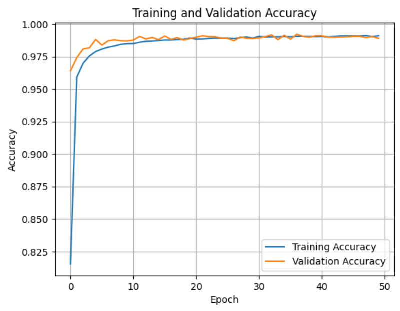
- **ResNet34:** The baseline model achieved a validation accuracy of 99.0% and a test accuracy of 98.9%. The AVG-Mixed model achieved slightly higher accuracies compared to the baseline model, a validation accuracy of 99.2%, and a test accuracy of 99.0%. The AVG-TopK model also achieved slightly higher accuracies compared to the baseline model, a validation accuracy of 99.3%, and a test accuracy of 99.0%.

The results in Table 1 and Figure 10, 11, and 12 show that the baseline, AVG-Mixed, and AVG-TopK models of AlexNet, VGG16, and ResNet34 all achieved near-perfect accuracies. It is relatively easy to capture the simple and distinct features of handwritten digits.

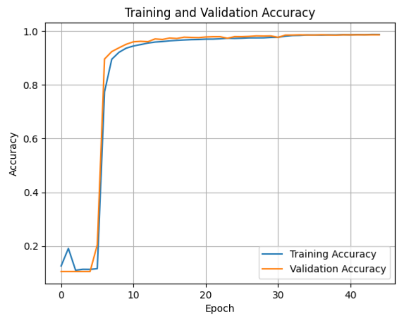
5.2.1 AlexNet



(a) AlexNet with Standard Pooling



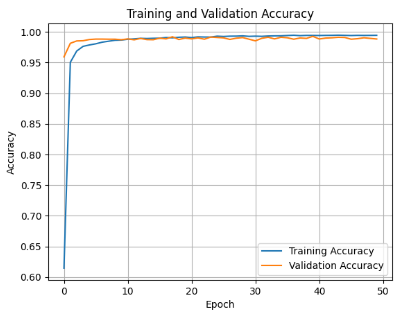
(b) AlexNet with AVG-Mixed Pooling



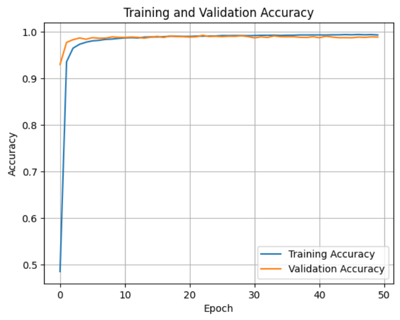
(c) AlexNet with AVG-TopK Pooling

Figure 10: Accuracy plots of AlexNet with different pooling methods on MNIST

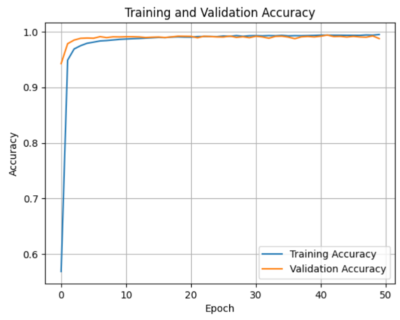
5.2.2 VGG16



(a) VGG16 with standard pooling



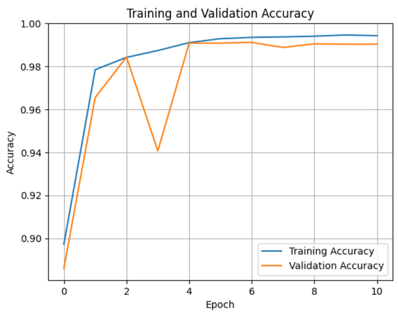
(b) VGG16 with AVG-Mixed pooling



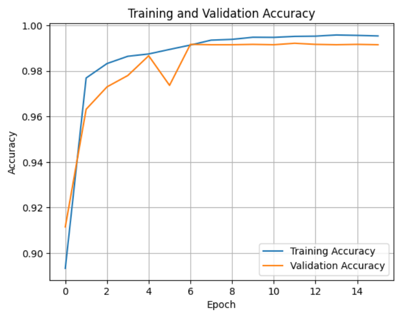
(c) VGG16 with AVG-TopK pooling

Figure 11: Accuracy plots of VGG16 with different pooling methods on MNIST

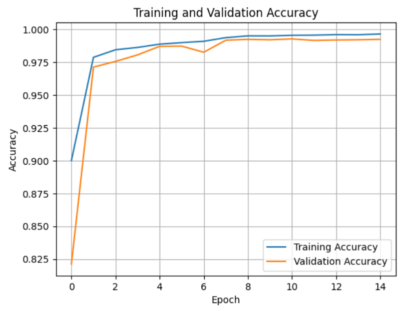
5.2.3 ResNet34



(a) ResNet34 with standard pooling



(b) ResNet34 with AVG-Mixed pooling



(c) ResNet34 with AVG-TopK pooling

Figure 12: Accuracy plots of ResNet34 with different pooling methods on MNIST

5.3 CIFAR-10

This section presents the experimental results evaluating the performance of AlexNet, VGG16, and ResNet34 using different pooling methods on the CIFAR-10 dataset. The standard architectures of these CNNs are modified by replacing the standard max pooling layers with the AVG-Mixed and AVG-TopK pooling layers. The performance of each architecture with these custom pooling methods is compared to the baseline architecture with standard max pooling.

Model	Accuracy(val)	Accuracy(test)
AlexNet		
Baseline	86.2%	84.8%
AVG-Mixed	84.8%	84.1%
AVG-TopK	86.4%	**85.5%**
VGG16		
Baseline	78.8%	77.6%
AVG-Mixed	79.8%	**78.6%**
AVG-TopK	N/A [1]	N/A [1]
ResNet34		
Baseline	78.2%	77.1%
AVG-Mixed	82.6%	**81.6%**
AVG-TopK	81.4%	80.2%

Table 2: Validation and test accuracies of the different models and pooling methods on CIFAR-10

The results in Table 2 demonstrate that:

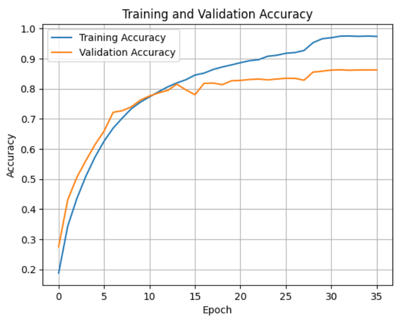
- **AlexNet:** The baseline model achieved a validation accuracy of 86.2% and a test accuracy of 84.8%. In the AVG-Mixed model, the validation and test accuracy are slightly lower, at 84.8% and 81.1%. The AVG-TopK pooling method achieved a validation accuracy of 86.4% and test accuracy of 85.5%, improving the performance by 0.2% and 0.7% compared to the baseline.

- **VGG16:** The baseline model achieved a validation accuracy of 78.8% and a test accuracy of 77.6%. The AVG-Mixed model improves the accuracies by 1.0%, to 79.8% and 78.6%, compared to the baseline model.

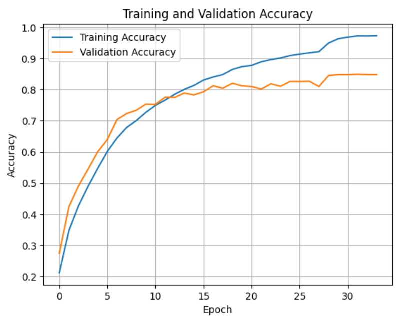
- **ResNet34:** The baseline model achieved a validation accuracy of 78.2% and a test accuracy of 77.1% The AVG-Mixed model improves the baseline accuracies by 4.4% and 4.5%, to 82.6% and 81.6%. The AVG-TopK model achieved a validation accuracy of 81.4% and test accuracy of 80.2%, which is an improvement of 3.2% and 3.1% over the baseline model.

[1] Results are missing due to a coding bug that invalidated the initial experiments

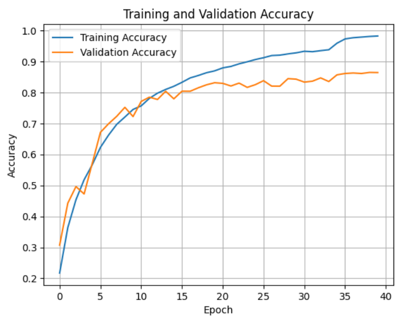
5.3.1 AlexNet



(a) AlexNet with Standard Pooling

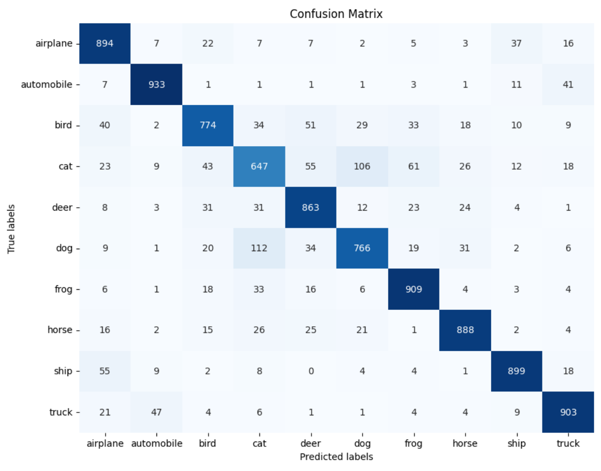


(b) AlexNet with AVG-Mixed Pooling



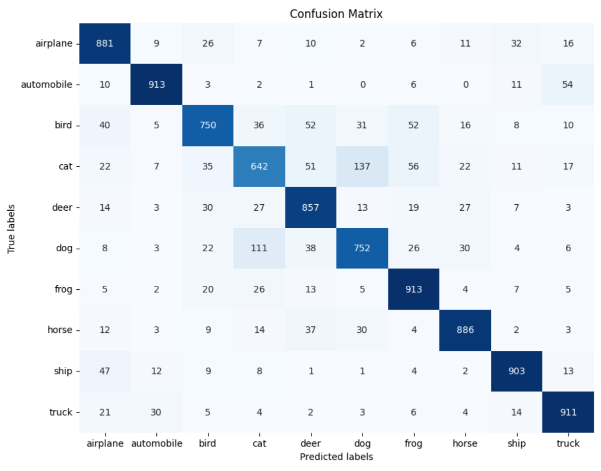
(c) AlexNet with AVG-TopK Pooling

Figure 13: Accuracy plots of AlexNet with different pooling methods on CIFAR-10



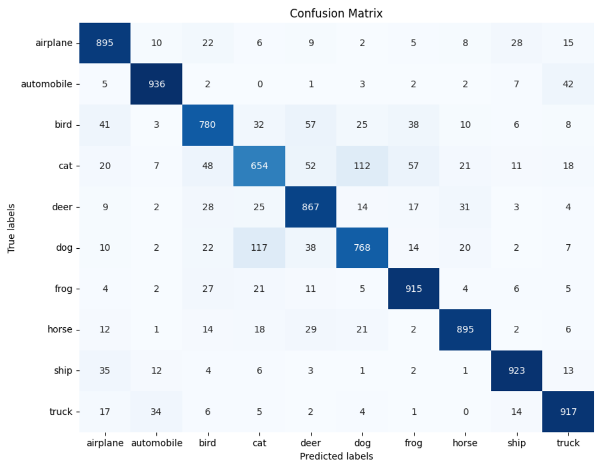
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	894	7	22	7	7	2	5	3	37	16
automobile	7	933	1	1	1	1	3	1	11	41
bird	40	2	774	34	51	29	33	18	10	9
cat	23	9	43	647	55	106	61	26	12	18
deer	8	3	31	31	863	12	23	24	4	1
dog	9	1	20	112	34	766	19	31	2	6
frog	6	1	18	33	16	6	909	4	3	4
horse	16	2	15	26	25	21	1	888	2	4
ship	55	9	2	8	0	4	4	1	899	18
truck	21	47	4	6	1	1	4	4	9	903

(a) AlexNet with Standard pooling



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	881	9	26	7	10	2	6	11	32	16
automobile	10	913	3	2	1	0	6	0	11	54
bird	40	5	750	36	52	31	52	16	8	10
cat	22	7	35	642	51	137	56	22	11	17
deer	14	3	30	27	857	13	19	27	7	3
dog	8	3	22	111	38	752	26	30	4	6
frog	5	2	20	26	13	5	913	4	7	5
horse	12	3	9	14	37	30	4	886	2	3
ship	47	12	9	8	1	1	4	2	903	13
truck	21	30	5	4	2	3	6	4	14	911

(b) AlexNet with AVG-Mixed pooling



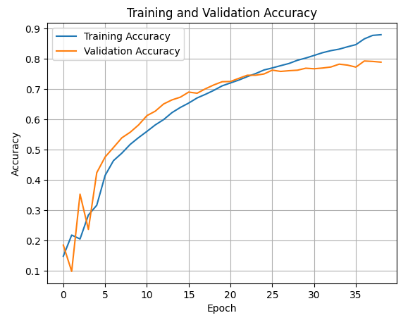
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	895	10	22	6	9	2	5	8	28	15
automobile	5	936	2	0	1	3	2	2	7	42
bird	41	3	780	32	57	25	38	10	6	8
cat	20	7	48	654	52	112	57	21	11	18
deer	9	2	28	25	867	14	17	31	3	4
dog	10	2	22	117	38	768	14	20	2	7
frog	4	2	27	21	11	5	915	4	6	5
horse	12	1	14	18	29	21	2	895	2	6
ship	35	12	4	6	3	1	2	1	923	13
truck	17	34	6	5	2	4	1	0	14	917

(c) AlexNet with AVG-TopK pooling

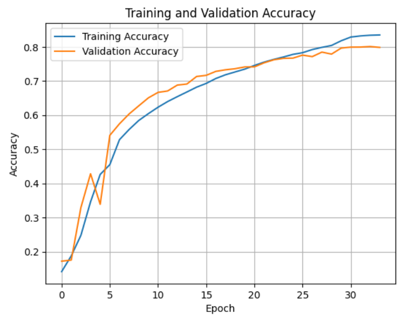
Figure 14: Confusion matrices of AlexNet with different pooling methods on CIFAR-10

The results in Table 2 demonstrate that the AVG-Mixed pooling method achieved worse validation and test accuracies, reduced by 1.4% and 0.7% compared to the baseline model. Additionally, the results in Table 2 demonstrate that AlexNet with the AVG-TopK pooling method achieved an improved performance compared to the AVG-Mixed and baseline models on the CIFAR-10 dataset. The validation and test accuracies are improved by 0.2% and 0.7%, compared to the baseline model. The confusion matrices in Figure 14 demonstrate that the three AlexNet models achieved similar correct predictions on the different classes of the CIFAR-10 dataset.

5.3.2 VGG16

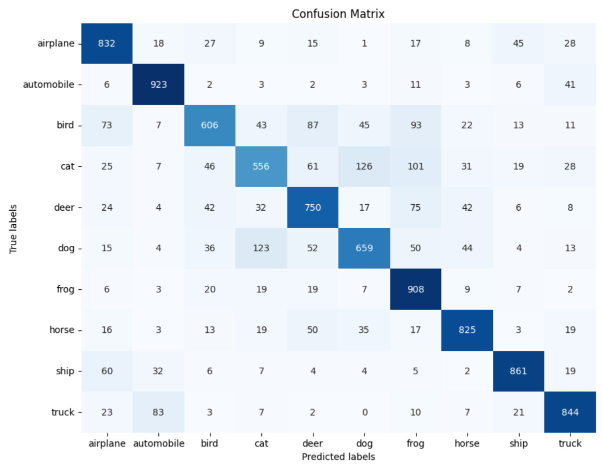


(a) VGG16 with standard pooling



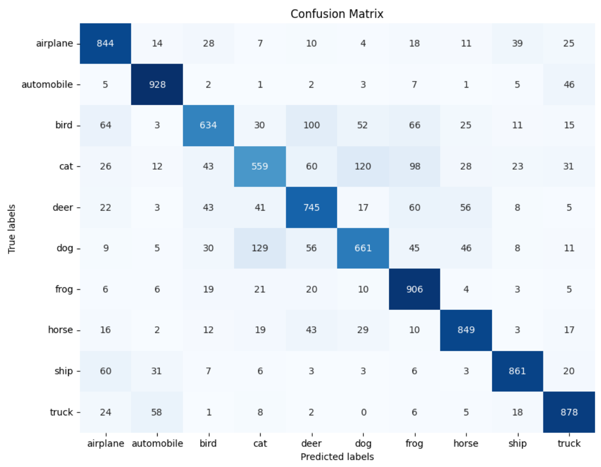
(b) VGG16 with AVG-Mixed pooling

Figure 15: Accuracy plots of VGG16 with different pooling methods on CIFAR-10



True \ Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	832	18	27	9	15	1	17	8	45	28
automobile	6	923	2	3	2	3	11	3	6	41
bird	73	7	606	43	87	45	93	22	13	11
cat	25	7	46	556	61	126	101	31	19	28
deer	24	4	42	32	750	17	75	42	6	8
dog	15	4	36	123	52	659	50	44	4	13
frog	6	3	20	19	19	7	908	9	7	2
horse	16	3	13	19	50	35	17	825	3	19
ship	60	32	6	7	4	4	5	2	861	19
truck	23	83	3	7	2	0	10	7	21	844

(a) VGG16 with standard pooling



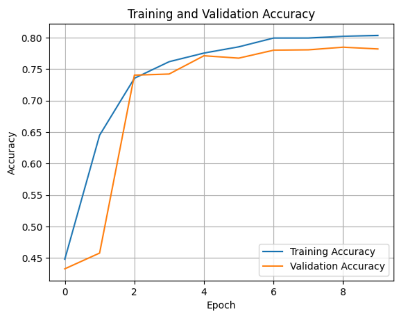
True \ Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	844	14	28	7	10	4	18	11	39	25
automobile	5	928	2	1	2	3	7	1	5	46
bird	64	3	634	30	100	52	66	25	11	15
cat	26	12	43	559	60	120	98	28	23	31
deer	22	3	43	41	745	17	60	56	8	5
dog	9	5	30	129	56	661	45	46	8	11
frog	6	6	19	21	20	10	906	4	3	5
horse	16	2	12	19	43	29	10	849	3	17
ship	60	31	7	6	3	3	6	3	861	20
truck	24	58	1	8	2	0	6	5	18	878

(b) VGG16 with AVG-Mixed pooling

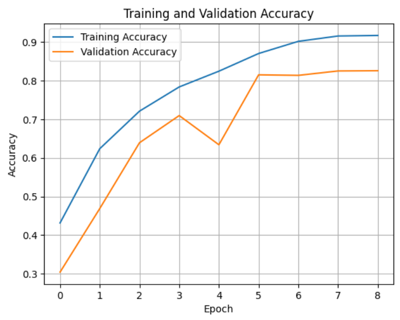
Figure 16: Confusion matrices of VGG16 with different pooling methods on CIFAR-10

The results in Table 2 show that the baseline VGG16 model achieved a solid performance with a validation accuracy of 78.8% and test accuracy of 77.6%. By incorporating the AVG-Mixed pooling method in VGG16, the model achieved an improvement of 1.0% in both the validation and test accuracy, compared to the baseline model. This implies that the AVG-Mixed pooling method can better capture and retain important features. The confusion matrices in Figure 16a and 16b show that the VGG16 with the AVG-Mixed pooling predicts each class, except for the 'deer' class, of the CIFAR-10 dataset equally or better than the baseline model.

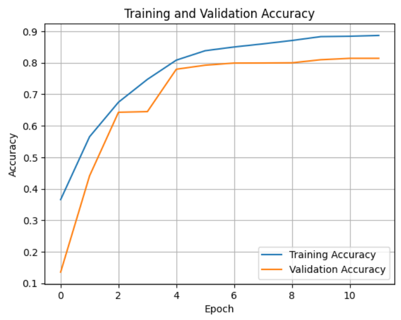
5.3.3 ResNet34



(a) ResNet34 with standard pooling

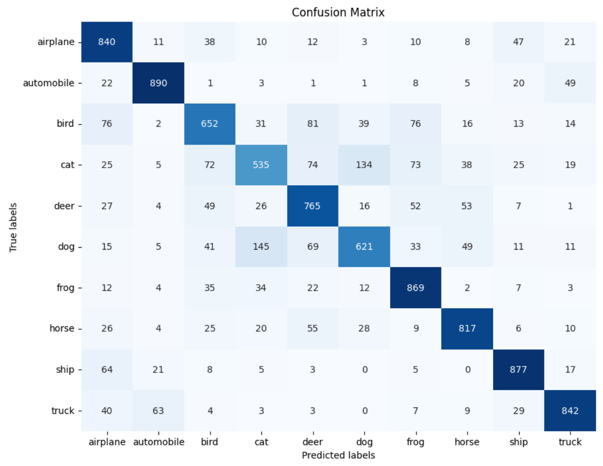


(b) ResNet34 with AVG-Mixed pooling

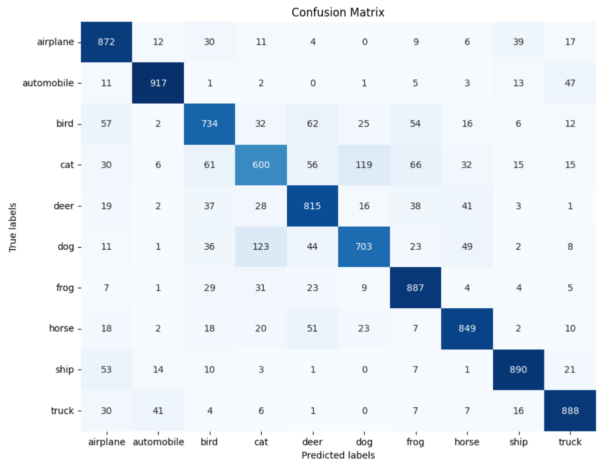


(c) ResNet34 with AVG-TopK pooling

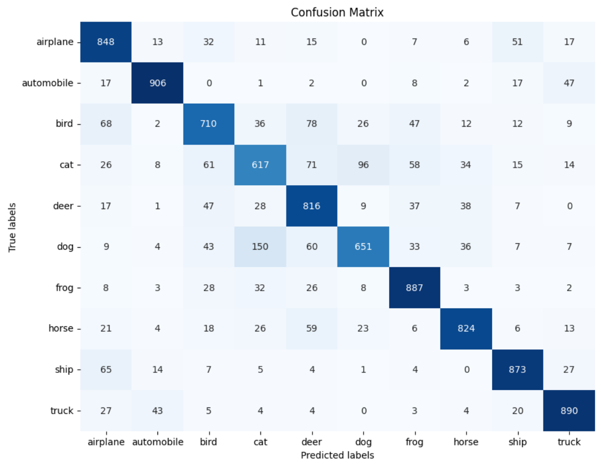
Figure 17: Accuracy plots of ResNet34 with different pooling methods on CIFAR-10



(a) ResNet34 with standard pooling



(b) ResNet34 with AVG-Mixed pooling



(c) ResNet34 with AVG-TopK pooling

Figure 18: Confusion matrices of ResNet34 with different pooling methods on CIFAR-10

The results in Table 2 show that the baseline ResNet34 model achieved a validation accuracy of 78.2% and a test accuracy of 77.1%. By replacing the max pooling layers with AVG-Mixed pooling, the model improves its performance significantly to 82.6% and 81.6%. This implies that the AVG-mixed pooling method improves the ResNet34 model's ability to capture important features or generalize better, compared to the standard pooling method used in ResNet34. Furthermore, the results of the ResNet34 model with AVG-TopK pooling also demonstrate an improvement over the baseline model, achieving validation and test accuracies of 81.4% and 80.2%. The confusion matrices in Figure 18c and 18a show a significant difference in the number of correctly predicted classes by the baseline model and the AVG-TopK model on CIFAR-10. More specifically, there are significant differences in the following classes: bird, cat, and deer. The AVG-TopK model achieved better predictions in those classes compared to the baseline model. Furthermore, the confusion matrix of the AVG-Mixed model in Figure 18b also shows significantly better performance in those three classes, along with an additional improvement in the 'dog' class.

5.4 CIFAR-100

This section presents the experimental results evaluating the performance of AlexNet, VGG16, and ResNet34 using different pooling methods on the CIFAR-100 dataset. The standard architectures of these CNNs are modified by replacing the standard max pooling layers with the AVG-Mixed and AVG-TopK pooling layers. The performance of each architecture with these custom pooling methods is compared to the baseline architecture with standard max pooling.

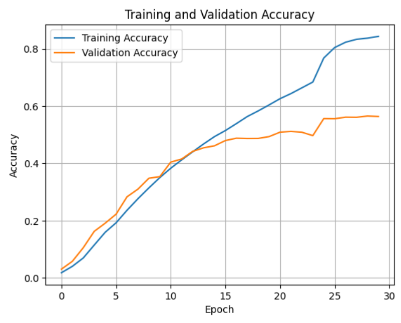
Model	Accuracy(val)	Accuracy(test)
AlexNet		
Baseline	56.4%	**58.0%**
AVG-Mixed	54.4%	55.2%
AVG-TopK	55.4%	56.8%
VGG16		
Baseline	53.0%	53.3%
AVG-Mixed	55.5%	**55.8%**
AVG-TopK	N/A [1]	N/A [1]
ResNet34		
Baseline	49.4%	50.9%
AVG-Mixed	51.3%	52.0%
AVG-TopK	54.0%	**54.9%**

Table 3: Validation and test accuracies of the different models and pooling methods on CIFAR-100

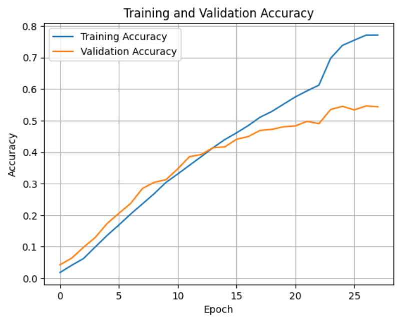
The results in Table 3 demonstrate that:

- **AlexNet:** The baseline model achieved a validation accuracy of 56.4% and a test accuracy of 58.0%. In the AVG-Mixed model, the achieved validation and test accuracy are slightly lower, at 54.4% and 55.2%. The AVG-TopK model achieved a validation accuracy of 55.4% and a test accuracy of 56.8%.
- **VGG16:** The baseline model achieved a validation accuracy of 53.0% and a test accuracy of 53.3%. The AVG-Mixed model improved the accuracies by 2.5%, to 55.5% and 55.8%, compared to the baseline model.
- **ResNet34:** The baseline model achieved a validation accuracy of 49.4% and a test accuracy of 50.9% The AVG-Mixed model improved the baseline accuracies by 1.9% and 1.1%, to 51.3% and 52.0%. The AVG-TopK model improved the accuracies by 4.6% and 4.0%, to a validation accuracy of 54.0% and test accuracy of 54.9%.

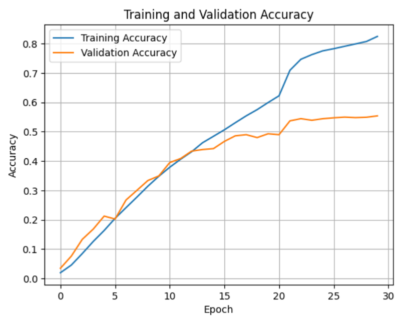
5.4.1 AlexNet



(a) AlexNet with Standard Pooling



(b) AlexNet with AVG-Mixed Pooling

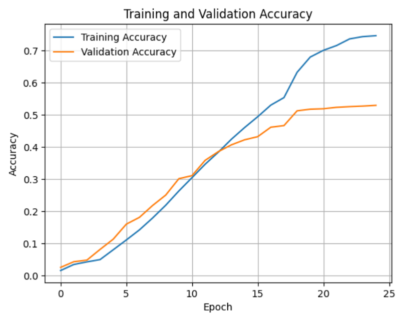


(c) AlexNet with AVG-TopK Pooling

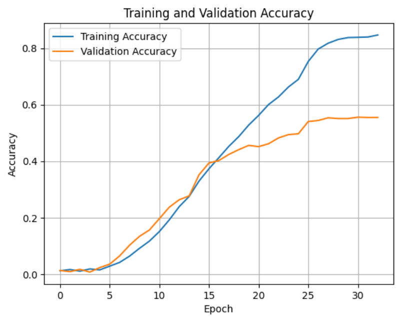
Figure 19: Accuracy plots of AlexNet with different pooling methods on CIFAR-100

The results in Table 3 show that the baseline AlexNet model achieved a validation accuracy of 56.4% and a test accuracy of 58.0%. The AVG-Mixed pooling method achieved a worse performance, with the model achieving a validation accuracy of 54.4% and a test accuracy of 55.2%. The AlexNet model with the integrated AVG-TopK pooling method achieved a validation accuracy of 55.4% and test accuracy of 56.8%, which is also worse than the baseline AlexNet but better than the AVG-Mixed pooling method. The plots in Figure 19 show that the AlexNet models are overfitting, indicated by the significant difference in the training and validation accuracy. The training accuracy significantly surpasses the validation and test accuracies.

5.4.2 VGG16



(a) VGG16 with standard pooling

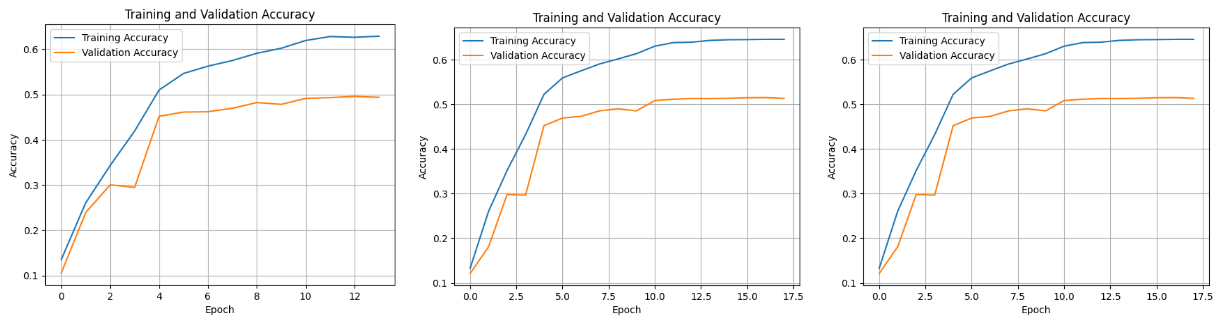


(b) VGG16 with AVG-Mixed pooling

Figure 20: Accuracy plots of VGG16 with different pooling methods on CIFAR-100

The results in Table 3 show that the baseline model achieved a validation accuracy of 53.0% and a test accuracy of 53.3%. The AVG-Mixed model improved the accuracies by 2.5%, to 55.5% and 55.8%, compared to the baseline model. Similarly to the AlexNet results on CIFAR-100, the training and validation accuracies of the VGG16 models, in Figure 20a and 20b, also indicated that the models are overfitting.

5.4.3 ResNet34



(a) ResNet34 with standard pooling (b) ResNet34 with AVG-Mixed pooling (c) ResNet34 with AVG-TopK pooling

Figure 21: Accuracy plots of ResNet34 with different pooling methods on CIFAR-100

The results in Table 3 show that the baseline model achieved a validation accuracy of 53.0% and a test accuracy of 53.3%. The AVG-Mixed model improved the accuracies by 1.9% and 1.1%, to 51.3% and 52.0%, compared to the baseline model. The AVG-TopK method significantly improved the performance by 4.6% and 4.0% to 54.0% and 54.9%.
Similarly to the AlexNet and VGG16 results on CIFAR-100, the training and validation accuracies of the ResNet34 models, in Figure 21, also indicate that the models are overfitting.

6 Discussion

This study evaluated the effectiveness of the AVG-Mixed and AVG-TopK pooling methods by integrating them into AlexNet, VGG16, and ResNet34 architectures and training the models on the MNIST, CIFAR-10, and CIFAR-100 datasets. A comparative assessment was conducted to determine the impact of the custom pooling methods on the model performance. The performance of the CNN models using the AVG-Mixed and AVG-TopK pooling methods was compared to the models using standard max pooling.

The experimental results of the three CNN architectures with the different types of pooling methods on the MNIST dataset are not insightful for this study as all models achieved near-perfect accuracy. MNIST is a relatively simple dataset with clear distinct classes, which makes it easy for models to achieve high performance. Therefore, the focus in this discussion is primarily on the CIFAR-10 and CIFAR-100 datasets, where the effectiveness of the proposed pooling methods is more discernible.

The experimental results of the AlexNet models with the AVG-Mixed pooling method demonstrate it achieved a slightly decreased performance on CIFAR-10 and CIFAR-100 compared to the baseline AlexNet model. The experimental results of the VGG16 and ResNet34 architectures with the integrated AVG-Mixed pooling on CIFAR-10 and CIFAR-100 showed improved performance compared to their baseline models with standard max pooling. This new method appeared to better capture and retain important features, by providing a balanced approach, compared to the standard max pooling method in the VGG16 and ResNet34 architectures. Moreover, the experimental results imply that the effectiveness of the AVG-Mixed pooling method improves in deeper and more complex architectures. The depth of the architecture may allow the models to learn more hierarchical feature representations, provided by the AVG-Mixed pooling method.

The experimental results of the models with the integrated AVG-TopK pooling on the CIFAR-10 and CIFAR-100 datasets also imply that the effectiveness of the pooling method varies, depending on the CNN architecture and the dataset. AlexNet with the integrated AVG-TopK pooling method on CIFAR-10 achieved a slightly improved performance compared to the baseline with max pooling. However, the AVG-TopK pooling method achieved a reduced performance on CIFAR-100 compared to the baseline AlexNet. The experimental results of the ResNet34 model with the Top-AVGK pooling method demonstrate significantly better performance on CIFAR-10 and CIFAR-100 compared to the baseline ResNet34 model. This may be due to their deeper architectures and residual connections which might be better equipped to handle the feature aggregation that is performed by the AVG-TopK pooling method.
Unfortunately, the results for VGG16 with AVG-TopK are missing, as noted in footnote [1]. The results could have provided insights into the effectiveness of the AVG-TopK pooling with smaller pooling sizes, namely of 2x2, compared to the 3x3 pooling sizes used in AlexNet and ResNet34. In the case of VGG16, with a pooling window of 2x2 and K set to 3, the AVG-TopK pooling operation would be similar to average pooling, as it averages the top three values out of four in each pooling window.

Prior studies have also demonstrated that custom pooling methods can outperform standard max pooling. The study in which the AVG-TopK pooling method was proposed, demonstrated that replacing the traditional pooling with the AVG-TopK pooling method in the LeNet architecture improves the performance in MNIST, CIFAR-10, and CIFAR-100 [33]. However, the improved performance was not consistent for all types of pooling sizes and values for the K parameter of the AVG-TopK pooling method [33]. This implies that the effectiveness of the AVG-TopK pooling method may depend on the pooling size and value of K.
In the study [33], experiments with transfer learning models, DenseNet169 [21], VGG16 [42], and Resnet50 [19], on the CIFAR-100 dataset have also been conducted, which showed that the VGG16 model could not learn [33]. The DenseNet169 [21] and ResNet50 [19] achieved improved performance with the AVG-TopK pooling method compared to their standard pooling methods [33].

This study has limitations. First, the dataset scope is limited, as the CNN models were trained and tested on standard datasets that do not include relatively complex images. Second, due to time constraints, experiments with different values for K for the AVG-TopK pooling method have not been conducted. In this study, the value of K in AVG-TopK pooling was set to 3, which might not be optimal. Different architectures and datasets may require different values for K to effectively capture important features. Third, during the implementation phase of this study, the choice was made to use the default weight initialization from Keras, Glorot initialization [15], for all models. This decision was made because AlexNet was unable to learn effectively, achieving 10% accuracy, with the original weight initialization described in its paper [23]. However, at a later stage, it was noted that the He weight initialization method is preferred for models using ReLU activation functions [18]. Since AlexNet, VGG16, and ResNet34 all utilize ReLU activation functions, the use of Glorot initialization could have potentially hindered their learning effectiveness. Lastly, while both the AVG-Mixed and AVG-TopK offer improved performance in some cases, their increased computational intensity is a notable disadvantage, compared to the standard max pooling operation in those CNNs. The max pooling method merely selects the maximum value within each pooling window. The AVG-TopK, on the contrary, requires first identifying the top values and then performing additional averaging. The AVG-Mixed method also requires more computation as both the average and maximum values within each pooling window have to be selected and then averaged. Considering both custom pooling methods require more computation power and time compared to traditional pooling methods, their practicality can be limited.

This study provides insights into the effect of the pooling method on the performance of different popular CNN models. This study contributes to the field of deep learning by focusing on one of the building blocks of CNNs: the pooling layer. This study provides insights for researchers that the performance of different CNN architectures could be improved by replacing the standard pooling operation with custom pooling methods. Moreover, it provides insights into the effect of custom pooling methods that combine aspects from the two traditional pooling methods, max and average pooling, on the performance of CNNs.

Future research can build on the findings of this study by integrating the two custom pooling methods into different types of CNN architectures, like GoogLeNet [43] or ResNet50 [19]. This could be valuable as the experimental results of the CNNs trained on CIFAR-10 and CIFAR-100 imply that the performance of deeper architectures improves with both new pooling methods. Also, in future experiments, the custom pooling methods could be evaluated on larger complex datasets as the experiments in this study were conducted using relatively small datasets. These further investigations with other CNN architectures and datasets could validate the generalizability of both the AVG-Mixed and AVG-TopK pooling methods. Additionally, future research could explore optimal values for the K parameter in the AVG-TopK pooling method to improve model performance.

7 Conclusion

This study introduced a new pooling method called AVG-mixed and builds upon recent research on the AVG-TopK pooling method [33]. Both pooling methods aim to improve the performance of Convolutional Neural Networks(CNNs) on image classification by providing a more balanced pooling approach, compared to the traditional pooling methods. In this study, the impact of both custom pooling methods on the performance of three benchmark CNNs, AlexNet [23], VGG16 [42], and ResNet34 [19], was investigated. The AVG-Mixed and AVG-TopK pooling layers, individually, replaced the max pooling layer in the baseline configuration of these networks. The CNN models were evaluated on three benchmark image classification datasets: MNIST, CIFAR-10, and CIFAR-

100. Moreover, a comparative assessment was conducted to evaluate the architectures' performance with the two custom pooling methods and the standard max pooling methods across the three datasets.
The experimental results demonstrate that ResNet34 consistently achieved improved results across MNIST, CIFAR-10, and CIFAR-100 with both the AVG-Mixed and AVG-TopK pooling methods, compared to the standard max pooling method in the baseline model. The two custom pooling methods were demonstrated to be effective in retaining important information from features for accurate classification in the ResNet34 model. Additionally, the VGG16 model achieved slightly higher performance with the AVG-mixed model on CIFAR-10 and CIFAR-100, compared to the baseline VGG16 model. The AlexNet model achieved improved performance on CIFAR-10 with the AVG-TopK pooling method, compared to the baseline model. The experimental results imply that the effectiveness of both AVG-Mixed and AVG-TopK pooling may depend on the complexity and depth of the CNN architecture, as well as the dataset used.
This study contributes to the field of deep learning by conducting a study that could provide valuable insights for future research on one of the building blocks of CNNs: the pooling layer. Additionally, the AVG-Mixed and AVG-TopK pooling methods could be used as replacements for standard pooling operations in different CNN architectures, potentially improving the performance.

References

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi. "Understanding of a Convolutional Neural Network". In: *Proceedings of the 2017 International Conference on Engineering and Technology (ICET)*. 2017. DOI: 10.1109/ICEngTechnol.2017.8308186.

[2] L. Alzubaidi et al. "Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions". In: *Journal of Big Data* 8.1 (2021), pp. 1–74. DOI: 10.1186/s40537-021-00444-8.

[3] M. Arham. *Diving into the Pool: Unraveling the Magic of CNN Pooling Layers*. Available at https://www.kdnuggets.com/diving-into-the-pool-unraveling-the-magic-of-cnn-pooling-layers. 2023.

[4] M. Basavarajaiah. *Maxpooling vs minpooling vs average pooling - Madhushree Basavarajaiah*. Available at https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9. 2021.

[5] Data Basecamp. *ResNet: Residual Neural Networks - Easily Explained!* 2023.

[6] L. Borawar and R. Kaur. "RESNET: Solving Vanishing Gradient in Deep Networks". In: *Lecture Notes in Networks and Systems*. Springer, 2023, pp. 235–247. DOI: 10.1007/978-981-19-8825-7_21.

[7] Y. Boureau, J. Ponce, and Y. LeCun. "A Theoretical Analysis of Feature Pooling in Visual Recognition". In: *Proceedings of the 27th International Conference on Machine Learning*. ACM, 2010, pp. 111–118.

[8] Y. Boureau et al. "Learning Mid-Level Features for Recognition". In: *Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010, pp. 2559–2566. DOI: 10.1109/CVPR.2010.5539963.

[9] Y. Chen et al. "A Deep Four-Stream Siamese Convolutional Neural Network with Joint Verification and Identification Loss for Person Re-Detection". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2018, pp. 1292–1301. DOI: 10.1109/WACV.2018.00146.

[10] D. Danushi. *Max Pooling*. Available at https://medium.com/@danushidk507/max-pooling-ef545993b6e4. 2023.

[11] K. Dhanwal, A. Mittal, and B. Kaushik. "A Comparison of Pooling Techniques for Neural Networks". In: *Artificial Intelligence Review* (2023). DOI: 10.1007/s10462-023-10631-z.

[12] K. Fukushima. "Artificial Vision by Deep CNN Neocognitron". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1 (2021), pp. 76–90. DOI: 10.1109/TSMC.2020.3042785.

[13] Z. Gao, L. Wang, and G. Wu. "LIP: Local Importance-Based Pooling". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 3354–3363. DOI: 10.1109/ICCV.2019.00345.

[14] H. Gholamalinejad and H. Khosravi. "Pooling Methods in Deep Neural Networks, a Review". In: *ResearchGate* (2020). URL: https://www.researchgate.net/publication/344277235_Pooling_Methods_in_Deep_Neural_Networks_a_Review.

[15] X. Glorot and Y. Bengio. “Understanding the Difficulty of Training Deep Feedforward Neural Networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*. Vol. 9. 2010, pp. 249–256.

[16] X. Glorot, A. Bordes, and Y. Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. Vol. 15. 2011, pp. 315–323.

[17] K. He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”. In: *Lecture Notes in Computer Science*. Springer, 2014, pp. 346–361. DOI: 10.1007/978-3-319-10578-9_23.

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

[19] K. He et al. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[20] E. Hssayni and M. Ettaouil. “A Novel Pooling Method for Regularization of Deep Neural networks”. In: *2020 International Conference on Intelligent Systems and Computer Vision (ISCV)*. IEEE, 2020, pp. 1–6. DOI: 10.1109/ISCV49265.2020.9204322.

[21] G. Huang et al. “Densely Connected Convolutional Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.

[22] K. Kavukcuoglu et al. “Learning invariant features through topographic filter maps”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 1605–1612. DOI: 10.1109/CVPR.2009.5206545.

[23] A. Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. Master’s thesis. Department of Computer Science, University of Toronto, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90. DOI: 10.1145/3065386.

[25] Y. LeCun, C. Cortes, and C. Burges. *MNIST Handwritten Digit Database*. Available at http://yann.lecun.com/exdb/mnist/. n.d.

[26] Y. LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[27] C. Lee, P. W. Gallagher, and Z. Tu. “Generalizing Pooling Functions in CNNs: Mixed, Gated, and Tree”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8.4 (2016), pp. 464–465. DOI: 10.1109/TPAMI.2017.2703082.

[28] Z. Li et al. “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022), pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.

[29] J. Liu et al. “A Learning-Based Frame Pooling Model for Event Detection”. In: *ResearchGate* (2016). URL: https://www.researchgate.net/publication/301841902_A_Learning-based_Frame_Pooling_Model_For_Event_Detection.

[30] J. Long, E. Shelhamer, and T. Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440.

[31] P. Mahajan. *Understanding Residual Network (ResNet) Architecture*. 2020. URL: https://medium.com/analytics-vidhya/understanding-resnet-architecture-869915cc2a98.

[32] K. Murphy et al. “Object detection and localization using local and global features”. In: *Lecture Notes in Computer Science*. Springer, 2006, pp. 382–400. DOI: 10.1007/11957959_20.

[33] C. Özdemir. “Avg-topk: A New Pooling Method for Convolutional Neural Networks”. In: *Expert Systems With Applications* 223 (2023), p. 119892. DOI: 10.1016/j.eswa.2023.119892.

[34] M. S. S. Patel and S. S. “Improved Spatial Invariance for Vehicle Platoon Application using New Pooling Method in Convolution Neural Network”. In: *International Journal of Advanced Computer Science and Applications* 13.7 (2022).

[35] M. Ranzato et al. “Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition”. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383157.

[36] R. Reshma and A. Jose Anand. “Predictive and Comparative Analysis of LENET, ALEXNET and VGG-16 Network Architecture in Smart Behavior Monitoring”. In: *2023 Seventh International Conference on Image Information Processing (ICIIP)*. IEEE, 2023, pp. 450–453. DOI: 10.1109/ICIIP61524.2023.10537732.

[37] A. M. Romano and A. A. Hernandez. “An Improved Pooling Scheme for Convolutional Neural Networks”. In: *2019 7th International Conference on Information, Communication and Networks (ICICN)*. IEEE, 2019, pp. 201–206. DOI: 10.1109/ICICN.2019.8834960.

[38] O. Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[39] F. Saeedan et al. “Detail-Preserving Pooling in Deep Networks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9108–9116. DOI: 10.1109/CVPR.2018.00949.

[40] P. Sermanet, S. Soumith, and Y. LeCun. “Convolutional neural networks applied to house numbers digit classification”. In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. 2012, pp. 3288–3291. URL: https://ieeexplore.ieee.org/document/6460867.

[41] Z. Shi, Y. Ye, and Y. Wu. “Rank-based pooling for deep convolutional neural networks”. In: *Neural Networks* 83 (2016), pp. 21–31. DOI: 10.1016/j.neunet.2016.07.003.

[42] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv* (2014). URL: https://arxiv.org/abs/1409.1556.

[43] C. Szegedy et al. “Going Deeper with Convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015. DOI: 10.1109/CVPR.2017.243.

[44] M. M. Taye. “Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions”. In: *Computation* 11.3 (2023), p. 52. DOI: 10.3390/computation11030052.

[45] M. M. Taye. “Theoretical understanding of convolutional neural network: concepts, architectures, applications, future directions”. In: *Computation (Basel)* 11.3 (2023), p. 52. DOI: 10.3390/computation11030052.

[46] N. Weber et al. “Rapid, Detail-Preserving Image Downscaling”. In: *ACM Transactions on Graphics* 35.6 (2016), pp. 1–6. DOI: 10.1145/2980179.2980239.

[47] M. Xin and Y. Wang. “Research on image classification model based on deep convolution neural network”. In: *EURASIP Journal on Image and Video Processing* (2019), p. 40. DOI: 10.1186/s13640-019-0417-8.

[48] R. Yamashita et al. “Convolutional Neural Networks: An Overview and Application in Radiology”. In: *Insights Into Imaging* 9.4 (2018), pp. 611–629. DOI: 10.1007/s13244-018-0639-9.

[49] D. Yu et al. “Mixed Pooling for Convolutional Neural Networks”. In: *Lecture Notes in Computer Science*. Springer, 2014, pp. 364–375. DOI: 10.1007/978-3-319-11740-9_34.

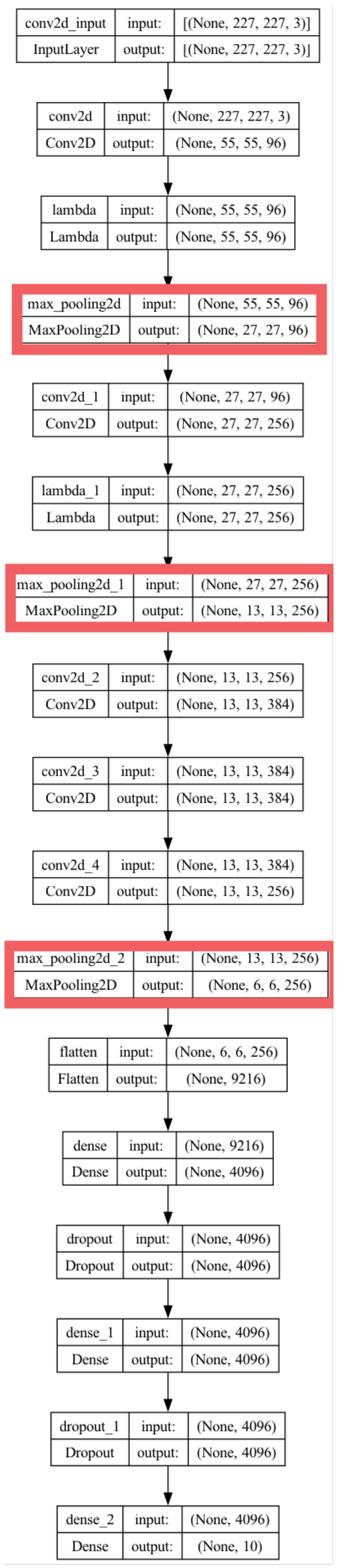
[50] A. Zafar et al. “A Comparison of Pooling Methods for Convolutional Neural Networks”. In: *Applied Sciences* 12.17 (2022), p. 8643. DOI: 10.3390/app12178643.

[51] M. D. Zeiler and R. Fergus. “Visualizing and Understanding Convolutional Networks”. In: *Lecture Notes in Computer Science* 8689 (2014), pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53.

[52] L. Zhao and Z. Zhang. “An Improved Pooling Method for Convolutional Neural Networks”. In: *Scientific Reports* 14.1 (2024). DOI: 10.1038/s41598-024-51258-6.

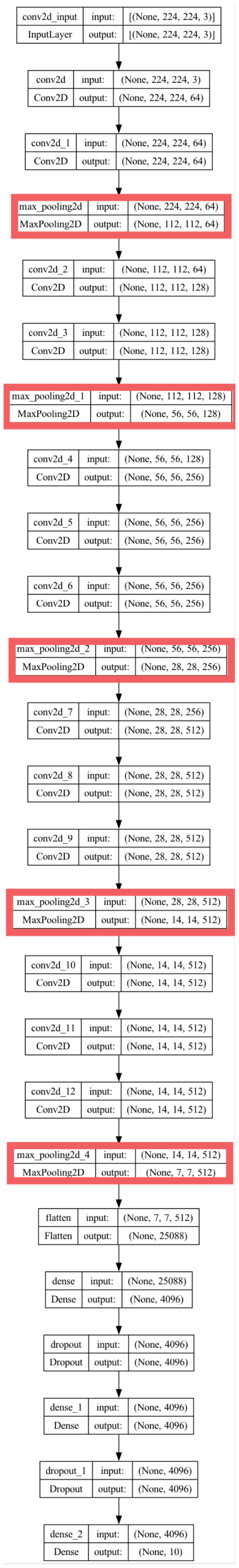
Appendices

A CNN Architectures



conv2d_input | InputLayer | input: [(None, 227, 227, 3)] | output: [(None, 227, 227, 3)]
conv2d | Conv2D | input: (None, 227, 227, 3) | output: (None, 55, 55, 96)
lambda | Lambda | input: (None, 55, 55, 96) | output: (None, 55, 55, 96)
max_pooling2d | MaxPooling2D | input: (None, 55, 55, 96) | output: (None, 27, 27, 96)
conv2d_1 | Conv2D | input: (None, 27, 27, 96) | output: (None, 27, 27, 256)
lambda_1 | Lambda | input: (None, 27, 27, 256) | output: (None, 27, 27, 256)
max_pooling2d_1 | MaxPooling2D | input: (None, 27, 27, 256) | output: (None, 13, 13, 256)
conv2d_2 | Conv2D | input: (None, 13, 13, 256) | output: (None, 13, 13, 384)
conv2d_3 | Conv2D | input: (None, 13, 13, 384) | output: (None, 13, 13, 384)
conv2d_4 | Conv2D | input: (None, 13, 13, 384) | output: (None, 13, 13, 256)
max_pooling2d_2 | MaxPooling2D | input: (None, 13, 13, 256) | output: (None, 6, 6, 256)
flatten | Flatten | input: (None, 6, 6, 256) | output: (None, 9216)
dense | Dense | input: (None, 9216) | output: (None, 4096)
dropout | Dropout | input: (None, 4096) | output: (None, 4096)
dense_1 | Dense | input: (None, 4096) | output: (None, 4096)
dropout_1 | Dropout | input: (None, 4096) | output: (None, 4096)
dense_2 | Dense | input: (None, 4096) | output: (None, 10)

Figure 22: Visualization of the AlexNet architecture, specifically highlighting the pooling layers that are being replaced by the custom pooling layers in the experiments



conv2d_input input: [(None, 224, 224, 3)]
InputLayer output: [(None, 224, 224, 3)]
conv2d input: (None, 224, 224, 3)
Conv2D output: (None, 224, 224, 64)
conv2d_1 input: (None, 224, 224, 64)
Conv2D output: (None, 224, 224, 64)
max_pooling2d input: (None, 224, 224, 64)
MaxPooling2D output: (None, 112, 112, 64)
conv2d_2 input: (None, 112, 112, 64)
Conv2D output: (None, 112, 112, 128)
conv2d_3 input: (None, 112, 112, 128)
Conv2D output: (None, 112, 112, 128)
max_pooling2d_1 input: (None, 112, 112, 128)
MaxPooling2D output: (None, 56, 56, 128)
conv2d_4 input: (None, 56, 56, 128)
Conv2D output: (None, 56, 56, 256)
conv2d_5 input: (None, 56, 56, 256)
Conv2D output: (None, 56, 56, 256)
conv2d_6 input: (None, 56, 56, 256)
Conv2D output: (None, 56, 56, 256)
max_pooling2d_2 input: (None, 56, 56, 256)
MaxPooling2D output: (None, 28, 28, 256)
conv2d_7 input: (None, 28, 28, 256)
Conv2D output: (None, 28, 28, 512)
conv2d_8 input: (None, 28, 28, 512)
Conv2D output: (None, 28, 28, 512)
conv2d_9 input: (None, 28, 28, 512)
Conv2D output: (None, 28, 28, 512)
max_pooling2d_3 input: (None, 28, 28, 512)
MaxPooling2D output: (None, 14, 14, 512)
conv2d_10 input: (None, 14, 14, 512)
Conv2D output: (None, 14, 14, 512)
conv2d_11 input: (None, 14, 14, 512)
Conv2D output: (None, 14, 14, 512)
conv2d_12 input: (None, 14, 14, 512)
Conv2D output: (None, 14, 14, 512)
max_pooling2d_4 input: (None, 14, 14, 512)
MaxPooling2D output: (None, 7, 7, 512)
flatten input: (None, 7, 7, 512)
Flatten output: (None, 25088)
dense input: (None, 25088)
Dense output: (None, 4096)
dropout input: (None, 4096)
Dropout output: (None, 4096)
dense_1 input: (None, 4096)
Dense output: (None, 4096)
dropout_1 input: (None, 4096)
Dropout output: (None, 4096)
dense_2 input: (None, 4096)
Dense output: (None, 10)

Figure 23: Visualization of the VGG16 architecture, specifically highlighting the pooling layers that are being replaced by the custom pooling layers in the experiments

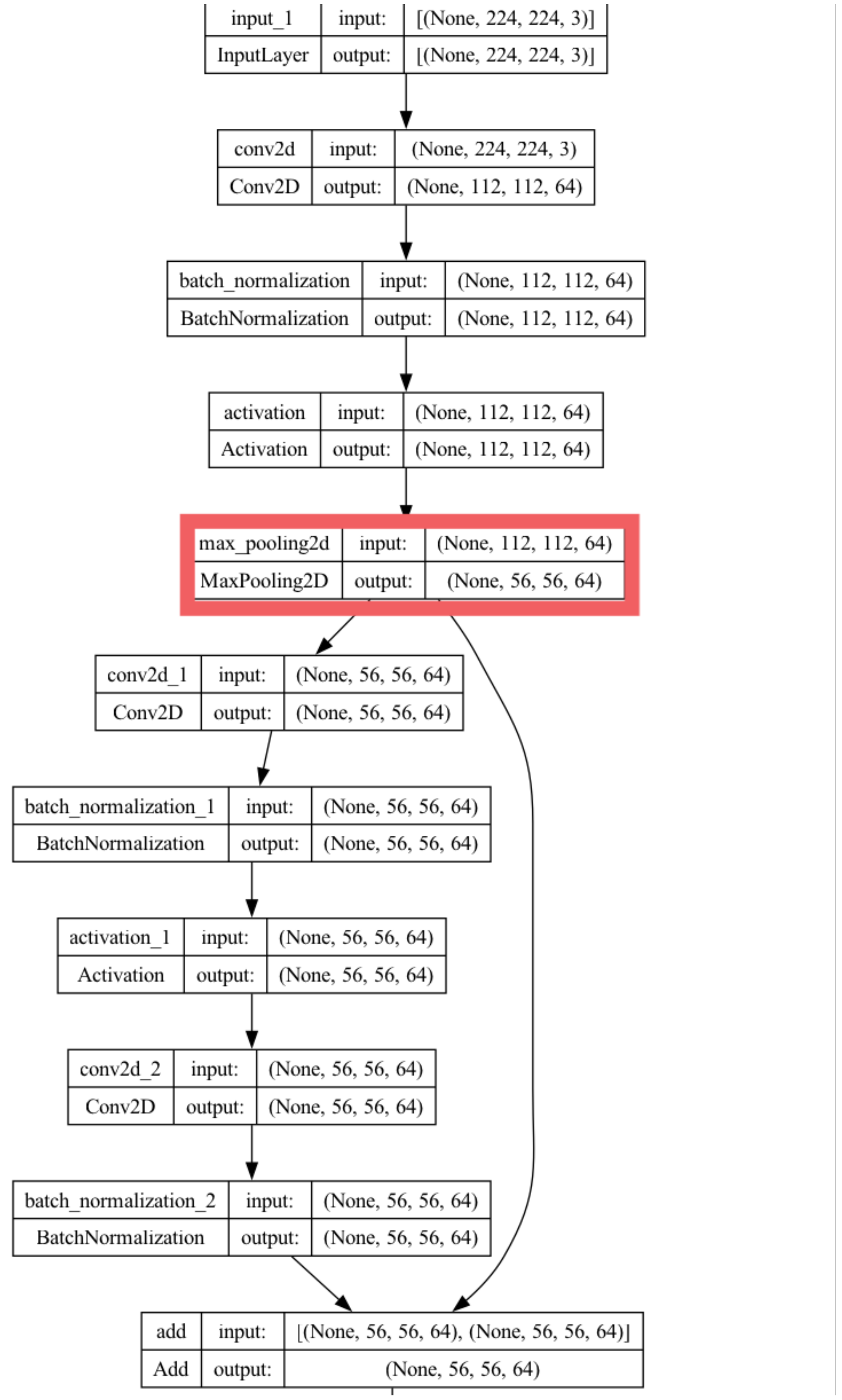


Figure 24: Partial visualization of the ResNet34 architecture. Due to space constraints, only a subset of the first layers is shown, specifically highlighting the only pooling layer that is being replaced by custom pooling layers in the experiments.