Investigating The Differences In Performance Between Traditional Neural Networks And Convolutional Neural Networks In Image Classification

COMS4030A
Adaptive Computation and Machine Learning

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Lindani Dlamini
1712359

Lindelani Dlamini
1898593

May 31, 2023

A group project submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfillment of the requirements for the degree of Bachelor of Science with Honours

Contents

List of Figures

Chapter 1

Introduction

Convolutional neural networks (CNNs) have become quite popular in image classification tasks more than the traditional fully connected neural networks because of their better performance with regards to processing data with a known grid-like structure such as an image. CNNs are still neural networks but they use convolution instead of traditional matrix multiplication in at least one of their layers [Ian *et al.* 2017]. A convolution is an operation on two functions, the input image and the kernel, to produce a feature map to extract the input image's features.

Traditional neural networks (TNNs) are universal approximators, which means that they can theoretically learn any function. However, they are not very efficient at learning functions that are spatially local, such as the features of an image. This is because TNNs treat all pixels in an image equally, regardless of their spatial relationship to each other. On the other hand. CNNs are designed to extract local information from images. They do this by using small filters or kernels that scan the image and capture local patterns or features such as edges, corners, and blobs. These filters are typically small in size and are applied across the entire image, with each filter producing a feature map that represents the response of the filter at each location.

In this project, we wish to investigate the difference in performance between these two neural network architectures to understand how their structures are adapted for image classification. We shall also explore some techniques on how to make our models more accurate.

Chapter 2

Methodology

2.1 Research Design

In general, the more layers we have in a network, we increase its knowledge capacity and its ability to understand the task as hand. More specifically, in CNNs, the more convolutional layers we have, the more high level features we extract to better classify the image. In the experiments to be conducted, we shall experiment with different numbers of layers in each network, integrate regularization and optimizers into the training process to obtain competent models.

2.2 The CIFAR-10 Dataset

For our project we will be looking at the task of image classification using the CIFAR-10 dataset. Researchers often use the CIFAR-10 dataset to benchmark their algorithms and compare their performance against state-of-the-art methods. It has become a standard dataset for evaluating the effectiveness of various deep learning architectures and techniques.

The CIFAR-10 dataset consists of 60,000 labeled images in 10 mutually exclusive classes, with a fixed size of 32x32x3 color channels [Krizhevsky and Hinton 2009]. The CIFAR-10 dataset is challenging because the images are low-resolution and contain variations in pose, viewpoint, lighting, and background. It poses a good test for developing and evaluating image classification algorithms.

2.3 Data Prepossessing

2.3.1 Data Pipeline

Our datasets are tensorflow datasets and we configure them in such a way that they make training the models computationally efficient. We employed buffered prefetching, which enables loading data from disk efficiently by overlapping data preprocessing and model execution while training. We also cache the images in memory after they're loaded off disk during the first epoch, and they're easily accessible during training.

2.3.2 Data Normalization

The RGB channel values are in the [0, 255] range. This is not ideal for a neural network and we opt to make our input values smaller. We thus normalize the values to be in the [0, 1] range, using the TensorFlow cast method.

2.4 Optimizers

2.4.1 Adam

The tensorflow Adam optimizer is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. It is computationally efficient, has little memory requirement, and is invariant to diagonal rescaling of gradients. Adam is well suited for problems that are large in terms of data parameters [tensorflow.org]. We have chosen it as our baseline optimizer since it has been shown to be effective on a variety of machine learning tasks, including image classification, natural language processing, and speech recognition.

2.4.2 Lion

Short for EvoLved Sign Momentum. This algorithm differs from various adaptive algorithms by only tracking momentum and leveraging the sign operation to calculate updates, leading to lower memory overhead and uniform update magnitudes across all dimensions [Chen *et al.* 2023]. It has demonstrated that it can aid tasks such as image classification attain outstanding results. Hence we have chosen to use it more extensively than Adam when training our models.

2.5 The Sparse Categorical Crossentropy Loss Function

For this project, we shall use tensorflow's Sparse Categorical Crossentropy loss function. We believe it's a good choice for image classification using CIFAR-10 because it is designed to handle sparse labels. It's also computationally efficient, which is important for image classification, where the model needs to be able to process a large number of images.

2.6 Sparse Cartegorical Accuracy

The Sparse Categorical Accuracy metric is a good choice for this project because it is designed to handle sparse labels such as CIFAR-10's 10 classes. Taking this into account, it calculates the accuracy of the model by counting the number of times the model predicts the correct class. It is also quite computationally efficient.

2.7 Regularization

2.7.1 Dropout Regularization

Dropout introduces noise and randomness into the network during training, which acts as a form of regularization. It prevents co-adaptation of neurons, encourages the learning of more robust features, and reduces the network's sensitivity to specific patterns in the training data. This can lead to better generalization performance on unseen data.

2.7.2 L2 Regularization

Weight decay, also known as L2 regularization or ridge regularization, is a regularization technique commonly used to prevent overfitting and improve the generalization performance of a model. It involves adding a regularization term to the loss function that penalizes large weights in the model. Ridge regularization has the effect of shrinking the weights towards zero. By penalizing large weights, it discourages the model from fitting noise in the training data and encourages it to learn more generalizable patterns. This regularization helps to reduce overfitting and improve the model's ability to generalize to unseen data. Even though it is specifically designed for linear models, we will look to apply it to the trainable parameters of the model to investigate it's effect.

2.8 Activation Functions

Most of our models will use a combination of the commonly used activation functions, Linear, ReLU, Sigmoid and Softmax.

2.8.1 Rectified Linear Unit (ReLU)

The ReLU activation function is one of the most widely used activation functions in deep learning. It returns the input value if it is positive and zero otherwise. ReLU introduces non-linearity and is computationally efficient. It helps in overcoming the vanishing gradient problem and accelerates the training process. ReLU, however, can lead to dead neurons (neurons that are never activated) as they become zero for negative inputs.

2.8.2 Sigmoid Activation Function

The sigmoid activation function maps the input to a value between 0 and 1. It is commonly used in binary classification problems where the network needs to output probabilities. Sigmoid function is smooth and continuously differentiable, making it suitable for gradient-based optimization algorithms. However, sigmoid is computationally demanding and suffers from the vanishing gradient problem, especially for extreme inputs, which can slow down training.

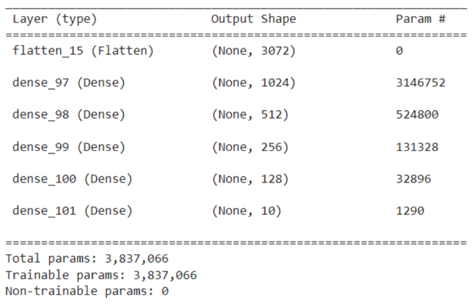
2.8.3 Softmax Activation Function

The softmax activation function takes a vector of arbitrary real values as input and transforms it into a probability distribution over multiple classes. Softmax ensures that the sum of the probabilities of all classes is equal to 1. It is normally used in the final layer of a neural network for multi-class classification tasks. Softmax function is differentiable, allowing for efficient training using gradient-based optimization algorithms.

2.9 Implementation

2.9.1 Traditional Neural Networks

TNN Model 1: Our model consists of 5 layers; 4 hidden layers and 1 output layer with no activation function. The first and third layer are sigmoid activated and the other are two relu activated. It also uses the Adam optimizer, SCCE loss function and SCA as an accuracy metric.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_15 (Flatten)        (None, 3072)              0

 dense_97 (Dense)            (None, 1024)              3146752

 dense_98 (Dense)            (None, 512)               524800

 dense_99 (Dense)            (None, 256)               131328

 dense_100 (Dense)           (None, 128)               32896

 dense_101 (Dense)           (None, 10)                1290

=================================================================
Total params: 3,837,066
Trainable params: 3,837,066
Non-trainable params: 0
```

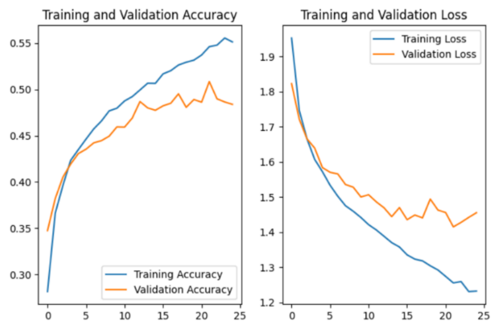
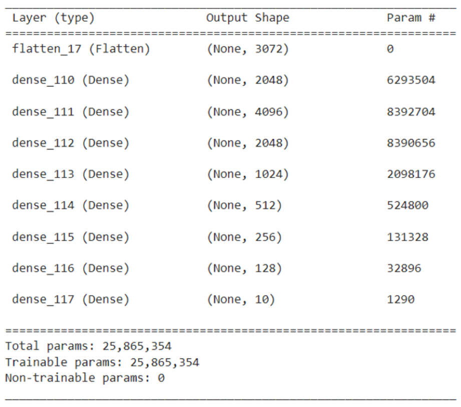


Figure 2.1: TNN Model 1 Summary and Accuracy & Loss Graphs

The use of sigmoid and relu activation functions, as opposed to the simpler linear activation function, allows our Neural Network model (of 3,837,066 parameters) to reach an accuracy of at least 48%. This is quite low, as this means that our classification model most likely fails 52% of the time. But is very expected and understandable of a Traditional Neural Network.
At over 10 epochs, out of a total of 25 and taking approximately 55 seconds to finish, we also start noticing some overfitting over the training data. The model isn't as good at classifying the validation data, as it is with the training data.

TNN Model 2: Our next model consists of 3 more hidden layers for a total of 8. This time, only the 2nd, 5th and 7th layers are relu activated, the rest are sigmoid. Even though it also uses the Adam optimizer, SCCE and SCA; because we use more sigmoid functions, and from having more layers and parameters in general, we expect our training to take longer.



```
Layer (type)                 Output Shape              Param #
=================================================================
flatten_17 (Flatten)         (None, 3072)              0

dense_110 (Dense)            (None, 2048)              6293504

dense_111 (Dense)            (None, 4096)              8392704

dense_112 (Dense)            (None, 2048)              8390656

dense_113 (Dense)            (None, 1024)              2098176

dense_114 (Dense)            (None, 512)               524800

dense_115 (Dense)            (None, 256)               131328

dense_116 (Dense)            (None, 128)               32896

dense_117 (Dense)            (None, 10)                1290

=================================================================
Total params: 25,865,354
Trainable params: 25,865,354
Non-trainable params: 0
_________________________________________________________________
```

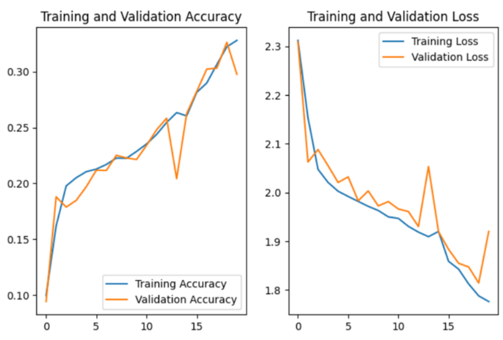
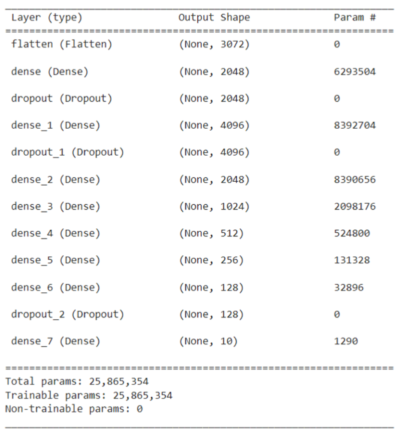


Figure 2.2: TNN Model 2 Summary and Accuracy & Loss Graphs

The more epochs and parameters we use, the higher the accuracy with respect to the training data. But also, with the more epochs we use, the more our traditional neural network model tends overfits to the training data. Not much overfitting is shown by our model, but at an accuracy of 30%, a loss of 1.88 and over 25 000 000 parameters; the model is able to poorly classify the training data, as relatively good as the validation data. Meaning it is most likely to make the wrong prediction approximately 70% of the time.

TNN Model 3: For our third and final TNN model, we keep the layers exactly the same as our previous model, except now we add some regularization. We add dropout regularization to the first, second and last hidden layers; and then add ridge regression to the last hidden layer. We also use a different, faster-converging optimizer; the Lion optimizer; instead of Adam.



```
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0

dense (Dense)                (None, 2048)              6293504

dropout (Dropout)            (None, 2048)              0

dense_1 (Dense)              (None, 4096)              8392704

dropout_1 (Dropout)          (None, 4096)              0

dense_2 (Dense)              (None, 2048)              8390656

dense_3 (Dense)              (None, 1024)              2098176

dense_4 (Dense)              (None, 512)               524800

dense_5 (Dense)              (None, 256)               131328

dense_6 (Dense)              (None, 128)               32896

dropout_2 (Dropout)          (None, 128)               0

dense_7 (Dense)              (None, 10)                1290

=================================================================
Total params: 25,865,354
Trainable params: 25,865,354
Non-trainable params: 0
_________________________________________________________________
```

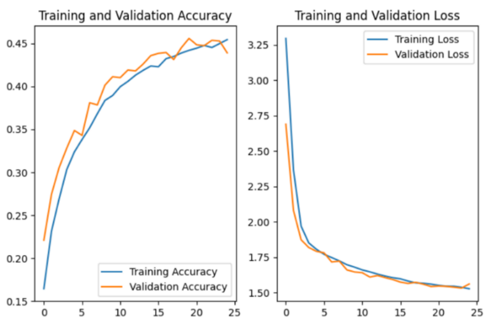


Figure 2.3: TNN Model 3 Summary and Accuracy & Loss Graphs

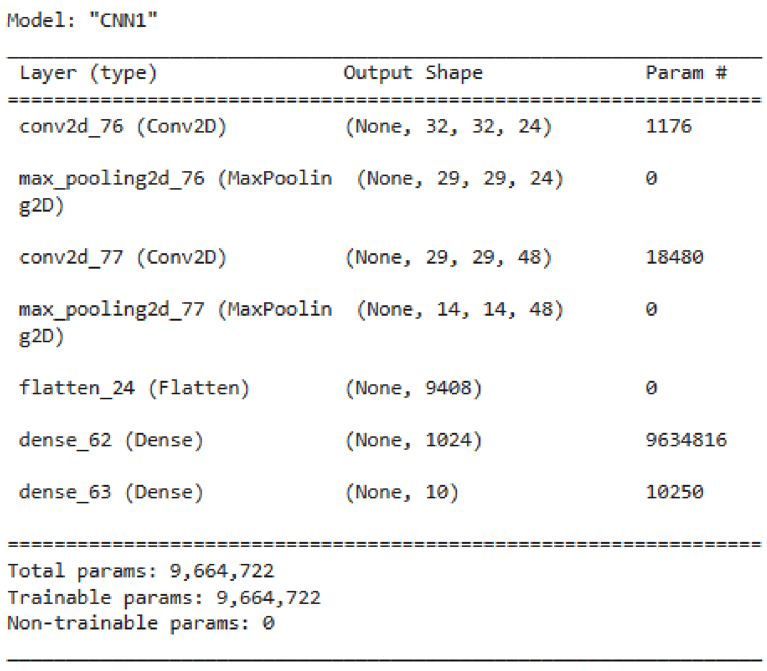
At 25 epochs, with an accuracy and loss of about 45% and 1.5 respectively, the model shows very little under and/or overfitting to the training data. This due to the high number of parameters (25,865,354), the Lion optimizer, the use of sigmoid and relu activation functions, and the added regularization which is normally used to particularly reduce overfitting.
We notice this is more accurate than the previous model, but given it is still just below even 50% accuracy; shows the TNNs inherently inefficient image classification capabilities.

In the next subsection, we will go on to investigate the more capable Convolutional Neural Network, for our image classification objective.

2.9.2 Convolutional Neural Networks

CNN1: Baseline Model

Our baseline model is a simple architecture with only a few convolutional layers, filters and pooling layers. This approach may be suitable for simpler image classification tasks or datasets with less variation in features. We began experiments with no regularization just as before with the TNNs and compiling with the ADAM optimizer.



```
Model: "CNN1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_76 (Conv2D)          (None, 32, 32, 24)        1176

 max_pooling2d_76 (MaxPoolin  (None, 29, 29, 24)       0
 g2D)

 conv2d_77 (Conv2D)          (None, 29, 29, 48)        18480

 max_pooling2d_77 (MaxPoolin  (None, 14, 14, 48)       0
 g2D)

 flatten_24 (Flatten)        (None, 9408)              0

 dense_62 (Dense)            (None, 1024)              9634816

 dense_63 (Dense)            (None, 10)                10250

=================================================================
Total params: 9,664,722
Trainable params: 9,664,722
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.4: CNN1 Model Summary

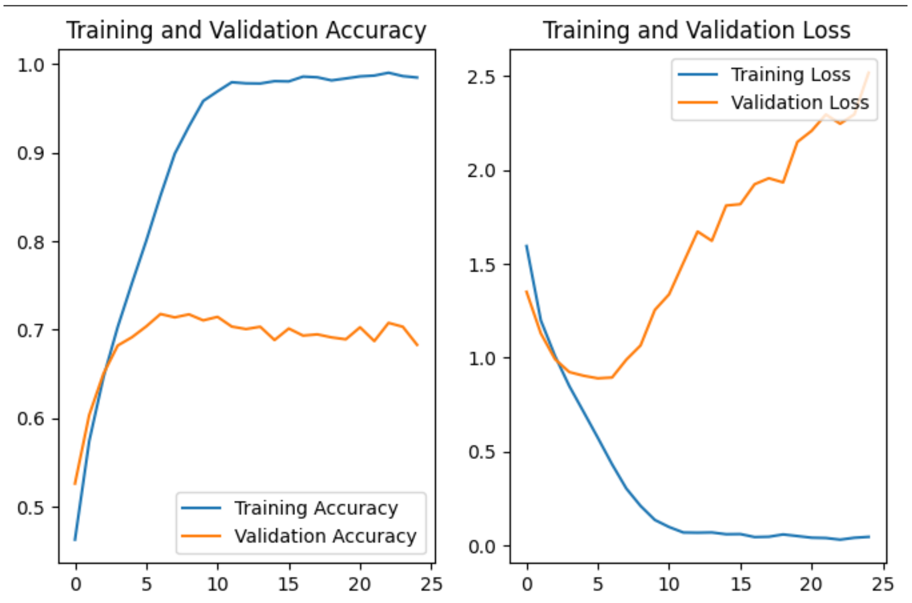


Figure 2.5: Visualization of CNN1 training

We see that already, the CNN achieves about 67.30% test accuracy with about 2x less parameters as the TNN. We notice maximum accuracy scores of about 71% during training with validation loss ¡ 0.9 in some epochs before overfitting kicks in. These values were not achieved by our most tweaked TNN.

However, the plots show that training and validation accuracy are off by large margins, as well as the training and validation losses. This is due to the overfitting that kicks in at about 5 epochs, which also causes model accuracy to drop.

- **CNN1 with Dropout Regularization**
  To reduce overfitting and the large margins of discrepancies between training and validation loss and accuracy, we also introduced dropout regularization to the network as we did with the TNNs.

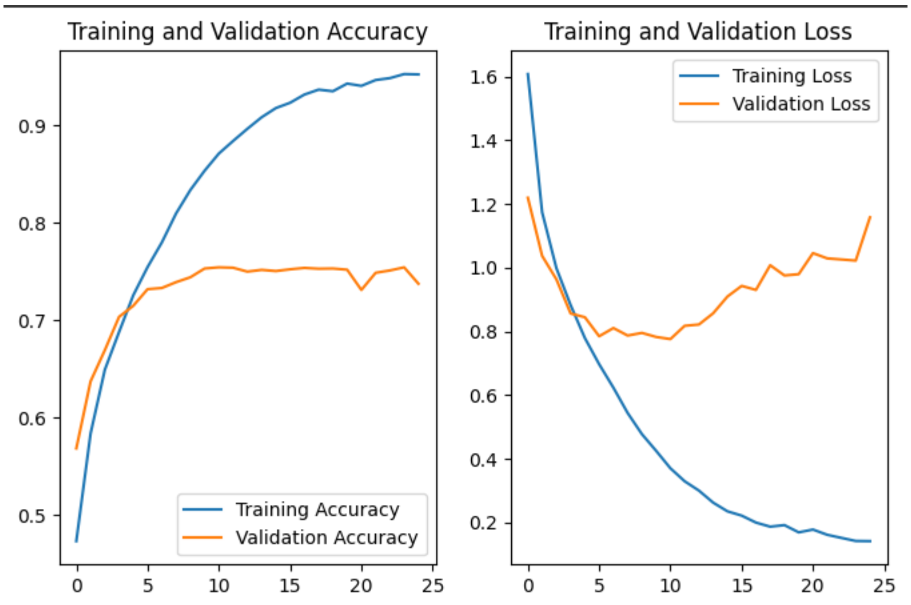


Figure 2.6: Visualization of CNN1 training with dropout

In doing so, we managed to retain most of the validation accuracy, attaining a test accuracy of about 73.87%. However, this architecture still peaked at this accuracy level and continued to overfit.

- **CNN1 with L2 Regularization**
  We set our regularization factor (a hyperparameter) to 0.00025 and applied L2 regularization to just 2 layers for now.

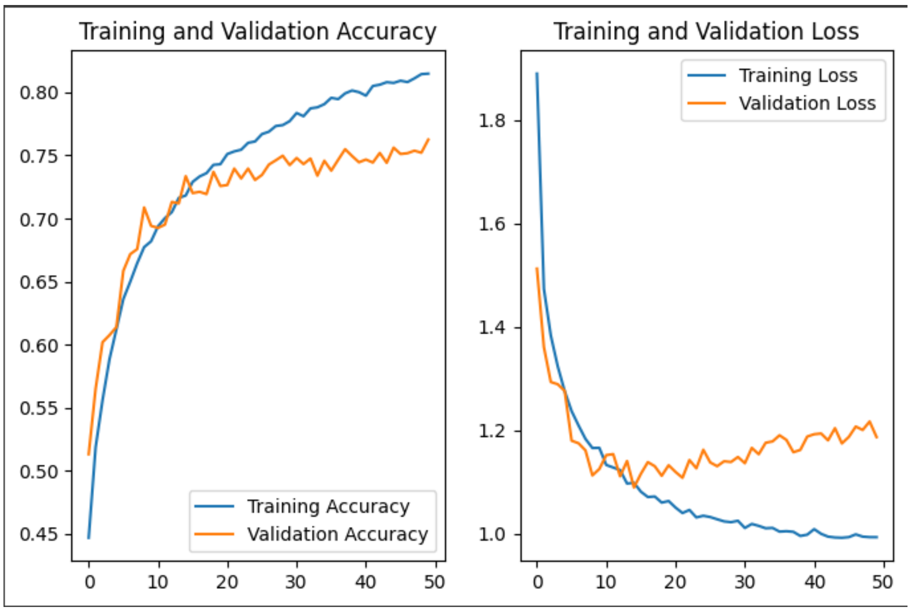
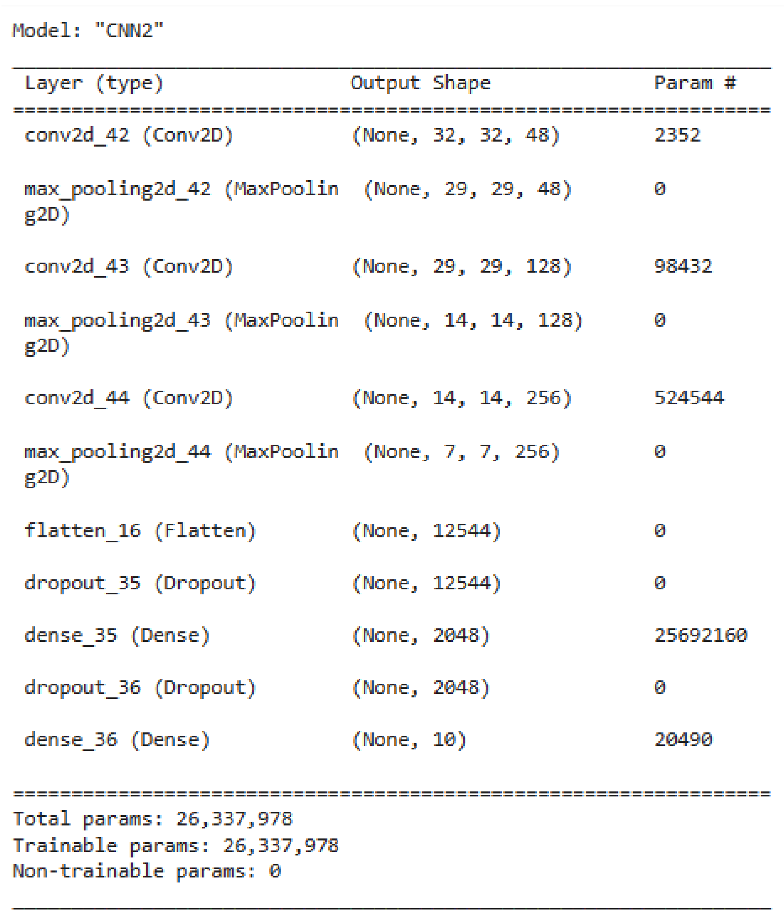


Figure 2.7: Visualization of CNN1 training with L2 regularization

This made training take a bit longer, that is, training accuracy did not quickly climb to over 90% as we saw previously and needed about 2x as much epochs to train. This is because the model was trying to minimize both the loss and the regularization term while learning smaller weights. Consequently, overfitting and the divergence in the performance measurement metrics was also reduced by a noticeable margin. This helped the model achieve a test accuracy of about 75.14%.

CNN2 : Added Convolutional & Pooling Layers

We have already seen the capabilities of CNNs in image classification from CNN1. Hence, in our next model, we aimed for higher accuracy by ramping up the convolutional and pooling layers, as well as the filters . This enables the network to capture more complex patterns and features from the input images. We believe this could be beneficial when dealing with more intricate image classification tasks or datasets with high variation in features such our chosen CIFAR-10.



```
Model: "CNN2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_42 (Conv2D)          (None, 32, 32, 48)        2352      
                                                                 
 max_pooling2d_42 (MaxPoolin  (None, 29, 29, 48)       0         
 g2D)                                                            
                                                                 
 conv2d_43 (Conv2D)          (None, 29, 29, 128)       98432     
                                                                 
 max_pooling2d_43 (MaxPoolin  (None, 14, 14, 128)      0         
 g2D)                                                            
                                                                 
 conv2d_44 (Conv2D)          (None, 14, 14, 256)       524544    
                                                                 
 max_pooling2d_44 (MaxPoolin  (None, 7, 7, 256)        0         
 g2D)                                                            
                                                                 
 flatten_16 (Flatten)        (None, 12544)             0         
                                                                 
 dropout_35 (Dropout)        (None, 12544)             0         
                                                                 
 dense_35 (Dense)            (None, 2048)              25692160  
                                                                 
 dropout_36 (Dropout)        (None, 2048)              0         
                                                                 
 dense_36 (Dense)            (None, 10)                20490     
                                                                 
=================================================================
Total params: 26,337,978
Trainable params: 26,337,978
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.8: CNN2 Model Summary

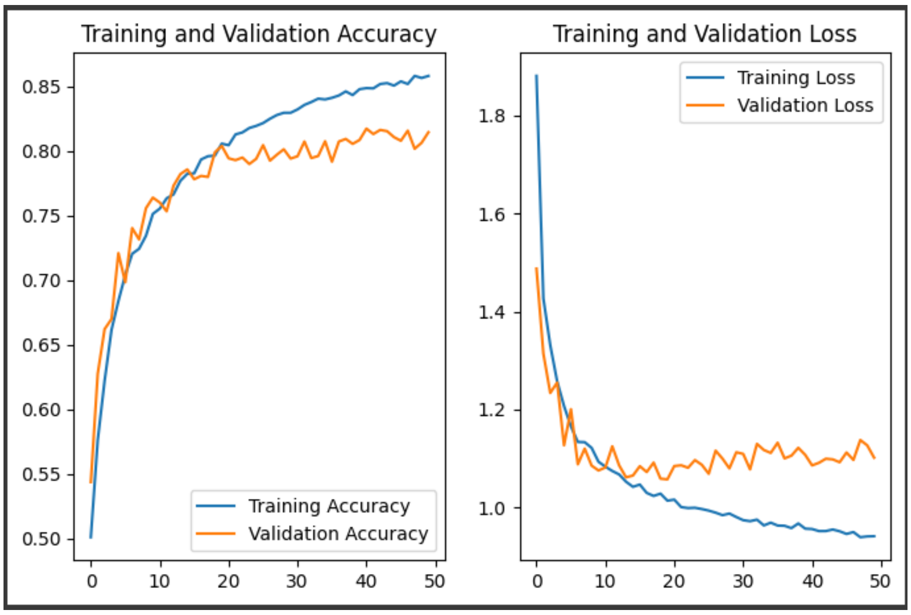


Figure 2.9: Visualization of CNN2 training

This time we set our regularization factor to 0.0005 and added l2 regularization in alternating layers in the network since it now has more layers. As expected, adding more complexity to the model improved its validation accuracy, we see it reaching 80% during training and a final test accuracy of 80.85% as well.

- **CNN2 with Lion Optimizer** We finally train CNN2 with the Lion optimizer.

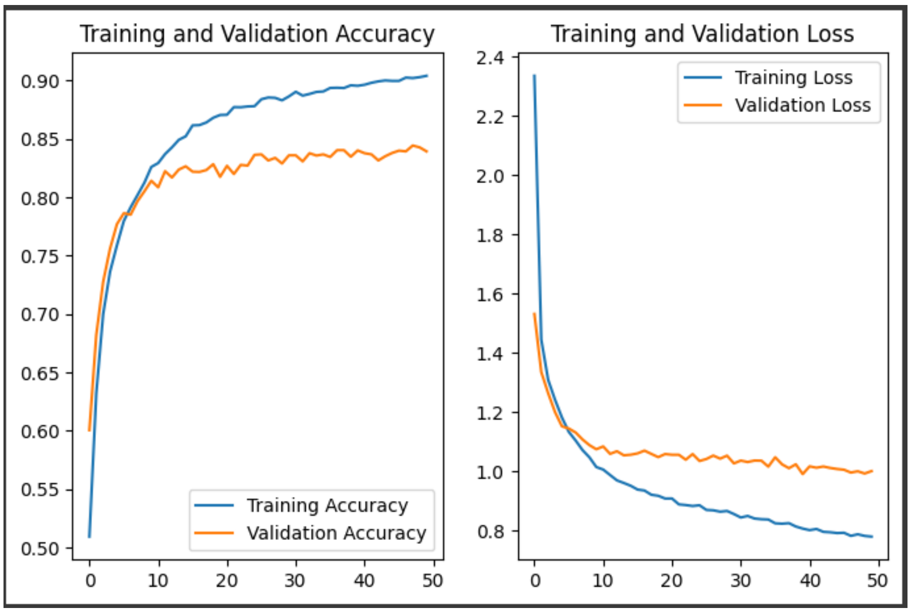


Figure 2.10: Visualization of CNN2 training with Lion optimizer

We notice how ”smoother” the curves have become, indicating that training was more stable with the Lion optimizer. As expected, it aided in increasing the accuracy of our model, helping attain a final test accuracy of 82.97%.

Chapter 3

Conclusion

We thus conclude our investigation into the performance of traditional neural networks (TNNs) and convolutional neural networks (CNNs) on the CIFAR-10 image classification dataset. We have experimented with different architectures, optimizers, and regularization techniques to improve the accuracy of our models.

From the above results, it appears that CNNs outperform TNNs on this image classification task. Granted, traditional neural networks (TNNs) are universal approximators with the ability to learn any function, however they have inherent shortcomings when it comes to image classification. Convolutional neural networks on the other hand are quite good at extracting local information from images. Using their small filters or kernels, they can scan the image and capture local patterns or features such as edges, corners, and blobs, making them particularly good for this task.

We also found that adding more layers and filters to our networks improved their performance and this was especially true for the CNNs. This is likely because as we add more layers, we also add more filters and pooling layers to the network, improving its ability to capture more complex patterns and features from the input images.

Additionally, we found that we can jointly leverage dropout and L2 regularization to minimize the discrepancy between training and validation loss and accuracy, effectively reducing overfitting and improving the accuracy of our models. Another finding we have made is that the choice of optimizer also plays a role stabilizing model training and also improving model accuracy.

References

[Chen *et al.* 2023] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. *Symbolic Discovery of Optimization Algorithms*, 2023.

[Ian *et al.* 2017] Goodfellow Ian, Bengio Yoshua, and Courville Aaron. *Deep learning: Adaptive computation and machine learning*, 2017.

[Krizhevsky and Hinton 2009] Alex Krizhevsky and Geoffrey Hinton. *CIFAR-10 and CIFAR-100*, 2009.

[tensorflow.org] tensorflow.org. *tf.keras.optimizers.Adam — TensorFlow v2.12.0*. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam. Accessed: [May 31, 2023].