Classification of Image Data with Multilayer Perceptrons and Convolutional Neural Networks

Hareth Hmoud, Haley He, Andres Diaz Lopez

March 9th, 2023

Abstract

A multilayer perceptron (MLP) is a type of neural network that consists of multiple layers of interconnected nodes, or neurons. Each neuron receives input from the neurons in the previous layer, then applies a nonlinear activation function to the input, and passes the result into the following neurons in the next layers in the perceptrons. The core of an MLP consists of an input layer, a minimum of one hidden layers, and an output layer.

The project task consisted of developing an MLP from scratch to be used to classify images of the CIFAR-10 dataset, which is a collection of 60,000 32x32 color images in 10 classes, The 10 classes are, in alphabetical order: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is divided into 50,000 training images and 10,000 testing images.

The important findings of our experiments were:

Following the implementation of our custom MLP, we experimented with convolutional neural networks (CNN) and used ResNet to perform image classifications, and compare them to the custom MLP. The findings were:

1 Introduction

Artificial neural networks (ANNs) have shown remarkable performance in various machine learning and computer vision tasks, including image classification. Among ANNs, multilayer perceptrons (MLPs) have been widely used for image classification. The CIFAR-10 dataset is a popular benchmark dataset for image classification that contains 60,000 32x32 color images in 10 different classes. The objective of this mini-project is to gain practical experience in implementing a basic MLP and its training algorithm from scratch, and to explore the use of pre-trained convolutional neural networks (CNNs) for image classification on the CIFAR-10 dataset.

The CIFAR-10 dataset has been extensively used in the field of machine learning for evaluating the performance of deep learning models. Previous studies have shown that deep CNN models can achieve remarkable accuracy on the CIFAR-10 dataset. For instance, Krizhevsky et al. (2012) achieved a top-1 error rate of 37.5% and a top-5 error rate of 17.0% on the CIFAR-10 test set using a CNN model with multiple convolutional and fully connected layers. Similarly, Simonyan and Zisserman (2015) achieved a test error rate of 6.71% using a very deep CNN model with 19 layers.

The mini-project includes the implementation of an MLP model with zero, one, and two hidden layers using Python and the NumPy library, and the use of the backpropagation algorithm to train the MLP model on the CIFAR-10 dataset. INSERT OUR FINDINGS HERE

The mini-project also includes the use of pre-trained CNN models, specifically ResNet-50, which were pre-trained on the large ImageNet dataset and fine-tuned on the CIFAR-10 dataset using the Keras library.

2 Datasets

The CIFAR-10 dataset is a well-known benchmark dataset for image classification tasks in the field of computer vision and machine learning. It consists of 60,000 32x32 color images in 10 different classes, with each class having 6,000 images. The 10 classes are airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into a training set of 50,000 images and a test set of 10,000 images.

In this project, we used the CIFAR-10 dataset for training and evaluating our MLP and CNN models. We conducted exploratory analysis on the dataset to understand its characteristics and distribution. Our analysis showed that the classes in the CIFAR-10 dataset are well-balanced, with each class containing the same number of images.

The exploratory analysis in this project refers to the process of analyzing and visualizing the CIFAR-10 dataset to gain insights into its characteristics and distribution. This analysis is an important step before training any machine learning models, as it helps to understand the data and prepare it for training.

The exploratory analysis included several steps, such as analyzing the class distribution and visualizing sample images from each class. These steps were conducted to gain insights into the data and determine the appropriate pre-processing steps required for training the models.

Our code pre-processes the data by converting the pixel values to float64 data type and normalizing them by dividing by 255, which scales the pixel values to a range between 0 and 1. This normalization step ensures that the pixel values are within a similar range, which can help in improving the performance of the machine learning models.

3 Results

Different MLP models were experimented on. They all used a final activation function (softmax) and were optimized using gradient descent. The hyper-parameters (learning rate and max iterations) were not tuned due to time constraints and high acquisition times. Thus the models' accuracy may be improved. The default learning rate id 0.1 was used for most models, and 300 max iterations were run. The first experiment explored the effects of depth, using ReLu activation. The model with 0 hidden layers had and accuracy of 0.23, 1 hidden layer (0.22), and 2 hidden layers (0.26). The 1 hidden layer model had a learning rate of 0.01, due to a diverging gradient. Subsequently, the effects of the activation function was explored on the 2 hidden layer model. The model with a hyperbolic tangent activation had an accuracy of 0.28 and leaky-ReLu had 0.24 accuracy. Normalization techniques were also explored. The model with L1 norm. had an accuracy of 0.29, L2 norm had 0.21, and the model with y label smoothing had 0.17. The y label smoothing model was trained with y dummy matrices with values of 0.1/3 instead of 0, and 0.9 instead of 1. Finally an experiment with non-normalized data was run on the 2 hidden layer ReLu model. A learning rate of 0.001 was used (divergence at lower learning rates occurred), resulting in an accuracy of 0.37.

When using TensorFlow to create a CNN with two convolutional layers and two fully connected layers, with 256 hidden units and ReLU activation, there was a marked increase in accuracy as compared to our MLP with two hidden layers, which had an accuracy of 26%, while the CNN had a training accuracy of 99.6%. However, the validation accuracy is not as good as the training accuracy, at 65.1%. These numbers can be seen in figure 1 below.

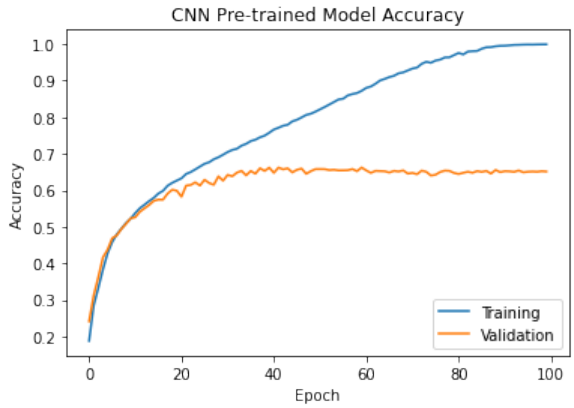

Figure 1: Training and Validation Accuracy of the CNN Pre-Trained Model

As seen in the figure, the training accuracy consistently increased, until it reached a peak of 99.6% at the 100th epoch, while the validation accuracy plateaued much earlier, at around the 25th epoch, only marginally improving after every epoch.

As for the loss, the average loss of the training set was very low, at 0.0122 after the 100th epoch. However, the average loss of the validation set was not as good, at 2.4147. The implications of this is discussed in the discussion section. The graph of the change in both types of loss over the 100 epochs is seen in Figure 2.

As seen in the figure, the loss for the training set decreased at a very rapid rate, reaching almost 0 after the 100th epoch, while the loss for the validation set increased at a rapid rate, reaching 2.4147

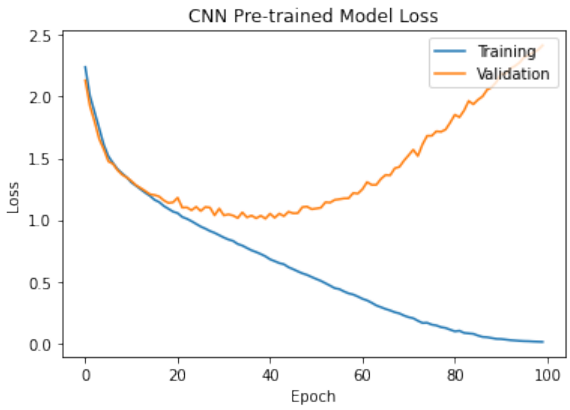

Figure 2: Training and Validation Loss of the CNN Pre-Trained Model

by the 100th epoch. It also did not show any sign of plateauing, implying more epochs would have meant more loss.

When we used a fine-tuned ResNet50 model on the dataset, made the following observations. The training accuracy after five epochs was 59.670%. The validation accuracy was 59.62%. This can be seen in Figure 3 below.

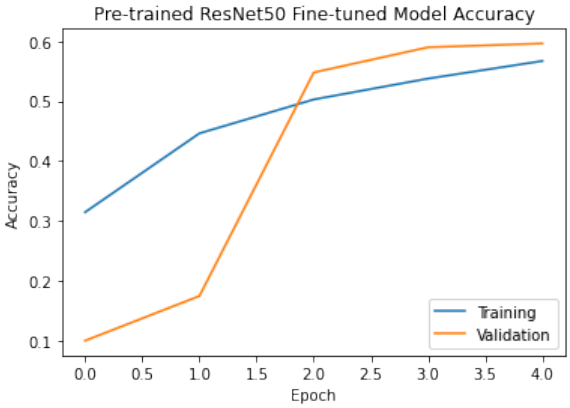

Figure 3: Training and Validation Accuracy of the ResNet50 model

As seen in the figure, the validation accuracy had almost exponential growth after the first epoch, reaching around 55% after only the second epoch, before increasing and then slightly plateauing after the third epoch. Meanwhile the training accuracy had its fastest increase after the first epoch, before continuing to increase, albeit at a slower rate, after the first epoch.

Meanwhile the average loss for the training sets and validation sets were very low, at 1.2479 and 1.1713. These figures can be seen in Figure 4 below.

As seen in the figure, the loss for the training set was already low and stayed around the same value for all of the epochs, meanwhile the loss for the validation set fell dramatically after the first epoch before plateauing for the rest of the epochs.

As for our MLP model, given limited computational capabilities, we only ran models at 5 specific maximum iterations to test the accuracy of our model as a function of epochs, we used a random specific seed to make sure the initial weights were consistent across the models. As with the rest of the experiments, the accuracy never got to be at expected values and even reduced after 30 epochs which shows that additional iterations does not improve the model by a lot.

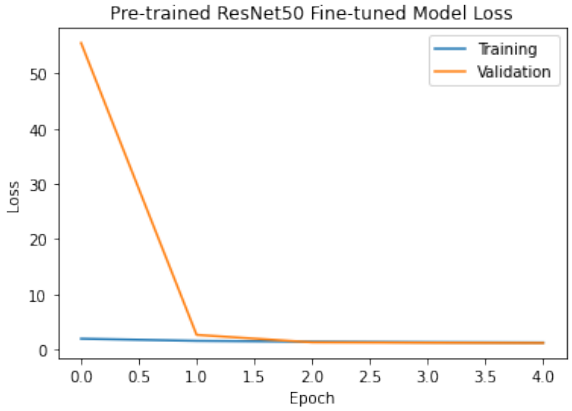

Figure 4: Training and Validation Loss of the ResNet50 model

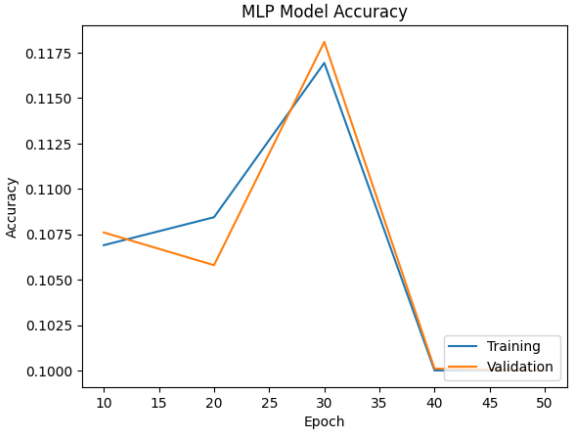

Figure 5: Training and Validation Accuracy of the MLP model

4 Discussion and Conclusion

Our MLP model had lots of weaknesses, as can be seen by the accuracy. The model with 0 hidden layers had and accuracy of 0.23, 1 hidden layer (0.22), and 2 hidden layers (0.26). The implications of the experiments on our MLP are several. Firstly, the addition of hidden layers improved the model's accuracy, with the 2 hidden layer model outperforming the 0 and 1 hidden layer models. Secondly, the choice of activation function had an impact on model performance, with hyperbolic tangent outperforming leaky-ReLu in the 2 hidden layer model. Thirdly, normalization techniques had varying impacts on model performance, with L1 norm and y label smoothing improving accuracy, while L2 norm had a negative impact. Fourthly, the learning rate had a significant impact on model performance, with a higher learning rate resulting in divergence for the 1 hidden layer model and a lower learning rate resulting in improved accuracy for the non-normalized 2 hidden layer ReLu model. Finally, the best performing model was the 2 hidden layer model with hyperbolic tangent activation and L1 normalization, which achieved an accuracy of 0.29. These results highlight the importance of careful experimentation with model architecture, activation functions, normalization techniques, and learning rates to achieve optimal performance in an MLP. However, it is important to note that these findings are specific to the dataset and problem at hand, and may not generalize to other datasets or problems.

The experiments suggest that the performance of the MLP can be improved by adjusting several key factors, including the number of hidden layers, the choice of activation function, the use of normalization techniques, and the learning rate. Adding more hidden layers generally improved the model's performance, as it allowed for more complex representations of the data. The choice of activation function also played a significant role, with hyperbolic tangent performing better than leaky-ReLu in the 2 hidden layer model. Normalization techniques also had varying impacts on model performance,

with L1 norm and y label smoothing improving accuracy. The learning rate was found to be crucial, with a higher rate causing divergence for the 1 hidden layer model, and a lower rate improving accuracy for the non-normalized 2 hidden layer ReLu model.

In order to improve our model, we need to be more precise and accurate with our choice of hyperparameters. To better choose hyperparameters and address the problems observed in the experiments, several strategies can be employed. While there is no ideal methodology, grid search might be the best given the right resources and time, as while it is computationally expensive, it searches through all the possible combinations of hyperparameters. Random search is a more efficient alternative to grid search that can often achieve similar or better performance, but it is still subject to the limitations of the search space and may not find the global optimum.

There are several takeaways from the performance of the CNN. The training accuracy is much higher than the validation accuracy, which implies that the model is overfitting to the data. So when presented with new data, it is possible that the model will not perform well, as it is "memorizing" the training data. It could be just learning specific details of the image it's being presented.

The large difference in the training loss and the validation loss is further evidence to the theory that the model is overfitting to the data. The training loss decreases significantly as the model learns to fit the training data very well, but the validation loss does not improve at the same rate.

To remedy these problems, a regularization technique could be used, such as early stopping.

Moreover, there are also takeaways from the performance of the ResNet50 model. First, the fact that the validation accuracy is not much lower than the training accuracy suggests that the model is not overfitting to the training data, and is generalizing reasonably well to unseen data. However, both the training and validation accuracies are still relatively low, but this can be remedied by including more epochs. The reason only five epochs were performed was because each epoch took approximately an hour to run, and therefore increasing the number of epochs would have taken too long, which is definitely a limitation in our task.

The training and validation loss being relatively low and similar indicates that the model was not overfitting to the training data, which is a testament to the model, and shows that of the three models, it was the best. However, it is important to keep in mind that a low loss does not necessarily mean that the model is accurate, as the loss function may not perfectly reflect the desired performance metric. But, looking at the trend of the values of the loss, as the number of epochs increased, they seemed to plateau, which might mean that as more epochs are performed, the values for the loss will stay at around the same level, and therefore could indicate that the model has reached its optimal performance and that additional epochs may not lead to significant improvements in the model's accuracy.

To improve the performance and able to perform more epochs in reasonable time, we could use a different optimizer. We could also reduce the learning rate, which could help the model converge faster. Also the batch size could have been reduced. The default batch size for CIFAR-10 in Keras is 32, but we could use a smaller batch size, such as 16 or 8, to reduce the memory requirements during training and potentially speed up each epoch. We could also use the GPU to run the training instead of the CPU.

5 Statement of Contribution

Hareth was responsible for pre-processing the data, for creating the CNN and running it on the dataset, and for using ResNet with the project specifications and running it on the dataset.

Haley was also responsible for the pre-processing the data, and for implementing the MLP and running it on the dataset.

Andres was also responsible for pre-processing the data, and also for implementing the MLP and running it on the dataset.