COMP 551 Applied Machine Learning
Mini-Project 2 - Classification of Image Data

Eren Ozturk
Department of Electrical and Computer Science Engineering McGill University

Ananthalekshmy Ambily
Department of Electrical and Computer Science Engineering McGill University

Nonlidji Merias Finoude
Department of Electrical and Computer Science Engineering McGill University

Abstract—**In this mini-project, we implemented a multi-layer perceptron model to categorize images into several classes. The CIFAR-10 dataset, which consists of 60000 tiny colour images categorized into 10 classes, was utilized for this work. We used multilayer perceptron models to perform categorization. We examined and contrasted the test and train performances of those two algorithms as a function of the epochs, batch sizes, and several other relevant parameters. We also looked at how the Multilayer Perceptron behaved when certain factors, such the number of layers, the number of units, and the kind of activation, were altered. A detailed analysis of the findingsare mentioned in this report.**

1. Introduction

Mathematical nonlinear solutions are used to handle classification issues in a variety of applications, including weather forecasting, speech recognition, stock market prediction, etc. For many purposes, including automatic image classification in social media and search engines, image tagging, etc., image classification is quite helpful. Categorization will be useful for the effective and improved analysis of surrounding scenes. It becomes quite difficult to identify the object in an image if there are issues with noise and low quality. To achieve more precise categorization outcomes, researchers use novel classification techniques [1]. A complicated form of the perceptron algorithm is the multilayer perceptron, which stacks layers of units, or neurons, each of which executes the perceptron algorithm. The input layer receives data and processes it before sending it to the other hidden layers. The output layer, which has exactly one neuron for each class and receives the input after it has passed through all the hidden layers, will produce a score that indicates how likely it is that the data belongs to that particular class. The class with the highest production is then chosen as the final classification. Each neuron in the most basic type of network is fully connected to every other neuron in the layer above; this is known as a dense layer. Each neuron in this scenario does a linear combination of the data from the preceding layer, multiplying each value for weight, and then adds a bias. After that, a nonlinear activation function is used, and the outcome is passed on to the following layer. We tried some of the more well-liked solutions out of the numerous ones that could be used for this activation function [3]. Prior to training the network, we first determine the discrepancy between the network's prediction and the actual label for each sample. The chain rule for partial derivatives is then used to determine the gradient that must be applied to the weights in order to minimize that error, and this error is then propagated back through the network, known as back-propagation.

2. Dataset

The CIFAR-10 (Canadian Institute for Advanced Research 10) dataset is a collection of 60,000 32x32 RGB images in 10 classes. Each class contains 6,000 images, out of which 5,000 are used for training and 1,000 for testing. The 10 classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The images in the dataset are of relatively low resolution (32x32 pixels) and contain significant variations in orientation, position, and lighting. CIFAR-10 has become a popular benchmark dataset for evaluating the performance of deep learning models in image classification. The 32x32 pixel images in the CIFAR-10 dataset are coloured and have RGB (Red, Green, and Blue) values assigned to each pixel. We require a 32x32x3 multidimensional array to reflect it.

3. Results

The CIFAR10 dataset was loaded and so as to perform the normalization, the pixel values of the images were transformed to have zero mean and unit variance. This was accomplished by dividing each pixel by the standard deviation of the pixel values after deducting the mean pixel value of the total dataset from each pixel. By guaranteeing that the input features have comparable scales and are centred around zero, normalization contributes to the stability and efficiency of the learning process. It also improve the generalization performance of the model by preventing overfitting. In [2], a similar approach was taken to overcome the gradient disappearance and overfitting phenomena, and has high stability and image classification accuracy. It provides

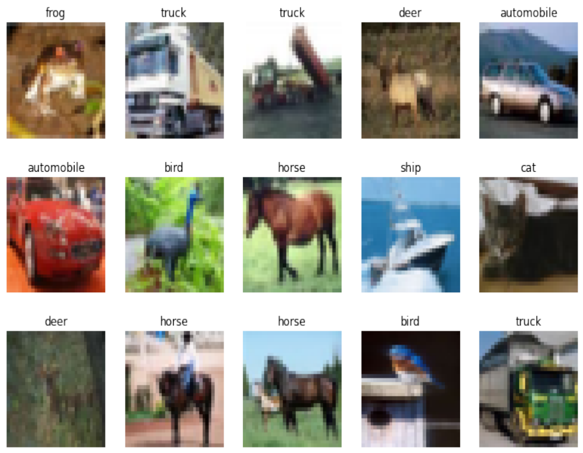
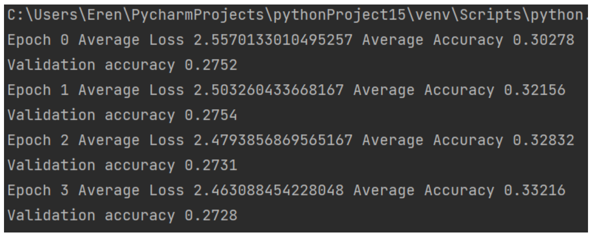


Figure 1. Images from CIFAR10 Dataset

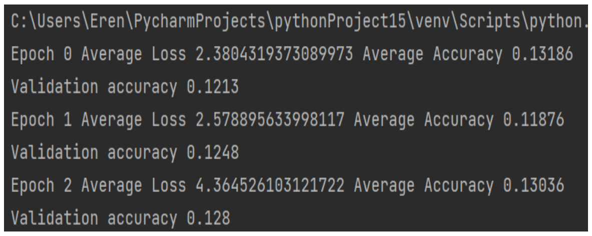
a novel perspective on image classification methods for lightweight convolutional neural networks.

For the implementation of Multi-Layer Perceptron, we defined the fit and predict function and the models were incorporated with gradient descent and stochastic gradient descent algorithms for optimal performance. Furthermore, the non-linearity transformations considered the following functions, namely, sigmoid, Relu, softmax, sigmoid derivative, softmax derivative. For the accuracy score estimation based on the true and target label, the function evaluate acc gave the outputs depending on the number of hidden layers and is shown below.



```
C:\Users\Eren\PycharmProjects\pythonProject15\venv\Scripts\python.
Epoch 0 Average Loss 2.5570133010495257 Average Accuracy 0.30278
Validation accuracy 0.2752
Epoch 1 Average Loss 2.503260433668167 Average Accuracy 0.32156
Validation accuracy 0.2754
Epoch 2 Average Loss 2.4793856869565167 Average Accuracy 0.32832
Validation accuracy 0.2731
Epoch 3 Average Loss 2.463088454228048 Average Accuracy 0.33216
Validation accuracy 0.2728
```

Figure 2. Accuracy score - Single layer



```
C:\Users\Eren\PycharmProjects\pythonProject15\venv\Scripts\python.
Epoch 0 Average Loss 2.3804319373089973 Average Accuracy 0.13186
Validation accuracy 0.1213
Epoch 1 Average Loss 2.578895633998117 Average Accuracy 0.11876
Validation accuracy 0.1248
Epoch 2 Average Loss 4.364526103121722 Average Accuracy 0.13036
Validation accuracy 0.128
```

Figure 3. Accuracy score - Hidden layer

The performance of the test data and the training data were analysed under different conditions of hyperparameters - batch size, kernel size, learning rate, filter size and pool size. The variations were as expected and the large data size did have a significant impact under each of these situations. These are shown in figures 3-10.

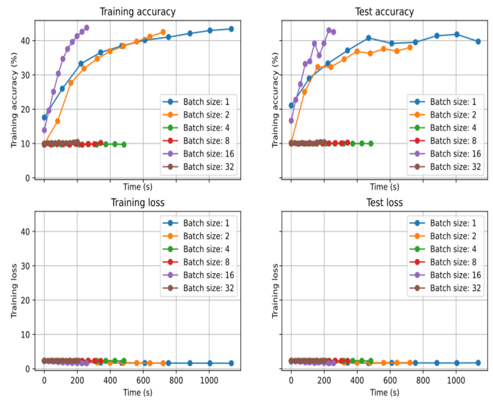


Figure 4. Effect of batch Size

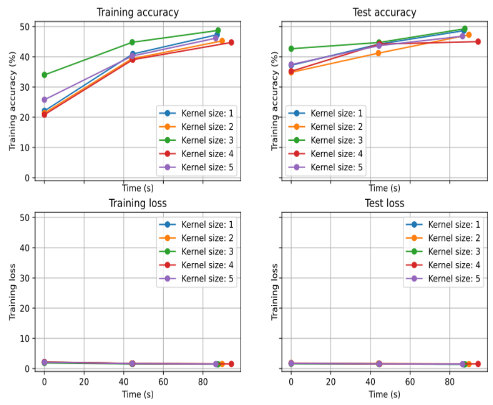


Figure 5. Effect of kernel Size

The three functions namely, initializeSimple, initializeOneLayer, initializeTwoLayers focused on implementing the third task of the project and the different test cases were run to determine the performance and its variation from the expected parameters. Based on the pre-trained model and the depth and width parameters being updated, the model showed an optimal performance with the ideal parameters being – Depth: 3, Width: 128, Batch size: 6, and Regularization/Dropout: None. It has to be noted that the model showed quite efficient performance with the testing data and came up with an accuracy approximately equal to 61%. The performance variation and gradually attaining the ideal state is depicted below from figure 11- 18. Figure 18 depicts the optimal performance of the model.

4. Discussion and Conclusion

In these experiments, we have attempted to implement a multi-layer perceptron, and have met some timing chal-

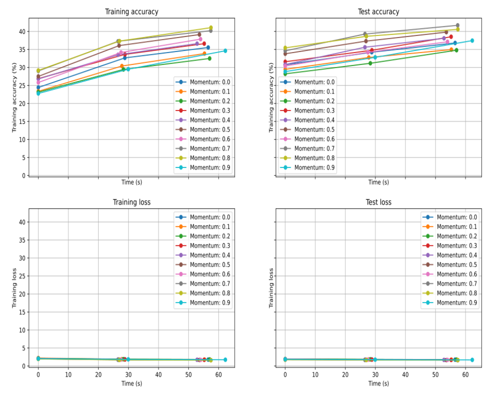


Figure 6. Effect of Momentum

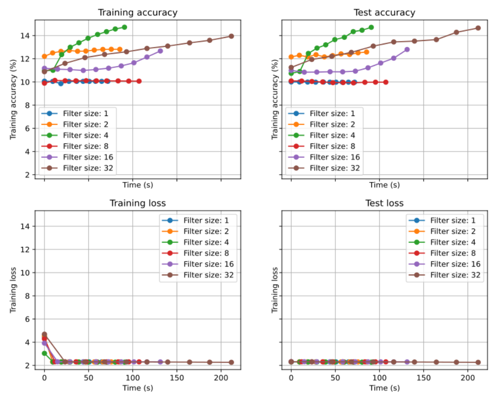


Figure 8. Effect of Filter size

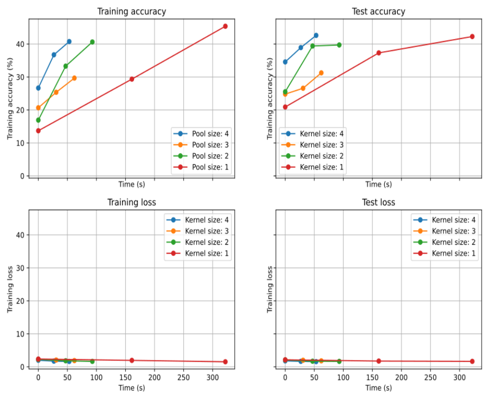


Figure 7. Effect of Pool size

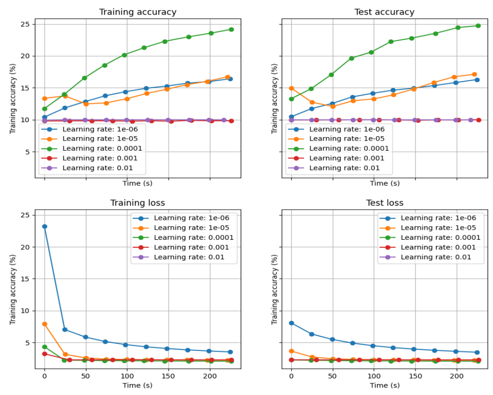


Figure 9. Effect of Learning rate

lenges. We were able to implement the simple 3072 - 10 topology as well as the 3072 - 256 - 10. Interestingly, the simpler topology has performed better in training and test, reaching about 27% test accuracy while the second model reached only 13% suggesting that there was a problem with the implementation that we could not attend to in time.

As for the models implemented using TensorFlow, our success rate was higher. Using the prebuilt model as the convolutional layers, we were able to reach a testing accuracy of 60-61% with a training runtime of 125 seconds on Colab's GPU's. We have attained a similar performance with the model that uses our own convolutional layers. Our model is faster, however, with a similar 2-3 minute runtime but on a low-end laptop instead of a Colab GPU.

We have found that both of these TensorFlow models are subject to similar tradeoffs in terms of training speed and testing accuracy. Overall, higher batch sizes confer a computational time advantage, at the cost of test accuracy. However, there are diminishing returns with reducing batch size. Reducing the batch size after a certain point yield the same accuracy at higher computational cost. For the convolutional net without pretrained layers, we have found that reducing the batch size from 16 to 8 only increases the computation time.

Depth of the network also increases performance, but that too has a point of diminishing returns. For the pretrained net, we have found that there is no advantage in speed or performance beyond 3 layers.

We have not observed a bound in the increase in training accuracy as a function of network hidden layer width. However, the test accuracy plateaus at around 61%, suggesting that higher widths lead to overfitting. However, there is a speed advantage to having a wider network, which also diminishes as the net widens. We have observed no extra benefit beyond 128-wide hidden layers on top of the pretrained convolutional set.

Learning rate has quite a binary effect on performance. If the learning rate is too high, the network will fail to converge. If it is too slow, then it will take too long to converge. We have experimented with different rates to find the largest rate that did not lead to divergence. One interesting avenue would be to find neural net properties

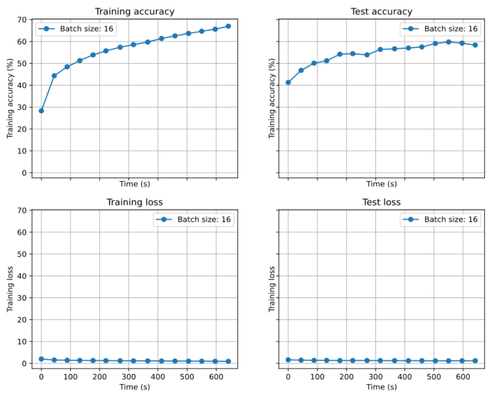


Figure 10. Performance with filter size-32, batch size-16 and iter-15

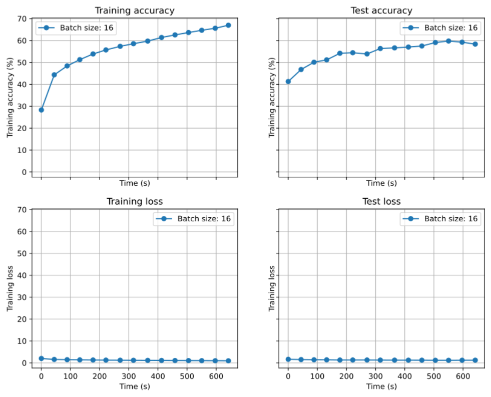


Figure 11. Performance with filter size-32, batch size-16 and iter-15

that affect the learning rate threshold.

For our own convolutional net, we have found that a kernel size of 3x3 leads to the best test accuracy, at no extra runtime cost. This is likely due to the fact that the image dataset has meaningful features at a 3x3 scope.

In terms of pool sizes, our net reaches the same level of accuracy at any pool size, but a pool size of 4x4 reaches that point at almost 1/6th the runtime without pooling.

Larger filters improve convergence and accuracy, at the cost of training time. We were curious if a larger filter size can sustain a higher batch size, resulting in a convergence that is both quick and accurate. When we tried a filter size of 32 and a batch size of 16, with all the other parameters being set to their optimum values, we were able to converge to a 62% test accuracy in around 200-300 seconds on a low-end laptop.

Overall, this mini-project was very instructive on the different tradeoffs of neural net topologies and hyperparameters. We have gained experience with using machine learning libraries to generate models and optimizing them.

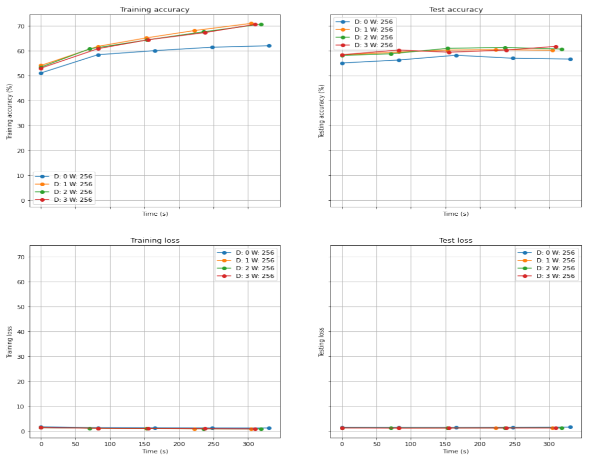


Figure 12. Performance with lower depth

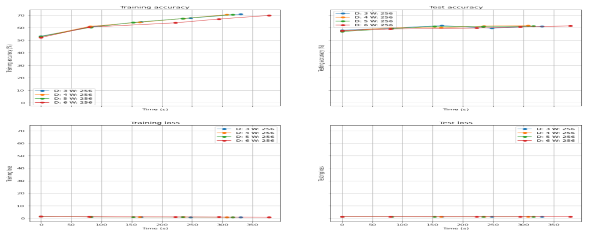


Figure 13. Performance with higher depth

5. Statement of Contributions

For this laboratory, the initial task division was such that Finoude would implement the multilayer perceptron, Ozturk would construct and optimize the convolutional networks using TensorFlow. The two would then merge their work and Ozturk would add the multilayer perceptron to the comparisons. Ambily would draft the report and place the figures in the relevant LaTeXformatting. However, with the limited workable implementation of mlp model, the work has to be repeated by the rest of the team members at the last moment which delayed the submission.

References

[1] Akshaya, B., and M. T. Kala. ”Convolutional neural network based image classification and new class detection.” 2020 International Conference on Power, Instrumentation, Control and Computing (PICC). IEEE, 2020.

[2] Ouyang, Wanqi, and Pan Zhu. ”A Lightweight Convolutional Neural Network Method for Image Classification.” 2022 2nd International Conference on Frontiers of Electronics, Information and Computation Technologies (ICFEICT). IEEE, 2022.

[3] Mahajan, Arpana, and Sanjay Chaudhary. ”Categorical image classification based on representational deep network (RESNET).” 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA). IEEE, 2019.

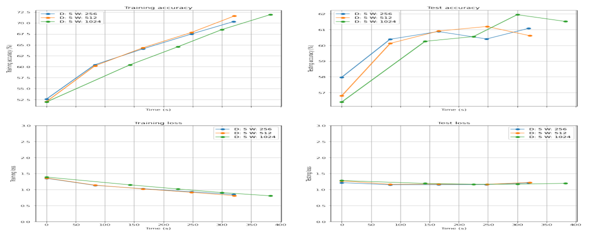


Figure 14. Performance with higher width

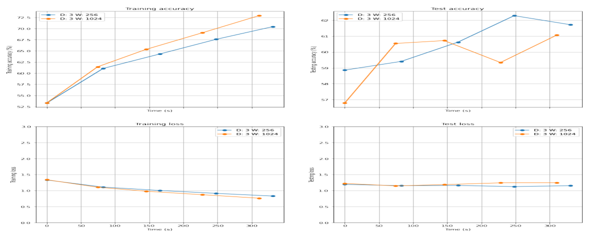


Figure 15. D=3, W=256,1024

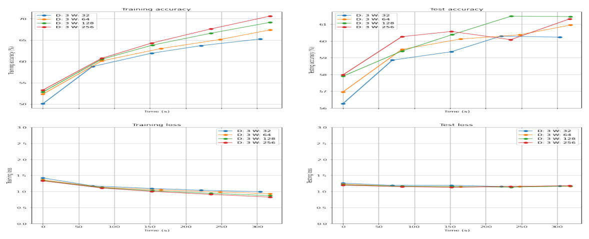


Figure 16. Performance under different widths

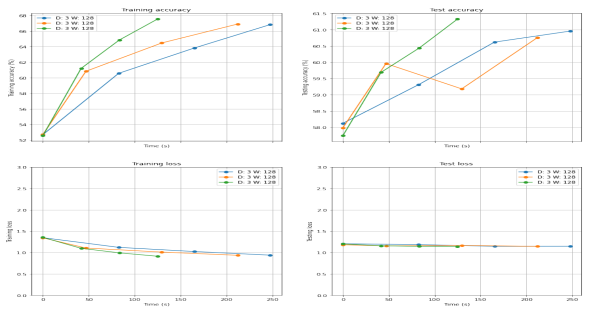


Figure 19. Ideal Performance - Batch size=64,D=3,W=128

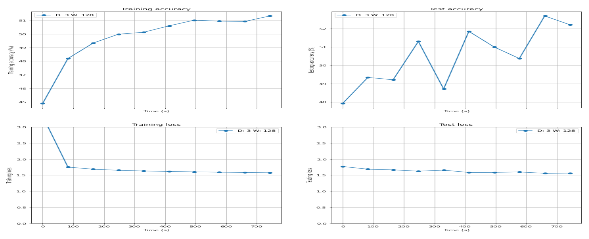


Figure 17. Performance with normalization=0.1

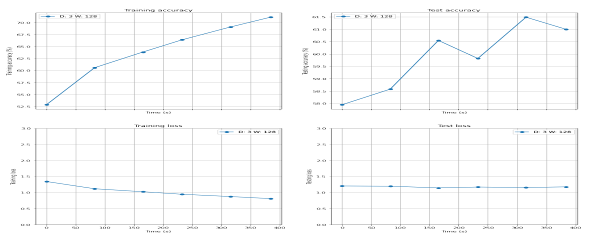


Figure 18. Performance with normalization=0.01