COMP 551 - Miniproject #2 Report

Perceptrons and Covolutional Neural Networks

Maxim Boucher - 260902501
Allison Mazurek - 260895254
Youssef Samaan - 261032708

***Abstract*—The aim of this project is to classify image data from the CIFAR-10 dataset using three approaches, namely Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), and pre-trained CNNs. This report compares various models in terms of accuracy and performance in order to compare the evolution of Machine Learning techniques and in order to better understand the power of pre-trained CNN models for use in Computer Vision applications. The models' hyper-parameters, which include learning rate, depth, and activation function type, are all evaluated with regard to the accuracy of the model in order to justify which values are the most optimal. The best results obtained with a CNN model resulted in a training and testing accuracy of 98.3% and 60.95%, respectively, while the best MLP model scored 60.06% and 43.28%, respectively. The CNN models took multiple hours of computation, including a pre-trained ResNet50 model, which resulted in very similar train and test accuracies, both of which were nearly 40%.**

I. Introduction

This project focuses on using artificial neural network models - Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) - to train and subsequently classify images of the CIFAR-10. Dataset into 10 classes. As per the project requirements, the MLP model is implemented using NumPy and backpropagation with the mini-batch gradient descent in order to find the optimal weights within a computationally acceptable time period. The TensorFlow library was used to implement the CNN models as well as to adapt the pre-existing CNN model, ResNet50. The data was loaded into the Jupyter Notebook using the Pickle library and was processed appropriately using NumPy. Normalized greyscale images were used as input to the MLP and CNN models, which gives a robust comparison between the neural networks. In order to optimize the computation period of the pre-trained CNN model and to allow a faster convergence, we increased the learning rate and added the option to specify batch sizes at the expense of a greater model loss. The MLP model that was used for this project split the input data into 200 mini-batches to train the neural network, training 1 batch every iteration. The models that were trained ran for 10000 iterations resulting in a total of 50 epochs.

II. Related Work

In this project, neural networks are used for image classification. Alex Krizhevsky and colleagues from the University of Toronto trained a deep convolutional neural network to classify 1.2 million high-resolution images [1]. The model, famously known as AlexNet, dominated the industry's lead competitor at the time by a whooping 9.8% in the ImageNet-2012 competition. This project uses the CIFAR-10 dataset, used by Alex Krizhevsky in one of his first papers [2], to study and learn pre-processing filters for input data in order to enhance the accuracy of image classification models. The CIFAR-10 training and testing dataset have been studied tremendously in the years since. Currently, on the *Image Classification on CIFAR-10* Leaderboard [3], the dataset has a testing accuracy of 99.5% using 632M parameters [4].

III. Dataset Description and Preprocessing

The CIFAR-10 dataset contains a total of 60000 color images, all having a resolution of 32 x 32 pixels. The images are evenly split into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. A 5:1 subset was taken for training and testing datasets, respectively, as suggested on the CIFAR-10 website. The test set contains 1000 images from each class.

The Pickle library was used to *unpickle* (extract) images, initially stored as arrays of length 3072 containing the 1024 red, green, and blue pixel channel values, respectively. These were vectorized into a more common image representation by grouping the RGB channels into arrays of size 1024x3 resulting in image data that could be easily displayed and processed. The MLP input layer is designed to take a one-dimensional vector as input. The color images were converted into grayscale in order to reduce the dimensionality of the input data. This comes at the expense of losing some information, such as color, which adds more features

and data to be analyzed. These images were normalized to a range of [0,1]. However, the raw data arrays of length 3072 are still able to be inputted into the MLPs; Multilayer Perceptron model input is invariant to the order of pixels inputted into the model and will produce a similar cost. However, the vectorized greyscale images were chosen as input in order to reduce computation time as well as to keep the input in a similar shape to those required for the CNNs, assuring a sound comparison between models. 2D Convolutional layers are used in the CNNs exhibited in this report, and as such, the greyscale images were resized appropriately for this task. The pre-trained ResNet50 architecture has been trained using color images and requires three-dimensional images as input to the network. Normalized RGB images were inputted into the model for better fairness across the models.

IV. Multilayer Perceptron Model and Implementation

A Multilayer Perceptron (MLP) model is a deep artificial neural network composed of perceptrons arranged in an input layer to receive the input data matrix (X), an output layer to describe the probability of classification for input, and an arbitrary number of fully-connected hidden layers of arbitrary width to process the data through the perceptrons. Each perceptron (the basic unit for MLPs) classifies a feature vector input x through the multiplication of a set of weights w, in addition to a bias b, and the application of a non-linear activation function ϕ:

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b) = \phi(w^T x + b) \tag{1}$$

From the weight scaling, bias offset, and applied non-linearity, theoretically, any continuous function could be modeled with an arbitrary accuracy if given a large enough depth. This theorem is known as the *Universality of MLP* [5]. The test accuracies for a 0,1,2 ReLU hidden layer trained network were found to be 28.42%, 41.89%, and 43.28%, respectively. The activation function, depth, and width applied to the model are design choices. However, the optimal value of weights and biases are learned from the training data using the MLP model. The algorithms for this learning process are backpropagation and gradient descent, both of which are implemented and described as follows:

a) Backpropagation computes the gradient of a loss function with respect to each weight using the chain rule. The gradient is computed on each layer, iterating backward in order to reuse the previously computed derivatives.

b) Minibatch Gradient Decent splits the training set into small batches to calculate errors and update the model weights per iteration. The time complexity of an MLP network is substantial as the model is fully connected and needs to compute many matrix multiplications. Mini-batch gradient descent improved time efficiency while training as it sums and updates a smaller subset of the weights as follows: $w = w - \alpha \Delta_w J(x^{i:i+b}, y^{i:i+b})$ where i is the number of iteration modulo 200, and b is the batch-size (250 training set). The model was trained on the full dataset multiple times.

V. Results

The naming convention of the models used below is in terms of the number of hidden layers (all using 256 units unless stated otherwise) with activation functions also listed, for example, TwoHidden(Leaky). If no activation function is stated, assume the model uses ReLU activation for all layers. As the task at hand is multiclass classification, the SoftMax function is used for the last activation layer. Figure 2 illustrates the training and test accuracies obtained from the MLPs with a learning rate of 0.0001 and are further listed in Table I, along with the accuracies for the CNN models.

The learning rate was set to 0.0001 for all the MLP models in order to properly compare the other hyperparameters. The learning rates were varied across several models in order to find the optimal, but only the results for the TwoHiddenLayer(ReLU) model are shown in Figure 1 to save space. It displays the training and test accuracy for a learning rate that is larger (α = 0.0005) and smaller (α = 0.00001) than the chosen learning rate. The tests indicate that a larger learning rate incurs an increased gap between the training and test accuracy. In other words, overfitting has occurred. Furthermore, it is less stable, and its larger oscillations clearly reveal the gradient's behavior of taking a "drunken walk" to the minima due to the larger step size. On the other hand, the lower learning rate exhibits less overfitting than the other two models, but the gradient does not move as fast to the minima, and it returns lower accuracies for only a slight decrease in overfitting when compared to the 0.0001 model.

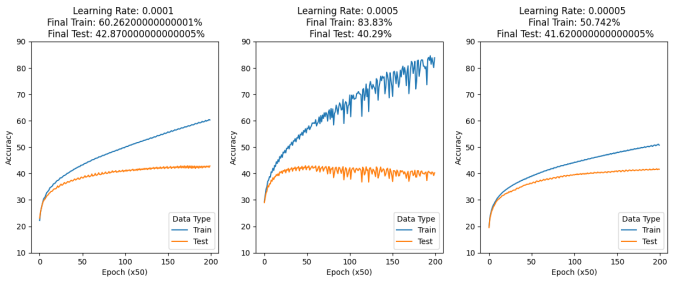

Fig. 1: Training and Test accuracies according to different learning rates for the TwoHidden model.

Model (ReLU if not specified)	Train Accuracy	Test Accuracy
NoHidden	31.046%	28.42%
OneHidden	51.714%	41.89%
TwoHidden	59.694%	42.43%
TwoHidden (Leaky)	60.06%	43.28%
TwoHidden (Tanh)	54.546%	42.49%
TwoHidden (L1 Regularization)	40.018%	35.07%
TwoHidden (L2 Regularization)	59.50%	42.97%
TwoHidden (ReLU) Not Normalized	33.467%	31.64%
TensorFlow CNN	98.3%	60.95%
ResNet50 Pretrained (256 layer)	38.89%	38.81%
ResNet50 Pretrained (256 layer x2)	40.43%	40.21%
ResNet50 Pretrained (128 layer x2)	39.00%	38.19%
ResNet50 Pretrained (256-128-64-32-16)	38.39%	38.19%

TABLE I: Train and Test accuracies of all MLP and CNN Models.

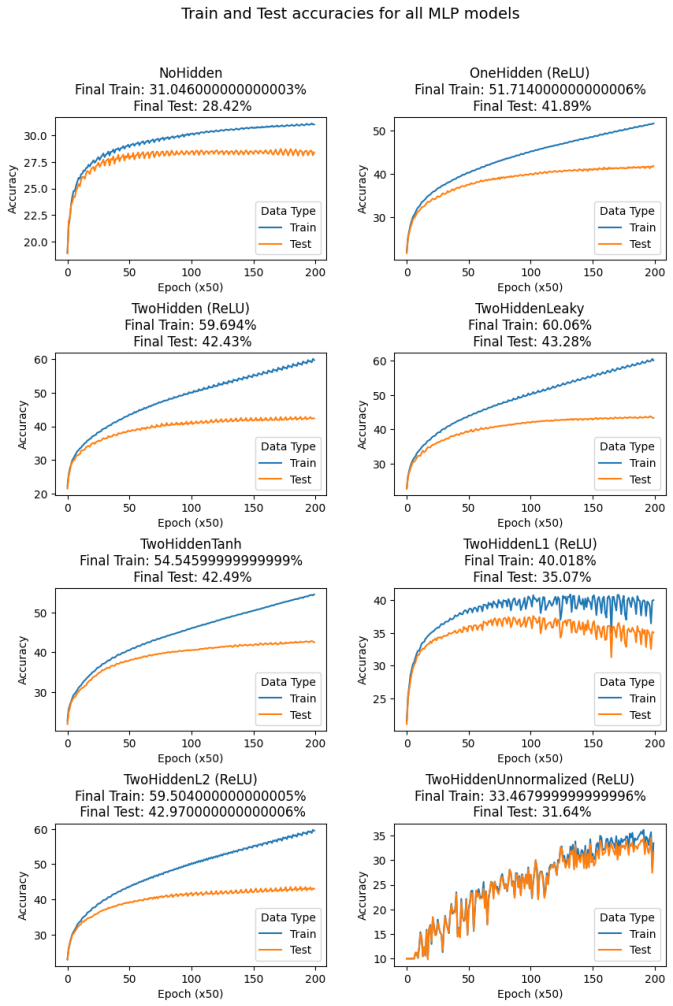

Fig. 2: Training and test accuracies for the best results obtained from the Multilayer Perceptron models.

A. *Task 1: Non-Linearity and Depth of MLP*

In this task, the goal was to study the effect of increasing depth in our MLP model. We trained three different MLPs: A model with no hidden layers (NoHidden), a second model with a single hidden 256-unit fully-connected layer (OneHidden), and a third model with two 256-unit fully-connected hidden layers. All three models use the ReLU activation function and have a learning rate of 0.0001. The time taken to train these models were very fast compared to the CNNs: the NoHidden model took 33s, the OneHidden model took 2m54s, and the TwoHidden model took 4m2s. The accuracies can be found in Figure 2. As described in Section IV, activation functions add non-linearity to the model parameters. The NoHidden, OneHidden, and TwoHidden models have test accuracies of 28.42%, 41.89%, and 42.43%, respectively. The model will no hidden layers (the linear model with only SoftMax applied) performs lower than OneHidden and TwoHidden where the non-linearity function was applied. This shows that the data was not linearly separable. Also, the increased depth did not have much effect, as the test accuracies are similar for these two models.

B. *Task 2: Comparison of Activation Functions*

Our task here was to compare the effects of the three different activation functions. As mentioned in the above subsection, the NoHidden, OneHidden, and TwoHidden models were trained with the ReLU activation function. In this section, we trained the TwoHidden model using the Leaky ReLU and the Tanh activation functions for both layers. The results can be found in Figure 2. Time-wise, the Leaky activation function took a very similar amount of time as ReLU. However, the tanh model took an additional 3 minutes to run due to the increased computation complexity of the function: $f(x) = max(0, x)$ vs. $f(x) = tanh(x)$. And the same for their derivatives. ReLU, LeakyReLU, and Tanh all have test accuracies within 1% of each other. However, the training accuracy decreases from ReLU to Tanh in the same order as they are listed. Both LeakyReLU and Tanh help with dead neurons by giving them a negative effect on the resultant output. However, this did not have as much of an effect because the other neurons compensated by having higher weights.

C. Task 3: Effect of L1 and L2 Regularization

In this task, our goal was to implement L1 and L2 regularization on the TwoHidden model and compare the results. These models took roughly a minute longer to compute than the standard TwoHidden, which was as expected, but the accuracies were all very similar (though slightly lower). However, there is a logical explanation for this: Our MLP is relatively small when compared to the size of the dataset, meaning that the model capacity is quite low. Regularization lowers this capacity further, hurting the performance.

It should also be noted that we used a Regularization Strength of 0.9 and a Learning Rate of 0.0001 in Figure 2. Too low or too high a mix of these parameters resulted in models with low accuracies.

D. Task 4: Effect of Unnormalized Input Data

In this task, an MLP was implemented with 2 hidden layers both using ReLU activation with 256 units. The input data, however, was not normalized, but still an array 1024 units long. With the same learning rate used previously, the model's accuracy did not rise as sharply and quickly as the other models, and it was a lot more volatile and susceptible to change compared to the other models while training. At first, this test resulted in overflow errors leading to a low training accuracy of 10%, meaning it classified all inputs as the same image type. In order to fix this issue, after some testing, the learning rate was decreased to 0.00008. We also decreased the standard deviation of the normal distribution used to initialize the weights in order to take into account the unnormalized values of the input dataset. The training and test accuracies after changing these input parameters were 33.46% and 31.64%. This is a worse accuracy compared to the normalized input result, though the overfitting was reduced.

E. Task 5: TensorFlow Convolutional Neural Networks

A CNN model, created using Tensorflow, was built with two convolutional layers and two 256-unit fully-connected (dense) layers with ReLU activation functions as well as a Softmax layer for classification and a 3x3 kernel. The two-dimensional normalized greyscale image data was reformatted and input into the CNN for a sound comparison to the MLP model (which requires one-dimensional input data). The results can be found in Figure 2, and further experiments on this model type are continued in a later subsection.

F. Task 6: Transfer Learning CNN Model using ResNet50

ResNet50, a convolutional network that is 50 layers deep, was used as the base for our Pre-Trained model. Using such a starting base allows the construction of accurate models in a timely manner, a technique known as Transfer Learning. In order to repurpose the re-trained model, the fully connected layers were removed, and the remaining (convolutional) layers were frozen. Two 256-unit ReLU-activated fully-connected layers plus a Softmax classification layer was then added on top. It took about 40 minutes to train on CoLab, and this pre-trained model has much less variance than our other tested models - the train and test accuracies differ by less than one percent. Several layer combinations: No layers, a single layer with 256 units, three layers with 256-128-64 units, and five layers with 256-128-64-32-16 units were trained in order to observe the effect of varying widths and depths. The increased depth, the longer the computation. The accuracies were all slightly lower than the originally chosen 256-256 model.

G. Exploratory Experiments

In addition to the required test against the MLP model and the Pre-Trained CNN, we ran several tests on the CNN from part 5 to measure the performance of different batch sizes, kernel/filter sizes, and the Dropout regularization technique. The results are shown in PLOTNAME. MaxPooling not only decreased the intensity of the overfitting and returned better test accuracy, but it also drastically decreased the time it took to train the model: The CNN with no MaxPooling took 53m25s to train, while the one with MaxPooling took only 11m24s. Thus we applied MaxPooling to the batch-size, filter-size, and dropout experiments to speed up the remaining tests. As such, the 128 and 256 batch-size versions similarly took 9m23s and 8m21s, respectively. The 5x5 and 7x7 filter versions took 16m31s and 15m22s, respectively, and the 0.2 Dropout version took 10m56s minutes while the 0.9 dropout took 11m24s minutes.

As we can see in Figure 4, a small amount of dropout improves the model, but too much drastically decreases the accuracy. Following from what we said in the section on Regularization, this is because dropout is reducing the capacity of a model which is already fairly small. Taking away too many nodes means there will not be enough remaining for accurate predictions.

Meanwhile, the batch smaller batch size of 128 took longer to train but resulted in a superior model. Finally, the 5x5 filter increased the accuracy slightly, but the 7x7 filter damaged the accuracy of the model, so one must

carefully select their filter size if they want to maximize a model's effectiveness. Another avenue of exploration would be to look into zero-padded filters.

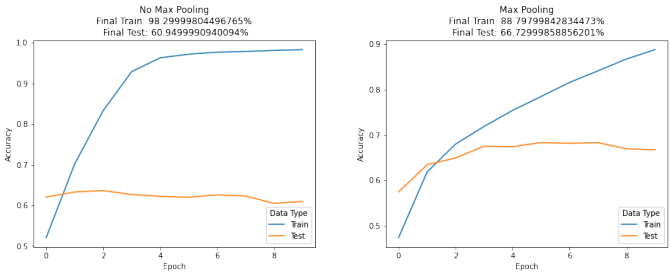

Fig. 3: CNNs with 2-2DConv Layers with and without max pooling afterward, and two ReLU Layers with 256 Units.

H. Performance Between Models

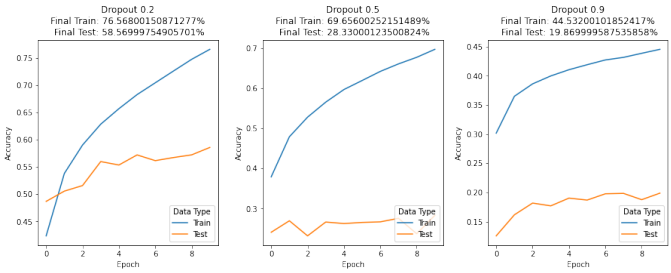

Fig. 4: CNNs with 2-2DConv Layers followed by Max Pooling, and 2ReLU(256 Units), with Dropout of 0.2, 0.5, and 0.9.

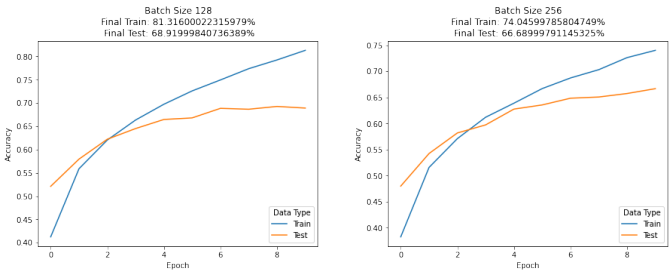

Fig. 5: CNNs with 2-2DConv Layers followed by Max Pooling, and 2ReLU(256 Units), with batch_size=128 and batch_size=256

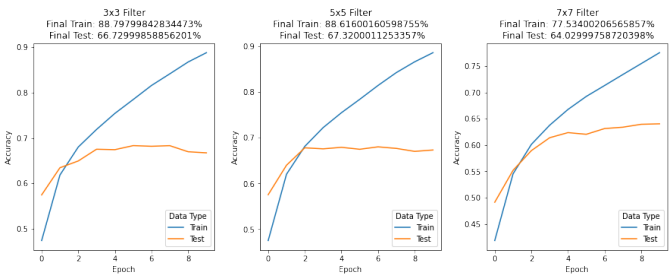

Fig. 6: NNs with 2-2DConv Layers followed by Max Pooling, and 2ReLU(256 Units), with filter sizes of 3,5, and 7

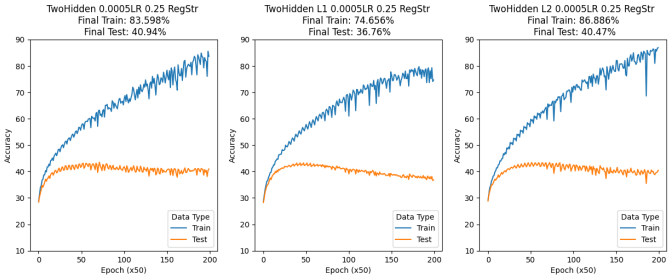

Fig. 7: 2Hidden(ReLU) with no regularization, L1 regularization, and L2 regularization

VI. Discussion and Conclusion

It was observed that having non-linearity and a deeper network helped the model perform better. Our results showed that ReLU produced the best results, followed by leaky and finally tanh. They also showed that L1 and L2 regularization did not have any positive effect on the model accuracy. Even though the MLP performed considerably well compared to the TensorFlow results, it is possible that given enough computation time, the model would produce as good as results as the one for TensorFlow. Further statistical evidence would be needed to ensure that these results hold and that they were not due to randomness from the weight and bias initialization. The pretrained ResNet50 model had a lower accuracy than our other models, which we can attribute to the fact that we were not able to tune any of the hyper-parameters aside from the number of layers and the number of units within those layers (which were shown to have relatively low effects). However, the TensorFlow CNN model had high accuracy despite some overfitting. The extra experiments in the "Exploratory Experiments" section showed us that MaxPooling is incredibly effective at reducing computation time and improving accuracy. Meanwhile, regularization techniques like L1/L2 regularization and dropout seem to have little effect on these models, likely to their already low size/capacity. Parameters like batch size and filter size have small effects but enough that attention should be paid to them to maximize accuracy and minimize runtime.

VII. Statement of Contributions

- Maxim Boucher: Helped with data preprocessing. Wrote and ran tests for Task 3. Wrote most of the "Results" section of and helped review the report.
- Allison Mazurek: Gathered and preprocessed the dataset. Reviewed Task 2. Ensured test input data was correct and wrote code for Task 3. Wrote and formatted the report.
- Youssef Samaan: Implemented Task 2. Helped run tests and debug the code. Helped give the main ideas of the report and reviewed the report.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 'ImageNet Classification with Deep Convolutional Neural Networks", 2012

[2] Alex Krizhevsky, 'Learning Multiple Layers of Features from Tiny Images", 2009.

[3] https://paperswithcode.com/sota/image-classification-on-cifar-10, Accessed:[09 03 2022]

[4] A.Dosovitskiy, L.Beyer, A.Kolesnikov, D.Weissenborn, X.Zhai, T.Unterthiner, M.Dehghani, M.Minderer, G.Heigold, S.Gelly, J.Uszkoreit, N.Houlsby, An Image is Worth 16X16 Words: Transformers for Image Recognition at Scale, ICLR 2021

[5] Kurt Hornik, Multilayer Feedforward Networks are Universal Approximators, 1989