MiniProject 3: Classification of Image Data

Alhusain Abdalla, Ege Yay, Hungi Kim - Group 61

***Abstract*— This project aims to evaluate and compare the performances of a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN) on classifying images from the renowned CIFAR-10 dataset. We tune several model-related and optimization-related hyperparameters for both networks in order to reach the best possible performance. For CNN, we experimented with the total number of networks parameters to be learned, number of layers, batch size, batch normalization, and learning rate. For MLP, we experimented with the number of units in each layer, the number of layers, and the activation function. This investigation of hyperparameters was based on recommendations from literature as well as personal trial and error. Eventually, we reached a testing accuracy of 92% for CNN and 15% for MLP.**

I. Introduction

In this project, we designed an MLP from scratch and a CNN using PyTorch to classify images from the famous CIFAR10 dataset. In our implementation, we worked on tuning several hyperparameters in order to reach the best accuracy on 10,000 testing samples that were never seen before by the networks. For the CNN, we experimented with different choices for optimization parameters such as the optimizer, learning rate, number of epochs, and batch size. We also experimented by tuning several model parameters such as the total number of parameters, number of layers, batch normalization, and dropout. As for the MLP, we experimented with the number of layers, number of units per layer, and the activation function. Finally, we compared the performance of MLP and CNN in terms of train loss, training accuracy, and testing accuracy. The results were interesting, since we were able to reach a highest accuracy of 92% for CNN, but only 15% for MLP.

A. Multilayer Perceptron

MLPs could be thought of as an extension of logistic regression and are considered the ”plain-vanilla” models of neural networks. Generally, an MLP includes several layers composing of several micro-components known as ”neurons”. An input layer receives the data, a number of hidden layers process the input and learn from it, and a final layer that outputs predictions about the input. In the training phase of the network, each neuron takes a weighted sum of some or all inputs, passes it through an activation function, and keeps propagating it through the network until an output is given. At the end of each forward pass, decisions at the output layer are compared to ground truth labels using a loss function, such as root mean squared error (RMSE) or cross entropy loss. Following that, the backward pass takes place. All the weights and biases from the forward pass are propagated backwards using the chain rule, partial derivatives of the loss function. The purpose from this back-propagation is to get a gradient (a landscape that represents error) in order to correct the weights and biases in the best possible combination that reduces the error calculated by the loss function.

B. Convolutional Neural Network

CNNs are a powerful version of neural networks with learnable weights and biases similar to MLPs. They are most powerful and most used with images. Generally, a simple CNN architecture would include at least 3 layers - a convolution layer, a pooling layer, and a fully connected layer. The building block of CNNs is the convolution operation itself, which extracts spatial features from the input and, passes them through a non-linear activation function and across the rest of network. The pooling layer reduces the dimensions of the input without jeopardizing its spatial information. The fully connected layer flattens the features outputted by the convolution layer and transforms them into final votes of the output, similar to that of an MLP.

II. Related Work

1) CNN: In Park's implementation of Convolutional Neural Network to classify the images from the CIFAR-10 dataset [1], his model consists of a total of 14 layers; including 4 convolutional layers with 64, 128, 256, 512 channels respectively. Each layer is followed by a max pooling layer which ReLU activation and batch normalization, and 4 fully connected layers. Adam algorithm is used as the optimizer. The accuracy starts at 21.7% on first epoch and improves up to 75.6% on tenth epoch.

2) Hyperparameter tuning for CNN: For CNN implementation, various settings and results for hyperparameter tuning were presented by a machine learning start-up, known as "Weights & Biases" [2]. Among their settings, the one that achieved highest test accuracy is as follows: batch size of 128, two convolutional layers with 64 and 512 respective channels, 50 epochs, 99 hidden nodes and learning rate of 0.0082.

3) Comparison of MLP vs CNN: In 2019, an implementation by Krzysztof Arendt actually compared the performance of MLP vs. CNN on the same CIFAR-10 dataset [3]. His results showed a 44% testing accuracy for MLP and 62% for CNN. He concluded that CNNs in general are more powerful than MLPs with image classification tasks. However, he mentioned that CNNs are limited in the sense that they are purely pattern based and have no understanding of the underlying images.

III. Dataset

CIFAR-10 dataset contains 60,000 32x32 RGB color images. It consists of 10,000 testing images, and 50,000 training images divided into five batches of 10,000 images each. Each image is labelled as one of the following 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. There exists no overlapping images between any two labels.

Data Augmentation

For CNN, we implemented data augmentation, which can help achieve a better result of image classification by adding slightly transformed data to the training dataset. It allows for more accurate prediction by providing different versions of the same object in an image, such as different positions, different colors, rotations, enlargement and so forth.

IV. Proposed Approach

A. Batch Normalization

Batch normalization (BN) refer to standardizing the input, meaning that a transformation is applied to keep the mean of all activations closer to 0, and the standard deviation of all activations closer to 1. Experiments show that BN reduces the sensitivity of the network to the initial weights and increases the training speed (allowing for much higher learning rates). Therefore, in our CNN, we batch-normalized the output at each convolutional layer using the BatchNorm2d operation in PyTorch.

B. Network Structures

We designed the general illustration of our network's structure as shown in Fig. 1. The network has a total of 14 convolutional layers, 3 max pooling layers, 1 average pooling layer, and 3 fully connected layers. Each convolutional layer consists of a convolution operation, followed by batch normalization and a RELU activation function. For the MLP, we designed a 4 layer network with a total of 1,100 perceptrons (or neurons, or units). The choice of these specific structures was based on recommendations from literature as well personal trial and error, as we started from two very basic networks and moved our way towards the best models. (as shown in following sections).

C. Loss Function and Optimization

Based on literature, we chose Cross Entropy Loss for the loss function and Adaptive Momentum (ADAM) as the optimizer for CNN. For MLP, we implemented a Sigmoid Cross Entropy Loss for the loss function and a Stochastic Gradient Descent as the optimizer.

V. Experiments and Results

A. CNN Hyperparameters Tuning

We began by building upon a basic network of 2 convolution layer, few parameters (6000 learnable parameters), no batch normalization, and a data loader with batch size 32. Our aim was to tune these parameters and take our network slowly to

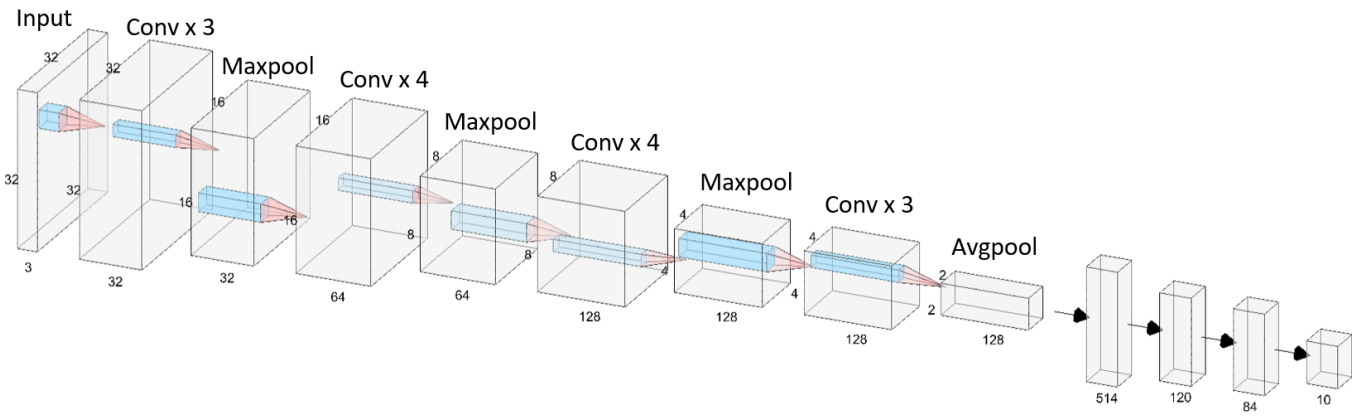

Fig. 1. Structure of our CNN

develop the most efficient model. Table I shows a summary of our experiments and their results (note the transition between experiments, each experiment builds upon the best result from the previous experiment). We conducted all experiments on 100 epochs.

TABLE I

DETAILS OF EXPERIMENTS PERFORMED

Experiment	Parameters	Test Acc. (%)	Train Acc. (%)
Batch Norm	without BN	70.5	68.9
	with BN	72.1	69.2
Batch Size	32	72.1	69.2
	128	72.4	69.7
# parameters	62006	72.4	69.7
	257022	80.1	81.7
# layers	2	80.1	81.7
	128	91.0	99.1

1) Batch Normalization: We began by testing two versions of the basic network we described above; one without any batch normalization and one with batch normalization following each convolution layer (between the convolution operation and the RELU function), the results are shown in Fig. 2. In addition to the improved accuracy, BN also decreased training time.

2) Batch Size: After implementing BN, we wanted to test the effect of increasing batch size. We tested with batch sizes 32 and 128 as shown in Fig 3. There was a minor increase in test accuracy and a noticeable increase in training speed.

3) Number of Parameters: Now that our batch size increased from 32 to 128, we wanted to test the effect of increasing the number of learnable

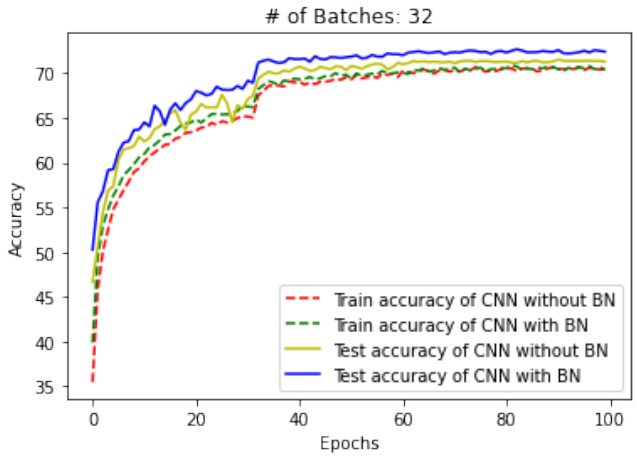

Fig. 2. Experimenting with Batch Normalization

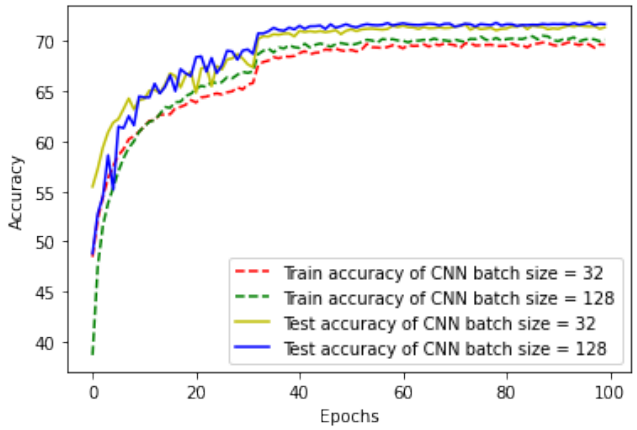

Fig. 3. Experimenting with batch size

parameters in the network (without actually increasing the number of convolution layers). This can be achieved by increasing the number of filters (channels) in a single convolution layer. The total number of parameters in a convolution layer is calculated as following:

$$(m \times n + 1) \times k \tag{1}$$

where m and n are the height and width of the filter, 1 is the bias term at each filter, and k is the number of filters (or channels). We increased the number of parameters from around $60,000$ to $250,000$ parameters and a major increase in test accuracy was noticed (a jump from around 70% to 81%)

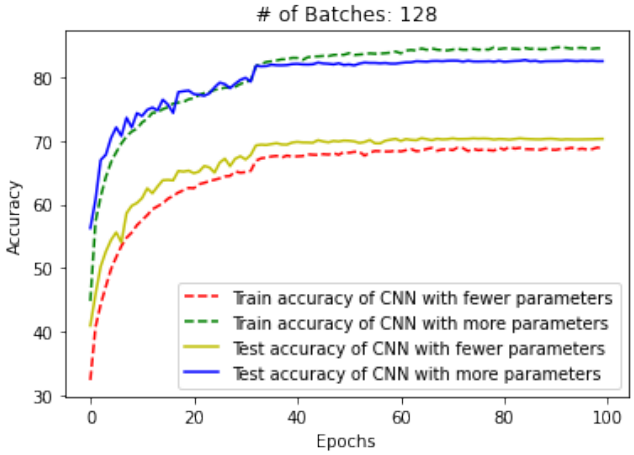

Fig. 4. Experimenting with the number of parameters

4) Number of Layers: Finally, we investigated the effect of increasing the number of layers on the accuracy of our model. To do that, we added another 12 convolution layers (total 14) and reduced the filter window size to preserve some x,y dimensions until the end. Since we started with a 32×32 dimensions, we calculated the required modifications to our convolutions layers in such a way that maintained a 1×1 x,y dimensions before the fully connected layers through the following equation:

$$Output = \frac{w - k + 2p}{s} + 1 \tag{2}$$

Where w refers to window size of the input, k refers to kernel size of the operator, p refers to padding size, and s refers to the stride. The results of this experiment were astonishing as well, since the testing accuracy increased by more than 10% and surpassed the 90% accuracy mark as shown in Fig. 5.

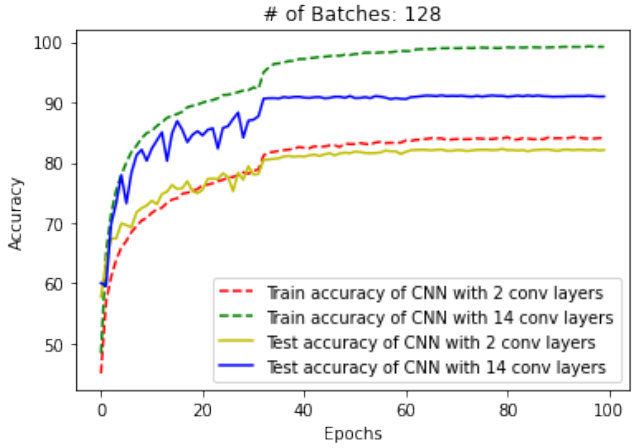

Fig. 5. Experimenting with the number of layers

B. MLP Hyperparameters Tuning

For MLP, we began with a basic network consisting of 4 layers, a total of 155,490 learnable parameters (sum of all learnable weights and biases), and a sigmoid activation function. Our goal was to experiment with these parameters to slowly develop the most efficient model.

1) Number of layers: We compared two networks with 4 and 5 layers, while maintaining the total number of neurons (or units) per network fixed to 90 units. Fig. 6 shows that a 5 layer network gives a slightly higher accuracy after 100 epochs. However, it takes substantially more training time.

2) Activation Function: We experimented with 3 different activation functions. Sigmoid activation, ReLU activation, and Logistic activation. As seen in Fig. 7, although ReLU activation showed the steepest increase in accuracy, the Sigmoid function resulted in the final highest accuracy. All in all, the differences in accuracy were not significant.

3) Number of perceptrons: The number of learnable parameters can be determined by the total number of connections between the neurons and the biases. To increase the number of learnable parameters, we increased the number of perceptrons or "units" within each layer. Fig. 8 shows the comparison between a network of 90

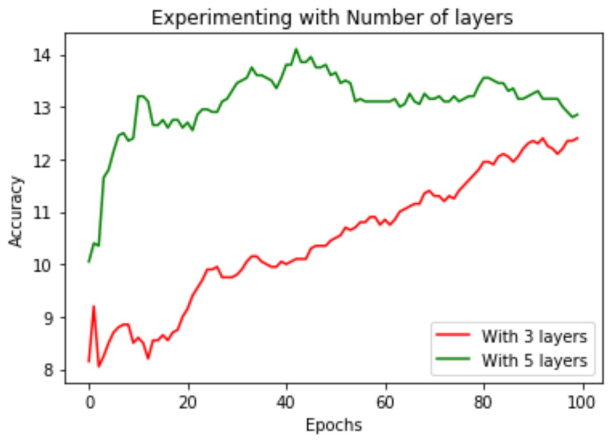

Fig. 6. Experimenting with the number of layers for MLP

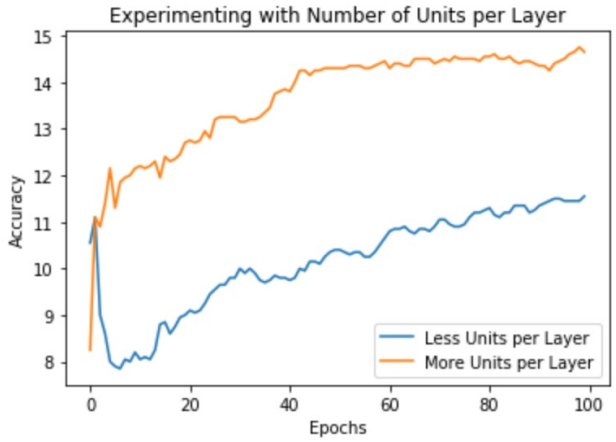

Fig. 8. Experimenting with the number of units for MLP

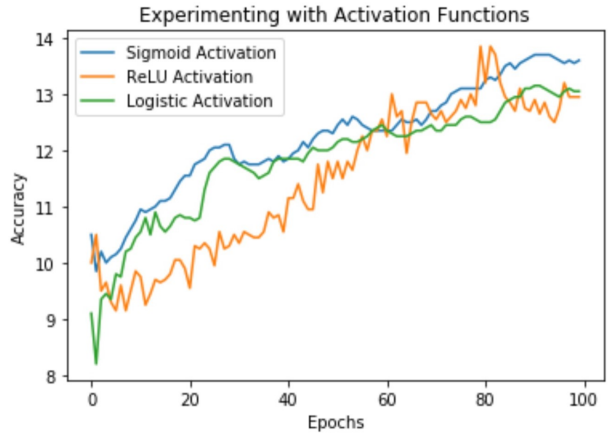

Fig. 7. Experimenting with the activation function for MLP

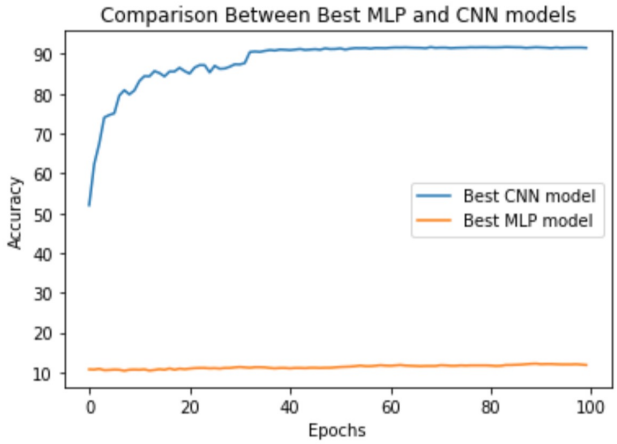

Fig. 9. Comparing best CNN and MLP models

units (after the input layer) and a network with 1,100 units. The experiment showed a substantial increase in testing accuracy when the number of units increased.

C. CNN vs. MLP

After performing all these experiments, we were able to finally reach the best CNN and MLP models to compare their performances. The best CNN model reached with a fairly high accuracy of 92% after adding a couple of dropout layers between the convolution layers to slightly improve the testing accuracy and training time. The MLP however, reached a much lower testing accuracy of around 15%. Fig. 9 presents the accuracies of both neural network models and the significant gap between them.

VI. Discussion and Conclusion

We implemented an MLP from scratch and a CNN using the famous PyTorch library. The goal was to find the most accurate models through tuning the hyperparameters of both networks. Results showed a significant superiority of the CNN's accuracy over the MLP's. In modern computer vision tasks, MLPs are no more deemed sufficient for several reasons [4]. First, the number of parameters can grow very high, which is inefficient due to the redundancy in higher dimensions. In addition, it takes the inputs as flattened vectors and completely ignores the spatial information. This explains why the CNN gave much better accuracy than the MLP. An interesting observation was that increasing the number of learnable parameters for both networks gave the most substantial increase in accuracy.

VII. Statement of Contributions

Alhusain implemented the CNN and its experiments. Ege implemented the MLP and its experiments. Hungi assisted in the writeup.

References

[1] P. Chansung. Cifar-10 image classification in tensorflow. [Online]. Available: https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c

[2] C. V. Pelt. Optimizing cifar-10 hyperparameters with wb and sagemaker. [Online]. Available: https://www.wandb.com/articles/running-sweeps-with-sagemake

[3] K. Arendt. Cifar-10 images: classification using mlp vs. cnn. [Online]. Available: https://krzysztofarendt.github.io/2019/01/29/cifar-10.html

[4] Uniqtech. Multilayer perceptron (mlp) vs convolutional neural network in deep learning. [Online]. Available: https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1