COMP 551
Mini Project 3: Classification of Image Data

Stephanie Li, Sully Dawood, Jerry Dong *McGill University*
GROUP 113

***Abstract*—In this project, the performance of two neural networks Multilayer Perceptron (MLP) and Convolutional Neural Network(CNN) was investigated on the CIFAR-10 dataset. The image data was preprocessed by normalizing and one-hot encoding the inputs. In addition to implementing the MLP from scratch, we investigated the effects of hyperparameter tuning, activation function selection, and regularization techniques to identify the best models for this dataset.**

I. INTRODUCTION

In this project, we explore two machine learning models on the CIFAR-10 dataset and analyze their performance using accuracy as a metric. Given that CIFAR-10 is an image classification problem, the two models we investigated are well-suited to this task.

1) Multilayer Perceptron (MLP): MLPs are also known as feed-forward neural networks. They take inspiration from how visual processing is handled by the human brain: given a set of inputs to the eyes, neurons integrate this information with increasing complexity layer by layer until a recognizable output is achieved. This system is mimicked by the MLP, with each layer feeding inputs to a layer above to form a set of ”activations” that indicate the excitability of that neuron. What this model learns is how much to weigh each activation as input to the neurons in the next layer. We also transform these inputs using activation functions applied at each layer. Common activation functions include the sigmoid and tanh functions. However, recent literature suggests that the ReLU function works better due to its ability to avoid vanishing gradients as the inputs deviate from 0.

The final output of the MLP is processed through another activation function (usually sigmoid for binary classification or softmax for categorical classification).

The optimal weights of the MLP are found through gradient descent, which makes use of the Jacobian matrix of the partial derivatives of the loss function (i.e. logistic loss for binary classification or cross-entropy loss for categorical classifcation) with respect to each weight in the network. This process is iterated through a number of epochs until we reach a local minimum of the cost function, which we can assume is close to a global minimum because of the landscape of the cost function.

We can also choose to use regularization methods including L2, dropout, and batch normalization, which will be discussed in this paper.

Additionally, some hyperparameters that must be tuned in MLP models include the percentage of dropout nodes during training, the number of layers, the number of activation nodes in each layer, and the number of epochs. The latter three of these will be explored in the results section.

2) Convolutional Neural Network (CNN): The most well-known model for learning from image data is convolutional neural networks (CNN). Drawing inspiration from MLPs, CNNs add to the basic concept of feed-forward layers using filters to learn spatial hierarchies of features. This exploits the fact that images contain sparse information that can be better represented by applying many filters of few dimensions. By greatly reducing the number of parameters required to represent a given image, the use of filters make training more feasible. In other words, convolutional layers enable parameter sharing and for sparse connections. The convolutions additionally help prevent overfitting by allowing translational invariance: the filters are tuned to identify specific features regardless of where they appear in the image, which is critical to the task of image classification (e.g. a cat in the left corner should not be categorized differently from a cat in the right corner).

More generally, CNNs take an image and pass it through a series of convolutional, pooling and fully connected layers to yield an output that is passed through an activation function .

The term “kernel” refers to the sets of parameters for each filter that is applied in convolution operations. The first convolutional layers repeatedly apply kernels to perform feature extraction, yielding a feature map that represents the properties of the inputs. Secondly, the pooling layer reduces the in-plane dimensionality of feature maps to decrease the number of learnable parameters. Finally, the features are mapped by a fully connected layer and an activation layer. CNNs are typically structured in repeating layers of convolutions, pooling, and fully connected nodes.

Training the network involves determining kernel parameters in convolution layers and weights that minimizes the discrepancy between the predicted and true labels. This is achieved through backpropagation and gradient descent, in a similar fashion to the process of training MLPs [1].

A. The Task

This project investigates the performance of MLPs and CNNs as we compare the classification methods on image data from the CIFAR-10 dataset that is partitioned into training, validation, and testing sets. Backpropagation and mini-batch gradient descent with momentum were implemented from scratch for MLP. Preexisting code for a CNN from PyTorch was adapted to yield the results presented in this paper[2].

B. Related Work

On the CIFAR-10 data set, an abundance of research has been conducted in enhancing the performance of CNNs in image recognition. Lin et. al proposes the use of more complex activation functions[3]. Whereas, Stollenga et. al. demonstrates techniques to improve class inference [4]. The authors dynamically alter convolutional filter sensitivities during classification to improve performance. Other studies investigate improved regularization and introducing multiple convolutions in between pooling layers[5].

Finally, Chan et.al evaluates a deep learning network on the CIFAR-10 set using principal component analysis, binary hashing, and block-wise histograms[6]. They demonstrate that despite having a lower accuracy compared to CNN, there may be merit in using the network as a baseline for large-scale image classification. We took inspiration from these studies in constructing the models presented in this paper.

II. Dataset and Preprocessing

A. CIFAR-10

Collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, the CIFAR-10 is a labelled subset of approximately 80 million images [7]. The data set itself is comprised of 60 000 32x32 coloured images. This is further partitioned into testing and training sets containing 50 000 and 10 000 images respectively. We then further split the training set into training (40 000 images) and validation (10 000 images). The images are separated into 10 mutually exclusive classes, each assigned labels using the integers 0-9. Classes range from animals such as ”cat” or ”dog” to vehicles like ”truck” and ”plane”. The data files were then ”pickled” using cPickle.

B. Data Preprocessing

Unlike regular classification problems where the data itself has a basic set of features, in textual data analysis, the first step is to try to break the data down into features. The following section incisively explains the various processing steps the group undertook to extract features for the dataset.

1) Normalization: To normalize the input data, we used torchvision.transforms.Normalize to achieve a range of [-1,1]. This normalizes the channels of the inputs, which is necessary to ensure all the data falls into the same range, allowing us to achieve a more stable convergence of model parameters.

2) One Hot Encoding: We one-hot encoded the labels used in MLP training with sklearn.preprocessing.label_binarize, which transformed the integer classes to a binary vector of 10 columns.

C. Data Analysis

III. Results

In this section, we will briefly discuss the logic of the two models implemented, the process of hyperparameter tuning, and compare the results achieved.

Table I: Hyperparameter(s)

Algorithm	**Hyperparameters**
MLP	Number of epochs, batch size, learning rate, momentum, L2 lambda
CNN	Number of epochs, batch size, learning rate and momentum

A. Multi-layer Perceptron (MLP)

The weights were initialized using the Xavier initialization scheme, where we set the weights in each layer to a normal distribution between the values of $\pm \frac{\sqrt{6}}{\sqrt{num\_inputs + num\_outputs}}$

[8] The first hyperparameters we tuned were the number of hidden layers and the number of activation units. We settled on a 3-layer MLP with 500 units in each hidden layer and a bias term. When trying higher numbers of layers (10 or more), we ran into issues with numerical stability, causing the program to crash. This was probably due to limitations in our implementation.

To increase the speed and likelihood of convergence, we implemented momentum in the mini-batch gradient descent. The beta hyperparameter for momentum was tuned to 0.9 with a learning rate of 0.005.

We also analyzed the difference between using ReLU vs sigmoid activation functions in the hidden layers (Figure 1). We noticed that ReLU allowed the model to achieve a higher validation accuracy in fewer epochs because of its ability to avoid vanishing gradients.

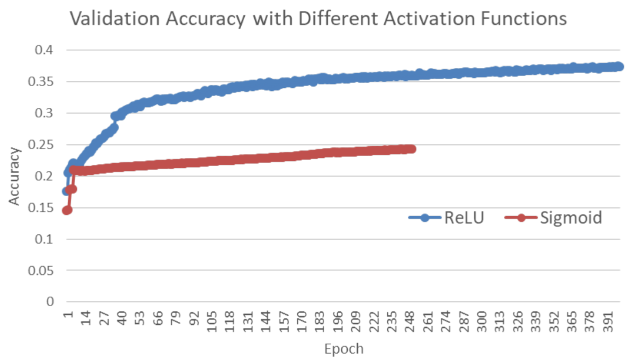

Figure 1: Summary - MLP validation accuracy as a function of epochs using different activation functions

In coding the model, we attempted to implement batch normalization, but this was not completed due to time constraints. However, we did apply L2 regularization to the activation functions to prevent overfitting and noticed that this allowed the model to achieve a higher accuracy over fewer epochs (Figure 2). The lambda value chosen for L2 was 0.001.

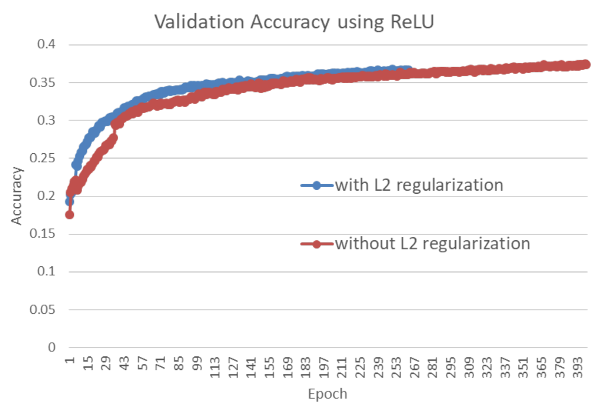

Figure 2: Summary - MLP validation accuracy as a function of epochs with and without L2 regularization ($\lambda = 0.001$)

In plotting the training and validation cost over the number of epochs, we noticed that the validation cost began to increase for the validation set beyond 150 epochs. The validation accuracy also did not significantly increase beyond this point (Figure 3). Because of this, we decided to stop training the model at 150 epochs to avoid overfitting.

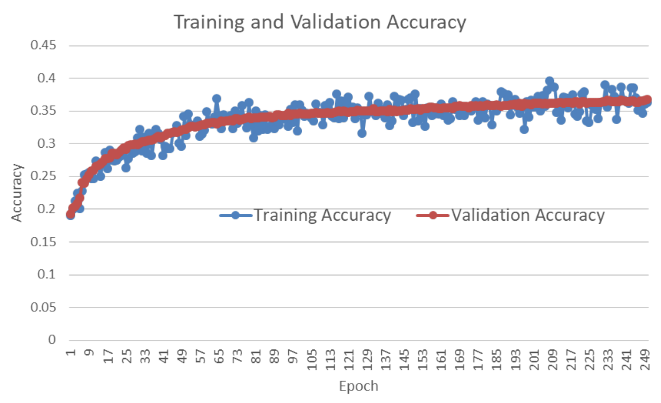

Figure 3: Summary - MLP training and validation accuracy as a function of epochs

B. *Convolutional Neural Network*

The neural network was defined to take 3-channel images as input. The hidden layers from input to output were as follows: conv1-pool-conv2-fc1-fc2-fc3; the final activations were then passed through a softmax function to yield the final output. Consistent with most categorical classification problems, cross-entropy loss was utilized as the loss function, with schoastic gradient descent used in the optimization process. We then tested the trained network on the test data.

Furthermore, we found that although there was an initial increase in accuracy of the network on the test set, the accuracy curve eventually decreased as the number of training epochs was increased (Figure 4), which signals overfitting. The range of training epochs we analyzed was 1 to 20. The best performing model was achieved after four epochs, yielding a test accuracy of 64.3%. The best performing class was car at 76% and the worst performing class was dog at 38%. These trends in class accuracy were generally consistent over the number of training epochs.

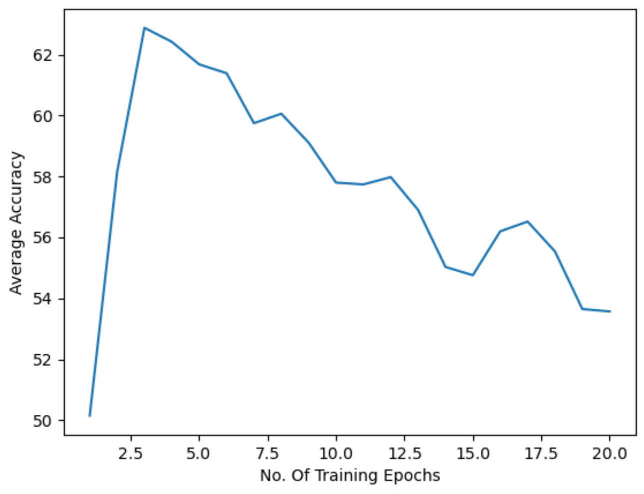

Figure 4: Summary - CNN test set accuracy as a function of Training Epochs

The following graphs reflect the logic of stochastic gradient descent: the cross-entropy loss decreases over the number of mini-batches as we iteratively update our weights towards the negative gradient of the loss function.

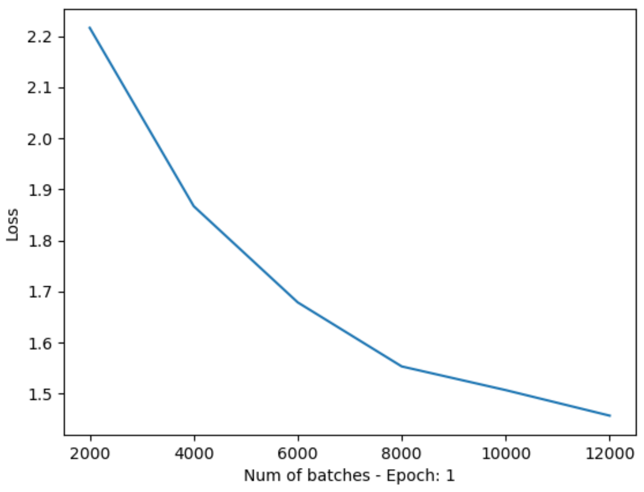

Figure 5: Training loss as a function of mini-batches (computed every 2000 mini-batches) in epoch 1

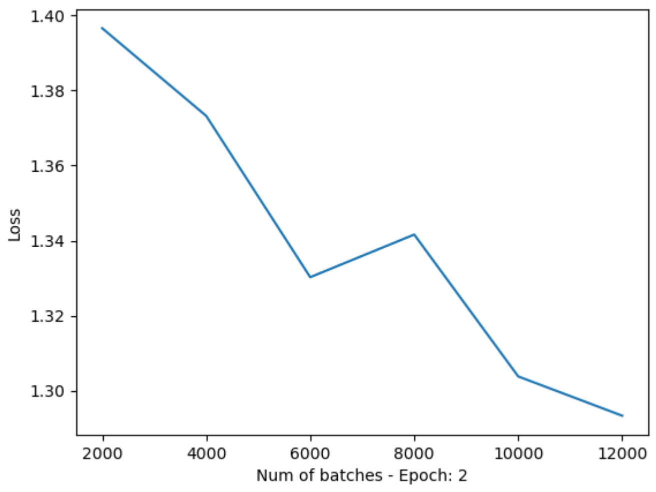

Figure 6: Training loss as a function of mini-batches (computed every 2000 mini-batches) in epoch 2

Table II: Results

Algorithm	**Best Hyperparameter(s)**	**% Accuracy**
MLP	epochs: 150, batch size: 500, learning rate: 0.005, momentum: 0.9, regularization: ’l2’, λ : 0.001	37.02%
CNN	epochs: 4, batch size: 2000, learning rate: 0.001, momentum: 0.9	64.3%

IV. Discussion and Conclusion

Neural networks make use of adaptive bases, meaning they have the ability to approximate any continuous function with arbitrary accuracy. However, this also speaks to their ability to overfit to training data. As such, regularization is crucial in training neural network models. One large limitation in our implementation of the multilayer perceptron was the lack of regularization. Although we used L2 regularization to penalize large weights that are usually indicative of overfitting, we did not successfully implement more powerful methods of regularization including dropout and batch normalization due to time constraints of the project. For future work, implementing these strategies is necessary to increase the generalization of our model to achieve higher model accuracy on unseen data. To show how these methods would have altered our model, we will explain these regularization methods below.

Dropout is a technique used where randomly selected neurons are ignored during training. This regularization method stochastically sets to zero the activation of hidden units for each training case at training time to prevent over-fitting and decrease interdependent learning amongst neurons [9]. Since dropped-out units cannot influence other retained units, this disrupts any co-adaptation of feature detectors [10].

Another useful method of regularization is batch normalization. At a high level, batch normalization preprocesses every layer of the network such that the activations obey a Gaussian distribution, improving the rate of convergence. To increase representational power despite this normalization, we then multiply the activations by some γ and add some β. As normalization is a differentiable operation, this also makes it easy to add to the back-propagation function. We can thus make the gammas and betas learnable parameters that can be optimized through gradient descent alongside the network weights.

During training, we would also keep track of the mean and variance of each layer computed over the training inputs and use these values to normalize the layers at test time. This has been shown to be an extremely powerful regularization technique that can be more effective than implementing dropout [11].

In comparison to the MLP, the CNN achieved much higher accuracy and trained significantly faster. To explain this, it is important to note that the number of activation sizes gradually reduced over the fully connected layers (e.g. 10,080 for the second fully connected layer and 840 for the last fully connected layer). Thus, the convolutional and pooling layers allowed for optimization and efficiency via parameter sharing and sparsity of connections. This was critical in allowing us to efficiently train the model despite the high number of layers. Ultimately, this is what allowed the CNN to achieve a better accuracy than the MLP; using convolutional layers enabled us to train a network with much greater depth for little computational cost. In comparison, the MLP required a higher number of nodes per layer to achieve the same representational power, limiting the number of layers that would be computationally feasible to train.

In comparing the run times of the two algorithms, we also noted that training the MLP took significantly more time than the CNN. Although this was partly due to implementation differences, it was largely because of the high number of parameters in each layer of the MLP.

V. Statement of Contributions

The collaborative work of this team involved everyone contributing to data processing and writing the report. Jerry contributed to the introduction, literature survey and references in the report.

Stephanie contributed to designing and implementing the multilayer perceptron. She also implemented backpropagation, mini-batch gradient descent algorithm and experimentation.

Sully applied and compared performance of the convolutional neural network, and trained/tested performance of the model with respect to epochs.

References

[1] R. Yamashita, M. Nishio, R. Do, and K. Togashi,Convolutional neural networks: an overview and application in radiology, *Insights Imaging*, vol.9, pp.611-629, 2018.
[2] TRAINING A CLASSIFIER,2017.
[3] M. Lin, Q. Chen and S. Yan,Network In Network, *arXiv*, 2014.
[4] M. Stollenga, J. Masci, F. Gomez, J. Schmidhuber,Deep Networks with Internal Selective Attention through Feedback Connections,*arXiv*, 2014.
[5] J. Springenberg, A. Dosovitskiy, T. Brox and M. Riedmiller. STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET, *arXiv*, 2015.
[6] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. PCANet: A Simple Deep Learning Baseline for Image Classification?,*arXiv*2014.

[7] A. Krizhevsky, V. Nair, and G.HintonThe CIFAR-10 dataset,2009.
[8] J.Dellinger. Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming, 2019.
[9] H. Wu and X. Gu,Towards dropout training for convolutional neural networks,*Neural Networks*, 2015.
[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting,*Journal of Machine Learning Research*, 2014.
[11] T. Chang. Implementing Batch Normalization in Python,