# COMP 551 mini project 3

Abdulla Alseiari Kenji Marshall Mian Hamza

April 20, 2020

**Abstract**

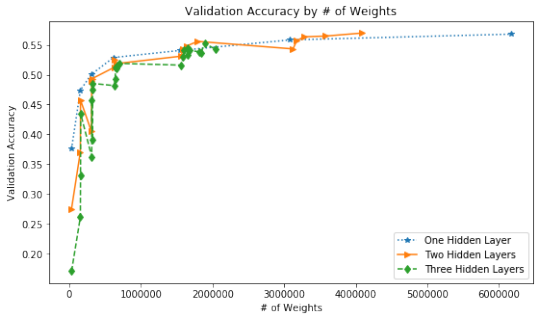
This mini project explored the performance of the Multilayer Perceptron and the Convolutional Neural Network (CNN) on the CIFAR-10 dataset. This dataset has 60000 RGB images of 10 different classes. In order to approach this problem, a flexible Multilayer Perceptron implementation was done from scrach in Python. 51 different architectures were explored through the lens of runtime and performance. We also showed the impact choices for dropout ratio or hidden layer activation function can have. We also trained and tested one CNN architecture based on LeNet-5 composed of two convolutional layers and two fully connected layers. Even with extensive experimentation the Multilayer Perceptron (56%) did not perform as well as the CNN (62%) on the test set. This demonstrated the inherent limitations of a fully-connected architecture for computer vision problems. In the future, further optimization over network architectures and parameters would be ideal.

## 1 Introduction

Image classification techniques are becoming essential methods in the modern digital realm. In parallel with other data classification tasks, image recognition has made huge advancements in the past few decades. Two of the current models for image classifications are the Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN). Both techniques use backpropagation to classify images based on their visual contents. In this project, we will implement these two models and compare their performance using the CIFAR-10 dataset.

## 2 Related work

Figure 1: Validation accuracy for one, two, three layer networks against model complexity




In the literature review, the CIFAR-10 dataset has been tested for image classification enormously. Thus, it has become a benchmark for this type of task. Different neural network models on the CIFAR-10 dataset have yielded a wide range of accuracies ranging from 75.86% [1] up to 96.53% [2]. In 2015, McDonnell and Vladusich's paper proposed an easy to implement and very fast to train CNN model, however, the model lag behind compared to state-of-the-art models in performance at their time [1]. In contrast to that, Benjamin Graham developed a CNN model with fractional max-pooling that achieved an extremely high accuracy of about 96% [2]. Other researchers focused on studying different activation functions since they are a significant component of neural networks. One group that achieved a 92.49% accuracy rate proposed using an adaptive activation function by using gradient descent for each neuron to learn a parametrized piecewise linear activation function independently [3]. They concluded that their work "suggested that the standard one-activation-function-fits-all approach may be suboptimal"[3].

## 3 Dataset and setup

The dataset used was the CIFAR-10 dataset, which "consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images." The images are colored so we the dimension of each image is 32x32x3 [4].

There are 6 batches total in CIFAR-10, five training batches and one test batch. Each batch has 10000 images. The test batch is obtained by randomly selecting 1000 images from each class. The training batches contain the rest of the images in a random order. Among the training batches, there are exactly 5000 images from each class.The 10 classes of the CIFAR-10 dataset are: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck.

The experiment was setup in a relatively simple manner, where there are essentially three components: 1) Unpacking, 2) Implementation, 3) Testing. In part 1, that CIFAR-10 dataset is unpacked and unpickled from its original array into images of size 32x32x3, and the data is normalized. Part 2 consists of the implementation part, which is the implementation of MLP and CNN to classify the image data. Here we run the classifier through multiple epochs of the CIFAR-10 training batches. Part 3 consists of the testing stage, here we test the trained classifier from part 2, on the testing batch of the CIFAR-Dataset.

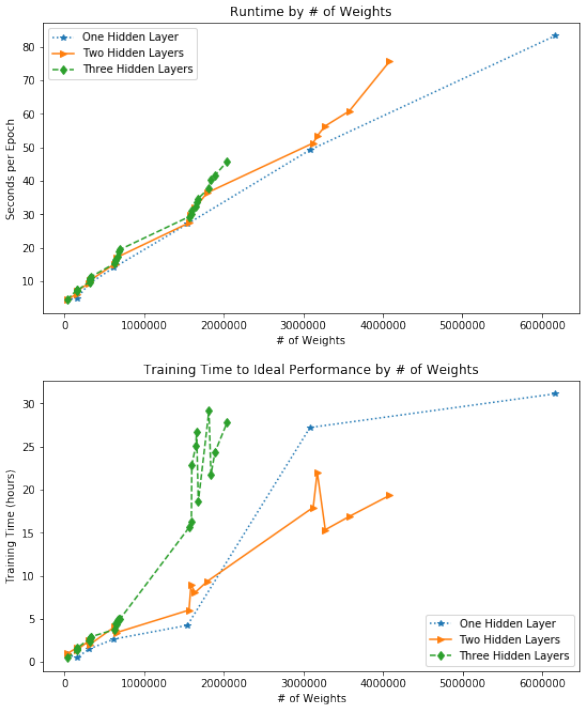



Figure 2: Top: seconds taken per epoch during training. Bottom: total hours needed to reach validation accuracy maximum.

# 4 Proposed approach

## 4.1 Multilayer Perceptron

The Perceptron is a historically meaningful model that prefaced the exploration of the complex biologically motivated models so prevalent today. It was initially proposed in 1958 as a computational manifestation of 'connectionist' theories for neural activity [5]. The Perceptron mimics this representation of memory learning the connecting weights such that if a layer of input represents the desired class (such as a cat or a dog), then it will be 'more strongly' connected to the corresponding output node. This idea is extensible to finding non-linear decision boundaries as well, by adding multiple layers between input and output. This is termed the Multilayer Perceptron [6]. It can also learn to recognize multiple different categories by adding more output nodes.

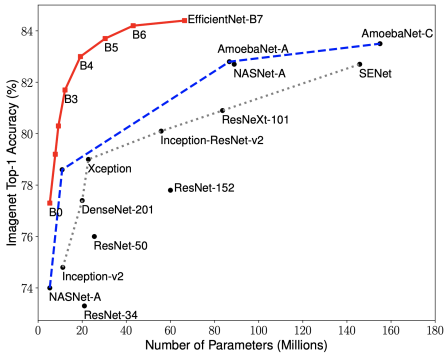



Figure 3: Impact of increasing parameters on test accuracy, models are derived from different literatures [7]

The Multilayer Perceptron was implemented from scratch in Python. The object instance is fully specified by providing the architecture (number of nodes in each layer from input to output), the number of epochs, the batch size, the learning rate, the hidden layer activation, the output layer activation, and the dropout ratio. The class supports any fully-connected architecture from the Perceptron, to a deep feed-forward neural network. It supports backpropagation with ReLU, Leaky ReLU, or sigmoid hidden layer activation functions, paired with sigmoid (binary) or softmax (multiclass) output activation functions. It also supports mini-batch gradient descent with any specified batch size which can be accompanied by any degree of dropout regularization.

In order to explore the pairing between the Multilayer Perceptron and the CIFAR-10 dataset, a range of 51 architectures were tested with 0-3 hidden layers all between 10 and 2000 nodes. All optimization was performed using a stratified train/validation split on the original CIFAR-10 training data before the best architectures were evaluated on the test set. Additionally, the relationships between runtime and validation accuracy with the number of parameters, the number of layers, the hidden layer activation function, and the dropout ratio were explored.

### 4.2 Convolutional Neural Network

The modern Convolutional Neural Network (CNN) is generally considered to have first been proposed by Yann LeCun in 1998 [8]. The network is motivated by the ability of simple filters to extract features like edges from an image via the convolution operation. A CNN architecture specifies the size of these filters, and learns the optimal weights to actually define the filters by backpropagation. These convolution layers are also complemented by pooling layers which reduce the dimensionality of data, such as by reducing adjacent features to their average or their maximum [6]. Finally, networks also often use standard fully connected layers to make predictions after the data has passed through a series of convolution and pooling layers.

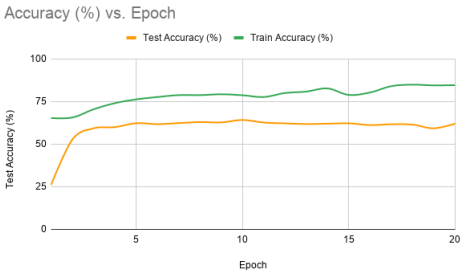



Figure 4: CNN accuracy vs epochs

The architecture of the CNN implementation consists of 2 convolution layers followed by 2 fully connected layers as shown in figure 8, the first convolution layer is a 2D Convolution with a kernel size of 5x5, this is then followed by 2x2 max pooling. After the image data is pushed through the convolution in the first layer the swish function is applied before the data is pooled. After the first layer the height and width of the data decreases from 35x35 to 14x14, whereas the number of channels increases during the convolution from 3 to 6. The second layer of the neural network is almost identical to the first, except the number of channels increases from 6 to 16 during the second convolution. After the two first layers, we form the first fully connected layer, where the data is flattened 400 units which are densely connected to 120 units. This layer is then fed into a second fully connected layer with 84 units. The second fully connected layer is then fed into the output layer with 10 nodes.

## 5 Results

### 5.1 Multilayer Perceptron

The Multilayer Perceptron was trained with 51 different architectures on the CIFAR-10 dataset. Each architecture was trained with a dropout ratio of 0.5, and a learning rate of 0.05 or 0.1. Depending on the complexity of the architecture, they were trained for 1000-3000 epochs.

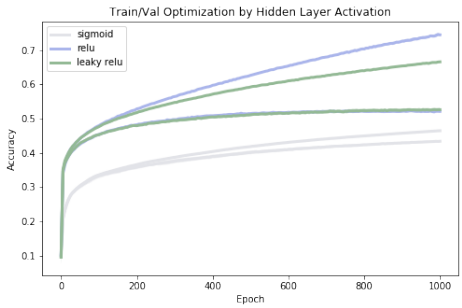



Figure 5: Impact of hidden layer activation function on optimization of a 200-200 hidden node architecture

The optimization pathway varied significantly depending on the architecture. A sampling of these are shown in Figure 6. These examples were chosen to showcase the influence of architecture on model training. On the left, we have no hidden nodes and have initialized a Perceptron. Since the data isn't linearly separable, we get caught in an optimization cycle as the decision boundary fails to stabilize. In the middle, we have a single hidden layer with 2000 nodes, which gives over 6 million parameters; this causes a huge gap between training and validation accuracy, showcasing a variance problem. Finally, with a middling model complexity on the right, we have less of a variance problem but the training accuracy fails to breach 60% in 1000 epochs – this model has issues with bias.

However, although variance issues become more significant with more parameters, in general increasing model complexity does give better performance on the test set. Figure 1 shows the 51 architectures plotted by their ideal validation accuracy against the number of weights included in the model. Regardless of depth, performance seems to move similarly with respect to the number of weights, reaching a ceiling at around 57%. Interestingly, increasing depth seems to yield little benefit over increasing width. This indicates that compounding non-linearities isn't useful for this problem – this might show that CIFAR-10 data isn't complex in a way that can be modelled by multiple hidden layers, and that performance is inherently limited by the fully-connected structure. In the end, the best performing architecture was the 1000-1000 hidden node architecture which achieved 56.97% accuracy on the validation set, and 56.01% accuracy on the test set.

To gain more insight into how model performance changes with increasing complexity, confusion matrices for each architecture were recorded as well. In Figure 10, a confusion matrix for one of the simplest architectures (10 hidden nodes) is compared against a more complex architecture (1000-1000 hidden nodes). In the simpler architecture, we

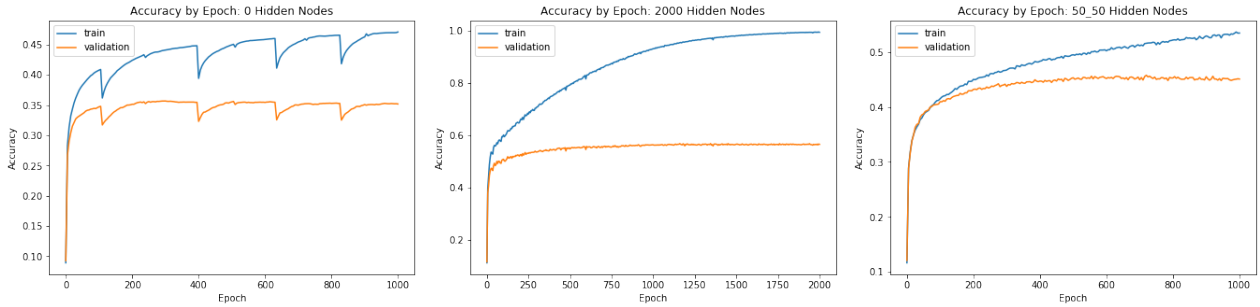



Figure 6: Optimization trajectory of various Multilayer Perceptrons. Architectures from left to right: 0 Hidden Nodes, 2000 Hidden Nodes, 50-50 Hidden Nodes

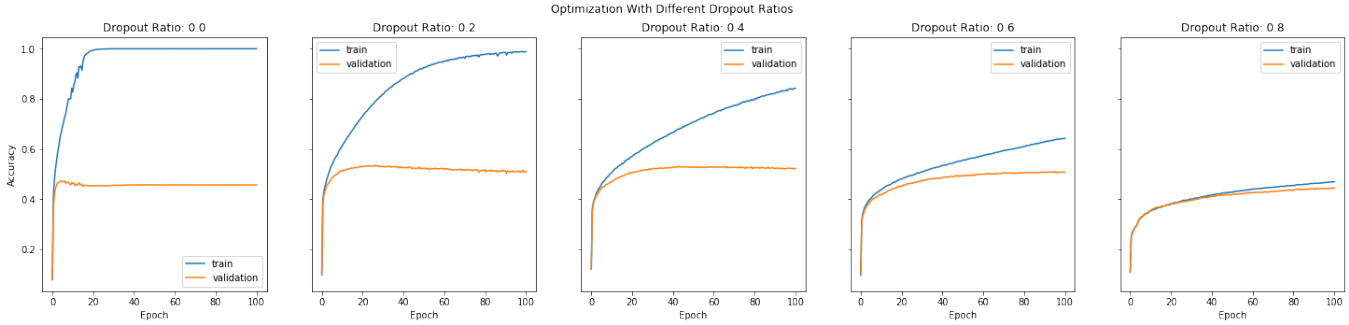



Figure 7: Impact of dropout ratio on optimization of a 200-200 hidden node architecture

see that the central patch of animal classes is the source of most problems. The network seems complex enough to recognize an animal, but doesn't have the ability to distinguish between them. As such, it resorts to deer, frogs, or dogs. With more weights, it's powers of observation increase, and this issue is ameliorated along with predictable confusion between ships and planes or trucks and cars.

We also explored how increases in model complexity impact the runtime for the training algorithms. In Figure 2, the seconds per epoch are shown against model complexity on the top. On the bottom, the total hours to reach the maximum for validation accuracy is shown for each architecture. As expected, the seconds per epoch roughly show the linear time complexity gradient descent has with the number of weights. However, this doesn't tell the whole story, as different architectures require different amounts of epochs to reach ideal performance. Notably, when increasing to three layers, the train time jumps significantly; the largest three-layer networks took over a day with less than half the weights of the largest single-layer network.

Finally, the effects of choices in dropout ratio and hidden layer activation function were compared. In Figure 7, different dropout ratios are compares on a 200-200 hidden node architecture. As expected, higher levels of dropout decrease the variance problem, but eventually cause issues with bias. In Figure 5, ReLU, Leaky ReLU, and sigmoid activation functions are compared on the same architecture. Leaky ReLU is used with a 'leak' of 0.1. The sigmoid activation function results in extremely slow training for both train and validation; this showcases the vanishing gradient problem of the logistic function for activations of large magnitude. The ReLU family performs identically on the validation set.

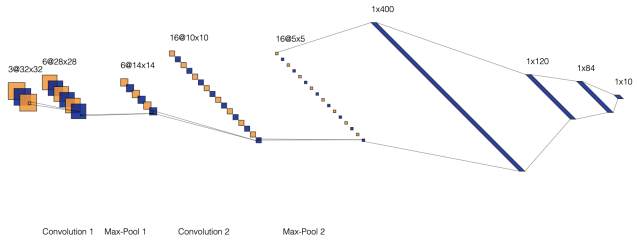



Figure 8: Diagram of Layers in CNN

## 5.2 Convolutional Neural Network

For the Convolutional Neural Network, it was apparent that there are a select few implementations that work well on the CIFAR-10 Dataset. Therefore,the CNN was implemented by referencing past literature. The CNN

implementation was a modified version of LeNet-5 by Yann Le Cun, where instead of the softmax activation function, the swish activation function is used [8]. The choice of Swish as an activation function was inspired by Google Brain, and it has been consistently shown to outperform or match ReLU. Other activation functions were also considered such as GELU or Leaky ReLU but overall swish gave the overall best performance [9].

```
Accuracy of plane : 64 %
Accuracy of   car : 73 %
Accuracy of  bird : 43 %
Accuracy of   cat : 43 %
Accuracy of  deer : 56 %
Accuracy of   dog : 53 %
Accuracy of  frog : 62 %
Accuracy of horse : 65 %
Accuracy of  ship : 77 %
Accuracy of truck : 66 %
```

Figure 9: Accuracy of individual CIFAR-10 classes

accuracy to cat and bird with 44% accuracy's.

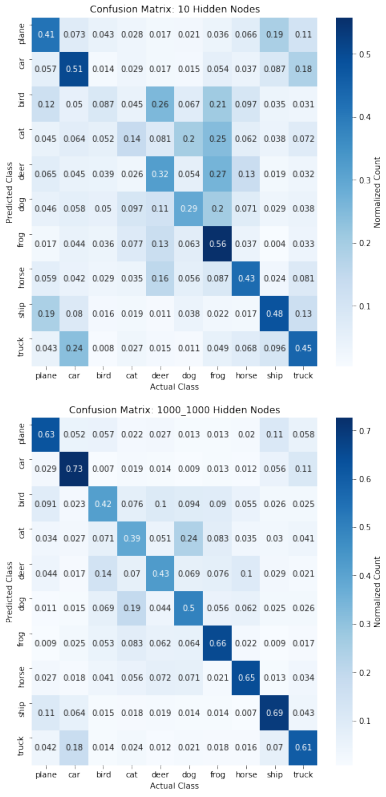



Figure 10: Left: confusion matrix for 10 hidden node architecture. Right: confusion matrix for 1000-1000 hidden node architecture.

As shown in figure 4 the convolutional neural network trained relative quickly. After 3 epochs the testing accuracy was 59.3%, which is around 92% of the highest train accuracy of 64.17%. After 20 epochs the training accuracy reached a plateau value of 62%. As for the testing accuracy, due to the effects of overfitting, the training accuracy was higher than the test accuracy and plateaued at 85%. The training accuracy is around 23% higher than the test accuracy; the CNN had some variance issues with the data.

The CNN had a slew of different training accuracy for various classes, as shown in figure 9, which varies from ship with a 77%

The relative simplicity of this convolutional neural network was the leading cause for the smaller number of epochs that were used before reaching a plateau. For a network with greater complexity such as ResNet, the network requires more epochs for training. More complex models with a greater number of parameters are able to reach higher test accuracy, granted that the CNN is properly built. This correlation is shown in figure 3. However as new techniques and methods are developed this correlation will likely weaken [10].

# 6 Discussion and Conclusion

This mini project yielded a lot of significant findings. For one, the limitations of a fully connected network on computer vision tasks were fully exposed. Regardless of extensive experimentation with model complexity, performance was capped at 56.97% on the validation set and 56.01% on the test set. It makes sense that a fully connected architecture is sub-optimal. For one, in any image many of the pixels are 'dead' or uninformative, while only a subset will contain the actual object of interest. A fully-connected architecture has trouble with a strategy for flexibly locating the important pixels. For example, a convolutional layer could use multiple filters to identify vertical or horizontal edges or structures of different colours, dynamically striding across each local region. A fully connected architecture is limited to one, static filter per local region; it might take the first layer to find all the vertical edges, but that's not enough information to describe an image.

Within our study of the MLP we saw how the different hyperparameters make a distinct impact on the optimization process. Increasing weights led to a linear increase in runtime per epoch; increasing dropout reduced variance in exchange for more bias; using a sigmoid hidden layer activation slowed training to a crawl due to vanishing gradients. This showed how training an MLP requires extensive, computationally intensive optimization to come up with the right hyperparameter recipe. Future directions for this component would be to explore the extremes of model complexity; this was limited in the project time frame due to computational expense. Additionally, a more thorough grid search over architectures in conjunction with dropout ratio would have been ideal.

# 7 Statement of Contributions

Kenji Marshall implemented and ran experiments for the Multilayer Perceptron. Hamza Mian implemented and ran experiments for the CNN. Everyone contributed equally to the report.

# References


[1] Mark D McDonnell and Tony Vladusich. Enhanced image classification with a fast-learning shallow convolutional neural network. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2015.

[2] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[3] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[4] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[5] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] *PapersWithCode*. PapersWithCode. URL `https://www.google.com/url?sa=iurl=https://paperswithcode.com/sota/i` 1587408288606000$source = imagescd = vfeved = 0CAIQjRxqFwoTCLiPouuS9egCFQAAAAAdAAAAABAD$.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665`.

[9] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2018. URL `https://openreview.net/forum?id=SkBYYyZRZ`.

[10] Simone Bianco, Rémi Cadène, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *CoRR*, abs/1810.00736, 2018. URL `http://arxiv.org/abs/1810.00736`.