Statistical Methods for Machine Learning Hyperparameters on Neural Networks

Enes Sezgin

July 2022

I declare that this material, which now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Introduction

Neural networks are a learning model that mimics human brain structure. It has neurons, layers, and connections between them. Layers and connections are used for generating a prediction by using our data. Hyperparameters are variables that allow us to adjust our model to our needs. These hyperparameters can be batch size, epochs, number of layers and neurons, optimizers and learning rates.

This project is based on the classification of cat and dog images. I feed the model with around 25000 different cat or dog images, then test the model to reach minimum loss and maximum accuracy. Using this context, hyperparameters are examined on two different neural network structures, multi-layer perceptron(MLP) and convolutional neural networks(CNN). Different hyperparameters are tested with various values and conclude how they affect the outcome.

Project is created using TensorFlow 2[1] library and Keras[2]. To test the model, I used 5-fold cross validation and get folded dataset using sklearn[3] library.

2 Default Values

Before testing different hyperparameters, I determined some default values for both structures. The reason for that is that we need some values to compare and reach a conclusion.

For MLP, you can see the default values. To train the model, I rescaled all the images to 100x100 and used grayscale. However, despite all configurations, MLP is not performant enough to successfully predict in this project. The reason is that MLP is not advanced enough to distinguish complex image patterns. So loss values will always be high.

For CNN, you can see the default values. To train the model, I rescaled all the images to 64x64 and used RGB values. I could not use higher resolution because of the lack of memory space. Since 100x100x1 and 64x64x3 are close to each other, I basically used similar memory space while training for both structures.

Table 1: Default Values for MLP

Hyperparameters	**Default Values**
Image Size	100x100x1
Batch Size	32
Number of Epochs	20
Hidden Layers	1
Number of Neurons	64
Learning Rate	0.001
Optimizer	ADAM

Table 2: Default Values for CNN

Hyperparameters	**Default Values**
Image Size	64x64x3
Batch Size	32
Number of Epochs	20
Hidden Layers	3
Number of Neurons	64
Learning Rate	0.001
Optimizer	ADAM

I wanted to start with the epoch for testing hyperparameters because selecting an appropriate epoch will help to compare hyperparameters better and help to identify overfitting and underfitting values.

3 Epoch

Epoch in machine learning means one complete pass of the training dataset through the algorithm. It is one of the most critical hyperparameters because we can observe underfitting or overfitting via epoch count. Underfitting and overfitting are terms used for describing the performance of our model. When our model underfits, that means that the loss of training and testing are both high, which can be interpreted as our model did not learn enough from our dataset. So we can increase the epoch count. On the other hand, when our model overfits, that means the loss of our training is low, but the loss of testing is high. That can be interpreted as our model fitted too much to our dataset, and it does not have the ability to correctly interpret data from outside. So we should decrease the epoch count to prevent this.

To be able to observe our new hyperparameters while tuning, we need to find a suitable epoch count interval. By doing that, we will achieve the opportunity of seeing critical points on the graph. Hence, I tested starting models with a high interval of epoch count, and

detected a suitable interval.

For MLP, I tested epoch values 5 to 40. As you can see in Figure 1 after around epoch n 10, loss values started to separate. This is the optimal value with loss 0.68. So, I will observe other hyperparameters around this value. However, as I mentioned before, the loss values are both high for training and test loss.The reason is that the model is inadequate to process the dataset; therefore, it underfits.

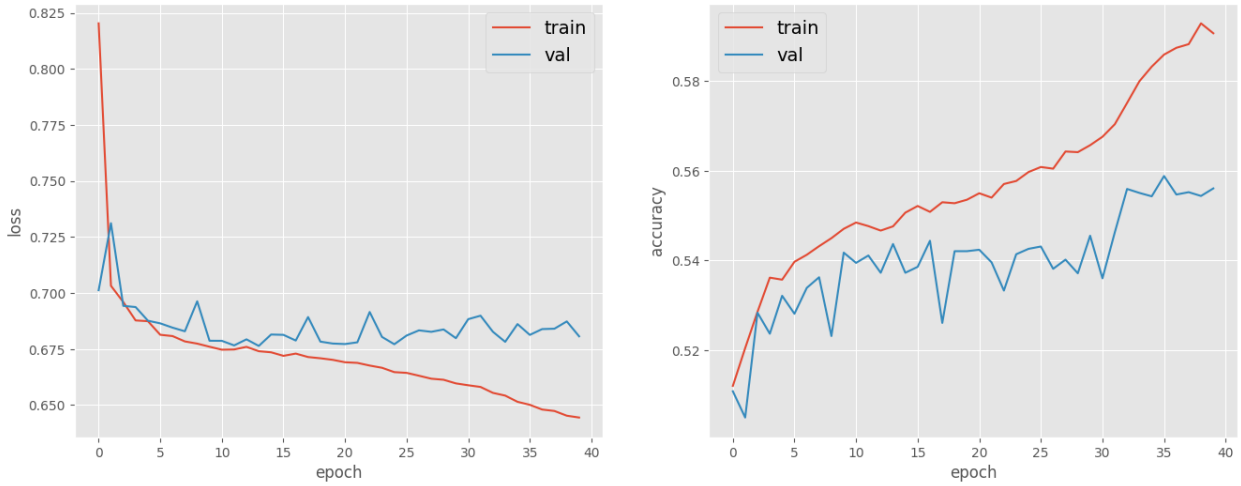


Figure 1: Impact of Epoch on MLP

For CNN, I tested epoch values 5 to 50. As you can see Figure 2 after around epoch 9, loss values started to separate. This is the optimal value with loss 0.36. I will consider around this value. The values are promising with low loss and high accuracy.

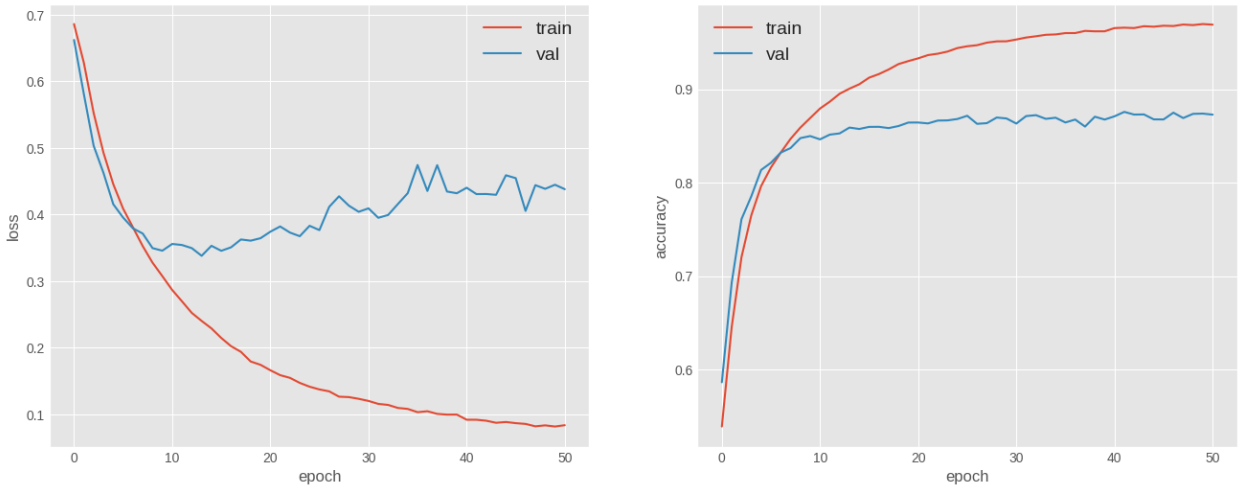


Figure 2: Impact of Epoch on CNN

For next step, I continue with another important hyperparameter: batch size.

4 Batch Size

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. To give an example, let's say we have 100 data samples. If we have batch size 25, we divide our dataset by 4, and after every 25 samples, we update our model.

The importance of batch size is related to our algorithm: gradient descent. Gradient descent is a first-order iterative optimization algorithm for finding the local minimum of a function. It is used during the back-propagation step of training our model. Back-propagation tries to find weights that provide a local minimum, or at least try to move values near to the local minimum. The batch size shows its effect at this point. Choosing a big batch size provides more approximated change in the way of local minimum; however, selecting a small batch size can provide drastic changes depending specific samples of current batch. Hence, the batch size is one factor that determines the stability of the learning process.

In my experiments, for CNN, I tested a wide range of batch sizes (32 to 256, 32 interval). As you can see in Figure 3, our test loss increases as the batch size increases. You can see the difference of training and loss values between batch size 32 and 256 in Figure 4. With less loss and more accuracy, smaller batch size is more optimal.

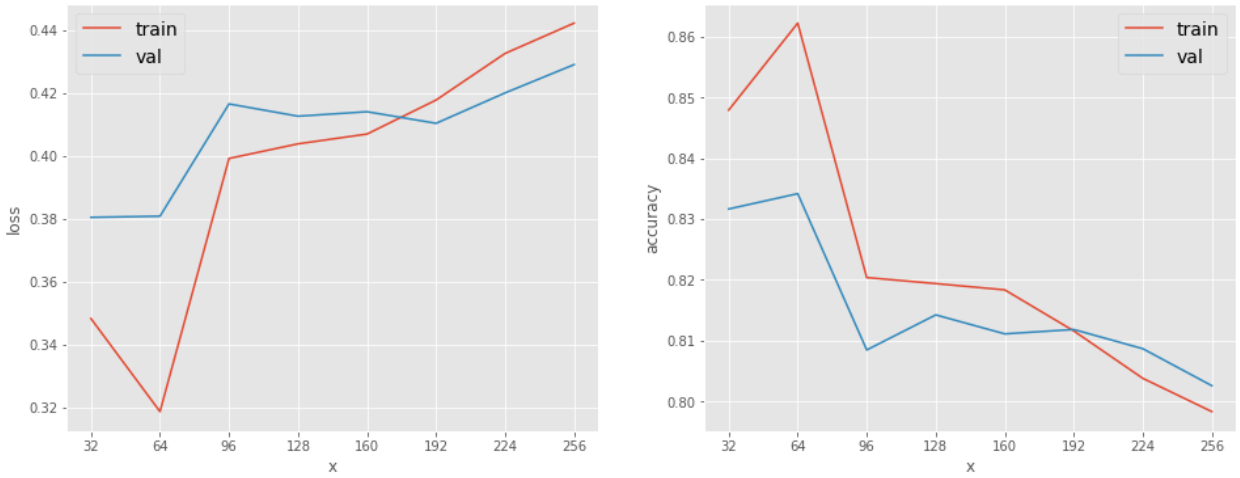


Figure 3: Batch Size on CNN

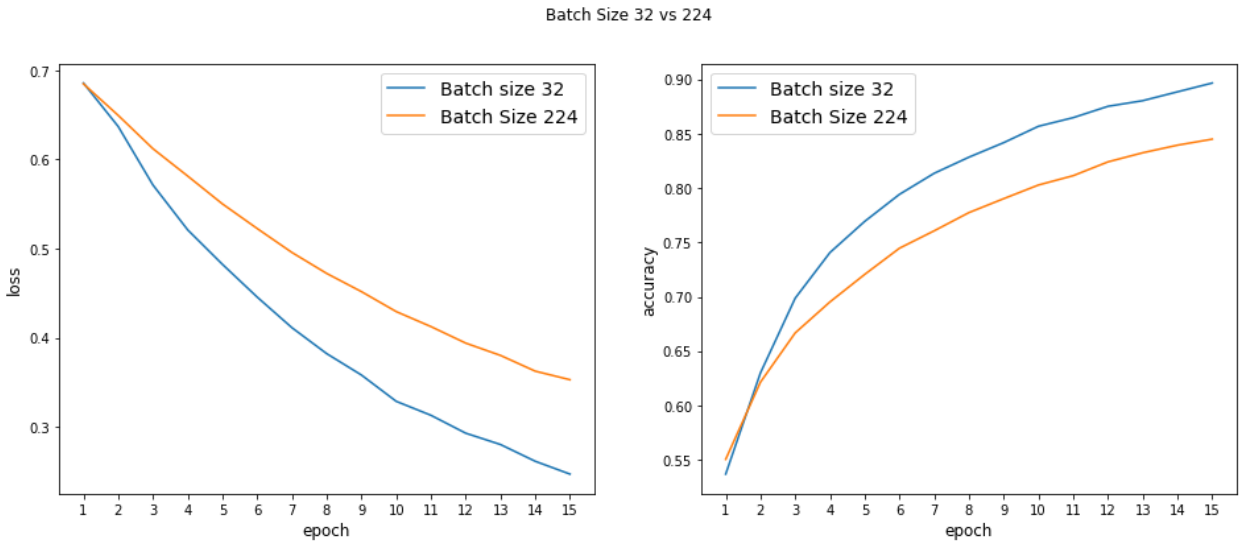


Figure 4: Batch Size 32 vs 224

To be able to analyze better, I tested again with smaller intervals on smaller sizes (16 to 80, with 16 intervals). You can see the graph in Figure 5. It seems like 32 is a good number to choose, because it has smallest loss with around 0.32.

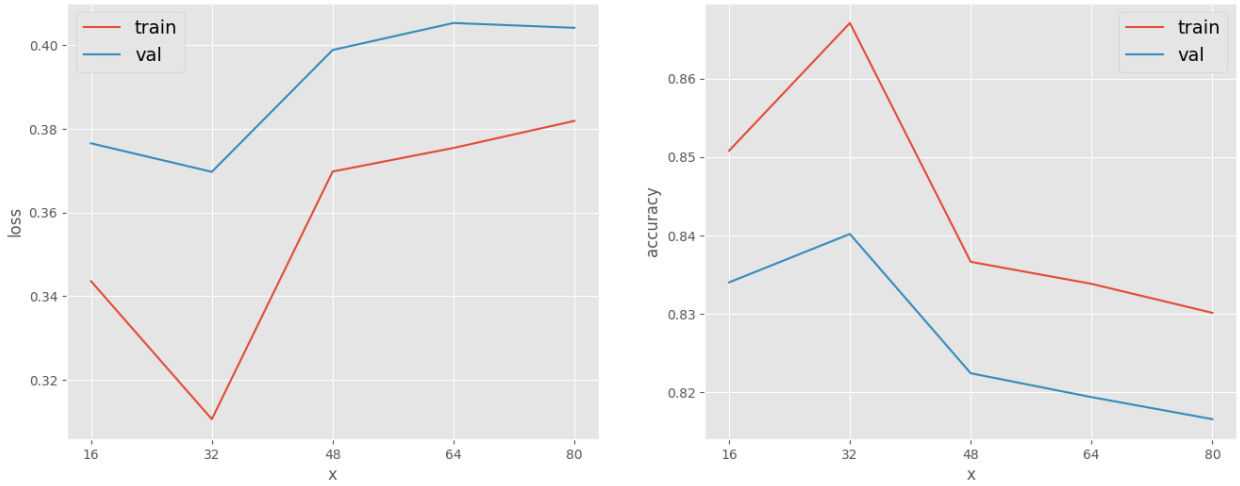


Figure 5: Batch Size on CNN with smaller intervals

Another drawback of big batch size is memory limit. While experimenting, batch size 256 caused crashes due to use of high gpu memory. So smaller batch size is good in terms of memory usage.

For MLP, I tested values between 16 and 112. There was no significant changes on final loss values of model. You can see the graph in Figure 6.

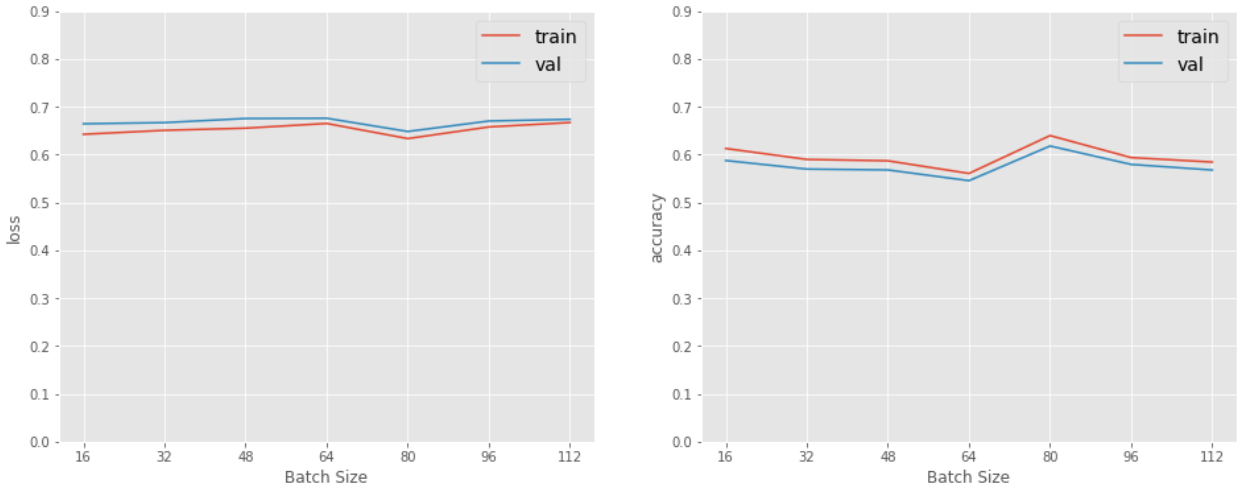


Figure 6: Batch Size on MLP

With considering these experiments, I decided to continue testing other hyperparameters with epoch 10 and batch size 32. Next hyperparameter I will examine is about structure of neural network models.

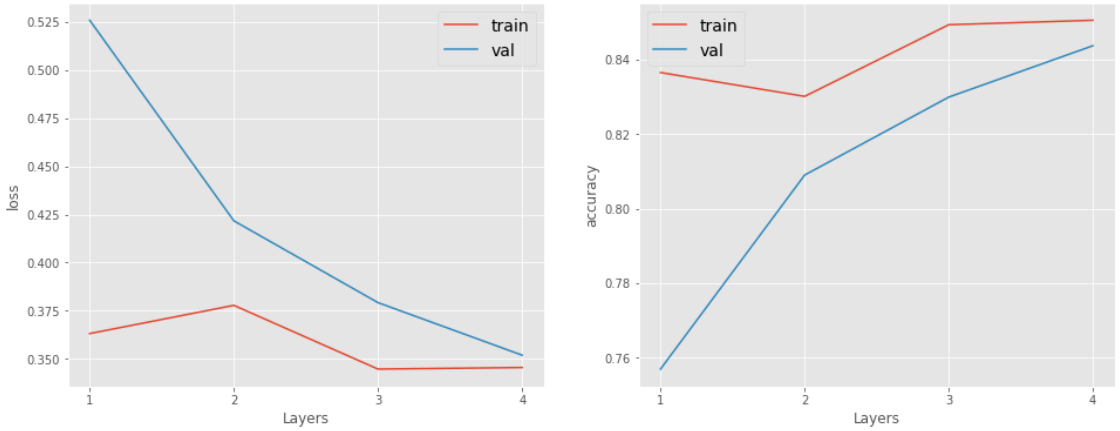
5 Layers

5.1 Number of Layers

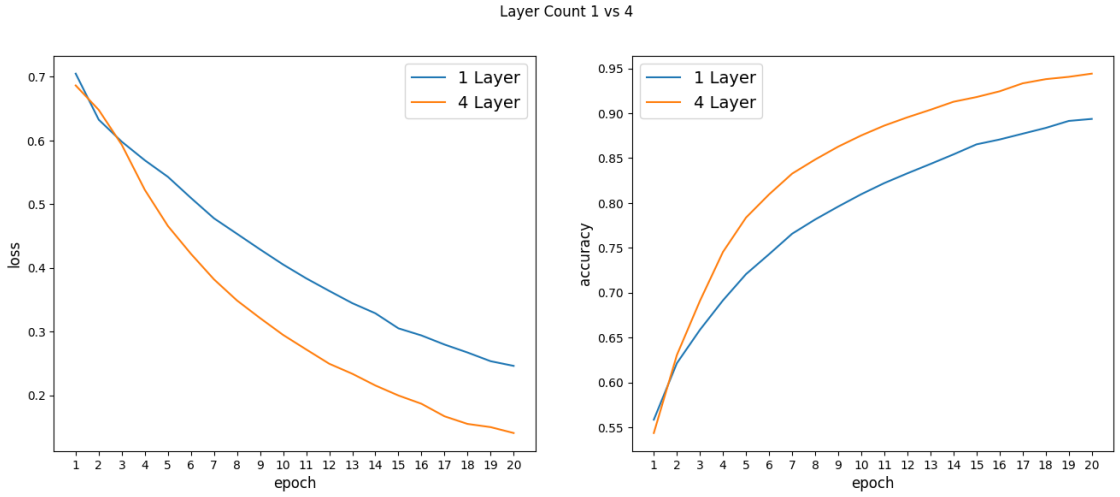
Layers in neural networks are node groups that are stacked together. Layers are connected one by one with weights and influence each other. There are three types of layers. Firstly, the input layer is where we put our inputs. The second type is the output layer. This layer gives our model's prediction. Lastly, hidden layers are between input and output layers, and their nodes take in a set of weighted inputs and produce an output through an activation function.

The hidden layer count is significant because "training" means adjusting the variables of hidden layers. A number of layers less than we need can lead to limitations on learning due to inadequate complexity; a layer count more than we need can lead to overcalculation and waste of resources.

For testing the layer count on CNN, I created a module consisting of a convolutional layer and a max pooling layer. I tested by changing the count of this block. The tested values are 1-4. Bigger values were not possible because convolutional layers take chunks of the image in every step of CNN to create an input for the next layer. Each chunk gives a value, creating a new image-like input with these values. Since input count is reduced in every step, 5 convolutional layers were impossible to test in this configuration. As you can see in Figure 7a, best loss value was at 4 hidden layer with value 0.35 and you can see comparison between 1 layer and 4 layer in Figure 7b.



(a) Hidden Layer Count on CNN



(b) 1 Hidden Layer vs 4 Hidden Layer on CNN

For testing the layer count on MLP, I tried 1 to 10 layer count. As you can see in Figure 8, layer count does not change loss significantly and losses are still high. I will continue experimenting using 4 hidden layers for both structure.

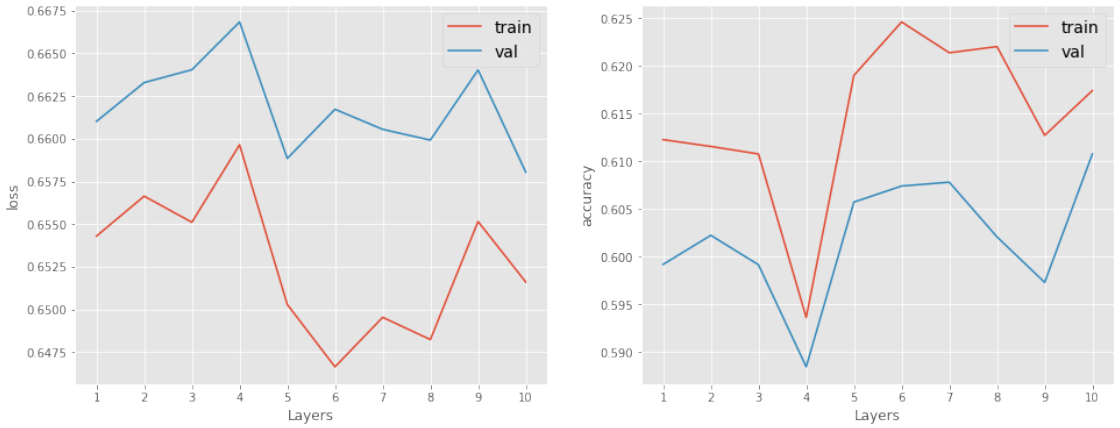


Figure 8: Hidden Layer Count on MLP

5.2 Number of Nodes

Nodes in neural network are units that have weighted connections and transfer functions that combines the inputs and gives outputs. Nodes are then organized into layers to comprise a network. Number of nodes is important because for example, only one node in a layer will undercut prediction ability because only one variable decides on our output.

For CNN, I changed node count in each layer. The node counts are between 16 and 128, with interval 16. As you can seen in Figure 9, low node count causes underfitting due to low capability of training. In these situations 112 for layer count seems like a good value to choose for with loss value 0.335.

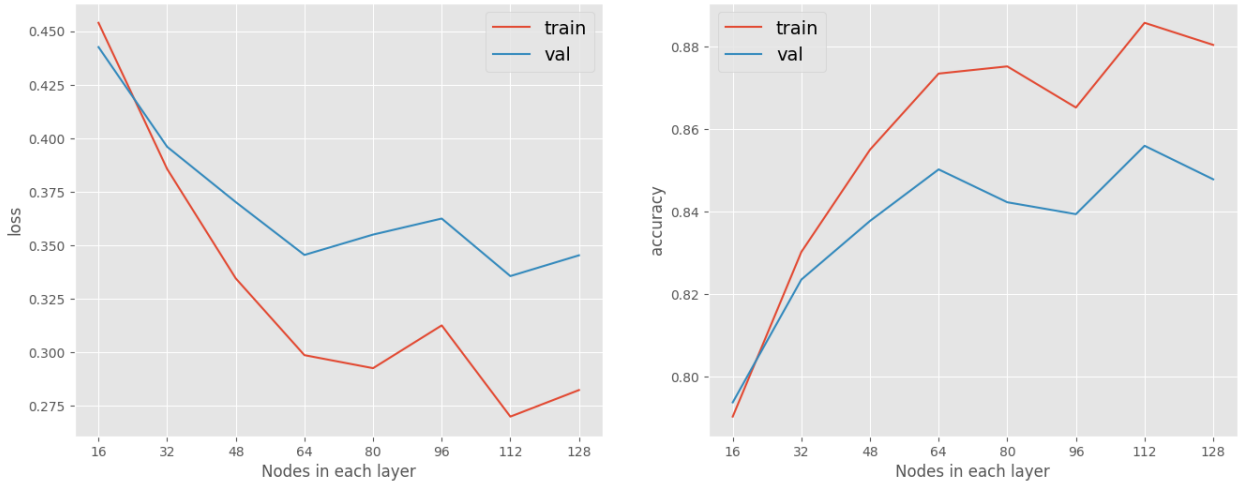


Figure 9: Node Count on CNN

You can see difference of training process between node count of 16 and 112 in Figure 10.

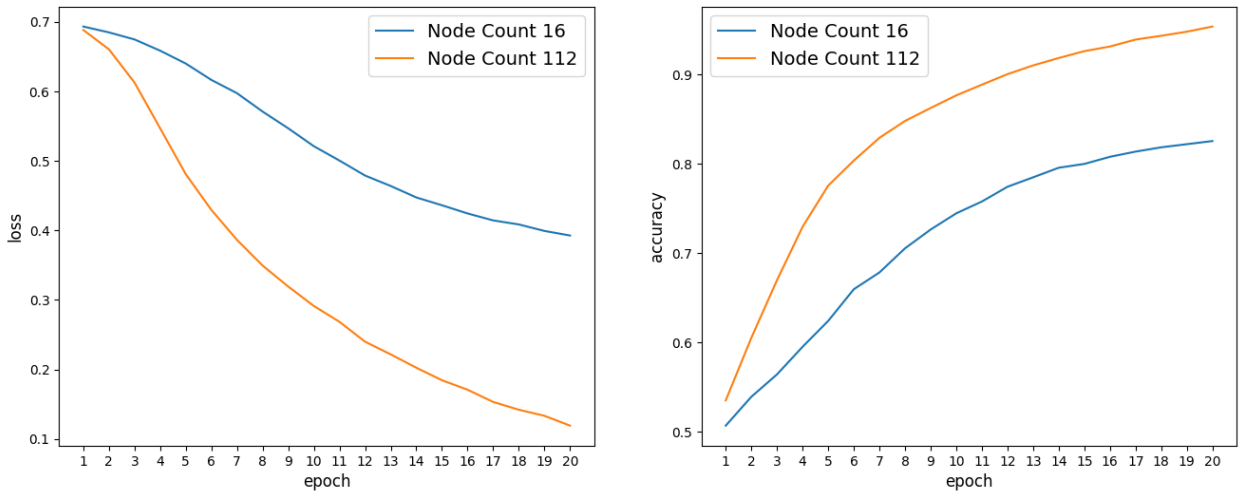


Figure 10: Node Count 16 vs 112 on CNN

For MLP, I tested node counts 16 to 128 with interval 16. As you can see in Figure 11, there are little improvements with bigger node counts, but still MLP has very high loss.

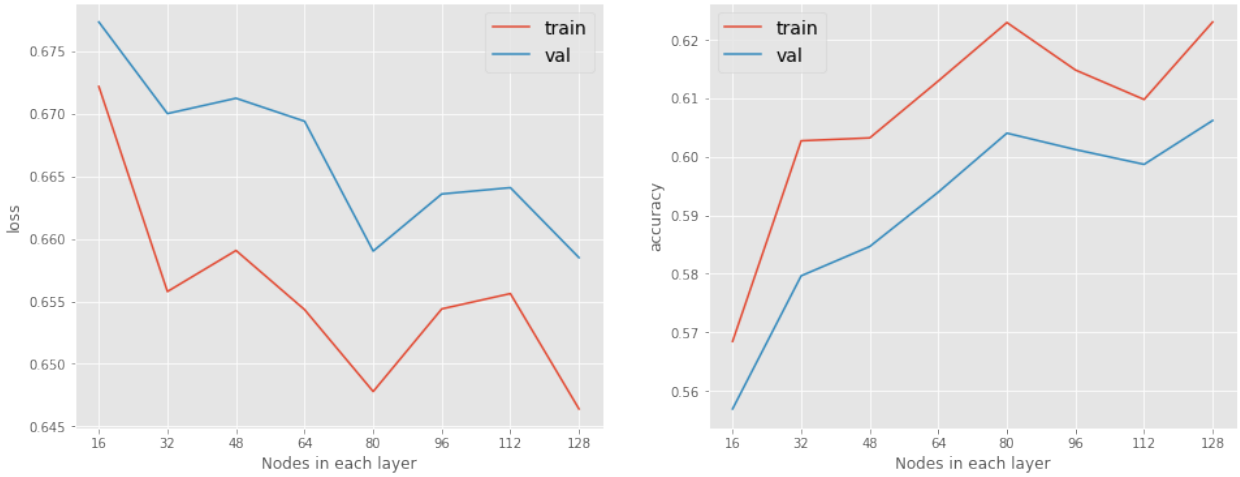


Figure 11: Node Count on MLP

In conclusion, I will continue the experiment with 4 hidden layer and 112 nodes for each structure.

6 Optimizers and Learning Rate

As I mentioned before, neural networks are trained using the gradient descent algorithm. With the number of samples determined by the batch size, we calculate the direction we should go to reach a local minimum of the function. We decide how far we will go in this direction with the learning rate. In other words, the amount that the weights are updated during training is referred to as the learning rate. Hence, the learning rate controls how quickly the model is adapted to the problem. A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck.

Optimizers in neural networks are algorithms that adjust variables like weights and learning rates to optimize the model. Actually, gradient descent algorithm is an optimizer. Since optimizers determine how the model will be efficient and how to lower the loss, it is a crucial variable.

To test the learning rates on CNN, I selected an optimizer, ADAM. I then picked a range to test, which is 1e-5 to 1. As you can see in Figure 12, when the learning rate is too small and too big, the results are not good in terms of accuracy and loss. So, I will continue with the 0.001 learning rate and test the optimizer algorithms.

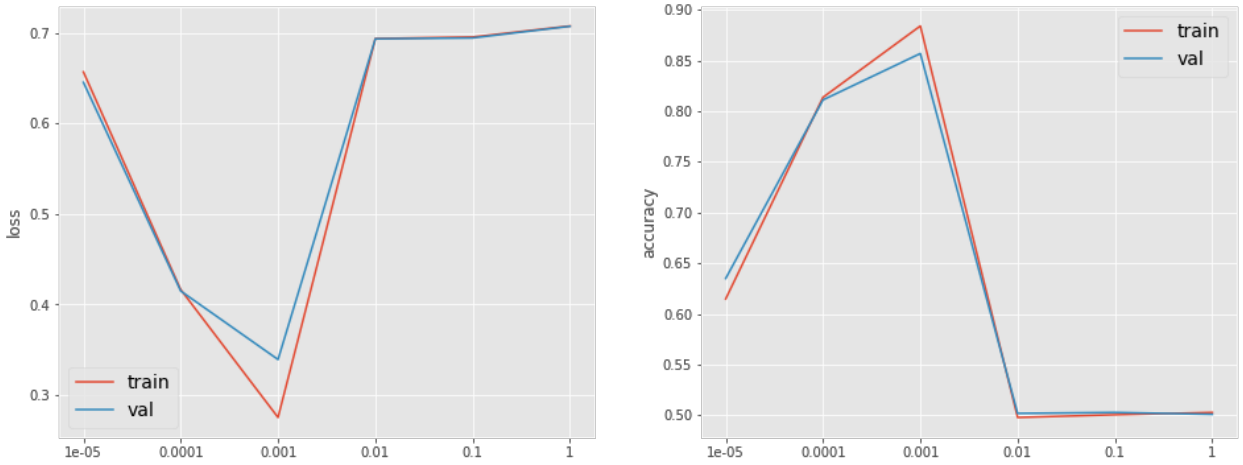


Figure 12: Learning Rates on CNN

To test optimizers on CNN, I selected 3 optimizers: ADAM, ADAMAX, and SGD. As you can see in the Figure 13, ADAM and ADAMAX are close to each other and better than SGD. In conclusion, choosing ADAM or ADAMAX and 0.001 learning rate will be the best option.

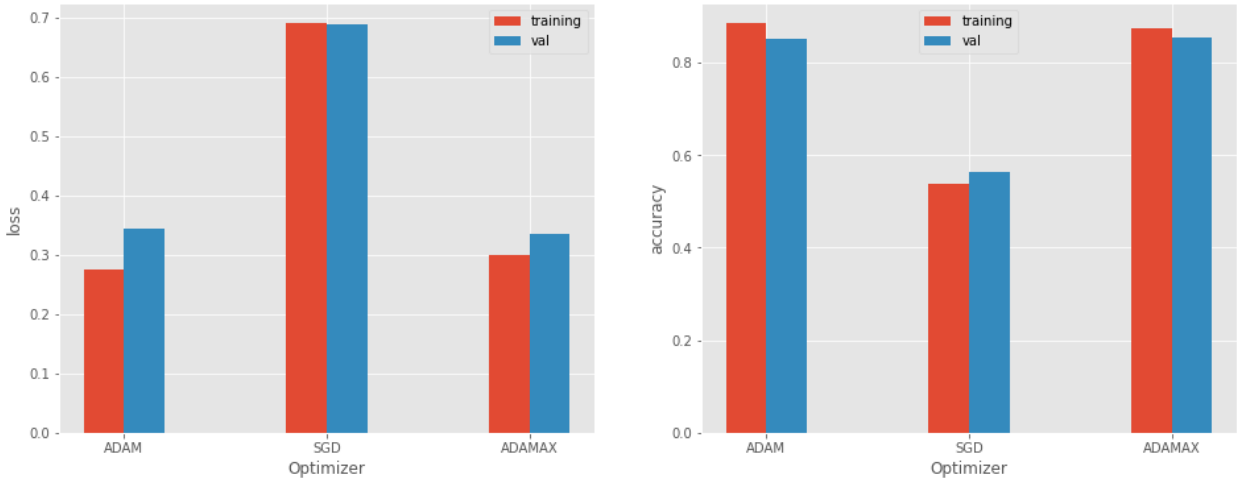


Figure 13: Optimizers on CNN

For MLP, I again used 0.001 learning rate and tested optimizers. As expected, there was no improvement in models when optimizers changed. You can see this in Figure 14.

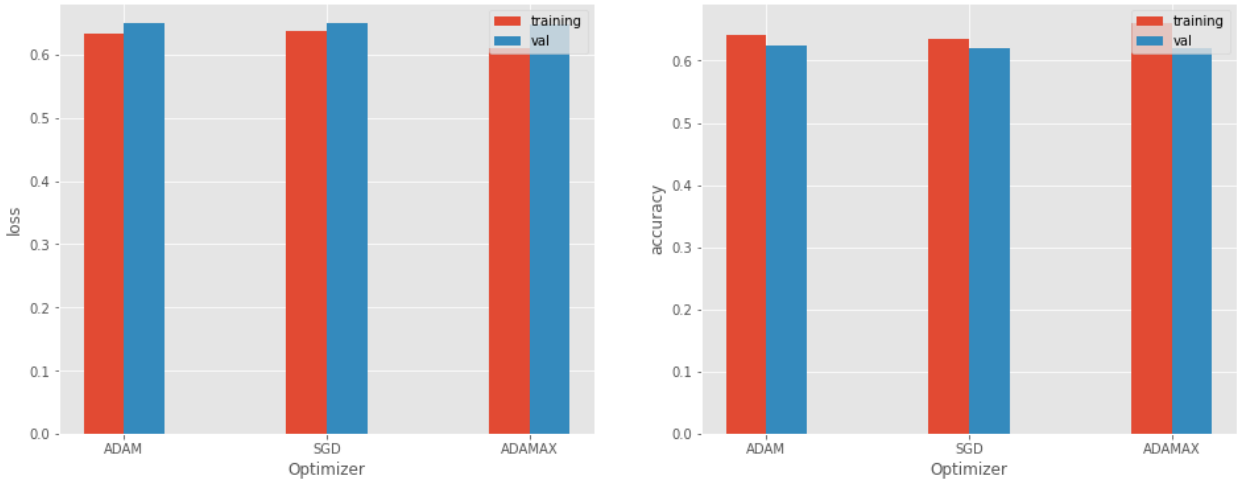


Figure 14: Optimizers on MLP

7 Conclusion

In conclusion, two neural network architecture responds differently to hyperparameters. And each hyperparameter's impact on the model accuracy is different. If we compare the two architectures, CNN was the most suitable solution to solve the image classification problem. Its accuracy reached up to 83% in the last tests. On the other hand, MLP reached a maximum of 62% accuracy, and its response to hyperparameter changes was not stable.

For further experiments, image input resolution, activation functions and dropout rates can be examined.

References

[1] Martin Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.*, Software available from tensorflow.org. 2015. url: https://www.tensorflow. org/.

[2] François Chollet et al. *Keras*, https://keras.io. 2015. API design for machine learning software: experiences from the scikit-learn project, Buitinck et al., 2013.

[3] Buitinck et al. *API design for machine learning software: experiences from the scikit-learn project*, 2013.