Hyperparameter Tuning in Neural Networks with Cross Validation

Berke Turanlioglu

I declare that this material, which I 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.

Abstract

This project experiments with different hyperparameters in different neural network architectures to understand what their values should be under different conditions. The dataset that neural networks worked on both the grayscaled and colored images of cats and dogs. To lower the loss and raise the accuracy for the unseen data, hyperparameters are tuned in neural networks with 5-fold cross validation. Their averaged value got discussed to pick the best. The results showed that hyperparameters should not be the same for every type of dataset and neural network. They act different and have unique results at every iteration. Thus, it was simply proved that the hyperparameter tuning is a crucial thing to be applied for every neural network to get the best result for its case.

1 Introduction

Neural networks are the similar structures to human brains with their neurons and have numerous connections between them to predict over a topic they specialized on. There are several neural network architectures with different connections or dimensions, for instance. In general, these networks learn the data with many iterations over and over, and aim to guess the unseen data with the minimum error margin.

Before the programmer teaches and feeds the neural network with enough data, there are some properties called *hyperparameters* to set before starting. Number of epochs, batch size, learning rate, number of neurons and optimizers are some of the examples to these hyperparameters.

In this project, two different neural network architectures, which are multilayer perceptron and convolutional NN, learned and tried to predict whether the shown image is cat or dog with highest accuracy and lowest loss as possible. While doing it, several hyperparameters were tweaked to experiment whether which hyperparameter should be what according to its results.

2 Methodology

Dataset is taken from the university's website. It consists of approximately 25000 images of cats and dogs, equivalently. After removing couple of images that were broken, the dataset was prepared with their labels as binary classification, where an image labeled as 0 if it is a cat, 1 if a dog. One can note that these images were downscaled to 64x64 pixels to compute faster. Since they were in order of cats and dogs, images were then shuffled with their labels in order to make neural network models learn better. In addition, an array for the grayscaled images were also created for the multilayer perceptron.

Two different neural network architectures have been analyzed, which are multilayer perceptron (MLP) and convolutional neural network (CNN). These networks were constructed with Keras [1] library under TensorFlow [2]. For the purpose of this project, they were used for hyperparameter tuning with 5-fold cross validation. To tune, GridSearchCV method from scikit-learn library [3] was used.

Several parameters were looked into to optimize the neural networks. For MLP, six parameters were experimented in pairs. They are the number of epochs and batch size, optimizer and its learning rate, activation function and the number of neurons, respectively. In each step, their best values were taken and implemented as a fixed value for the next tuning operation.

For CNN, in addition to the hyperparameters that were mentioned before, dropout value was tuned as well. However, due to memory issues, they could not be examined in pairs, but one after another. Lastly, the final version of the CNN with the hyperparameters that gave the best accuracy were run to see the final result.

3 Experimental Results

3.1 Multilayer Perceptron

Firstly, MLP model were created and tuned afterwards. First run was done and the hyperparameters can be seen in Table 1. Third fold was the most accurate one and if we look further into the scores of this fold (see Figure 1), accuracies and loss values for both training and test sets are not that successful. For the training set, while the accuracy is at most approximately 0.63, its loss value is 0.6429. For the test set, which was initialized as the testing fold, the accuracy is 0.618 and the loss value is 0.653.

Since there is no incline for the loss value after some epoch, we can say that there is no overfitting. Plus, there might be an underfitting since all the scores are a bit lower than usual. Hence, the aim with the hyperparameter tuning is to improve the accuracies and decrease the loss values as much as possible.

Hyperparameters	Values
Number of epochs	30
Batch size	60
Optimizer	SGD
Learning rate	0.001
Activation function	ReLU
Number of neurons	256

Table 1: Initial run of the MLP.

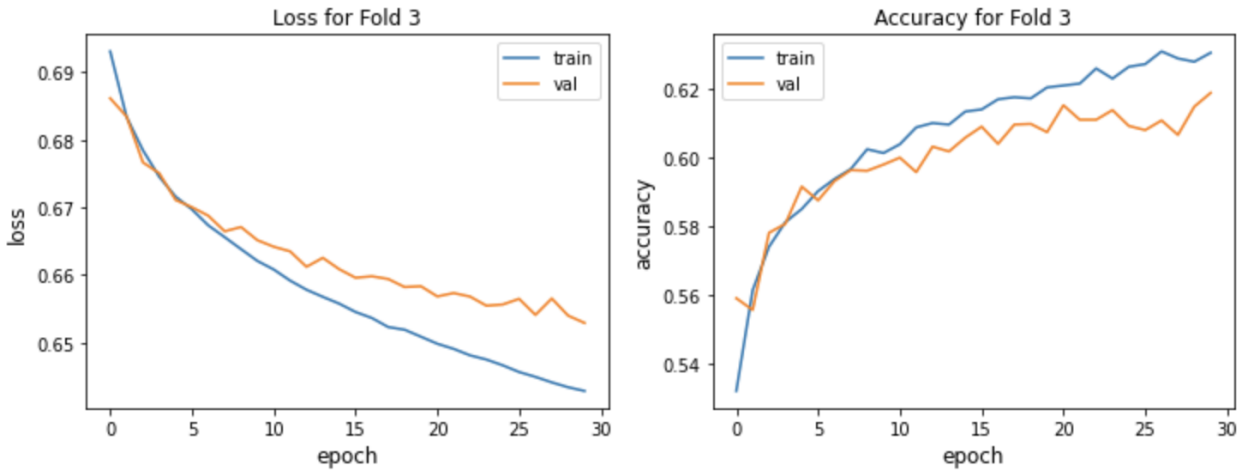

Figure 1: Graphs for loss values and accuracies of training and test sets for the MLP.

The number of epochs and the batch size were the first two to be tuned for this model. They were meant to be 20, 30, 40, and 40, 60, 80, respectively (see Table 2). With grid search, the algorithm looked for every combination of them and pulled out the most accurate one out of them.

One can see from the Table 2 that it gives the best accuracy when the number of epochs is equal to 30, and the batch size is 80. In addition, the model behaved worse when there are 20 epochs. This situation also implies that the model lacks learning well and there is no overfitting. On the other hand, there is not a strict discrimination in terms of the batch size. All batch sizes acted similar to each other with different number of epochs.

Apart from not getting much improved model, the loss value dropped to approximately 0.56. Getting a decrease from 0.64 to 0.54 shows that the tuning is on the right way and has to improve more.

After this iteration, all other tuning tests were done with the fixed epochs and batch sizes, since the most optimal ones were found. Then, the next step was to find the optimizer and its learning rate that gave the most accurate result. Parameters for the optimizers were SGD, Adam, and Adamax. For the learning rate, possible candidates were $1e$-3, $1e$-2, $2e$-2, $1e$-1.

# of epochs	Batch size	Accuracy
30	80	0.6447
40	40	0.6385
30	40	0.6350
40	80	0.6338
30	60	0.6334
20	60	0.6309
20	40	0.6308
20	80	0.6297
40	60	0.6125

Table 2: Tuning of the number of epochs and the batch size for the MLP.

Results (see Table 3) for this grid search were not developed much as it was desired. While it might be expected to have a better result than before since the model has two tuned hyperparameters at least, it is not the case according to the results. It proved that the SGD that has been already used initially was the true optimizer to be used. However, its learning rate should be $1e$-3, according to the grid search results.

Optimizers	Learning rates	Accuracy
SGD	0.0001	0.6408
SGD	0.01	0.6369
SGD	0.001	0.6348
Adamax	0.01	0.6256
Adamax	0.002	0.6244
Adamax	0.001	0.6226
SGD	0.002	0.6224
Adamax	0.0001	0.6194
Adam	0.01	0.6181
Adam	0.0001	0.6178
Adam	0.001	0.6153
Adam	0.002	0.6090

Table 3: Tuning of the optimizers and their learning rates for the MLP

The unorthodox thing here is that the previous model's accuracy (0.6447) is better than the current model's accuracy (0.6408). Previous model was calculated with SGD with 0.001

learning rate. However, current model calculated the accuracy as 0.6348 with the same hyperparameters. In conclusion, new best value was taken into consideration and new fixed learning rate became $1e$-3 according to the latest results, since the learning rate $1e$-2 can create lower accuracy results.

Loss values were also considered to pick the best learning rate, and they were both around 0.57 in both cases. Hence, they did not influence the final decision by oneself. Also, there is no proper development in terms of decreasing the loss while we are tuning the hyperparameters.

Activation functions	# of neurons	Accuracy
ReLU	384	0.6426
ReLU	256	0.6376
ReLU	64	0.6323
tanh	64	0.6289
ReLU	128	0.6279
tanh	256	0.6245
tanh	128	0.6228
tanh	384	0.6099
sigmoid	64	0.6020
sigmoid	256	0.5977
sigmoid	128	0.5954
sigmoid	384	0.5945
softmax	64	0.5825
softmax	128	0.5799
linear	64	0.5625
linear	384	0.5582
linear	256	0.5556
linear	128	0.5541
softmax	256	0.5481
softmax	384	0.5401

Table 4: Tuning of the optimizers and their learning rates for the MLP

Finally, last two hyperparameters were checked together, which are the activation functions and the number of neurons. Candidates for the first hyperparameter are softmax,

ReLU, tanh, sigmoid, and linear, respectively. For the second hyperparameter; 64, 128, 256, and 384 neurons were tested in a grid search.

In Table 4, accuracy values show that the model with ReLU is the fastest learner even with different number of neurons. Later, tanh follows it with significantly better performances with lower number of neurons at each step. It got better accuracy every time the number of neurons decreased. However, it is the opposite case for the ReLU. It has the most neurons at the top of the table.

Other three activation functions failed to learn the dataset more accurately. However, there is no strict thing as they cannot learn. They might work better with higher number of neurons, or they might learn with lower batch sizes and higher epochs since they are slower learners. In conclusion, ReLU is the best activation function that fit into our case with other previously fixed parameters.

Briefly, the hyperparameters that were changed after these tuning operations are the batch size, from 60 to 80, the learning rate, from $1e$-2 to $1e$-3, and the number of neurons, from 256 to 384. For the training set, accuracy jumped from approximately 0.63 to 0.7068, and loss value decreased from 0.6429 to 0.5670. In addition, for the test set, accuracy increased from 0.618 to 0.6426. These results show that there are certain improvements, yet the final values are not very reliable for a good neural network. This behaviour can be explained with the complexity of the data. Since the images were grayscaled and there could be very close gray colored squares, this dataset could be hard to work on with MLP. Therefore, next architecture that was experimented is the convolutional NN, which can handle RGB channeled images and recognize the images better.

3.2 Convolutional Neural Network

After experimenting the grayscaled image processing, convolutional NN was constructed to process the images with their RGB channels. Initial run was done with the parameters that can be seen in Table 5. For the training sets, average accuracy of the each fold is 0.8943, and the average loss value is 0.2511. For the test sets, the values are 0.8352 and 0.3903, respectively.

In Figure 2, there are the loss and accuracy values of the most accurate fold, which is 4. Compared to the first MLP graphs, this model is more leaning to overfit if it gets trained with more training data. Thus, the purpose is not to cause any single result of overfitting while increasing the accuracy. Loss value should decrease at the last epochs. If it bends and starts to increase after some epoch, then it would surely be an overfitting case.

Hyperparameters	Values
Number of epochs	20
Batch size	120
Optimizer	Adam
Learning rate	0.001
Activation function	ReLU
Number of neurons	64
Dropout	0.5

Table 5: First hyperparameters.

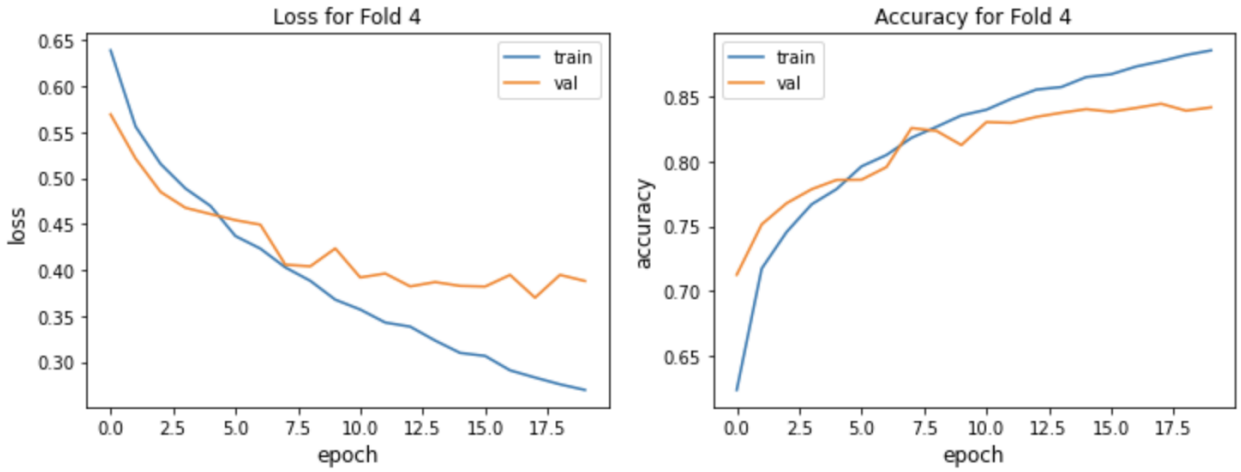

Figure 2: Graphs for loss values and accuracies of training and test sets for the initial CNN run.

At first, the number of epochs were increased to 30 to see the overfitting and not just assume. This iteration only done on the last model of the initial run, which is fold 5. In other words, fifth fold of the cross validation model was used as pre-trained model, and it was re-experimented with the initial epoch = 20. Then, 10 new epochs were added to it to see the performance of 30 epochs.

As expected, the loss value of the test set went from 0.2067 to the 0.23 at the end. Plus, the accuracy decreased from 0.92 to 0.90. One can note that these values are much better than the initial run because the model is pre-trained. However, it was used to see the overfitting and it proved with the increase of the loss from the 20^{th} to 30^{th} epoch.

Second hyperparameter is the batch size, and it was looked with 80 and 100 besides the 120 at the beginning. The results of these three models were close yet deterministic. The lowest accuracy came from the batch size 80 (see Table 6), since it increased the number of trained images most. There could have been a debate about selecting the batch size between 100 and 120, however, the initial run is already close to overfit. If the batch size gets lowered, the training set will be fed with more data and be more likely to overfit.

Batch size	Accuracy
120	0.8352
100	0.8334
80	0.8247

Table 6: Tuning the batch size for the CNN.

From now on, all of the further models were run with 20 epochs and batch size equal to 120. Next hyperparameter is the learning rate. Additional to the $1e$-3 in the initial run, $2e$-3 and $1e$-4 were tested. Unfortunately, the model with the learning rate $2e$-3 showed some overfitting results almost after the epoch 12 at each fold. Not all of the folds' loss values escalated, yet they all at least stopped improving.

The promising result here has come with the learning rate $1e$-4. Even though it has the lowest accuracy value for its test set, its graph looks better. It goes higher levels of accuracy in a correlated way with the training sets' accuracy and it might get better values if the model trains more. Because the training sets' accuracy is also lower than usual, which is also around 0.8. Plus, the loss values are higher, especially for the training set, where they both are around 0.4.

Learning rate	Accuracy
0.001	0.8352
0.002	0.8243
0.0001	0.7980

Table 7: Tuning the learning rate for the CNN.

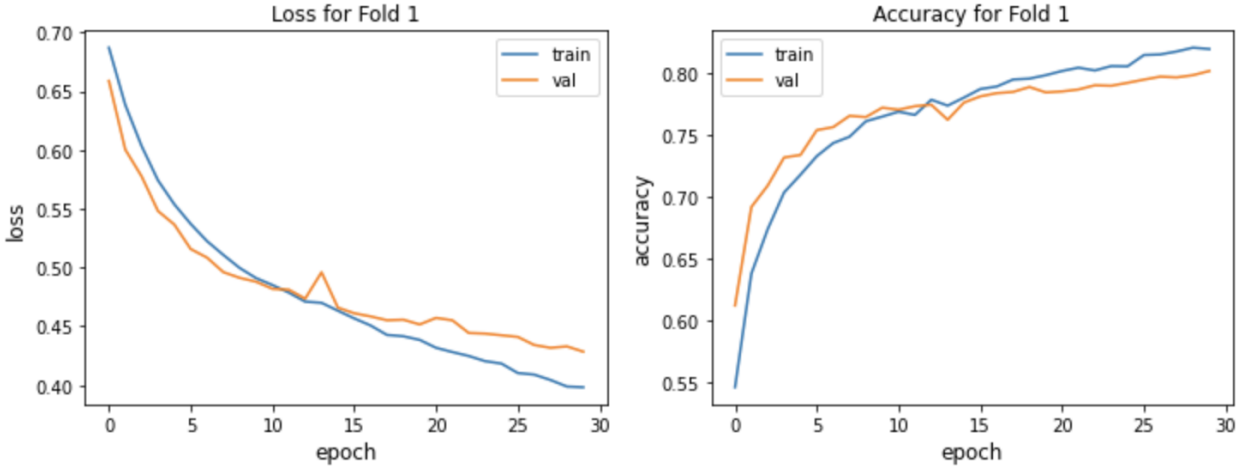

Figure 3: Graphs for loss values and accuracies of training and test sets when the number of epochs = 30, learning rate = 0.0001.

By believing the better shape of the model with the new learning rate, its epoch size re-changed to 30 to feed the training set with data more. In Figure 3, we can see that this modification did not change anything in a positive manner. Despite the fact that the number of epochs increased for the new learning rate, the results for the accuracy and the loss values are both similar to the previous. It determines that this learning rate is a slower learner than we expect. Hence, the best learning rate was selected as $1e$-3, from the initial run.

Optimizers were selected the same as the MLP architecture, which are SGD, Adam, and Adamax. Despite its result for the MLP, SGD optimizer did a miserable performance for the CNN (see Table 8). Its accuracy value got stuck at almost 0.59 while its loss value began at 0.69 and improved to only 0.672 as its best. Other than SGD, Adamax did well but not the best job. Its accuracy for the test sets converged around 0.81 and its loss value also converged between 0.40 - 0.45. Once again, the initial run's hyperparameter did the best job, which is Adam. There could have been different scenarios where Adamax would act better than Adam, however, double or triple hyperparameter checks could not be managed among the hyperparameters due to the lack of memory.

Optimizer	Accuracy
Adam	0.8352
Adamax	0.8136
SGD	0.5910

Table 8: Tuning the optimizer for the CNN.

Activation function hyperparameter was the easiest to decide which one was the best, unfortunately. ReLU did not give a chance to tanh and sigmoid functions, furthermore, they did not have a good performance indeed. With tanh, the model did not learn to predict the test sets. The accuracy and the loss values got stuck after some epochs. As usual, the fold closest to the average test accuracy was selected to visualize in Figure 4. It can be seen that both the loss and accuracy values are not improving. This case is much worse for sigmoid. Training and test sets' accuracies are almost the same, which are around 0.5. The loss values of both training and test sets are also very high than previous results - both approximately 0.69, without a sand grain size improvement. Therefore, ReLU is the right hyperparameter to move on.

Activation function	Accuracy
ReLU	0.8352
tanh	0.7805
sigmoid	0.5038

Table 9: Tuning the activation function for the CNN.

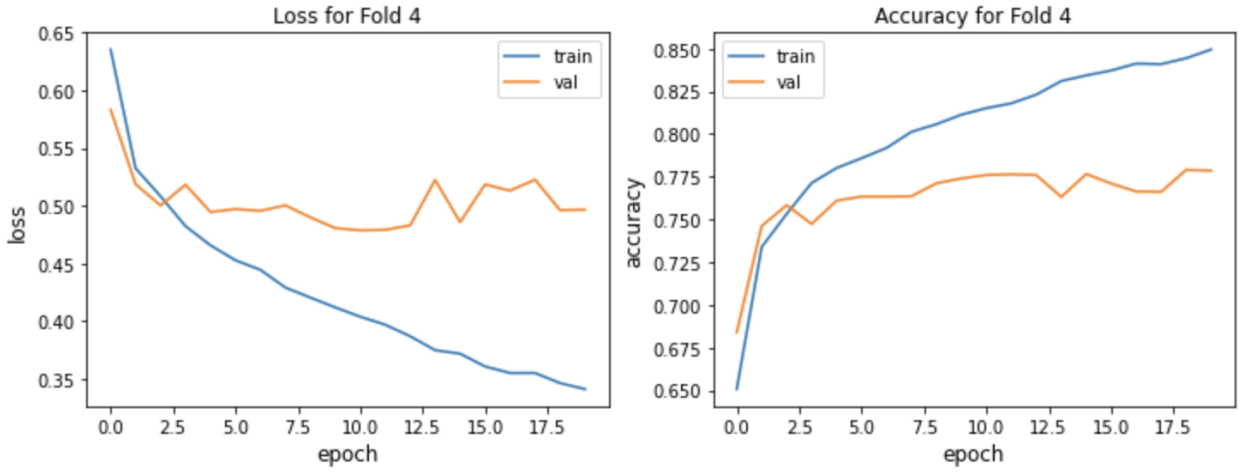

Figure 4: Graphs for loss values and accuracies of training and test sets when the activation function is tanh.

The number of neurons was the next to be tuned. Besides the initial 64 neurons, 48 and 128 were the number of neurons for other two 5-fold cross validation models. Results are in a similar pattern as the previous hyperparameters'. For instance, the model with the 128 neurons in its hidden layer overfit the data. The reason of the overfitting is more hidden layer neurons that are more detailed with training data. Which means that it trained the data more than usual. Statistically, its test accuracy level stuck around 0.83 after some epochs and its loss value also stuck around 0.4. The overfit can be seen from the Figure 5, the loss value got flat and at all of the folds after roughly the 10^{th} epoch.

# of neurons	Accuracy
64	0.8352
48	0.8328
128	0.8321

Table 10: Tuning the number of neurons for the CNN.

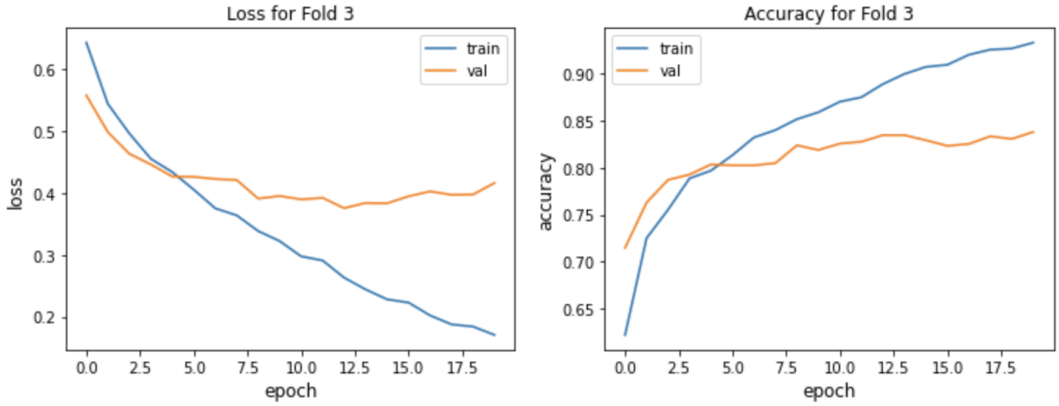

Figure 5: Graphs for loss values and accuracies of training and test sets when the number of neurons is 128.

For the hidden layer with 48 neurons, it also fell behind the desired result while its very similar to the initial run. However, we could get worse results if we want to train the model more because we have already seen that it is near overfitting and trying the find the hyperparameters that prevent this situation.

Finally, and differently from the tuned MLP hyperparameters, dropout value was examined. This method was applied both before and after the hidden layer to ignore some amount of neurons while training each time. Initially, it was determined as 0.5. While tuning, 0.2 and 0.8 were also experimented to improve the performance. In Table 11, there are the results of these models with different dropout values. Starting from the obvious one, 0.2 dropout caused an apparent overfitting (see Figure 6). It is also comprehensible from the training accuracy which is almost 0.96 on average, because it trained more than it needs. However, the loss value for the test set turns where it started at the last epoch. It went to almost 0.6.

Dropout	Accuracy
0.5	0.8352
0.8	0.8216
0.2	0.8146

Table 11: Tuning the dropout value for the CNN.

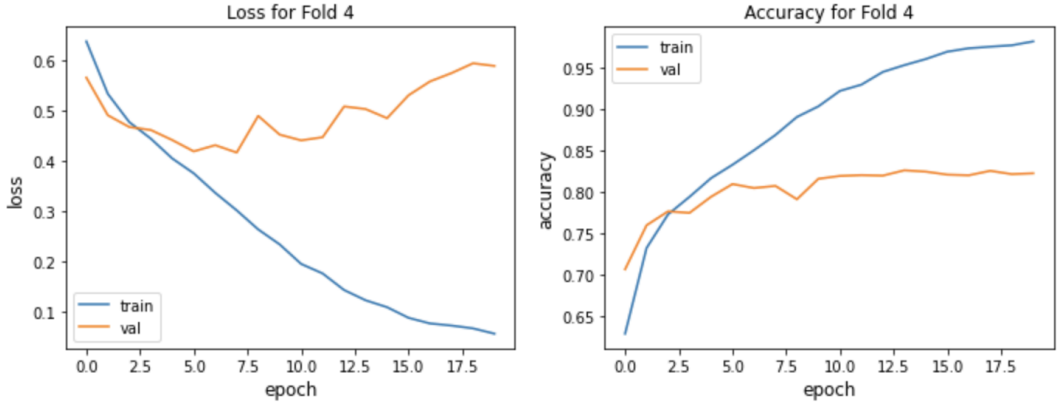

Figure 6: Graphs for loss values and accuracies of training and test sets when the dropout value is 0.2.

Besides the overfitting dropout and initial dropout values, last model with 0.8 dropout looks very promising with its graph (see Figure 7). For the first time for the CNN models, test sets developed better than the training sets, thanks to the high dropout value. The reason why the training accuracy is low is again the high dropout value. This time, since there is no signal for overfitting in the graphs, this model can be trained more with data to increase the accuracies and decrease the loss values. Hence, there is not a final decision yet for the dropout value. Before deciding it, one more tuning was done with the same dropout value but higher number of epochs.

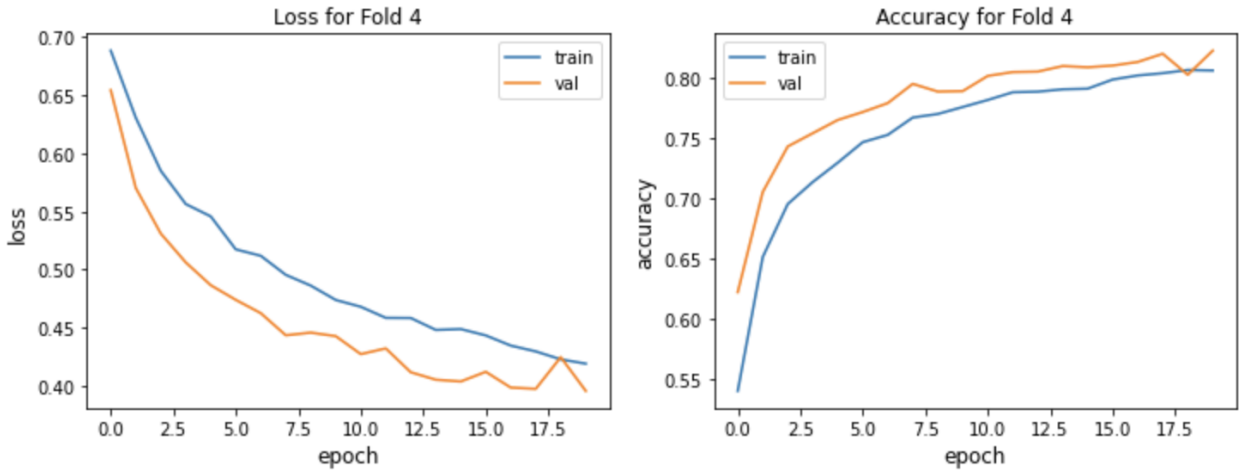

Figure 7: Graphs for loss values and accuracies of training and test sets when the dropout value is 0.8.

Similar to what was done at the first step of hyperparameter tuning, just to see the performance of higher number of epochs, only the last model, fifth fold was tested. Its epoch was increased to 40 and started from the initial epoch = 20. To visualize and have a better understanding, it then concatenated with the previous 20 epochs in loss and accuracy graphs.

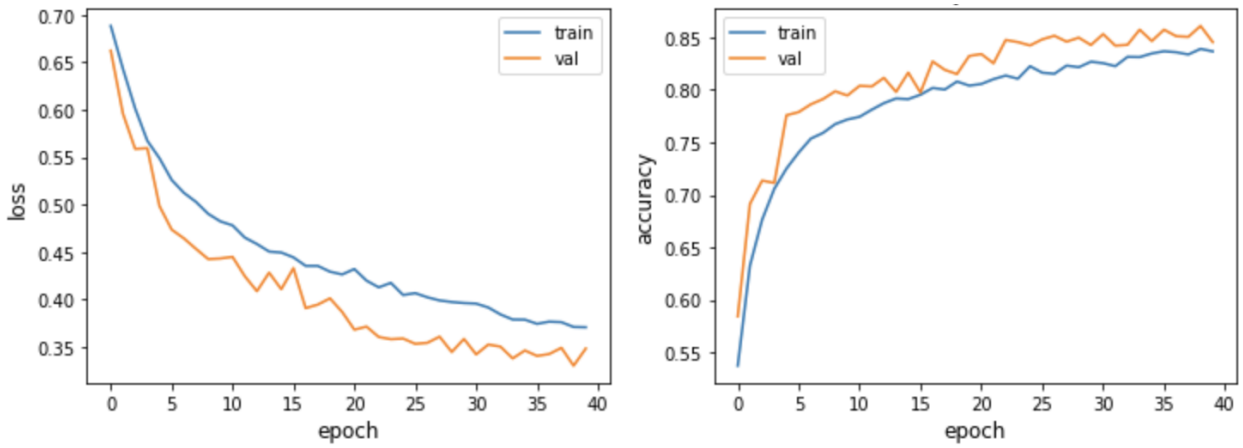

Figure 8: Graphs for loss values and accuracies of training and test sets when the dropout value is 0.8, and epoch = 40.

As anticipated, the results were better than the initial run at the beginning. This model has the test accuracy 0.8454 and the loss 0.3481, which are both more improved than the initial model with the dropout 0.5. Hence, the final run will be done with these two new hyperparameters.

At the end, after tuning the seven hyperparameters, there are only two parameters different than than the beginning and gave better results. Therefore, final model was designed with these hyperparameters and run once again. Fold closest to the average test accuracy can be seen in Figure 9.

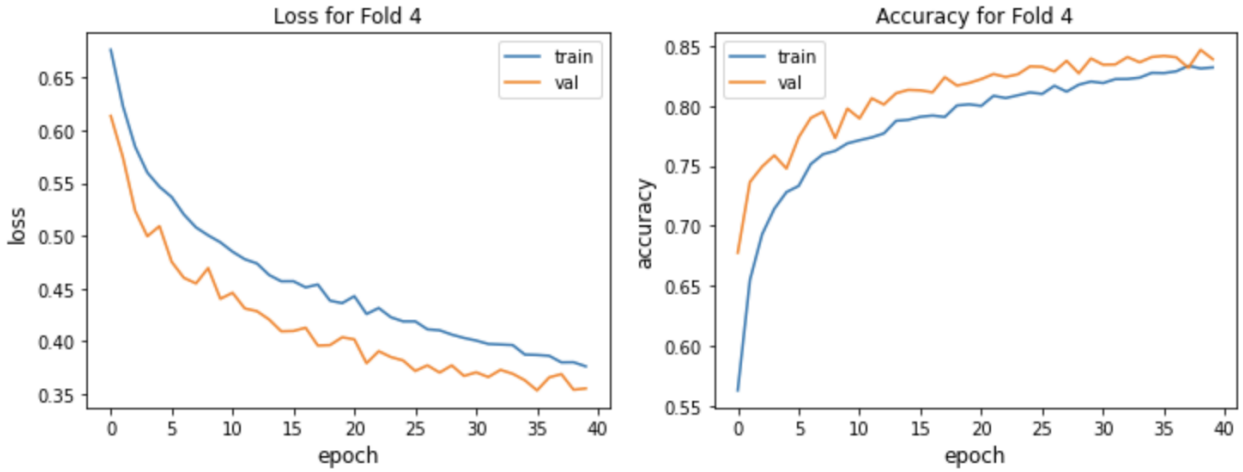

Figure 9: Final run of the tuned CNN.

Average of test loss and accuracy values were calculated and turns out that a slight improvement was gained with the final model (see Table 12). Final model's average test accuracy increased to 0.8379 from 0.8352, and its loss decreased to 0.3605 from 0.3903. In addition, its training accuracy decreased to 0.8342 from 0.8943, and its training loss value increased to 0.3723 from 0.2511. In terms of training sets, there is a negative improvement, however, thanks to high dropout value, results of the test set are the determiners. Hence, after experimenting the final model and comparing with the initial run, the convolutional NN architecture with this dataset works the best with 40 epochs, batch size equal to 120, Adam optimizer with learning rate $1e$-3, ReLU activation function, 64 neurons in hidden layer and 0.8 dropout values both before and after the hidden layer.

CNN models	Accuracy
Final	0.8379
Initial	0.8352

Table 12: Initial and final CNN models.

4 Results

In conclusion, two neural network architectures had much different results while using the same hyperparameters with different values. With tuning those hyperparameters, the MLP model was managed to get almost 64% accuracy for its test set, and CNN model got

almost %83. One can conclude that each hyperparameter had different impact on these two architectures and positively affected the final result.

To understand the differences among the architectures, they also have different input shapes in terms of the color of the images, hence, it should not be expected to have similar results for the both of them. CNN architecture did a better learning and have better results for the accuracy and loss.

Future works can be done for the CNN architecture, where can be tunings with multiple options without any memory issues. In addition, scaling of the images can be increased for the NN to understand and distinguish the pixels better.

References

[1] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[2] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.