CMPE452 Assignment Part-A

İsmail Kağan Acar

22244710065

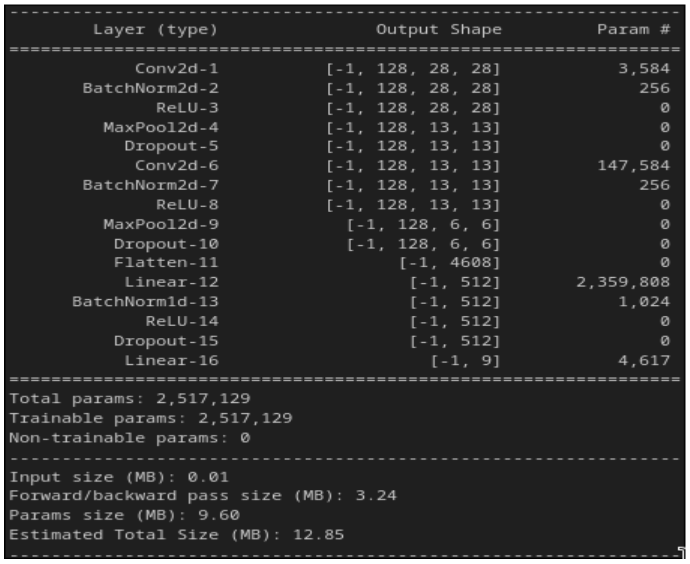
Model Sturcture

The assignment involved creating a Convolutional Neural Network (CNN) to classify medical images from the PathMNIST dataset. This dataset contains over 100,000 small images (28x28 pixels, 3 color channels) representing 9 different types of cancer.

The base model begins with basic data augmentation, including random rotations and color adjustments. Initially, horizontal flipping was implemented, but it negatively affected the model's performance, so it was removed. The CNN consists of two main layers, each using 128 filters with a 3x3 kernel size, followed by max pooling (kernel size 3, stride 2) and a 50% dropout rate. After these layers, a fully connected layer with 512 neurons and another dropout layer were added.

The model was trained for 20 epochs with a batch size of 128, using the Adam optimizer and CrossEntropyLoss. An early stopping mechanism was initially implemented but was later disabled after following the experiment guidelines.

The dataset was split into training (89,996 images), validation (10,004), and test (7,180) sets. These splits were provided by PathMNIST as separate datasets.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 128, 28, 28]           3,584
       BatchNorm2d-2          [-1, 128, 28, 28]             256
              ReLU-3          [-1, 128, 28, 28]               0
         MaxPool2d-4          [-1, 128, 13, 13]               0
           Dropout-5          [-1, 128, 13, 13]               0
            Conv2d-6          [-1, 128, 13, 13]         147,584
       BatchNorm2d-7          [-1, 128, 13, 13]             256
              ReLU-8          [-1, 128, 13, 13]               0
         MaxPool2d-9            [-1, 128, 6, 6]               0
          Dropout-10            [-1, 128, 6, 6]               0
          Flatten-11                 [-1, 4608]               0
           Linear-12                  [-1, 512]       2,359,808
      BatchNorm1d-13                  [-1, 512]           1,024
             ReLU-14                  [-1, 512]               0
          Dropout-15                  [-1, 512]               0
           Linear-16                    [-1, 9]           4,617
================================================================
Total params: 2,517,129
Trainable params: 2,517,129
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.24
Params size (MB): 9.60
Estimated Total Size (MB): 12.85
----------------------------------------------------------------
```

Base model summary

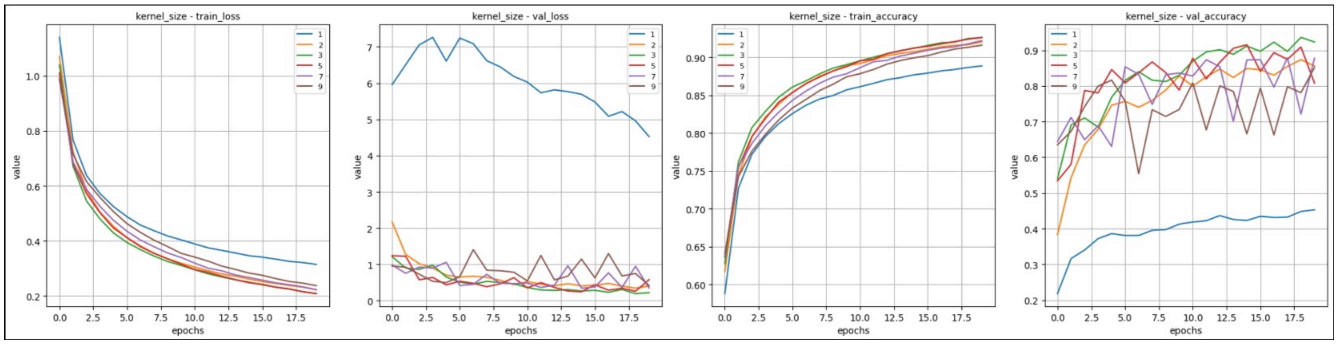
Experiments

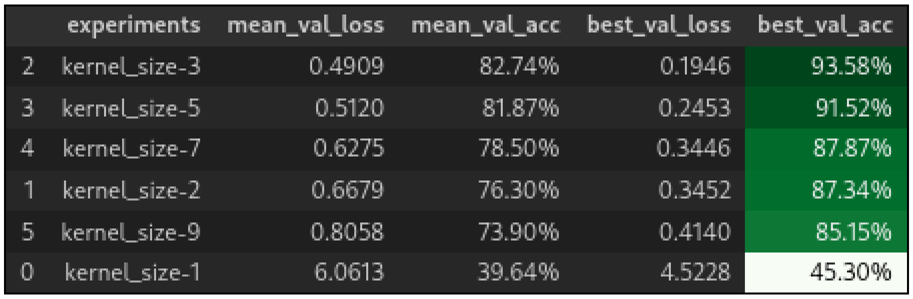
Experiments were performed on 11 different hyperparameters as specified in the guidelines. A significant portion of the experiments were performed and logged automatically using my custom grid search program. For each experiment, the training and validation losses, accuracies, and training duration were recorded in JSON files. This automated logging system made visualizing and analyzing the results more efficient.

The hyperparameters tested included:

- Kernel sizes of CNN (6 value)
- Dropout rates of CNN (6 value)
- Dropout rates of Classifier (6 value)
- Layers of CNN (6 value)
- Hidden neurons of Classifier (6 value)
- Max Pooling kernel sizes (6 value)
- Max Pooling strides (6 value)
- Epoch numbers (6 value)
- Batch sizes (8 value)
- Optimizers (3 optimizer type each with 2 difirent learning rate)
- Activation functions (3 function)
- Data Augmentation (only experimented on best model)

1. Kernel Size Experiments





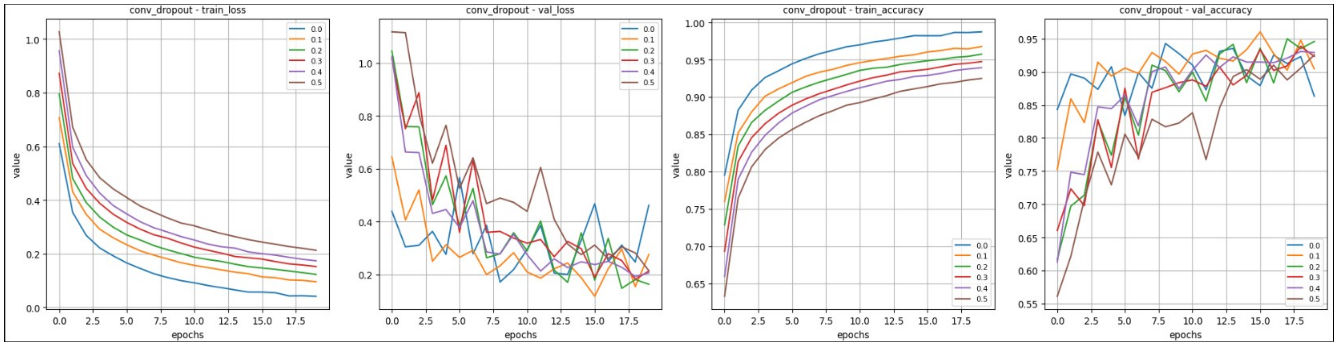
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
2	kernel_size-3	0.4909	82.74%	0.1946	93.58%
3	kernel_size-5	0.5120	81.87%	0.2453	91.52%
4	kernel_size-7	0.6275	78.50%	0.3446	87.87%
1	kernel_size-2	0.6679	76.30%	0.3452	87.34%
5	kernel_size-9	0.8058	73.90%	0.4140	85.15%
0	kernel_size-1	6.0613	39.64%	4.5228	45.30%

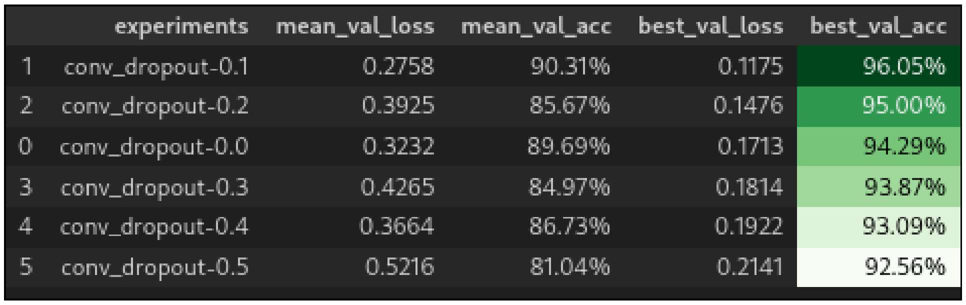
Generally, odd-numbered kernel sizes like 3×3 are preferred in CNNs. The experiments tested less common kernels, such as 1×1, which showed significantly worse performance. Kernel size selection is strongly related to input dimensions. For smaller input sizes, such as the 28×28 images in the PathMNIST dataset, it is easier to capture important features with smaller kernels. However, even with small sized 2×2 (and other even-numbered kernels performs poorly), the performance drop observed in the experimental results.

The base model uses standard 3×3 kernels, and experiments will continue with this configuration in the best model, as it is clearly the top-performing choice.

2. Dropout Rate Experiments

2.1 CNN Dropout





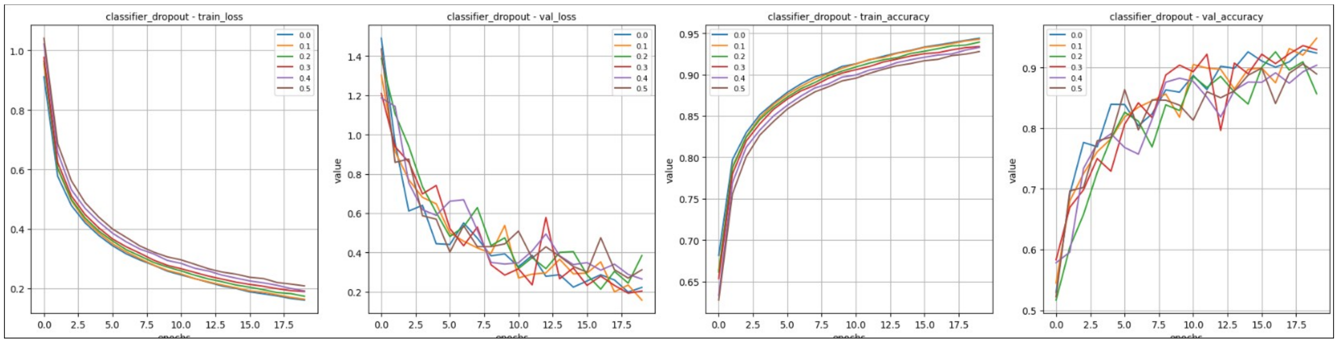
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
1	conv_dropout-0.1	0.2758	90.31%	0.1175	96.05%
2	conv_dropout-0.2	0.3925	85.67%	0.1476	95.00%
0	conv_dropout-0.0	0.3232	89.69%	0.1713	94.29%
3	conv_dropout-0.3	0.4265	84.97%	0.1814	93.87%
4	conv_dropout-0.4	0.3664	86.73%	0.1922	93.09%
5	conv_dropout-0.5	0.5216	81.04%	0.2141	92.56%

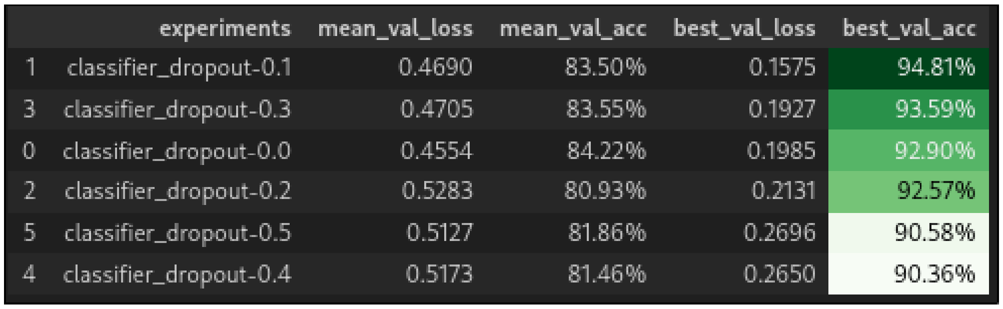
The model applied dropout after each convolutional layer. The base model initially used a 50% dropout rate. While dropout is effective for preventing overfitting, high rates can lead to underfitting. The 50% dropout rate in the base model appeared too aggressive, resulting in underfitting behavior.

Experimental testing showed 0% dropout achieved a decent accuracy score. However, complete removal of dropout risks potential overfitting. Both 20% and 10% dropout provided better performance while maintaining generalization. The 20% dropout rate's validation accuracy scores began with lower scores but eventually reached levels as high as the 10% dropout model. Additionally, an important observation from the experiments is that in the 11th epoch, the 50% dropout rate probably eliminated critical parts of the model, causing a significant negative spike in performance during that epoch.

The primary concern here is model overfitting at low dropout rates. To eliminate this risk, a 20% dropout rate appears safer to choose for the best model.

2.2 Classifier Dropout



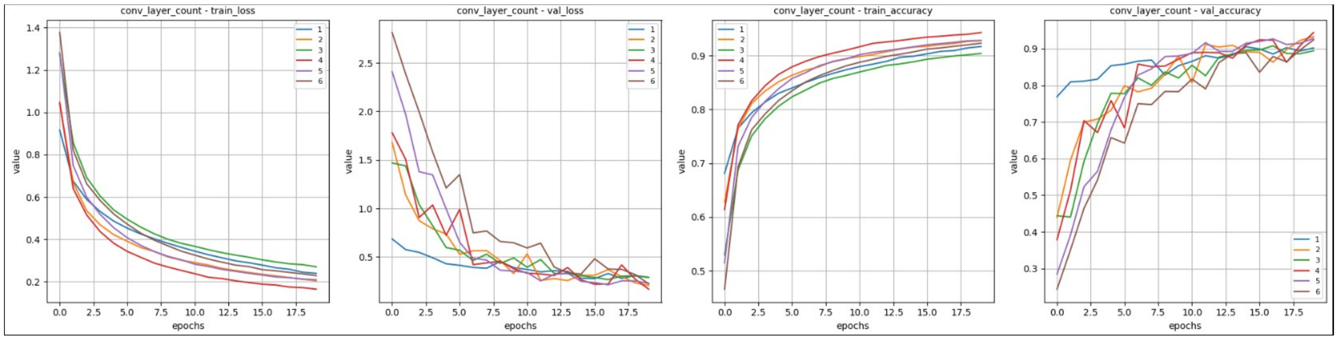


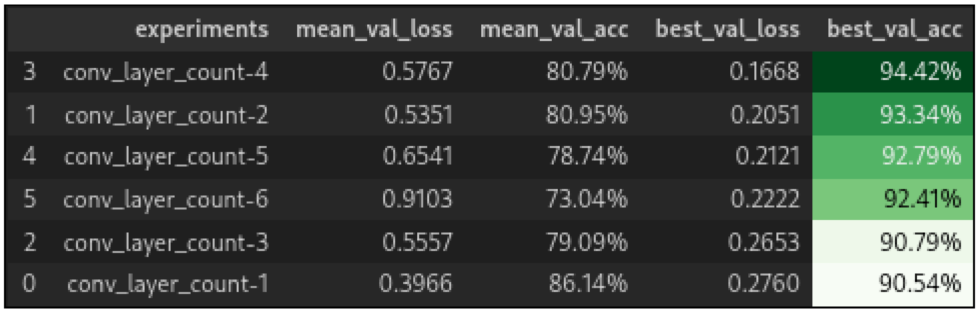
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
1	classifier_dropout-0.1	0.4690	83.50%	0.1575	94.81%
3	classifier_dropout-0.3	0.4705	83.55%	0.1927	93.59%
0	classifier_dropout-0.0	0.4554	84.22%	0.1985	92.90%
2	classifier_dropout-0.2	0.5283	80.93%	0.2131	92.57%
5	classifier_dropout-0.5	0.5127	81.86%	0.2696	90.58%
4	classifier_dropout-0.4	0.5173	81.46%	0.2650	90.36%

The model implements dropout only once between fully connected layers, resulting in a more limited impact on the classifier compared to the CNN sections. The base model implemented a 50% dropout rate in this configuration.

Experimental results showed slight improvement when using 30% dropout rate and performance drop when using 20% in these layers compared to the CNN's dropout performance. Along with the earlier findings, the 30% dropout rate gives optimal performance against 10% (to prevent overfitting) and chosed for best model training.

3. Layers of CNN Experiments



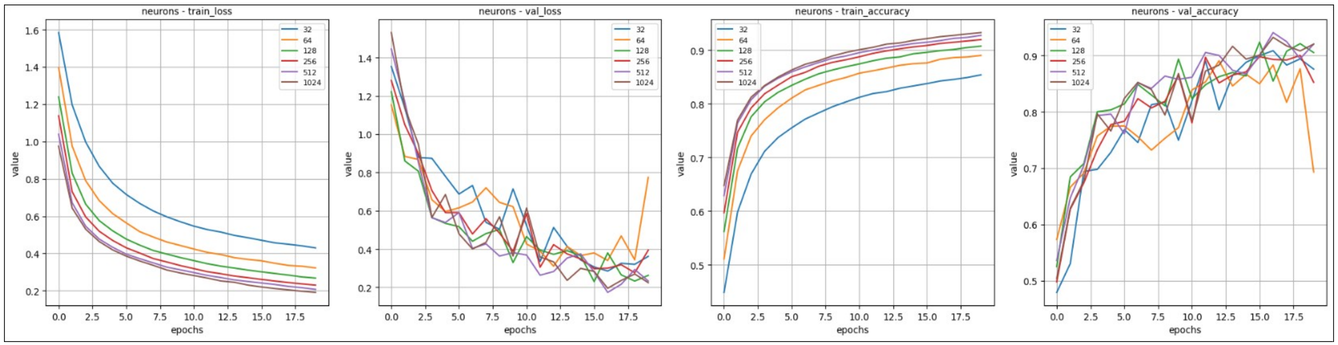


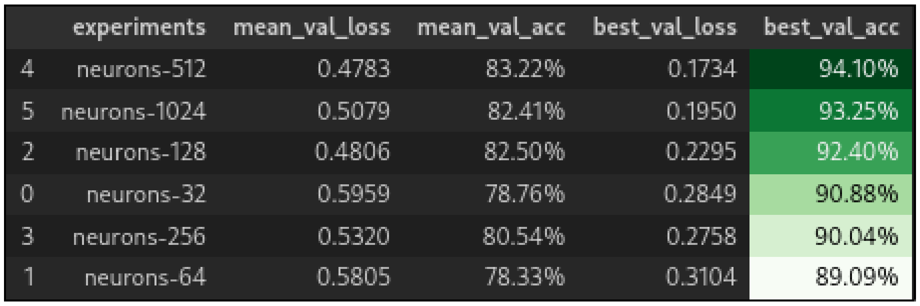
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
3	conv_layer_count-4	0.5767	80.79%	0.1668	94.42%
1	conv_layer_count-2	0.5351	80.95%	0.2051	93.34%
4	conv_layer_count-5	0.6541	78.74%	0.2121	92.79%
5	conv_layer_count-6	0.9103	73.04%	0.2222	92.41%
2	conv_layer_count-3	0.5557	79.09%	0.2653	90.79%
0	conv_layer_count-1	0.3966	86.14%	0.2760	90.54%

The base model used 2 CNN layers with a fixed filter size of 128 (same size for both layers). During experiments, each convolutional block was treated as a single unit containing all components (dropout, max pooling, activation functions, etc.). This approach caused an issue: when using more than 3 CNN layers, max pooling reduced the image dimensions too much, resulting in errors. To fix this, max pooling was limited to 3 layers and placed carefully. This limitation may raise questions about experiment quailty. (The issue here was using CNN’s layer padding as ‘same’ if a proper padding value is used this wouldn’t be a problem. All of the experiments made with padding=’same’)

Adding or removing CNN layers showed little impact in experiments. A better strategy might be testing different filter sizes per layer (like VGG models, which increase filters incrementally). The best-performing models had 4 or 2 CNN layers, with very small differences between them. The validation graphs gived very minimal advantage for 4 layers, so 4 layers chosed for the final model.

4. Hidden Neurons of Classifier Experiments





	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
4	neurons-512	0.4783	83.22%	0.1734	94.10%
5	neurons-1024	0.5079	82.41%	0.1950	93.25%
2	neurons-128	0.4806	82.50%	0.2295	92.40%
0	neurons-32	0.5959	78.76%	0.2849	90.88%
3	neurons-256	0.5320	80.54%	0.2758	90.04%
1	neurons-64	0.5805	78.33%	0.3104	89.09%

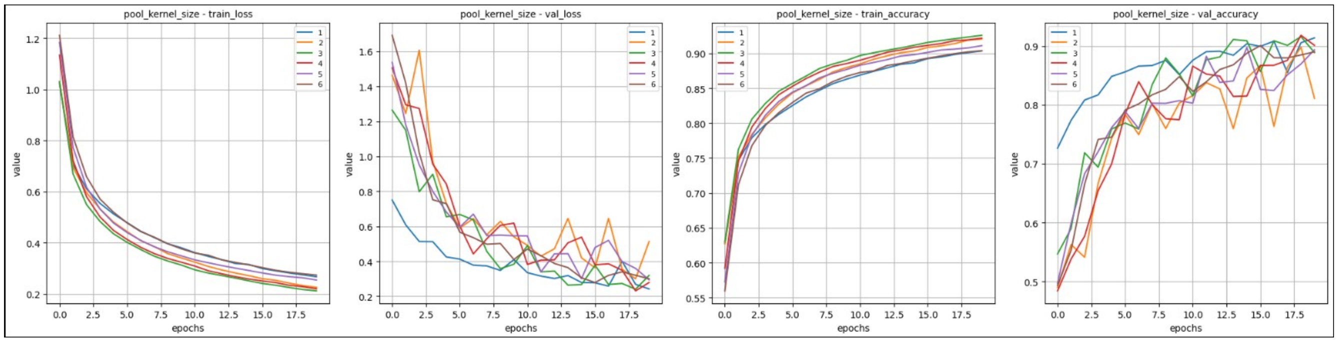
The base model used 512 neurons. Test results for 128, 512, and 1024 neurons were very close and it's hard to pick a clear winner. Even though 512 and 1024 are much bigger than 128, their final scores were almost the same.

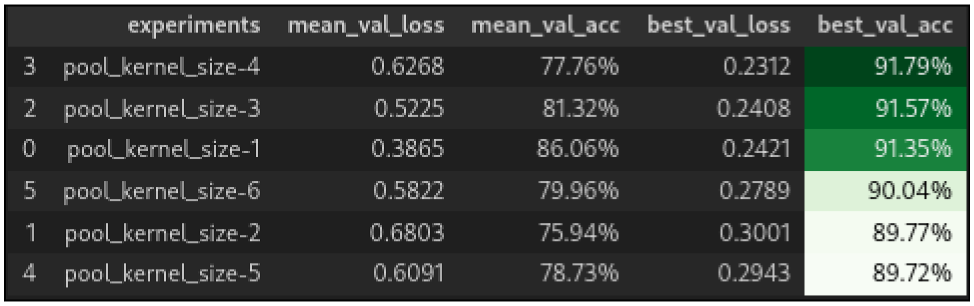
Surprisingly, using 8 times more neurons (like 1024 vs 128) didn't affect accuracy dramatically. This suggests changing neuron count doesn't really help here. At the 128 and 256 neuron counts model might reach the performance limit which more neurons are unnecessary at this point. Also experiments shows that 256 neurons performed poorly compared to way smaller size 32. To keep things simple and safe (safe is in terms of overfitting), neuron size is selected as 128.

5. Max Pooling Experiments

Max pooling normally implemented in each convolutional layer but limited at max 3 application because dimesion issues. Explained in detail in CNN layer count part.

5.1 Max Pooling Kernel Size



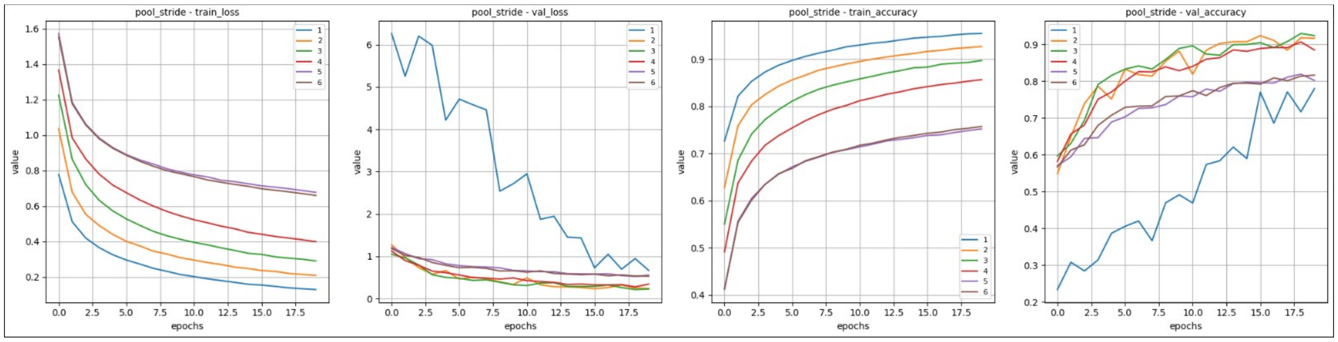


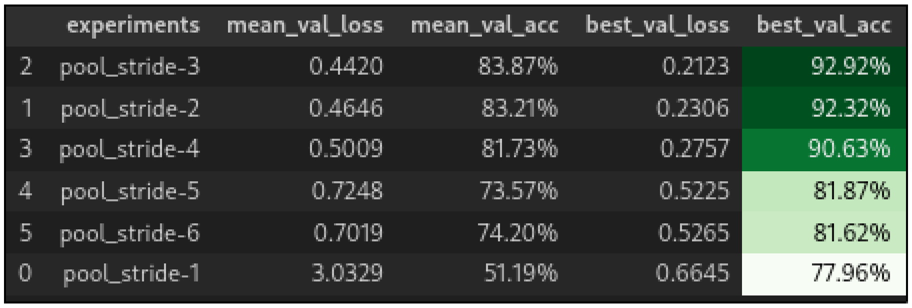
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
3	pool_kernel_size-4	0.6268	77.76%	0.2312	91.79%
2	pool_kernel_size-3	0.5225	81.32%	0.2408	91.57%
0	pool_kernel_size-1	0.3865	86.06%	0.2421	91.35%
5	pool_kernel_size-6	0.5822	79.96%	0.2789	90.04%
1	pool_kernel_size-2	0.6803	75.94%	0.3001	89.77%
4	pool_kernel_size-5	0.6091	78.73%	0.2943	89.72%

The base model used a max pooling kernel size of 3. Generally, 2x2 and 3x3 kernel sizes are used. A 2x2 kernel size is considered standard, while 3x3 is seen as aggressive. Experiment results among these kernel sizes are very similar to each other, except for 1x1, which does not apply pooling.

Experimental results show that the best value is obtained with a 4x4 kernel size. However, this size is not particularly useful (and very rarely used) and did not demonstrate strong generalization. Thus, sticking with the 3x3 kernel size remains optimal for the best model. On the other hand, despite being the standard choice, the 2x2 kernel size's performance is less reliable, as its graphs include several sharp spikes.

5.2 Max Pooling Stride



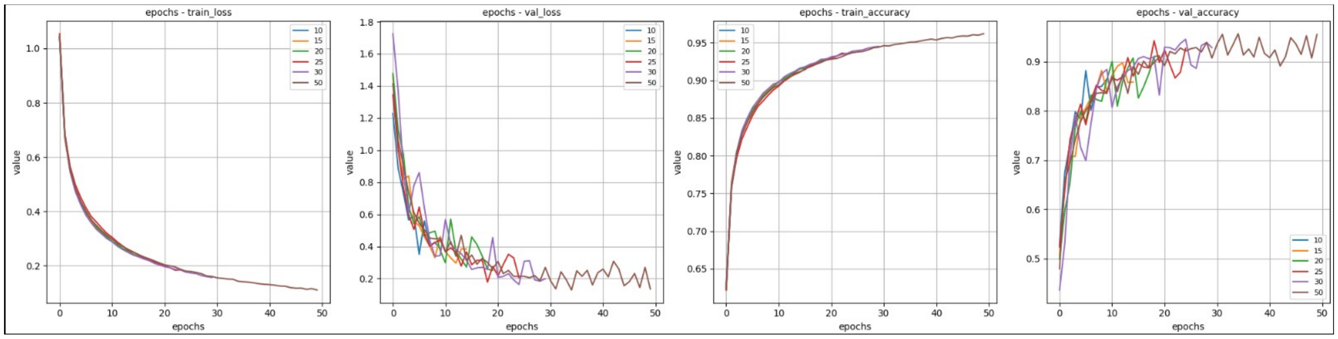


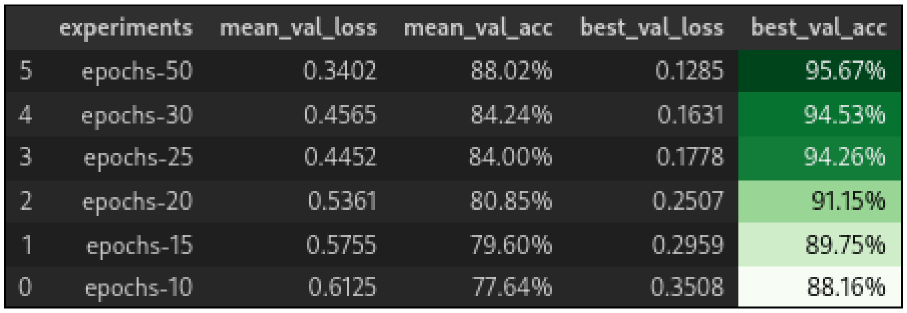
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
2	pool_stride-3	0.4420	83.87%	0.2123	92.92%
1	pool_stride-2	0.4646	83.21%	0.2306	92.32%
3	pool_stride-4	0.5009	81.73%	0.2757	90.63%
4	pool_stride-5	0.7248	73.57%	0.5225	81.87%
5	pool_stride-6	0.7019	74.20%	0.5265	81.62%
0	pool_stride-1	3.0329	51.19%	0.6645	77.96%

The most common value for stride is "2", as used in the base model. This means downsampling the resolution by half at each step.

Experiments clearly show that a stride of 1 would be a very poor choice. On the other hand, stride values of 2 and 3 performed very well. Similar to how a kernel size of 3 is considered a more aggressive approach for stride selection, stride 3 can sometimes lead to loss of features, but it showed a slight advantage over stride 2 in these experiments. To keep this minor advantage, stride 3 was selected for training the best model.

6. Epoch Number Experiments





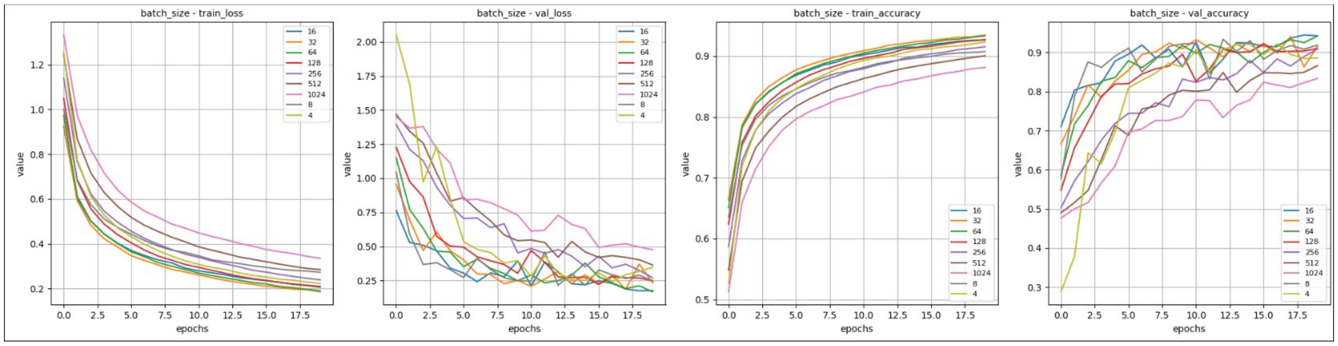
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
5	epochs-50	0.3402	88.02%	0.1285	95.67%
4	epochs-30	0.4565	84.24%	0.1631	94.53%
3	epochs-25	0.4452	84.00%	0.1778	94.26%
2	epochs-20	0.5361	80.85%	0.2507	91.15%
1	epochs-15	0.5755	79.60%	0.2959	89.75%
0	epochs-10	0.6125	77.64%	0.3508	88.16%

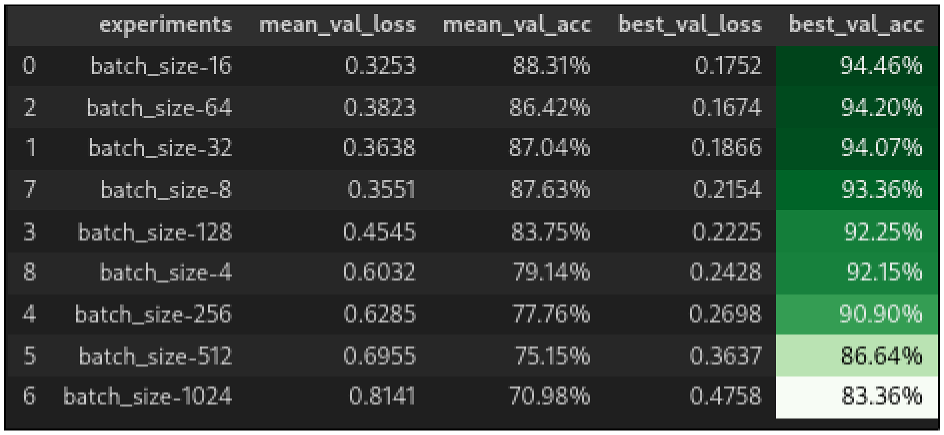
The epoch count was specified as 20 for the base model. Unnecessarily large epoch counts often lead to overfitting, while low epoch counts typically result in underfitting.

Experiments show that 10 or 15 epochs are not enough for optimal training. Additionally, after the 20th epoch, no significant improvement can be observed. Based on these findings, the ideal epoch count is selected as 20. If any overfitting issues occures reducing epochs to the 15 would be best.

Tests with Best Model: Best model trained with 25, 20, and 15 epochs individually and least overfitting result came with 15 epoch.

7. Batch Size Experiments





	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
0	batch_size-16	0.3253	88.31%	0.1752	94.46%
2	batch_size-64	0.3823	86.42%	0.1674	94.20%
1	batch_size-32	0.3638	87.04%	0.1866	94.07%
7	batch_size-8	0.3551	87.63%	0.2154	93.36%
3	batch_size-128	0.4545	83.75%	0.2225	92.25%
8	batch_size-4	0.6032	79.14%	0.2428	92.15%
4	batch_size-256	0.6285	77.76%	0.2698	90.90%
5	batch_size-512	0.6955	75.15%	0.3637	86.64%
6	batch_size-1024	0.8141	70.98%	0.4758	83.36%

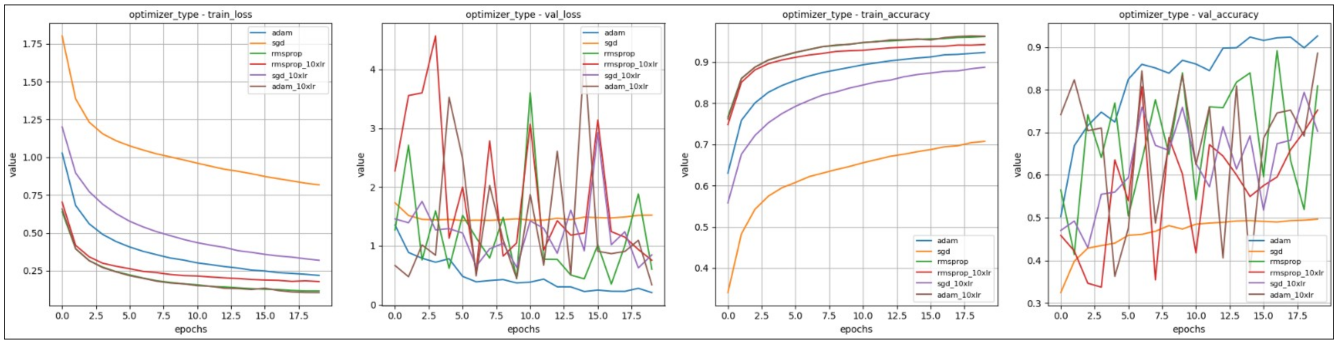
Initiatlly Batch size selected randomly 128 for the base model. Actually I didn't know it was so important for model. Turns out batch size is important both for model performance and computational cost.

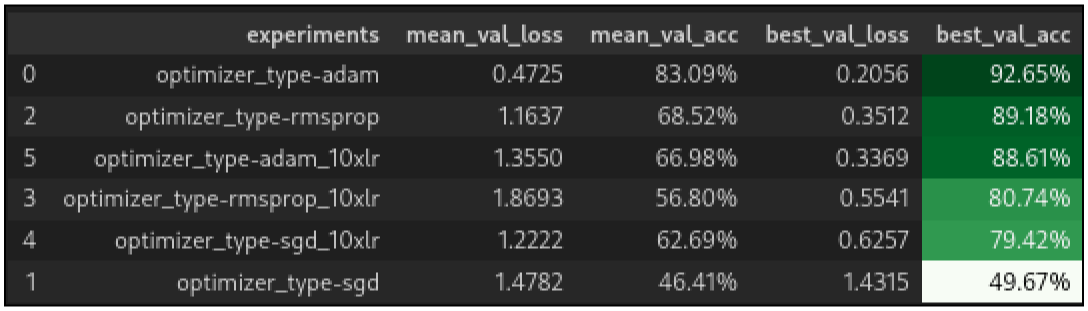
According to experiment results; while large batch sizes like 1024, 512, and 256 negatively affects the model performance, low batch sizes gives more convenient and similar results such as 16, 32, and 64. However mini-batches like 4 and 8 gives slightly more poor results but they are still better than large ones. From this findings batch size 16, 32, and 64 should be tested more for best model.

Tests with Best Model: Best model trained each of this values individually and best value decided as 64 in terms of test accuracy scores. Test accuracy scores:

- Batch size 64: %89
- Batch size 32: %84
- Batch size 16: %81

8. Optimizer Type Experiments



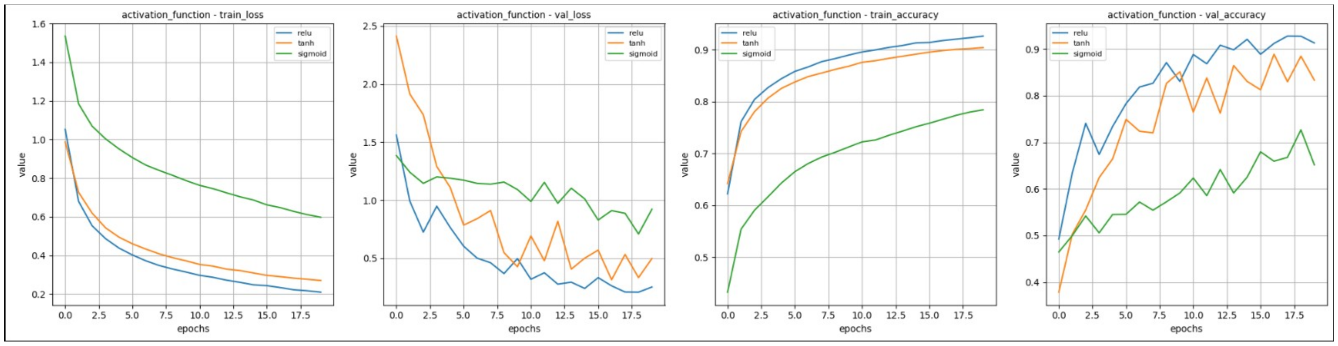


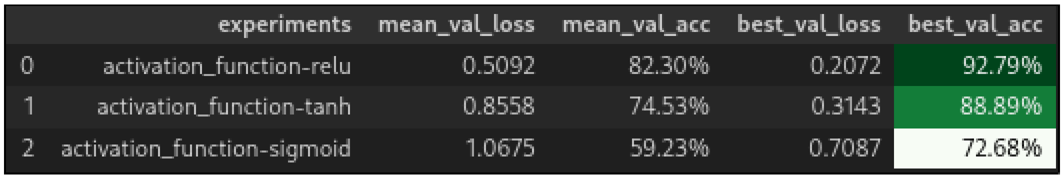
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
0	optimizer_type-adam	0.4725	83.09%	0.2056	92.65%
2	optimizer_type-rmsprop	1.1637	68.52%	0.3512	89.18%
5	optimizer_type-adam_10xlr	1.3550	66.98%	0.3369	88.61%
3	optimizer_type-rmsprop_10xlr	1.8693	56.80%	0.5541	80.74%
4	optimizer_type-sgd_10xlr	1.2222	62.69%	0.6257	79.42%
1	optimizer_type-sgd	1.4782	46.41%	1.4315	49.67%

The Adam optimizer is the most popular and simpler selection among all optimizers. So, the base model used the Adam optimizer with a 0.0001 learning rate. In truth, the learning rate alone deserves its own experiment, but tests were conducted for 0.0001 and 0.001 with Adam, RMSprop, and SGD. Momentum was set to 0.9 for RMSprop and SGD.

These experiments resulted in chaotic outcomes. Optimizer selection is a serious matter requiring careful tuning. Due to this complexity and limitations in technical skills of mine, all experiments except the base model's configuration failed. Therefore, Adam with a 0.0001 learning rate will continue to be used.

9. Activation Function Experiments





	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
0	activation_function-relu	0.5092	82.30%	0.2072	92.79%
1	activation_function-tanh	0.8558	74.53%	0.3143	88.89%
2	activation_function-sigmoid	1.0675	59.23%	0.7087	72.68%

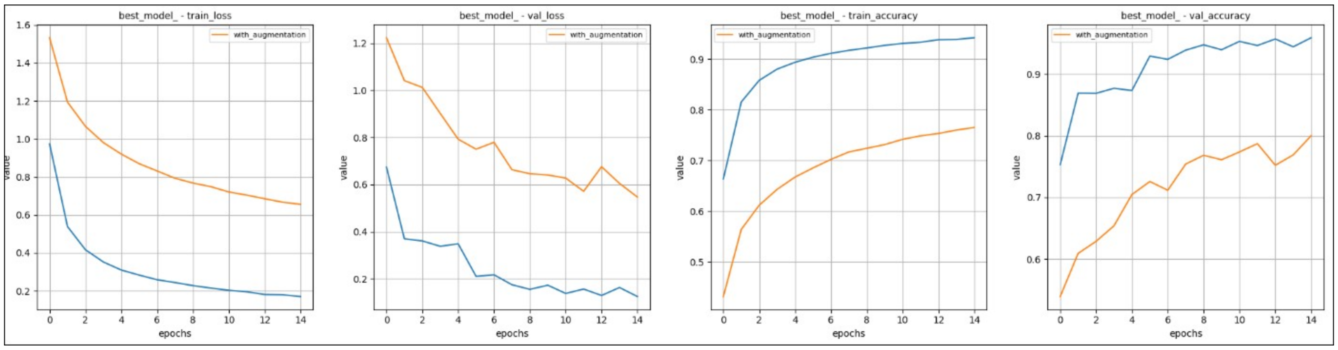
The base model uses the ReLU activation function in each convolutional and classifier layer.

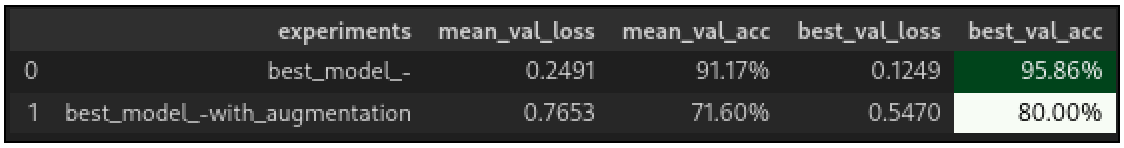
The experiment results clearly show that ReLU performs best compared to Tanh and Sigmoid. Sigmoid activation works best for probabilities in cases with fewer classes, but this dataset includes 9 classes, which is why it performs poorly. Because of reliability and good performance ReLU will continue to be used for training the best model.

10. Data Augmentation

In early stages of my experiments data augmentation seemed as a good idea but when things comes to practice, it was affecting in bad way. I wanted to demonstrate this the difference on best model. Here are the augmentations used:

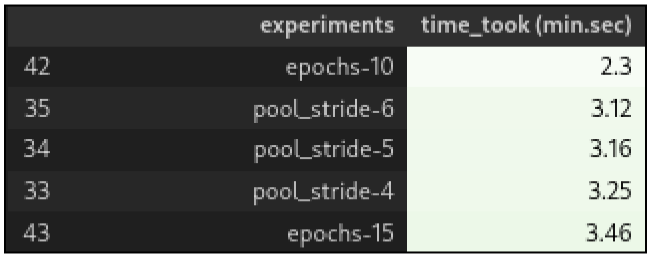
- RandomHorizontalFlip: Flips image vertically.
- RandomRotation: Randomly rotates the image respect to given parameters.
- ColorJitter: Randomly adjusts brightness, contrast, saturation, and hue.





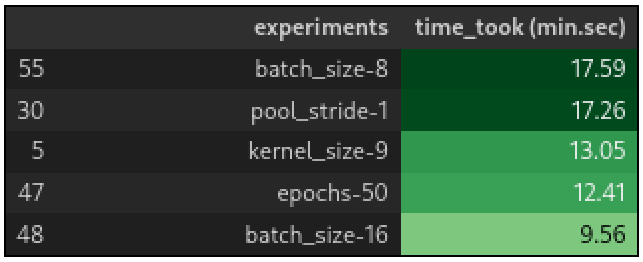
	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
0	best_model_-	0.2491	91.17%	0.1249	95.86%
1	best_model_-with_augmentation	0.7653	71.60%	0.5470	80.00%

How Much Time Took to Train each Model



	experiments	time_took (min.sec)
42	epochs-10	2.3
35	pool_stride-6	3.12
34	pool_stride-5	3.16
33	pool_stride-4	3.25
43	epochs-15	3.46

5 Fastest Trains



	experiments	time_took (min.sec)
55	batch_size-8	17.59
30	pool_stride-1	17.26
5	kernel_size-9	13.05
47	epochs-50	12.41
48	batch_size-16	9.56

5 Slowest Trains

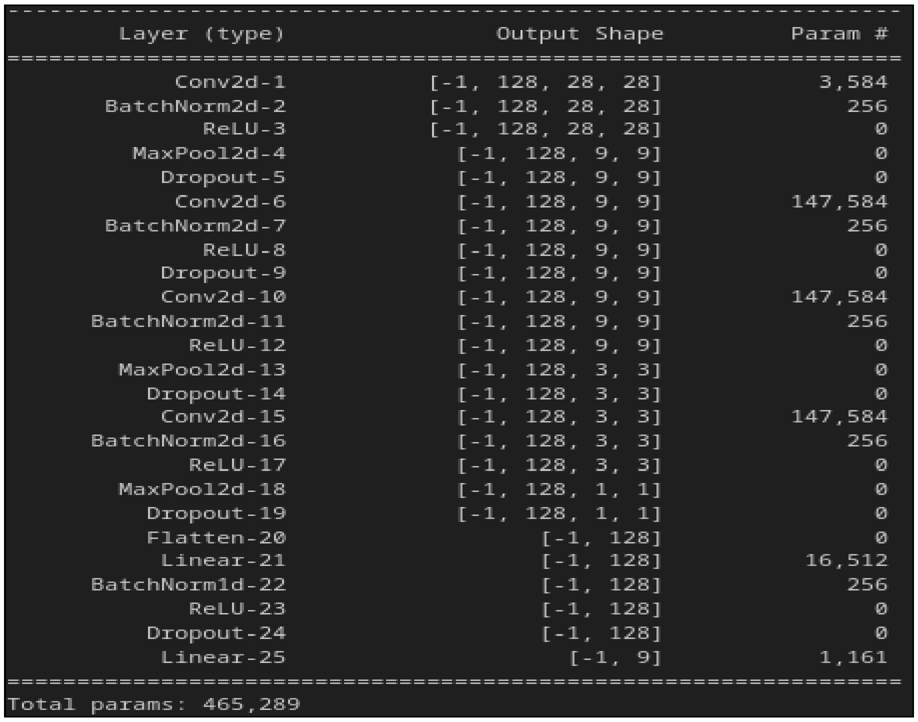
Total training time for all experiments took: 6 and half hours. (Trains are made in a personal computer so the result times are not isolated from other uses of pc.)

Best Model

According to all of the experiments, best resulting hyper parameters are listed:

- Kernel size of CNN : 3x3
- Dropout rate of CNN : %20
- Dropout rate of Classifier : %30
- Layers of CNN : 4
- Hidden neurons of Classifier : 128
- Max Pooling kernel size : 3
- Max Pooling stride : 3
- Epoch number : 15
- Batch size : 64
- Optimizer : Adam with lr=0.0001
- Activation function : ReLU

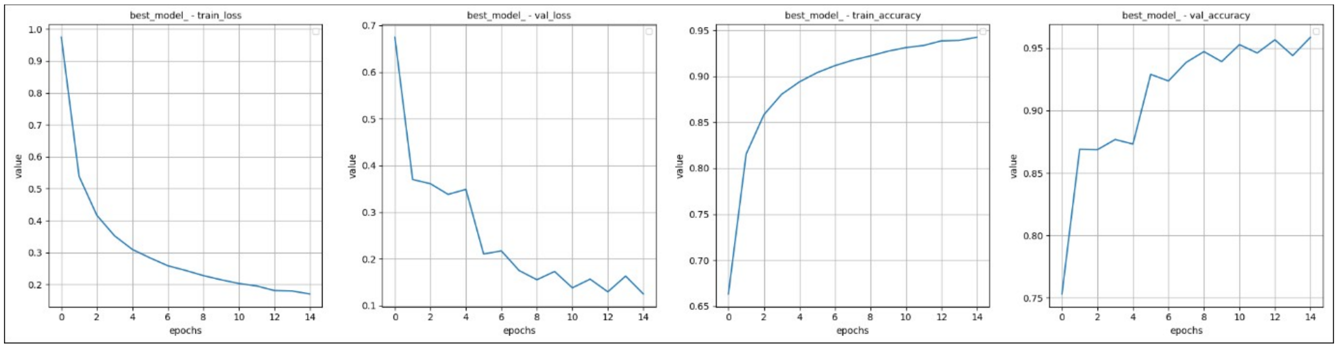
Best Model Architecture:

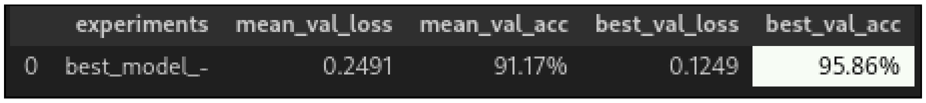


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 128, 28, 28]           3,584
       BatchNorm2d-2         [-1, 128, 28, 28]             256
              ReLU-3         [-1, 128, 28, 28]               0
         MaxPool2d-4           [-1, 128, 9, 9]               0
           Dropout-5           [-1, 128, 9, 9]               0
            Conv2d-6           [-1, 128, 9, 9]         147,584
       BatchNorm2d-7           [-1, 128, 9, 9]             256
              ReLU-8           [-1, 128, 9, 9]               0
           Dropout-9           [-1, 128, 9, 9]               0
           Conv2d-10           [-1, 128, 9, 9]         147,584
      BatchNorm2d-11           [-1, 128, 9, 9]             256
             ReLU-12           [-1, 128, 9, 9]               0
        MaxPool2d-13           [-1, 128, 3, 3]               0
          Dropout-14           [-1, 128, 3, 3]               0
           Conv2d-15           [-1, 128, 3, 3]         147,584
      BatchNorm2d-16           [-1, 128, 3, 3]             256
             ReLU-17           [-1, 128, 3, 3]               0
        MaxPool2d-18           [-1, 128, 1, 1]               0
          Dropout-19           [-1, 128, 1, 1]               0
          Flatten-20                 [-1, 128]               0
           Linear-21                 [-1, 128]          16,512
      BatchNorm1d-22                 [-1, 128]             256
             ReLU-23                 [-1, 128]               0
          Dropout-24                 [-1, 128]               0
           Linear-25                   [-1, 9]           1,161
================================================================
Total params: 465,289
```

Evaluation

Train - Validation Metrics





	experiments	mean_val_loss	mean_val_acc	best_val_loss	best_val_acc
0	best_model_-	0.2491	91.17%	0.1249	95.86%

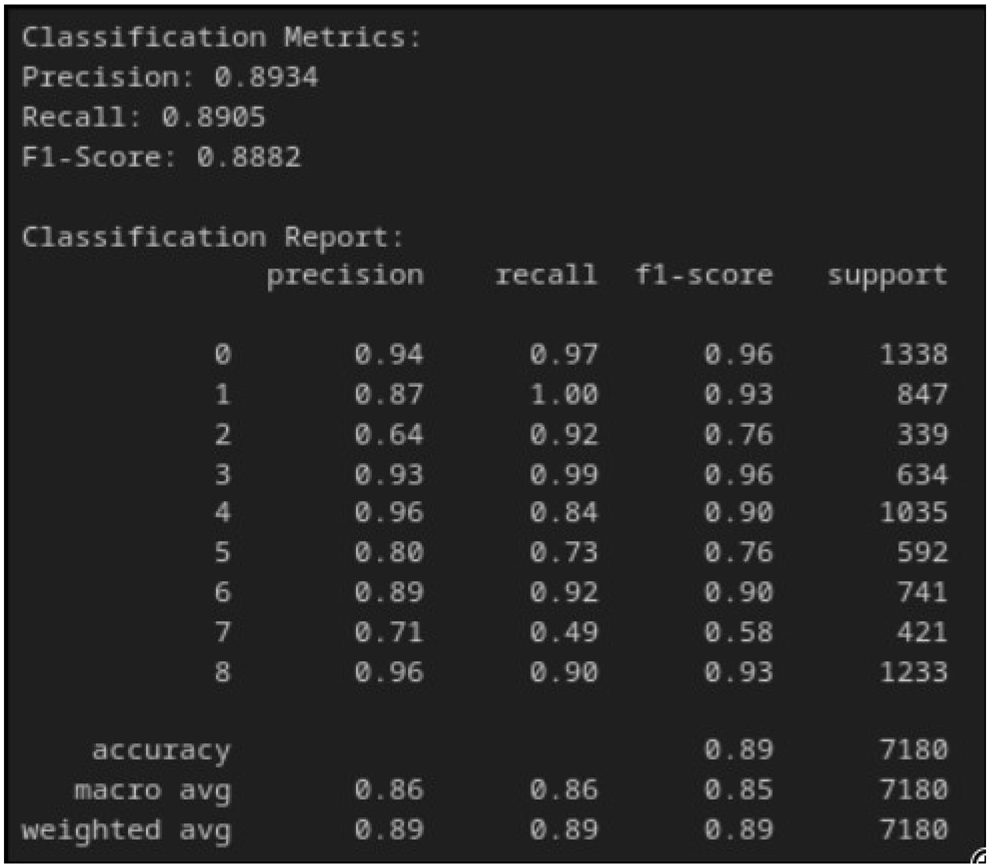
Best model trained in 3 minutes 50 seconds.Best model performed better from all previous experiments in terms of validation metrics. Validation graphs are nearly linear to train graphs which is a sign of good learning. There is no sharp spikes which also a good sign. Validation accuracy goes decently high up to %95.8 with mean of %91.1.

Test Results

Best Models performance on the test split:

- **Test Accuracy:** %89.05
- **Test Loss:** 0.4888
- **Precision:** %89.34
- **Recall:** %89.05,
- **F1-Score:** %88.82

Best Model's general performance can be considered as good except for 2nd and 7th classes. This difference might caused by imbalanced classes or just models issue. You can see the classes individual test scores from classification report and confussion matrix:

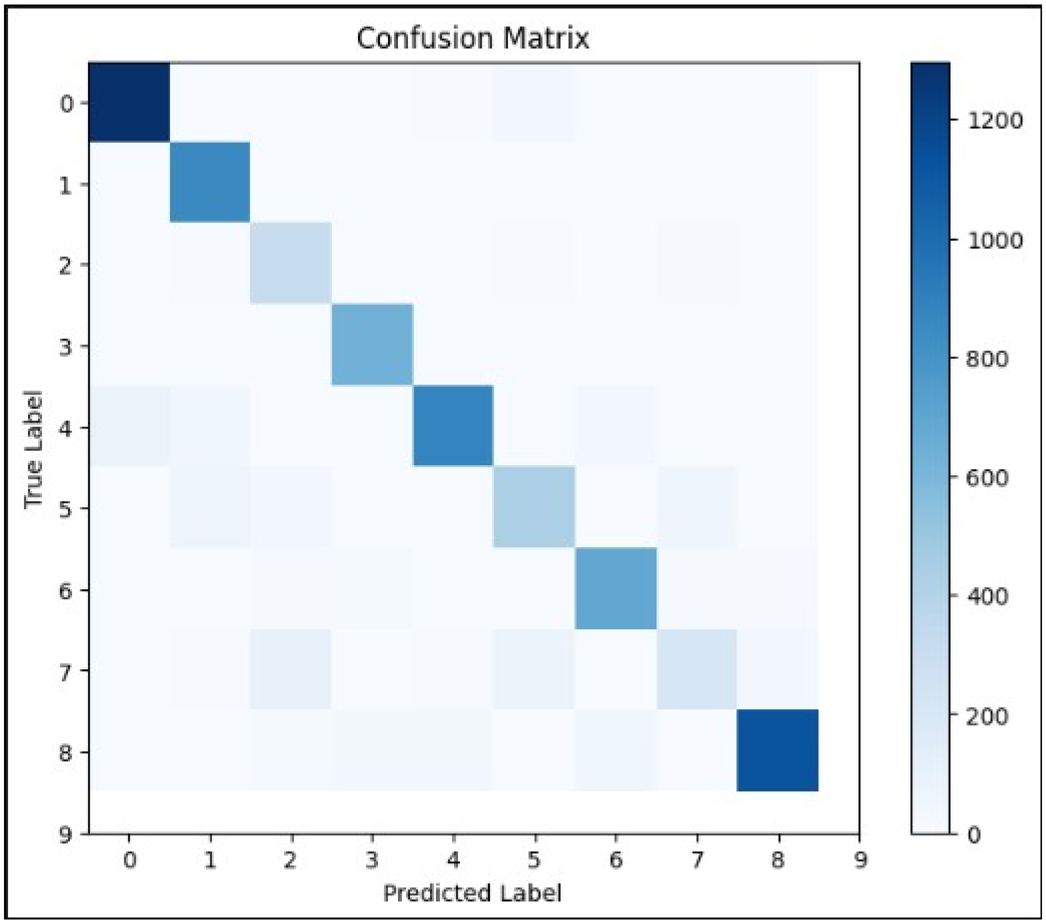


```
Classification Metrics:
Precision: 0.8934
Recall: 0.8905
F1-Score: 0.8882

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1338
           1       0.87      1.00      0.93       847
           2       0.64      0.92      0.76       339
           3       0.93      0.99      0.96       634
           4       0.96      0.84      0.90      1035
           5       0.80      0.73      0.76       592
           6       0.89      0.92      0.90       741
           7       0.71      0.49      0.58       421
           8       0.96      0.90      0.93      1233

    accuracy                           0.89      7180
   macro avg       0.86      0.86      0.85      7180
weighted avg       0.89      0.89      0.89      7180
```



Conclusion

Experimenting all hyperparameters on some model, provides really good understanding about each hyperparameter’s duty and effect to overall. With this approch you are not leaving anything to luck. It didn’t surprised me to obtain nearly 90% test accuracy because of my detailed and carefull work.

I gained a strong understanding about how batch size effects the models outcomes. Low and High batch sizes became a disadvantage and ı should have find the optimal value. I realized increasing dropout rates higher than 30% leads to underfitting and loss of important features. The next very important learning of mine was max pooling and using it without proper padding. When more than 3 convolution layer and max pooling added, model gived warnings about the input dimensions. It was becoming so small because of the lack of padding. I overcome this issue by not using more than 3 max pooling.

The findings showed that you should’t consider a parameter value as best if it gives highest validation accuracy or lowest loss. You really be carefull about overfitting. For instance using 20 or 25 epochs seemed reasonable for base model but when training the best model it was clear that model started to memorize after 15th epoch. So it is a good practice to never stop experimenting.

It is advised to do more experiments with considering this best model as base model because each hyper parameter works in a corolation. They might give better results with other parameters and their -recently titled as- “worse” values.