Project 3

A. Burak Gulhan

March 20, 2023

Contents

1 Part 1: Taiji Keypose Classification with MLP

1.1 Introduction

For this part of project 3, I used an MLP to improve Taiji Classification. I trained and tesed 2 different models, one model is the same as the baseline MLP model, but with an added ReLU layer, which makes the classification non-linear. The other model contains two layers, each followed by a ReLU layer.

NOTE: I did not put images for Baseline+ReLU, since there would be too many images in the report and because the results of Baseline+ReLU are almost identical to the improved model. However, the accuracy and std results for Baseline+ReLU are shown in table 1 and 2.

1.2 Running Code

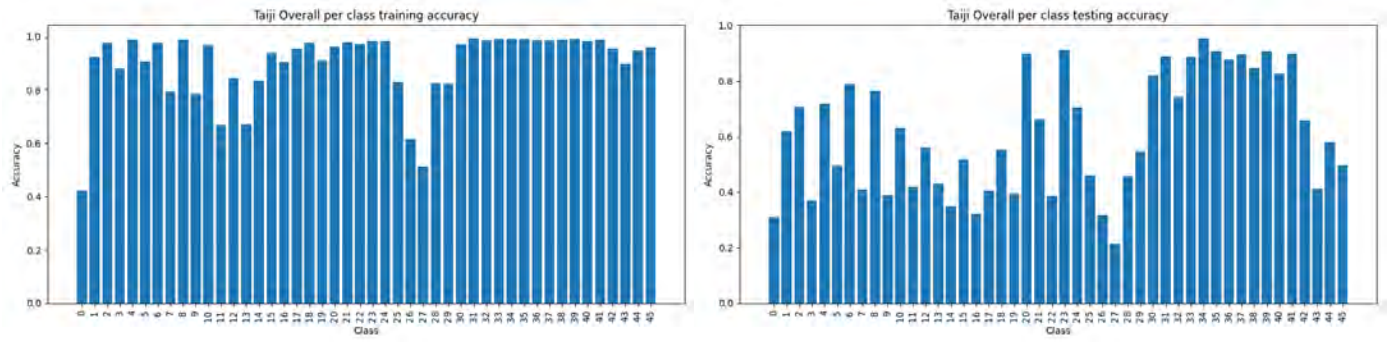
Use parameter '–model MLP2' to run baseline+ReLU model.
Use parameter '–model MLP3' to run improved model.

1.3 Baseline Model Results

The following images are and tables are results obtained for the baseline model for both full and lod4 datasets. Both models are trained with 10 epochs.

1.3.1 Full

The overall per class accuracy are shown in Fig. 1 The overall train and test confusion matrices are in Fig. 2. The test confusion matrix for subject 1 is shown in Fig. 3. The subjectwise accuracy is shown in Fig. 4. The training curve is shown in Fig. 5. The train and test accuracy and std is shown in table 1 and 2

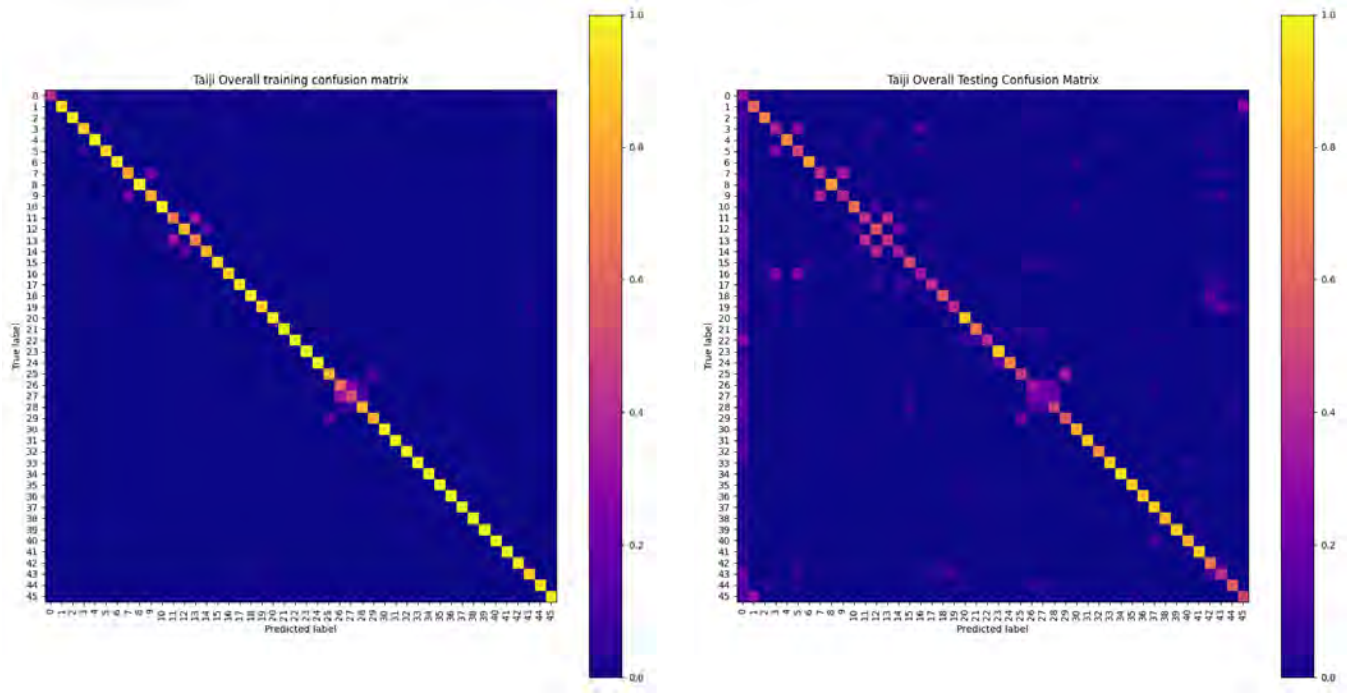


(a) train (b) test

Figure 1: Overall per class accuracy (using baseline with full dataset)

1.3.2 lod4

The overall per class accuracy are shown in Fig. 6 The overall train and test confusion matrices are in Fig. The test confusion matrix for subject 1 is shown in Fig. 8. The



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 2: Confusion matrices (using baseline with full dataset)

subjectwise accuracy is shown in Fig. 9. The training curve is shown in Fig. 10. The train and test accuracy and std is shown in table 1 and 2

1.4 Improved Model Results

The following images are and tables are results obtained for first improved (same as baseline but with two linear+ReLU layers) model for both full and lod4 datasets. Both models are trained with 10 epochs.

1.4.1 Full

The overall per class accuracy are shown in Fig. 11 The overall train and test confusion matrices are in Fig. 13. The test confusion matrix for subject 1 is shown in Fig. 12. The subjectwise accuracy is shown in Fig. 14. The training curve is shown in Fig. 15. The train and test accuracy and std is shown in table 1 and 2

1.4.2 lod4

The overall per class accuracy are shown in Fig. 16 The overall train and test confusion matrices are in Fig. 18. The subjectwise accuracy is shown in Fig. 19. The test confusion matrix for subject 1 is shown in Fig. 17 The training curve is shown in Fig. 20. The train and test accuracy and std is shown in table 1 and 2

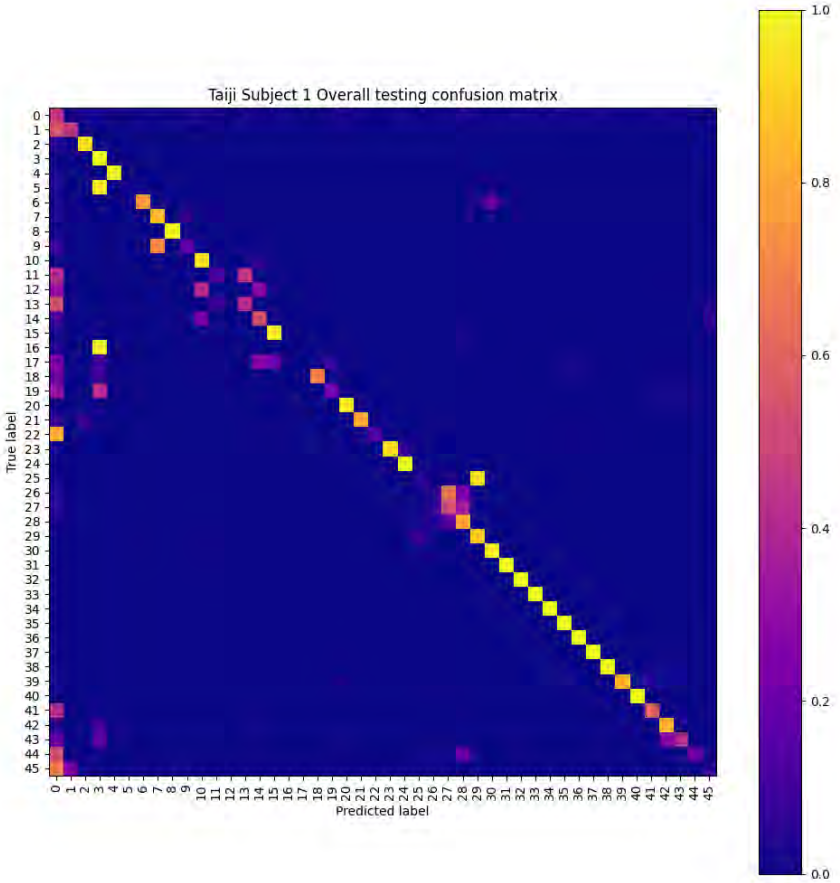


Figure 3: Subject 1 Test Confusion Matrix (using baseline with full dataset)

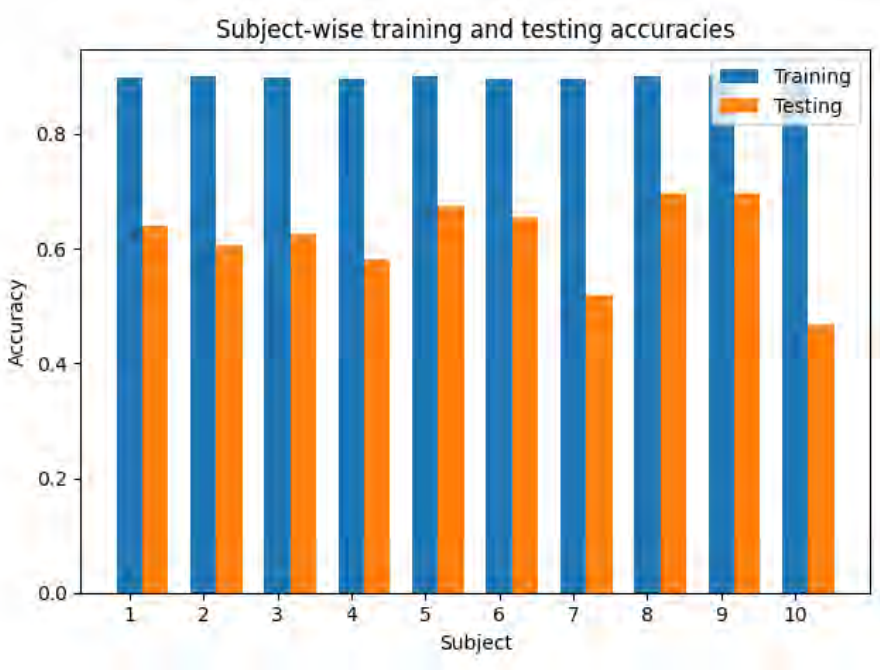


Figure 4: Subjectwise Accuracy (using baseline with full dataset)

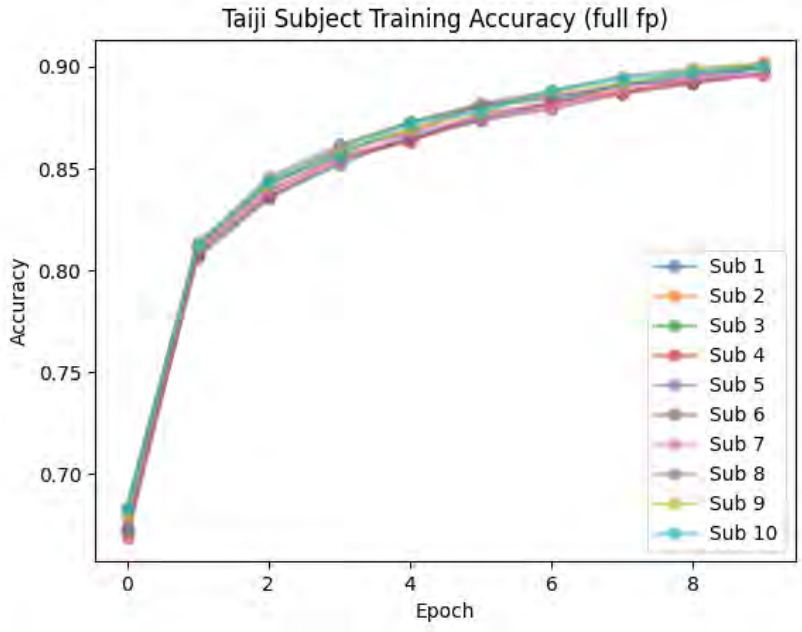
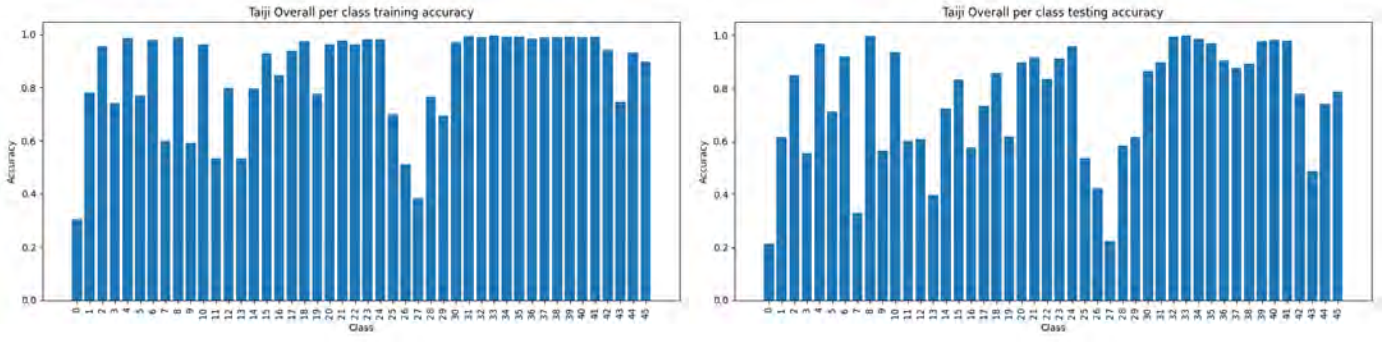


Figure 5: Training Curve (using baseline with full dataset)



(a) train (b) test

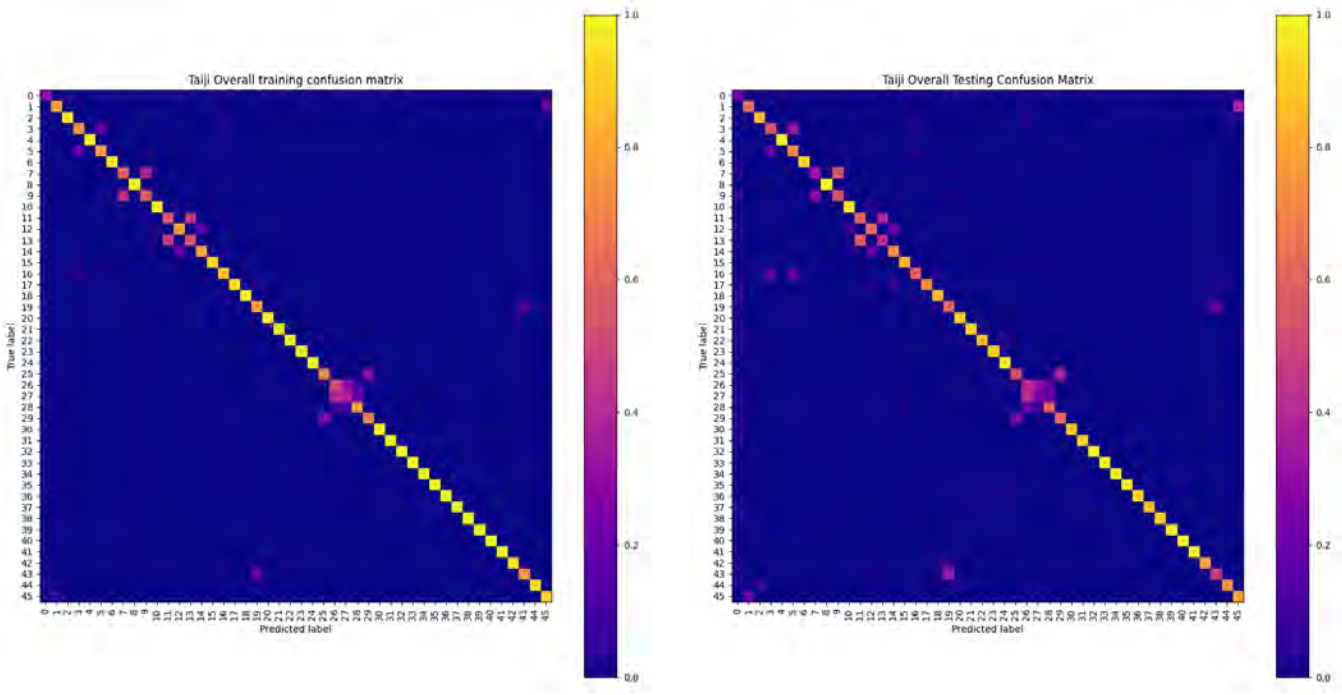
Figure 6: Overall per class accuracy (using baseline with lod4)

Table 1: Training accuracy and std

Method	Accuracy	Std
Baseline Full	0.8991	0.0020
Baseline lod4	0.8497	0.0024
Baseline+ReLU Full	0.9461	0.0016
Baseline+ReLU lod4	0.9122	0.0012
Improved Model Full	0.9364	0.0013
Improved Model lod4	0.9123	0.0018

Table 2: Testing accuracy and std

Method	Accuracy	Std
Baseline Full	0.6160	0.0720
Baseline lod4	0.7534	0.0490
Baseline+ReLU Full	0.6672	0.0414
Baseline+ReLU lod4	0.7698	0.0581
Improved Model Full	0.6968	0.0400
Improved Model lod4	0.7658	0.0469



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 7: Confusion matrices (using baseline with lod4)

1.5 Analysis of Results and Conclusion

In all models we can clearly see that using the Full dataset overfits, compared to lod4. We can observe this by the differences in the train and test results. Using the Full dataset we obtain a higher train, but a lower test compared to lod4. Another interesting observation between full and lod4 is that in the training curves and subjectwise accuracy figures, full obtains very similar accuracies for all subjects, while lod4 does not.

Adding a single ReLU to the baseline model gives significantly better results. This is because, ReLU is a nonlinear function, adding this allows our model to make non linear classification. In most subjects, there is an increase from using Baseline+ReLU, however for subject 8 and subject 9 this is not the case. Subject 8 has slightly lower accuracy in testing using ReLU with lod4 and subject 9 has the same accuracy. This may be because subject 8 and subject 9 are able to be classified linearly and that the Baseline+ReLU model overfits.

Lastly, the improved model which has two linear+ReLU layers, did not perform better than baseline+ReLU. In the training with lod4 it gets slightly lower accuracy but significantly lower std, the accuracy decrease may be due to randomness in model initialization. In the full dataset, the improved model performs better than baseline+ReLU.

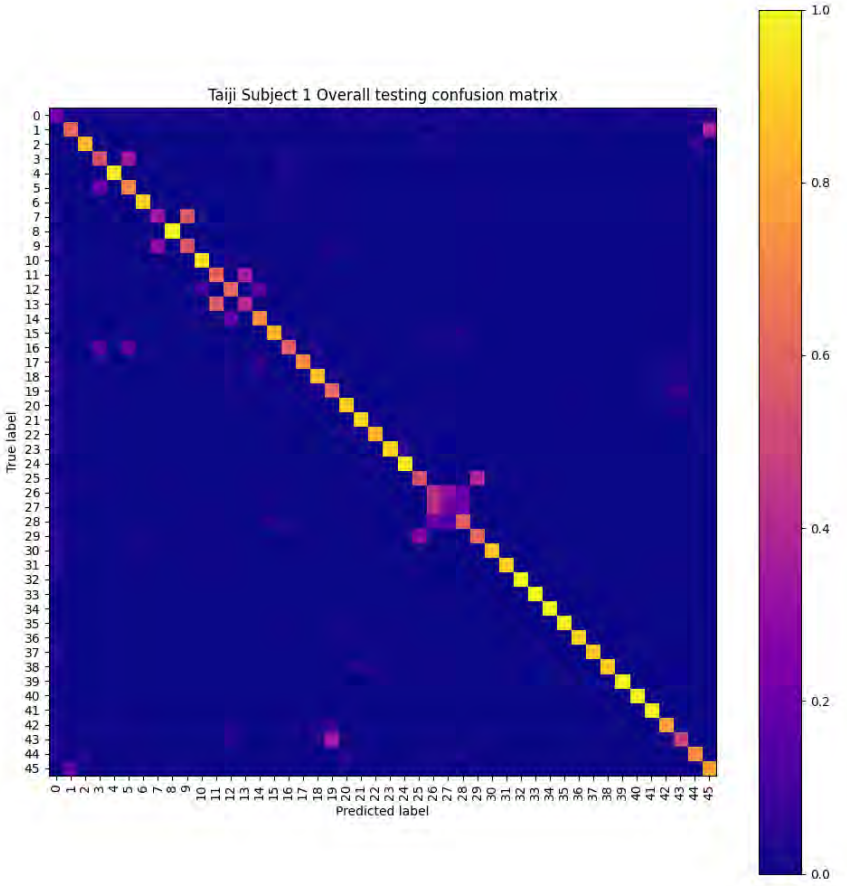


Figure 8: Subject 1 Test Confusion Matrix (using baseline with lod4)

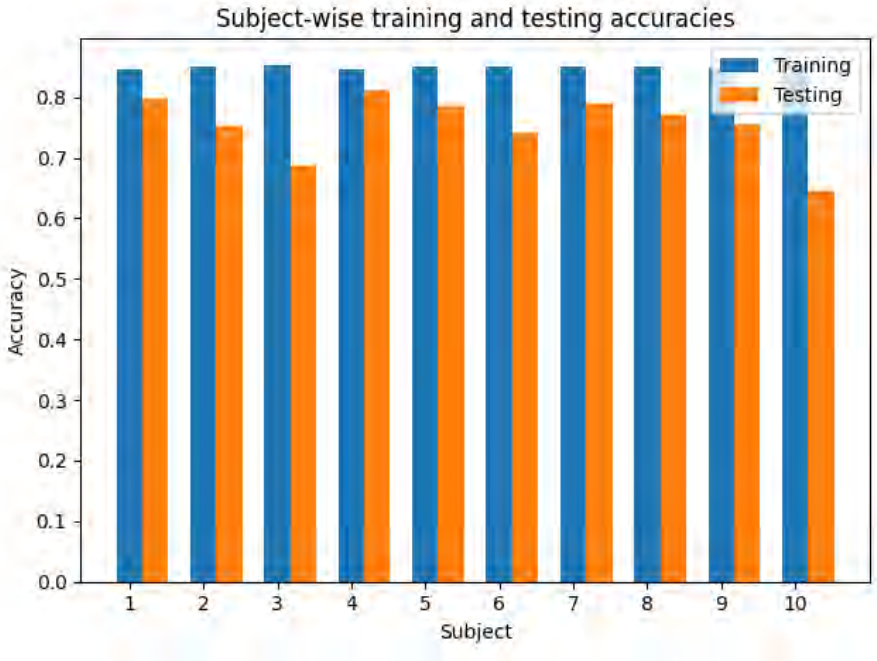


Figure 9: Subjectwise Accuracy (using baseline with lod4)

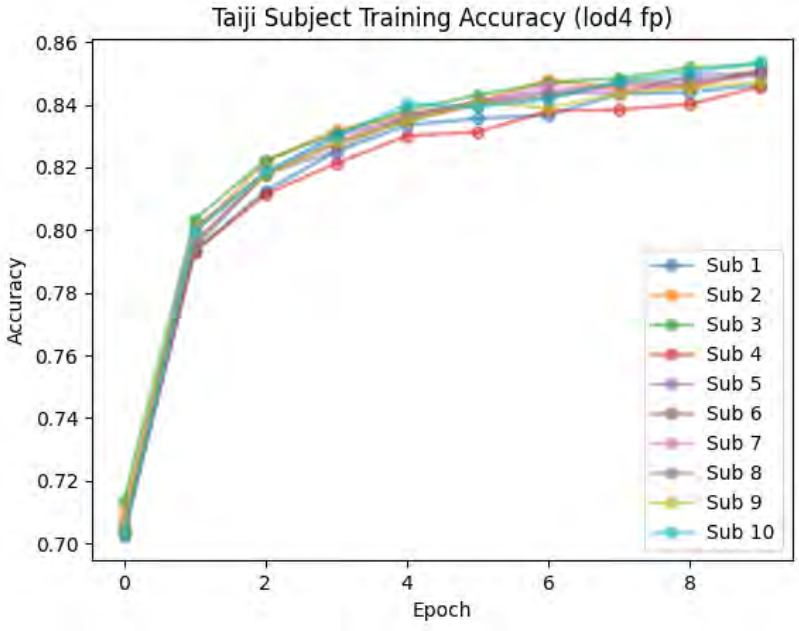
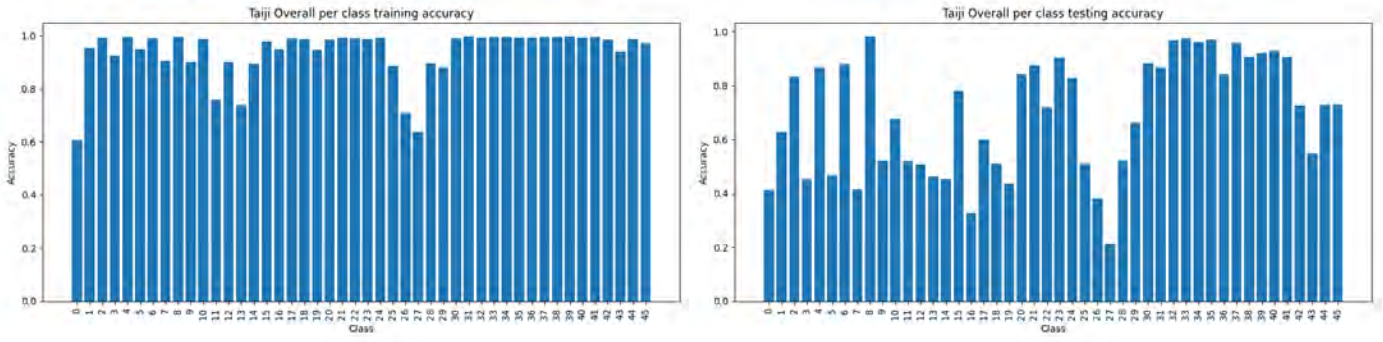


Figure 10: Training Curve (using baseline with lod4)



(a) train (b) test

Figure 11: Overall per class accuracy (using improved model with full dataset)

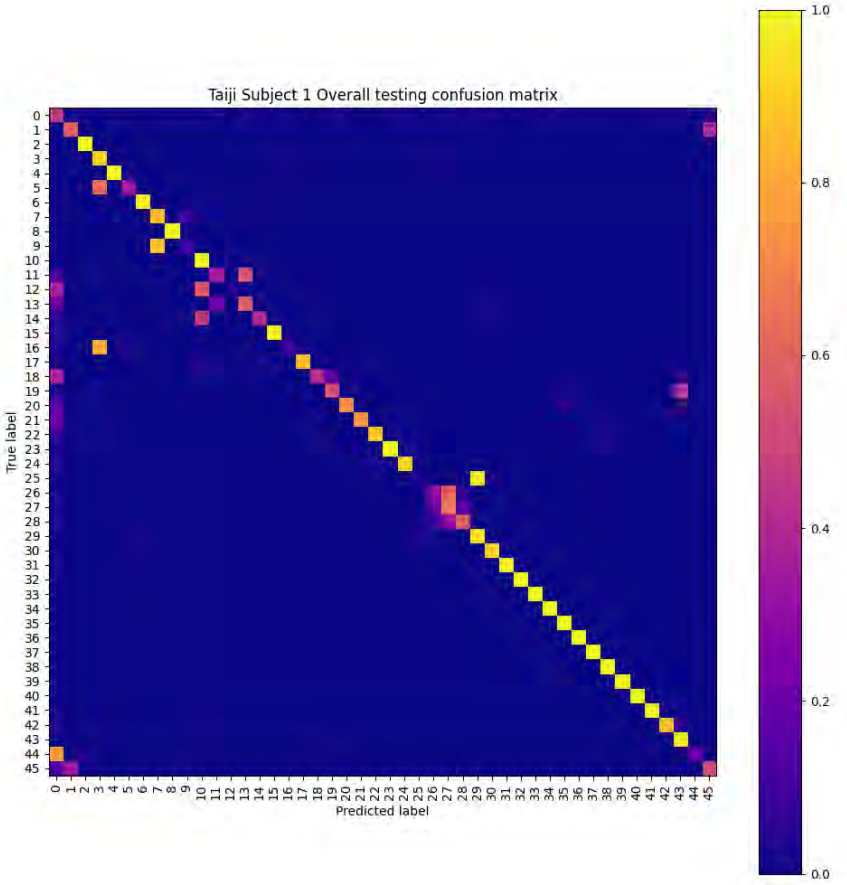
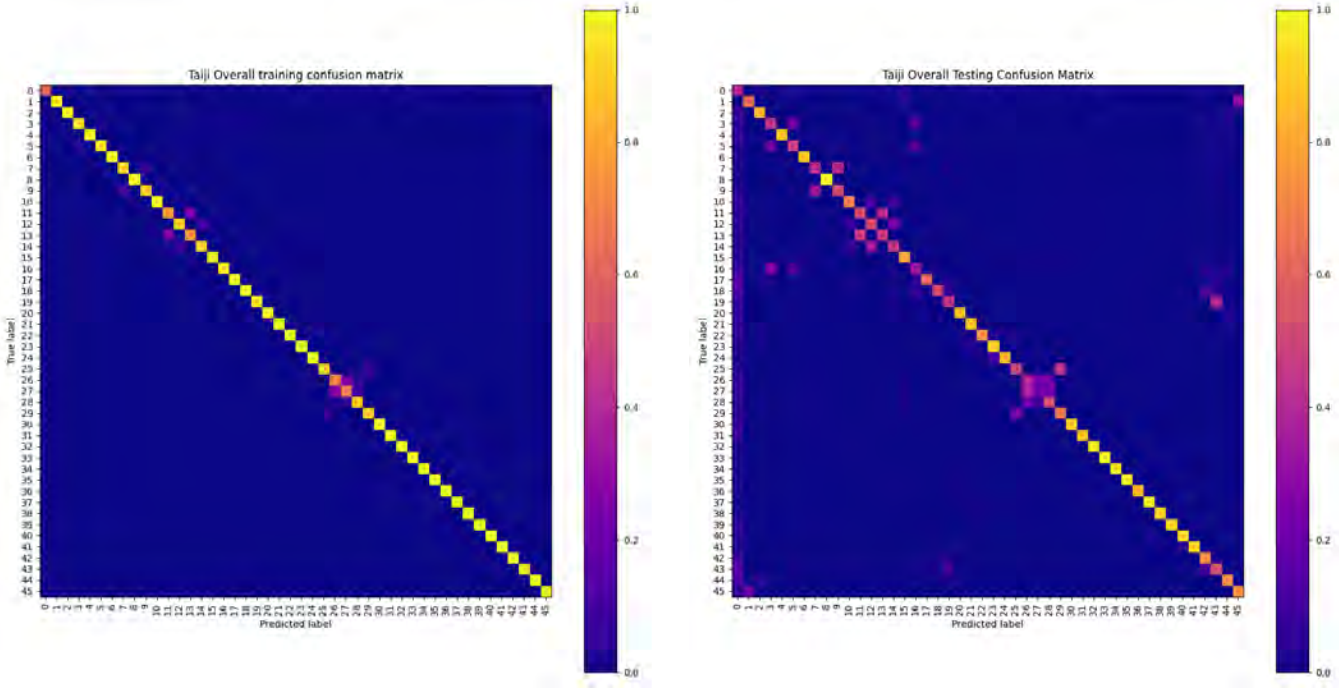


Figure 12: Subject 1 Test Confusion Matrix (using improved model with full dataset)



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 13: Confusion matrices (using improved model with full dataset)

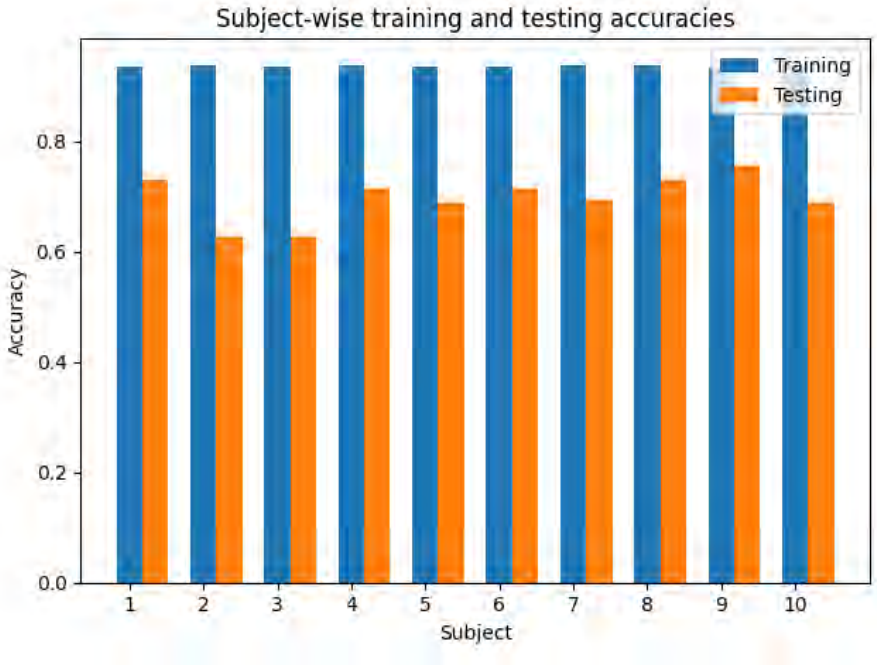


Figure 14: Subjectwise Accuracy (using improved model with full dataset)

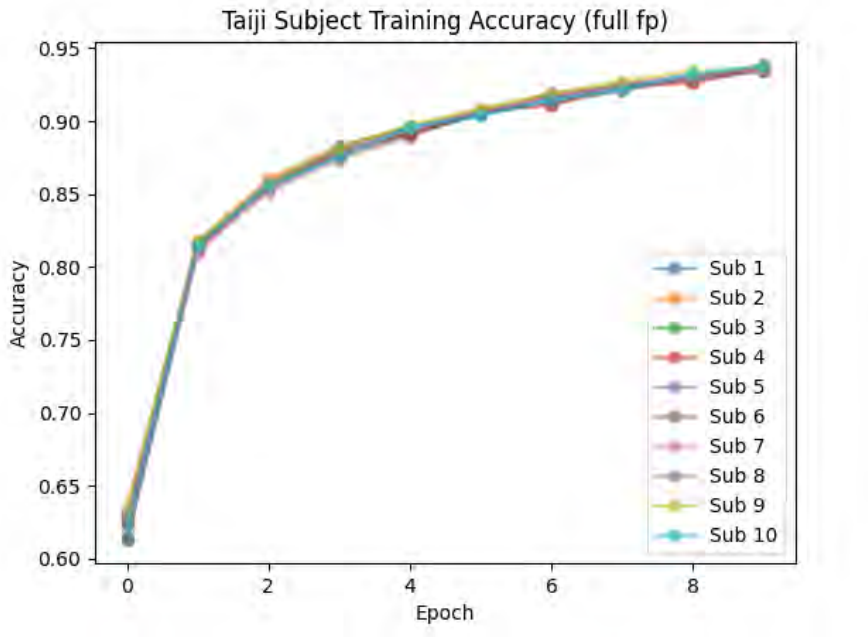
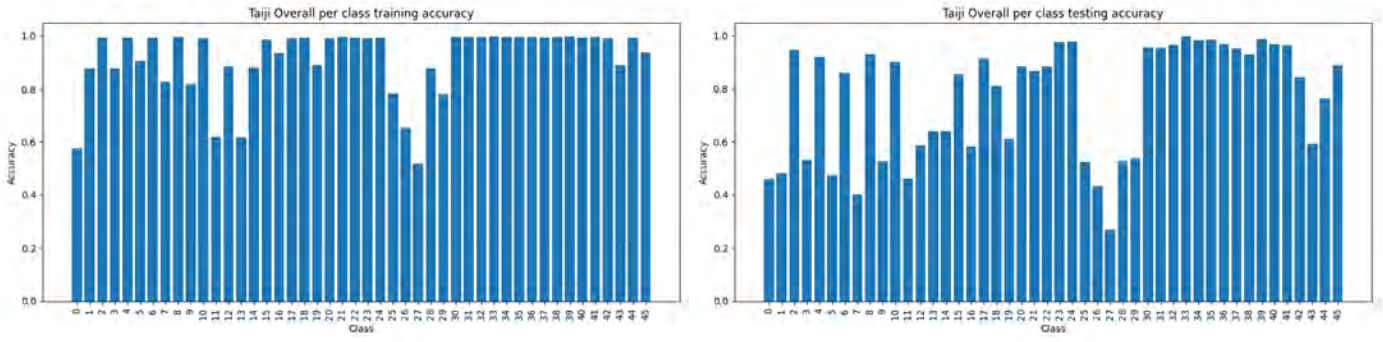


Figure 15: Training Curve (using improved model with full dataset)



(a) train (b) test

Figure 16: Overall per class accuracy (using improved model with lod4)

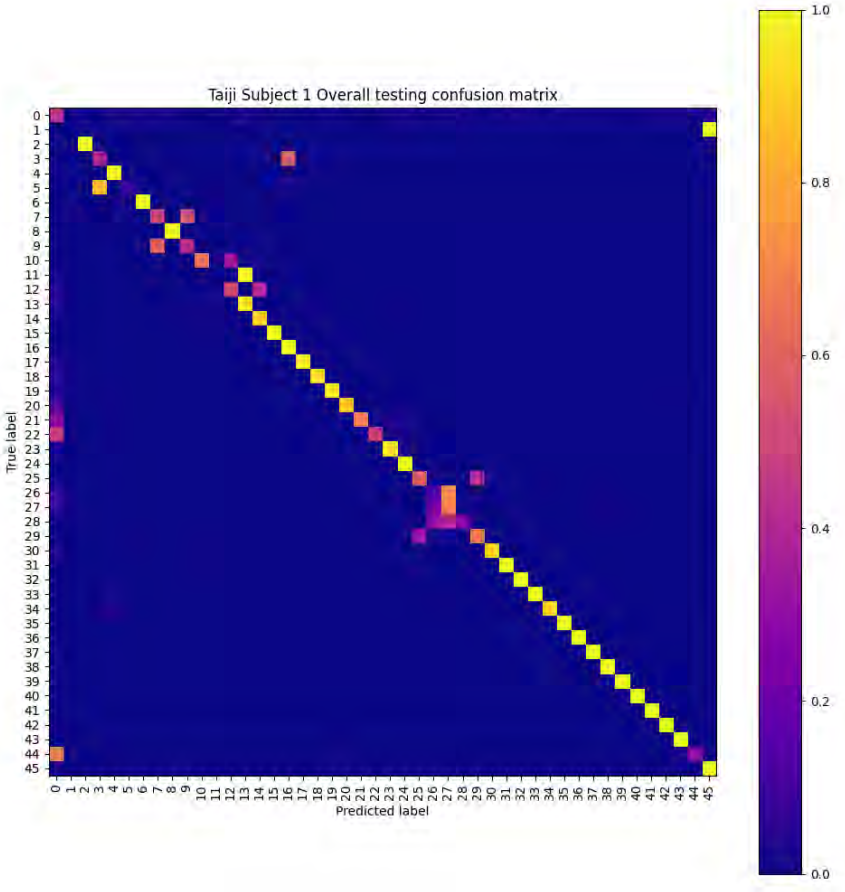
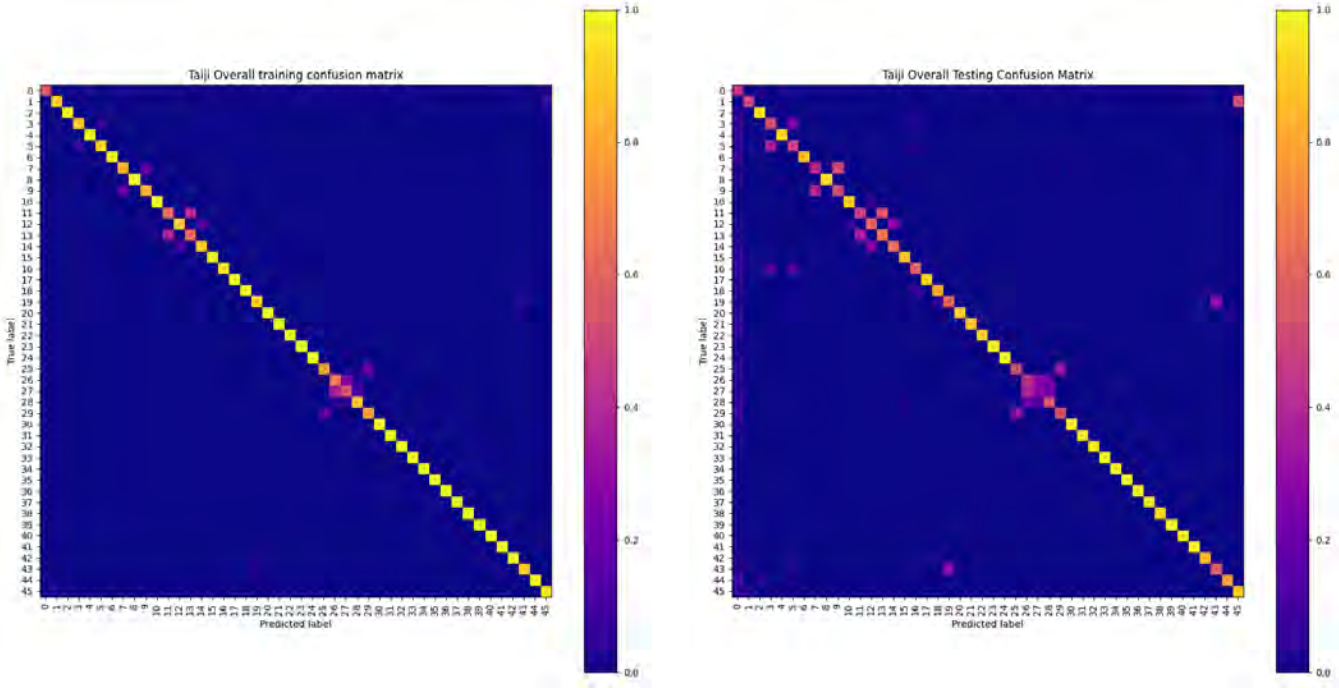


Figure 17: Subject 1 Test Confusion Matrix (using improved model with lod4 dataset)



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 18: Confusion matrices (using improved model with lod4)

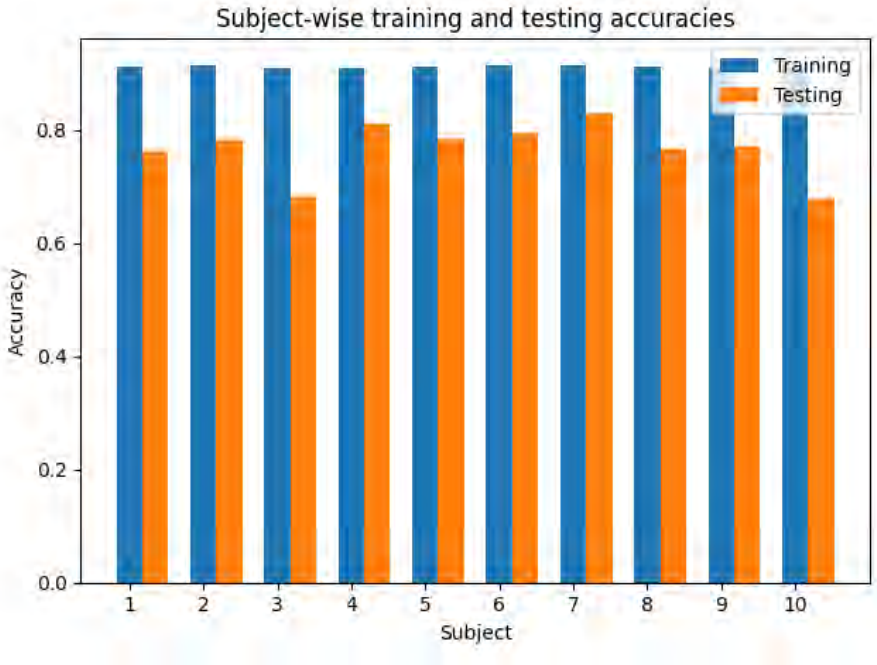


Figure 19: Subjectwise Accuracy (using improved model with lod4)

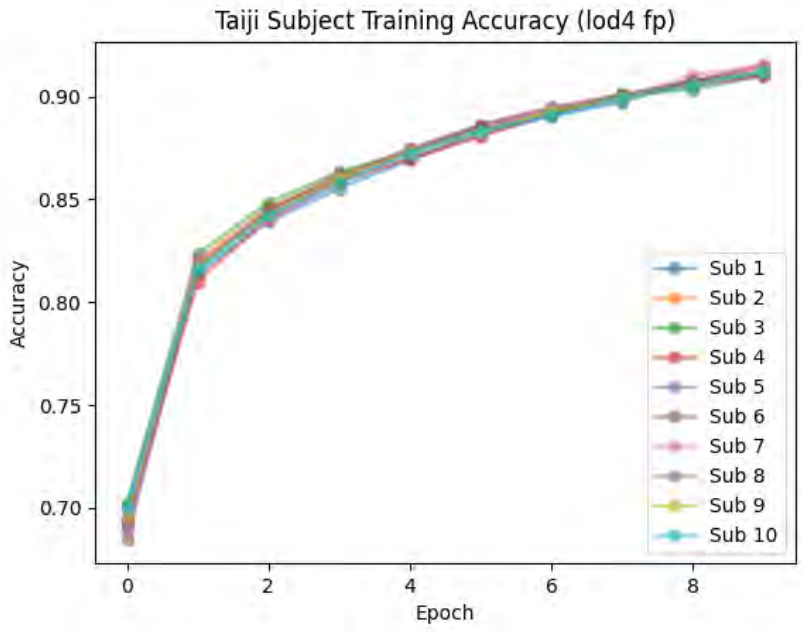


Figure 20: Training Curve (using improved model with lod4)

2 Part 2: Wallpaper Classification with CNN

2.1 Introduction

For this part of the project I implemented a CNN model that uses data augmentation to classify wallpapers. I used a model similar to what was used in the paper EscherNet, which was posted on Canvas.

I also added a dropout layer before each fully connected layer, since I found that even with data augmentation for the challenge dataset, the model has significant overfitting.

For augmentations, I first rotated each image randomly between 0 and 360 degrees. Then I removed black corners due to rotation. To do this I center cropped each image to 181 pixels, since each input image is 256x256 pixels and in the worst case of 45 degree rotation (or a multiple of 45) the maximum square without black edges would be 256*cos(45) = 181. Then I shifted the image up between -8 and 8 pixels up and right (I did not shift more, since shifting results in black edges and thus reduces the size of the image). Finally I zoomed between 100 to 200% of the image.

I also tried erasing random parts of the image as an additional transformation step, but I found that this requires more epochs to train and has worse results.

2.2 Running Code

Use parameter '–model CNN2' to run the updated architecture.

2.3 Baseline Model Results

NOTE: All results are trained with 10 epochs unless stated otherwise.

2.3.1 Test Dataset

Train and test accuracy and std are shown in Table 3 and 4. Overall per class accuracy for train and test is shown in Fig. 22. Overall train and test confusion matrix are shown in Fig. 23. t-SNE results are shown in figure 21.

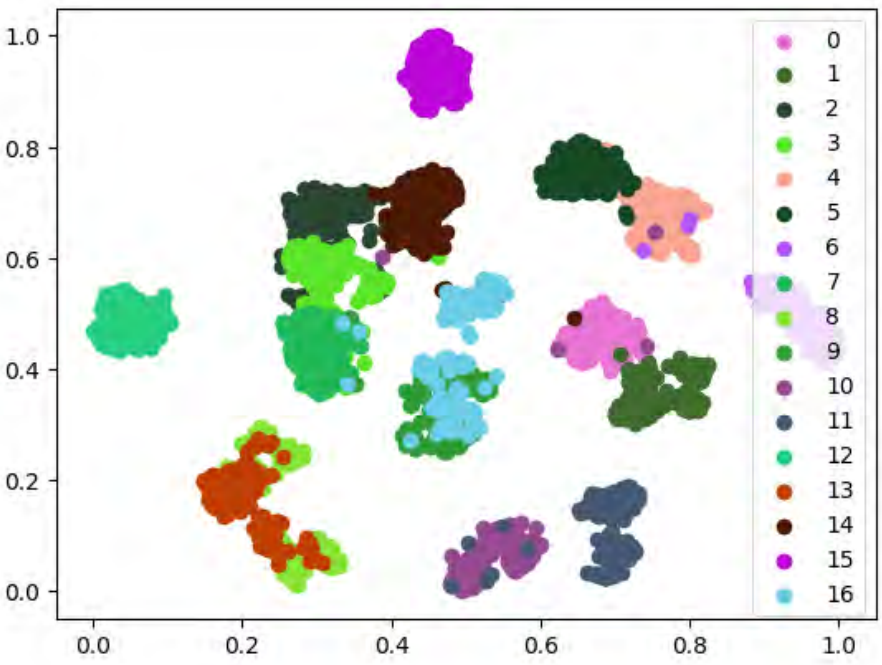
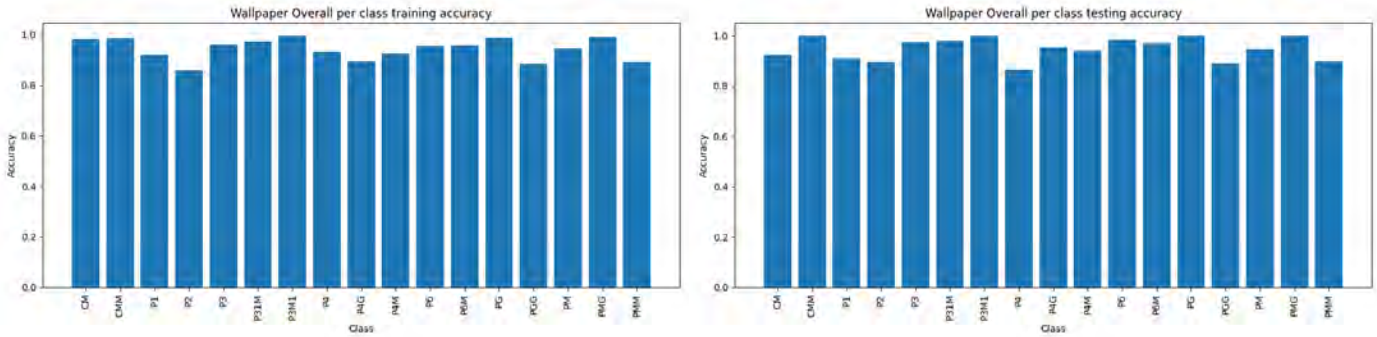


Figure 21: t-SNE plot for Baseline on 'test' dataset

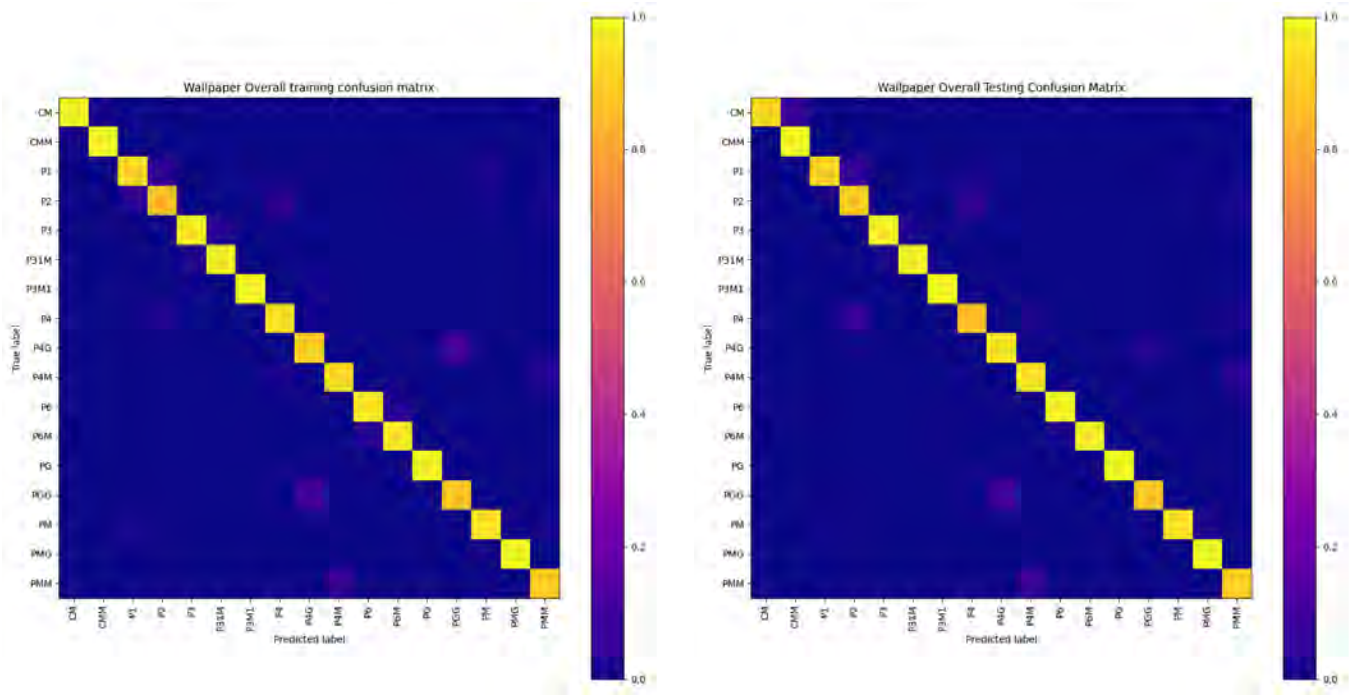


(a) train (b) test

Figure 22: Overall per class accuracy Baseline, using 'Test' dataset

2.3.2 Test Challenge Dataset

Train and test accuracy and std are shown in Table 3 and 4.



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 23: Confusion matrices using 'Test' dataset

2.4 Updated Architecture WITHOUT Data Augmentation

2.4.1 Test Dataset

Train and test accuracy and std are shown in Table 3 and 4. Overall per class accuracy for train and test is shown in Fig. 25. Overall train and test confusion matrix are shown in Fig. 26. t-SNE plot shown in Fig. 24

2.4.2 Challenge Dataset

Train and test accuracy and std are shown in Table 3 and 4. t-SNE plot shown in Fig. 27

2.5 Updated Architecture WITH Data Augmentation

NOTE: Using data augmentation leads to slower convergence (this can be observed in the training curve), therefore some results are trained with more than 10 epochs and are stated if so. I did not run 'Test' dataset to convergence due to limited time.

2.5.1 Test Dataset

Train and test accuracy and std are shown in Table 3 and 4. t-SNE plot shown in Fig. 28

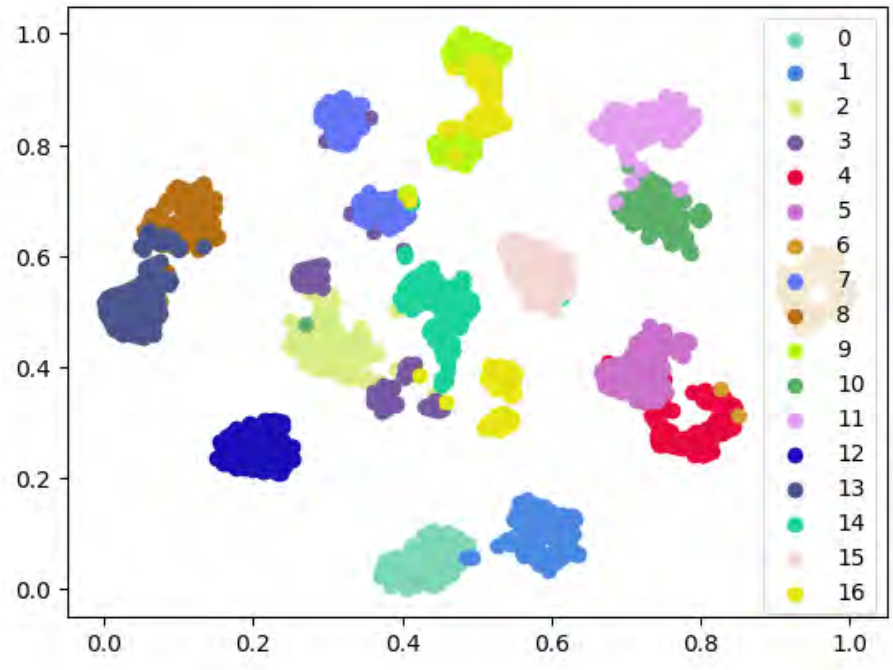
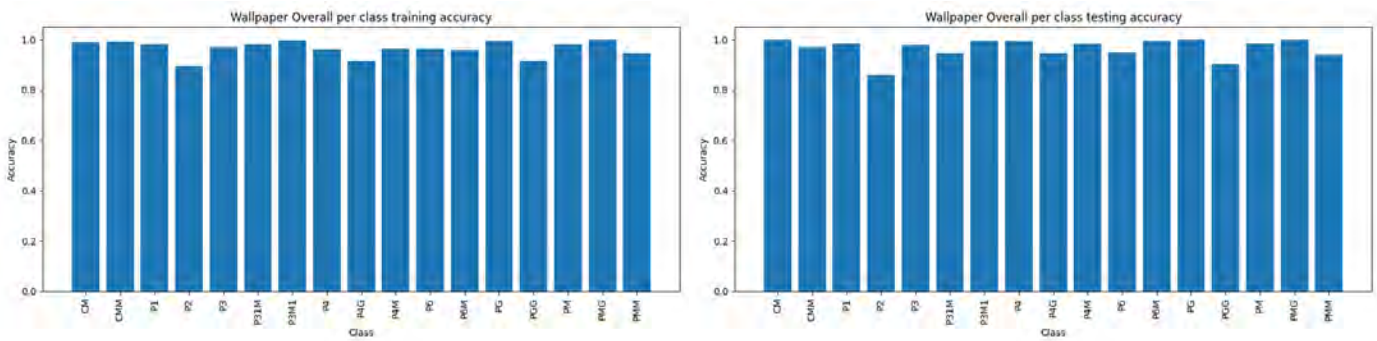
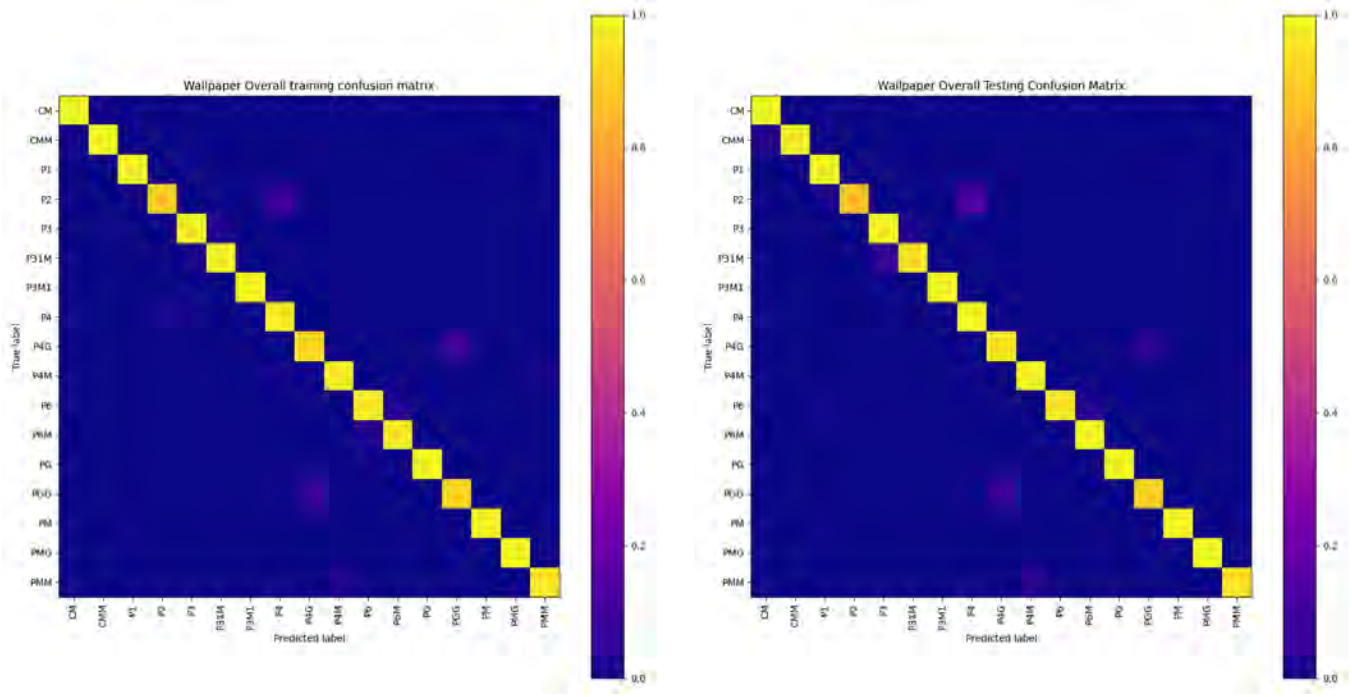


Figure 24: t-SNE plot for Updated Architecture on 'test' dataset



(a) train (b) test

Figure 25: Overall per class accuracy, using 'Test' dataset with Updated Architecture



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 26: Confusion matrices using 'Test' dataset

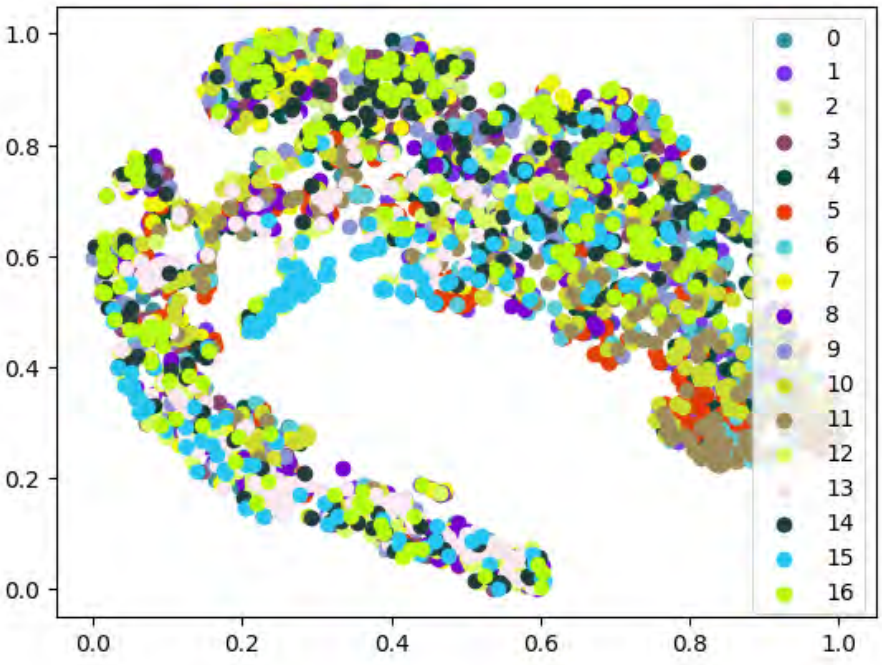


Figure 27: t-SNE plot for Updated Architecture on 'challenge' dataset

2.5.2 Challenge Dataset

Train and test accuracy and std are shown in Table 3 and 4.

Using 10 epochs Overall per class accuracy for train and test is shown in Fig. 31. Overall train and test confusion matrix are shown in Fig. 32. Training curve shown in Fig. 33. t-SNE plot shown in Fig. 29

Using 360 epochs Overall per class accuracy for train and test is shown in Fig. 34. Overall train and test confusion matrix are shown in Fig. 35. Training curve shown in Fig. 36. t-SNE plot shown in Fig. 30

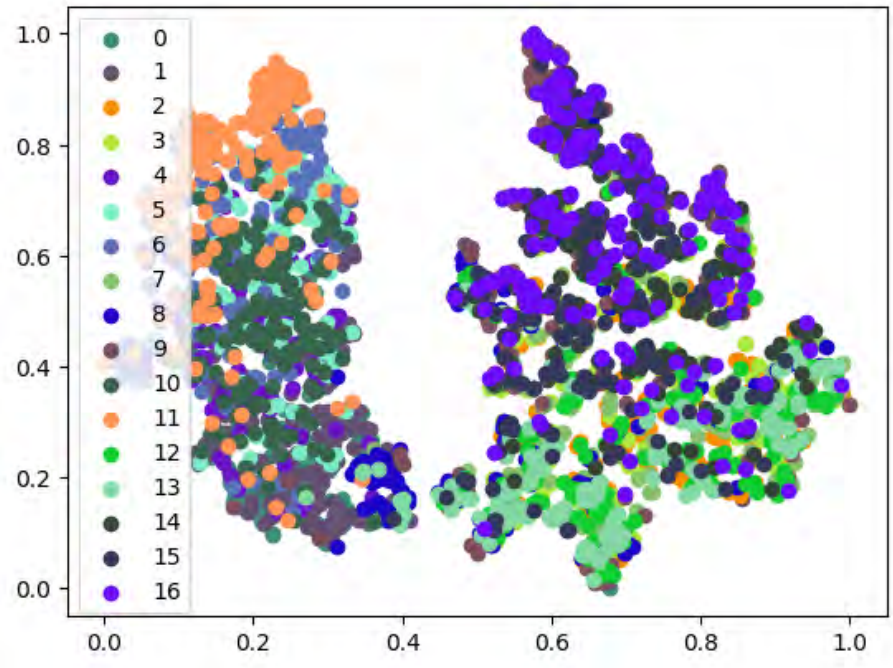


Figure 28: t-SNE plot for Updated Architecture + augmentation on 'train' dataset

Table 3: Training accuracy and std

Architecture+Dataset	Accuracy	Std
Baseline 'Test'	0.9444	0.0413
Baseline 'Challenge'	0.9294	0.0558
Updated 'Test'	0.9661	0.0308
Updated 'Challenge'	0.9712	0.0253
Upd+augmnt 'Test'	0.3913	0.1700
Upd+augmnt 'Challenge'	0.3564	0.2014
Upd+augmnt 'Challenge' 360 epochs	0.5450	0.1722

Table 4: Testing accuracy and std

Architecture+Dataset	Accuracy	Std
Baseline 'Test'	0.9491	0.0433
Baseline 'Challenge'	0.1094	0.1714
Updated 'Test'	0.9668	0.0376
Updated 'Challenge'	0.1312	0.2035
Upd+augmnt 'Test'	0.0815	0.2056
Upd+augmnt 'Challenge'	0.0853	0.1318
Upd+augmnt 'Challenge' 360 epochs	0.0635	0.0717

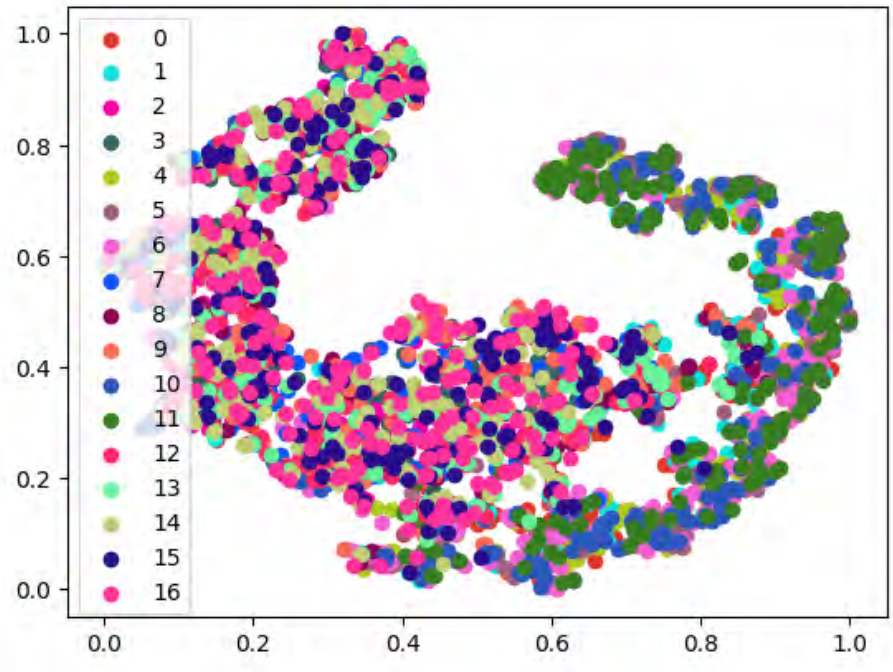


Figure 29: t-SNE plot for Updated Architecture + augmentation on 'challenge' dataset

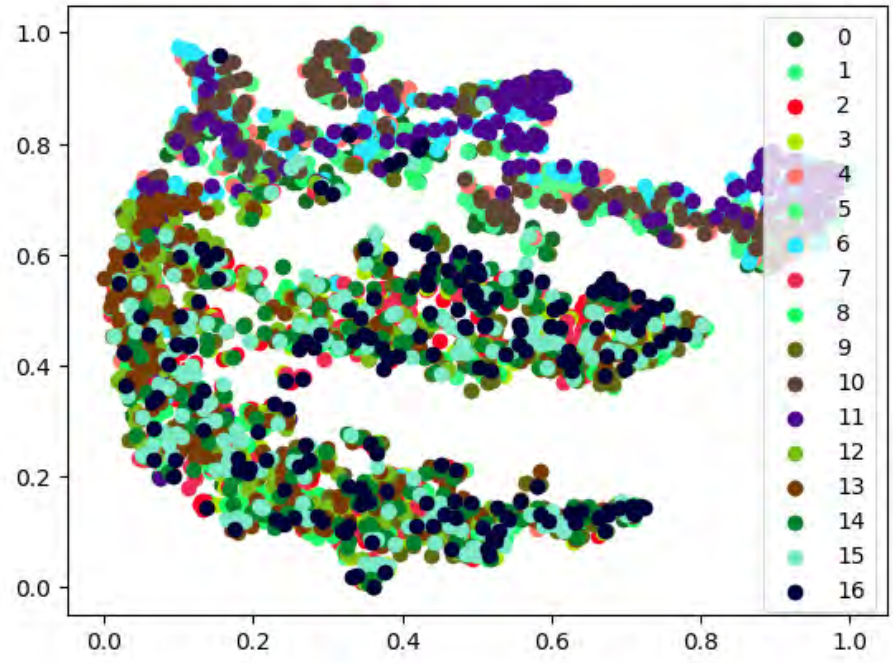
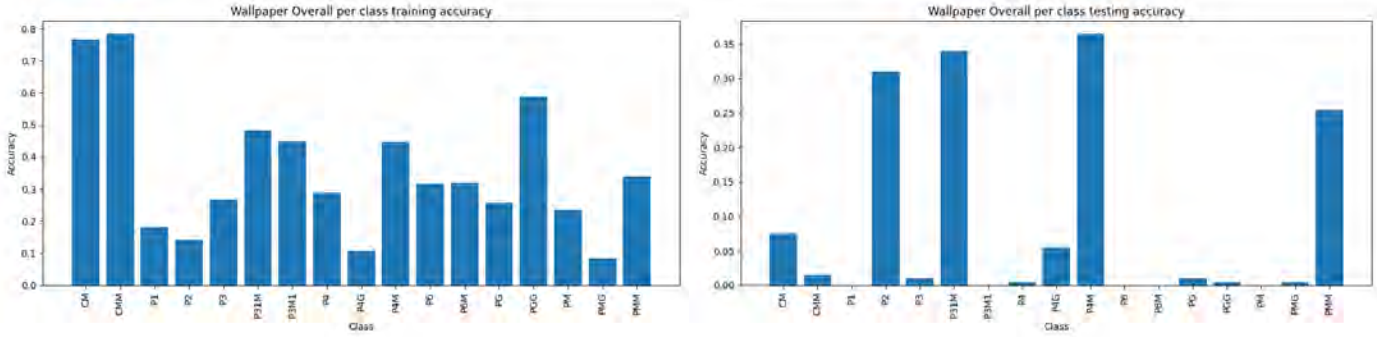
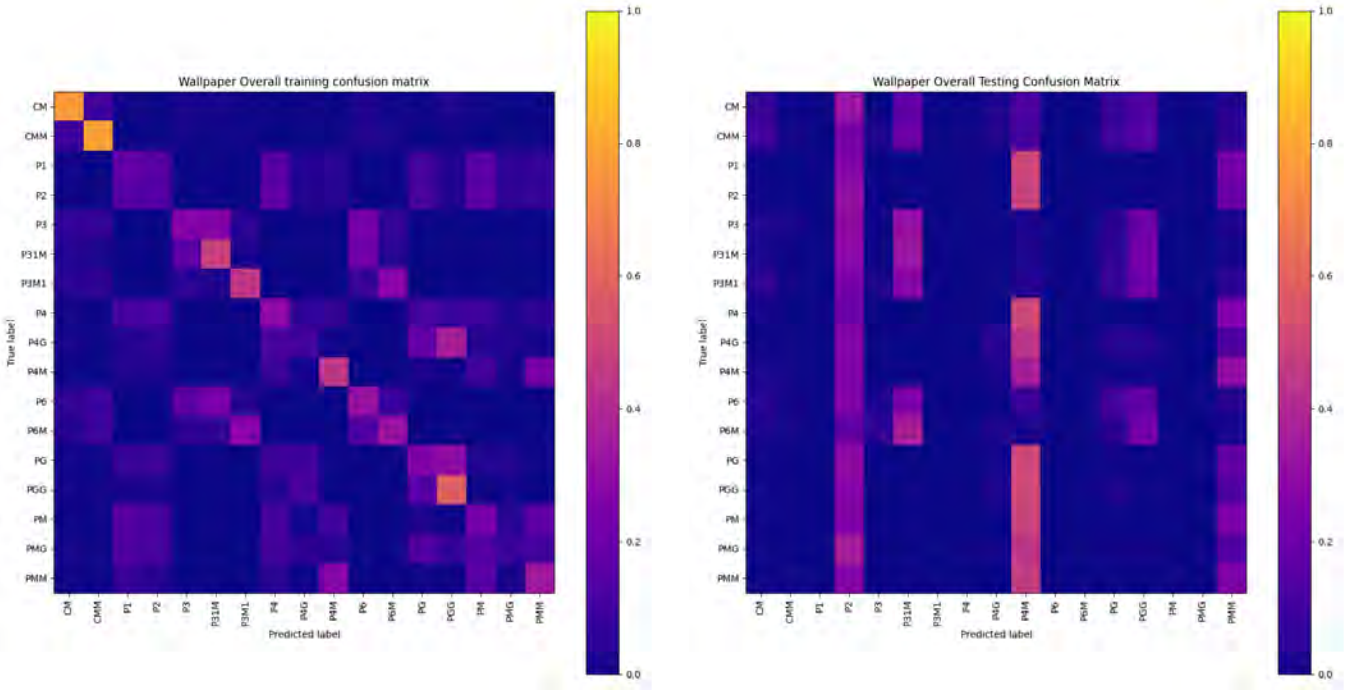


Figure 30: t-SNE plot for Updated Architecture + augmentation on 'challenge' dataset, trained with 360 epochs



(a) train (b) test

Figure 31: Overall per class accuracy, using 'Challenge' dataset with Updated Architecture + augmentation (10 epochs)



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 32: Confusion matrices using 'Challenge' dataset with Updated Architecture + augmentation (10 epochs)

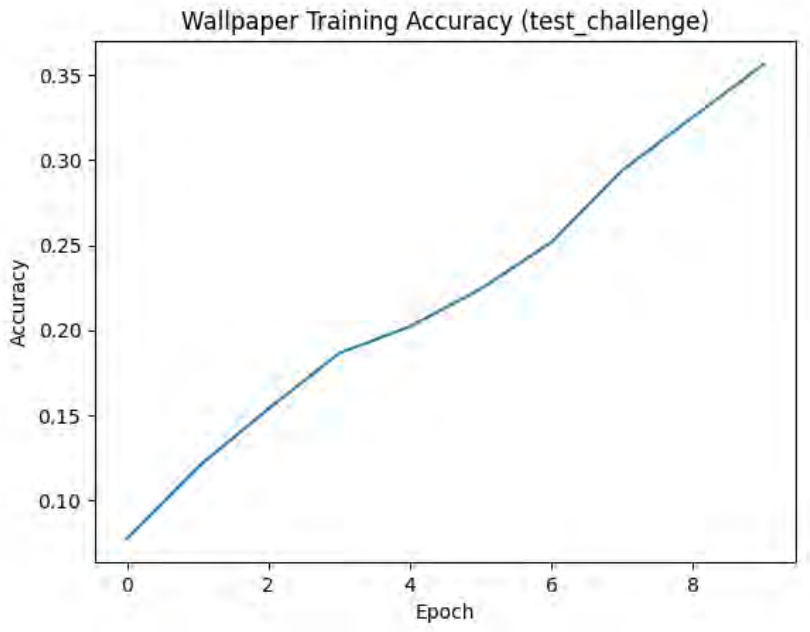
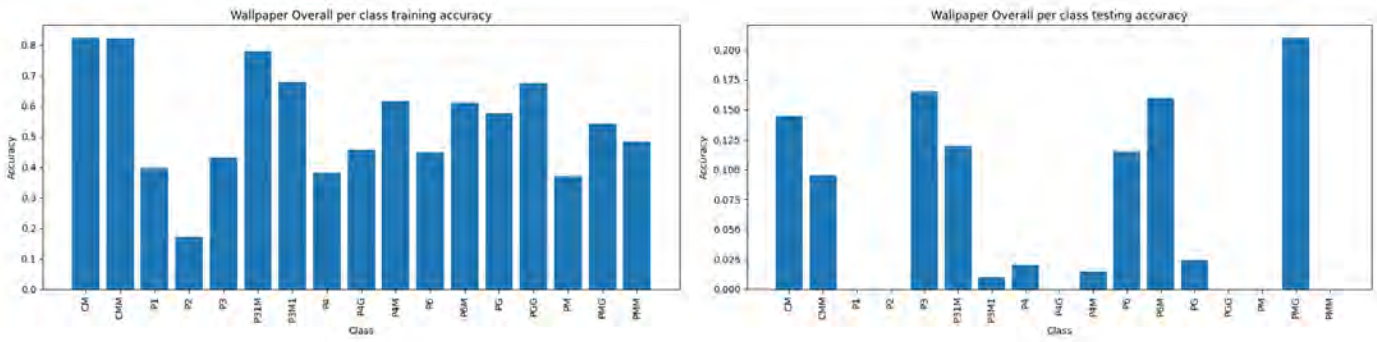
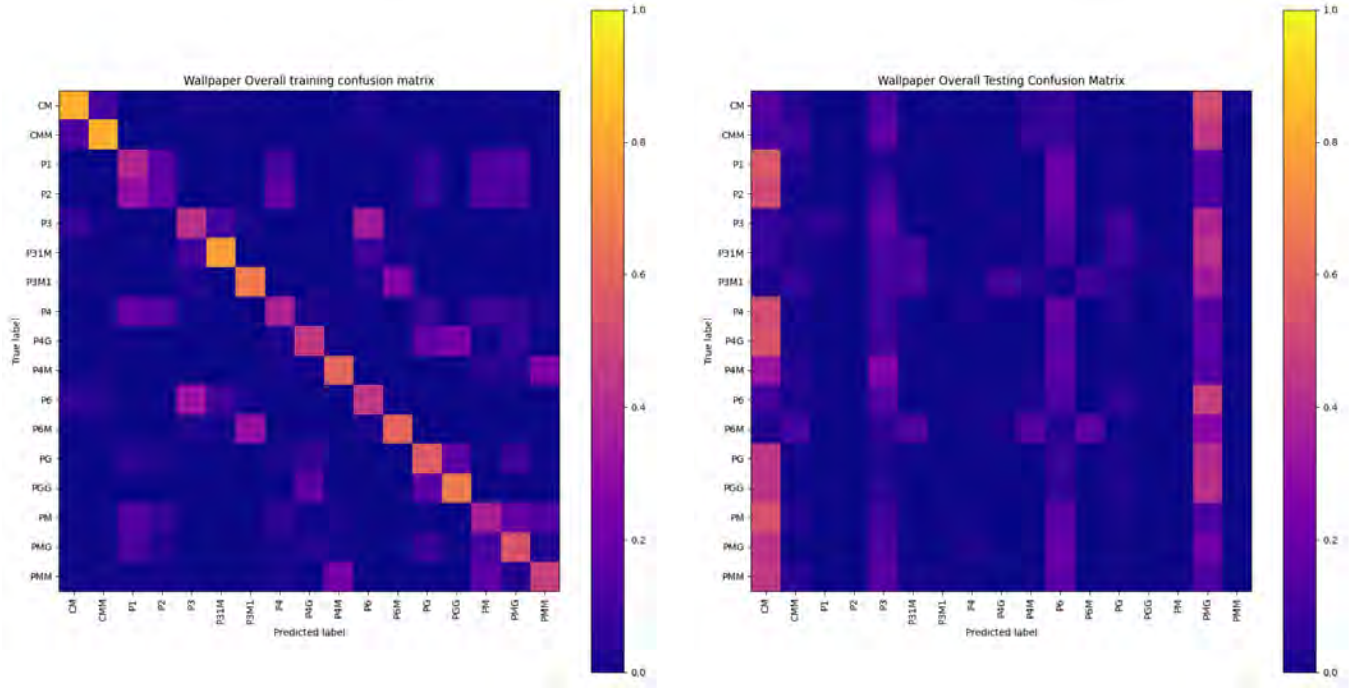


Figure 33: Training Curve using 'Challenge' dataset with Updated Architecture + augmentation (10 epochs)



(a) train (b) test

Figure 34: Overall per class accuracy, using 'Challenge' dataset with Updated Architecture + augmentation (360 epochs)



(a) Overall train confusion matrix (b) Overall test confusion matrix

Figure 35: Confusion matrices using 'Challenge' dataset with Updated Architecture + augmentation (360 epochs)

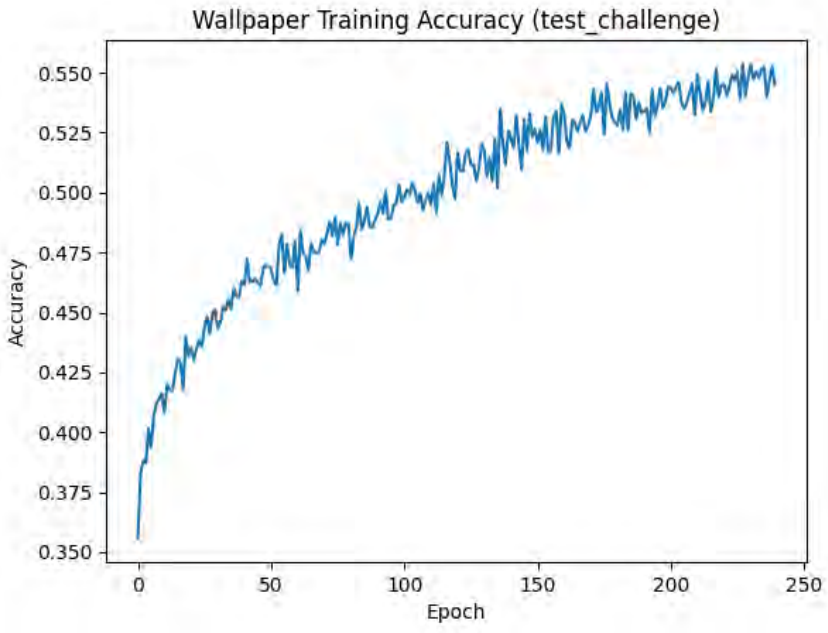
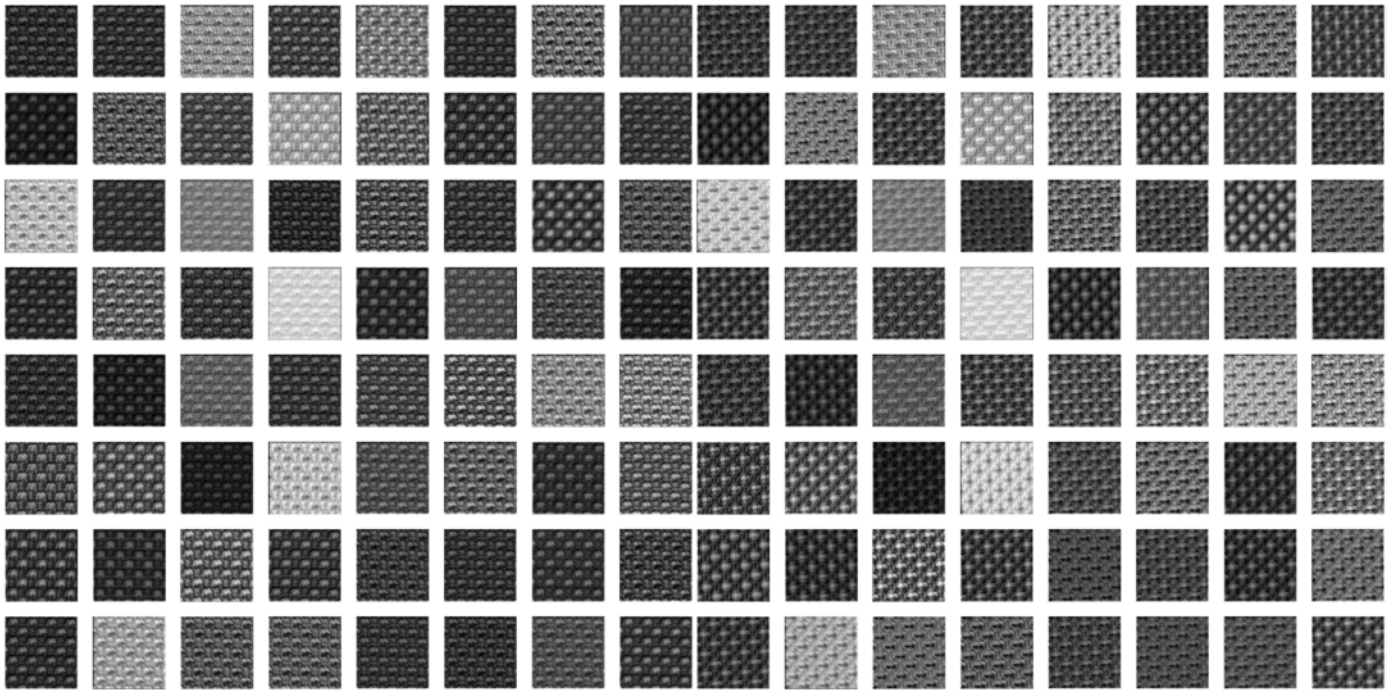


Figure 36: Training Curve using 'Challenge' dataset with Updated Architecture + augmentation (360 epochs) NOTE: Model was resumed from checkpoint, so not all epochs are visible

2.6 Feature Visualization

For feature visualization I used the last Conv2D layer in my model. All images were taken from test challenge dataset, and the first image in each wallpaper type folder was chosen as input. CM is shown in Fig. 37a, CM is shown in Fig. 37a, CMM in Fig. 37b, P1 in Fig. 38a, P2 in Fig. 38b, P3 in Fig. 39a, P3M1 in Fig. 40a, P4 in Fig. 41a, P4G in Fig. 40b, P4M in Fig. 42a, P6 in Fig. 41b, P6M in Fig. 43a, P31M in Fig. 43b, PG in Fig. 44a, PGG in Fig. 44b, PM in Fig. 45a, and PMG in Fig. 45b, PMM in Fig. 46a.



(a) Pattern CM Last Layer Visualization (b) Pattern CMM Last Layer Visualization

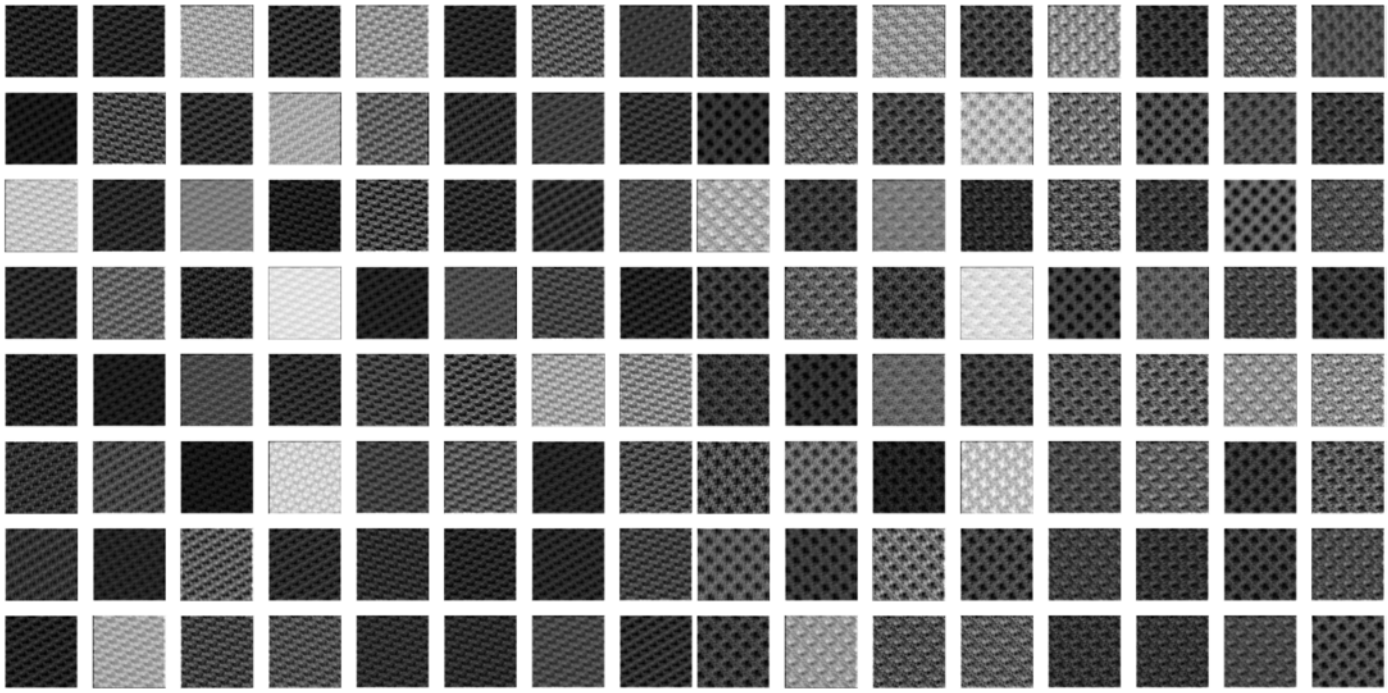
Figure 37

2.7 Analysis of Results and Conclusion

For my updated model, I implemented the model shown in the EscherNet paper, and I also added dropout layers before each fully connected layer to deal with overfitting.

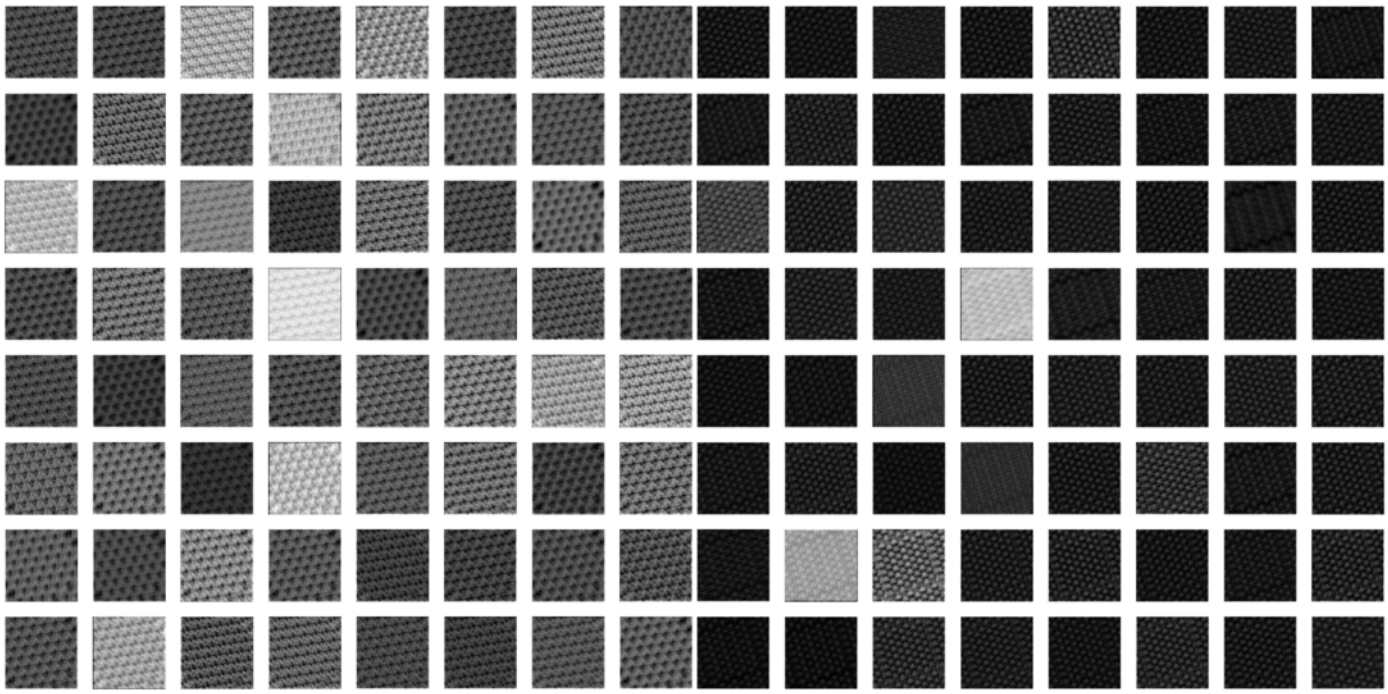
When training the baseline dataset, both baseline and the updated architecture (without augmentation) have good performance, with the updated architecture having slightly more accuracy and slightly less std. These results can be seen in the overall per class accuracy figures and confusion matrices. Both these architectures perform very bad on the challenge dataset, with the updated architecture performing slightly better. Looking at the t-SNE graphs we can also see that classes are clearly separated.

With the Updated architecture + data augmentation, training with only 10 epochs gives very bad results - only slightly better than random - for both the test and random datasets. This is due to the model converging very slowly, which can be observed in the training curve in Fig. 33. From the confusion matrices (Fig. 32) and overall per class accuracy figures (Fig. 31) we can see that the trained model only predicts 6 types of



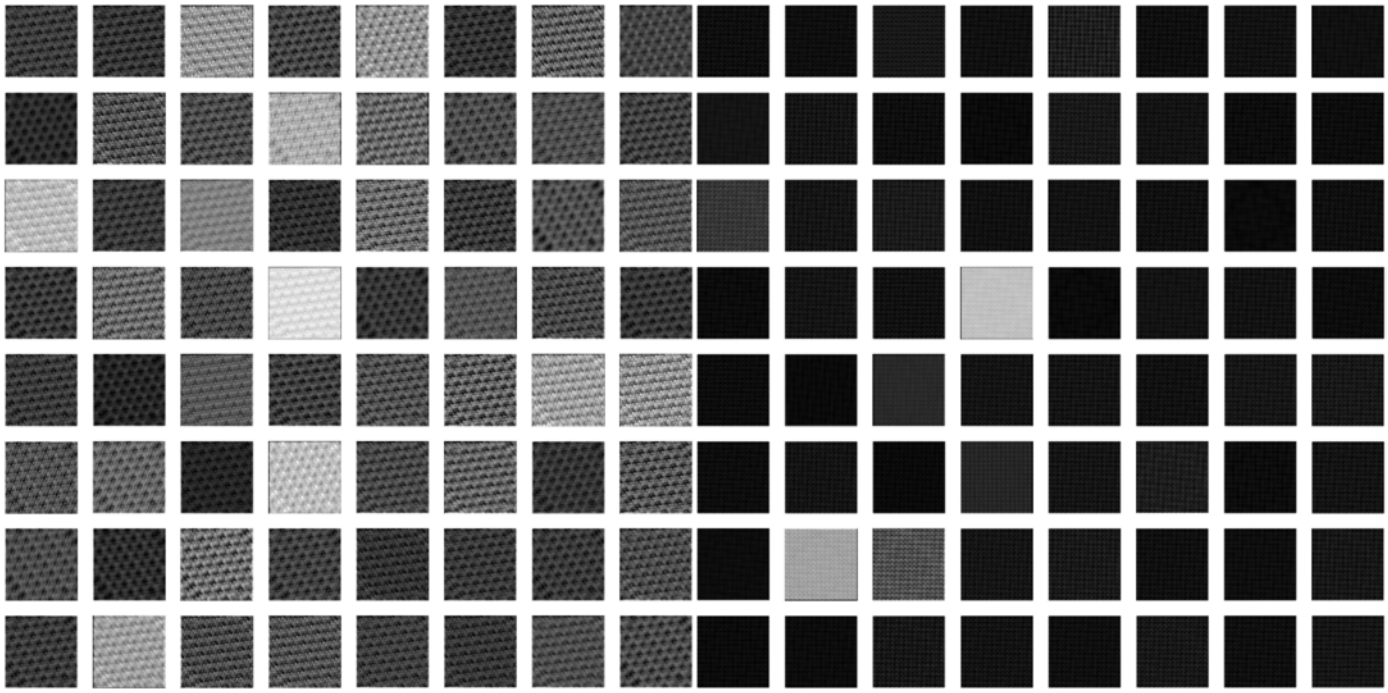
(a) Pattern P1 Last Layer Visualization (b) Pattern P2 Last Layer Visualization

Figure 38



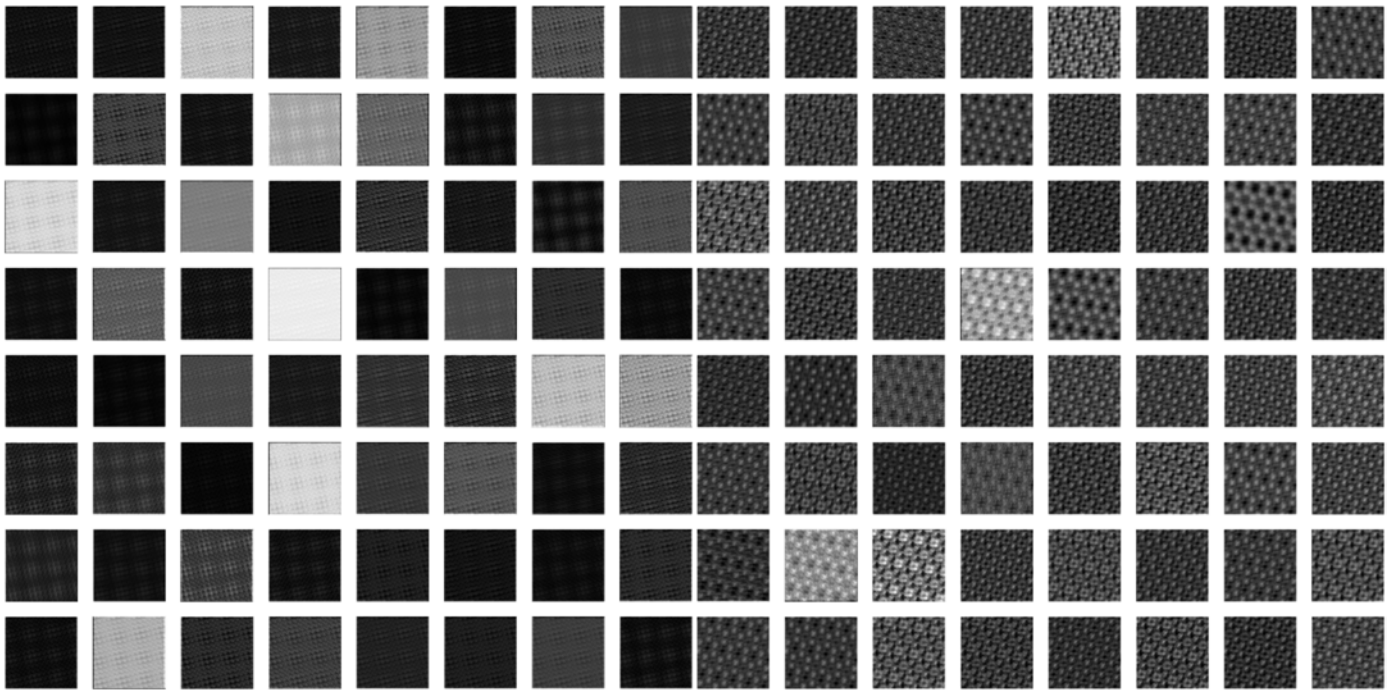
(a) Pattern P3 Last Layer Visualization (b) Pattern P31M Last Layer Visualization

Figure 39



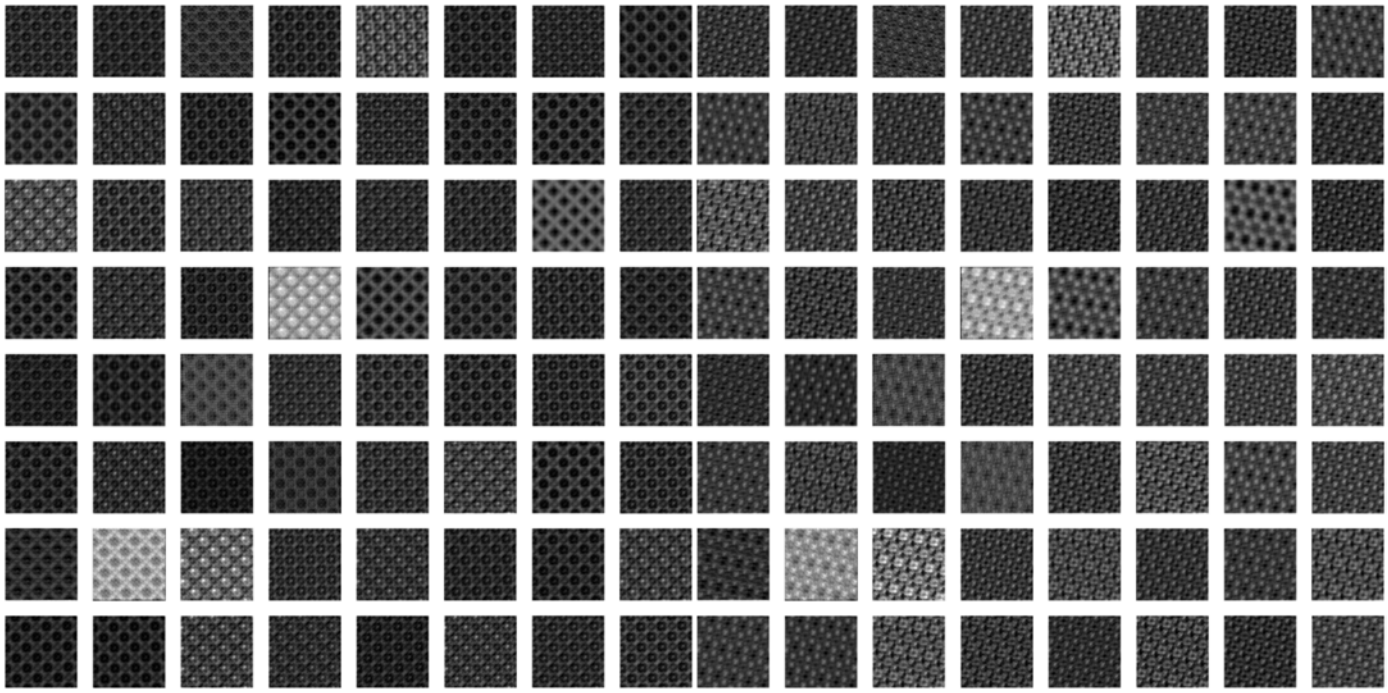
(a) Pattern P3M1 Last Layer Visualization (b) Pattern P4G Last Layer Visualization

Figure 40



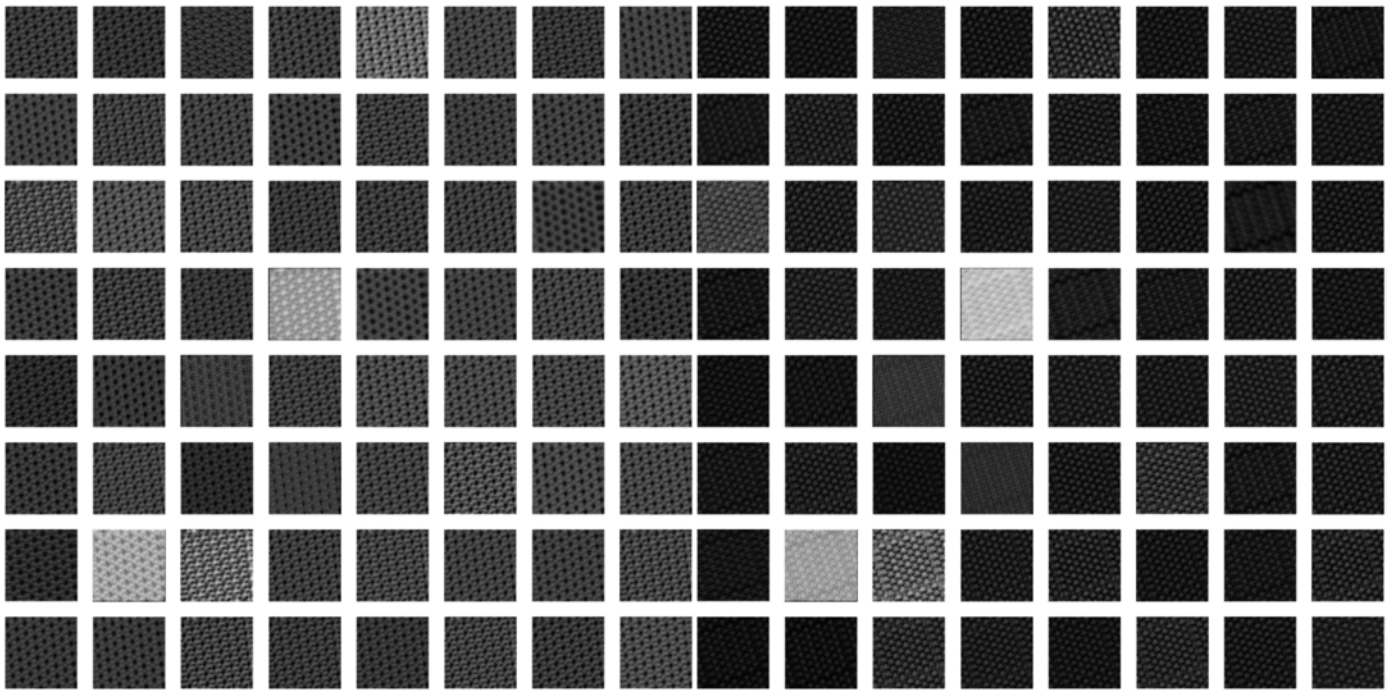
(a) Pattern P4 Last Layer Visualization (b) Pattern P6 Last Layer Visualization

Figure 41



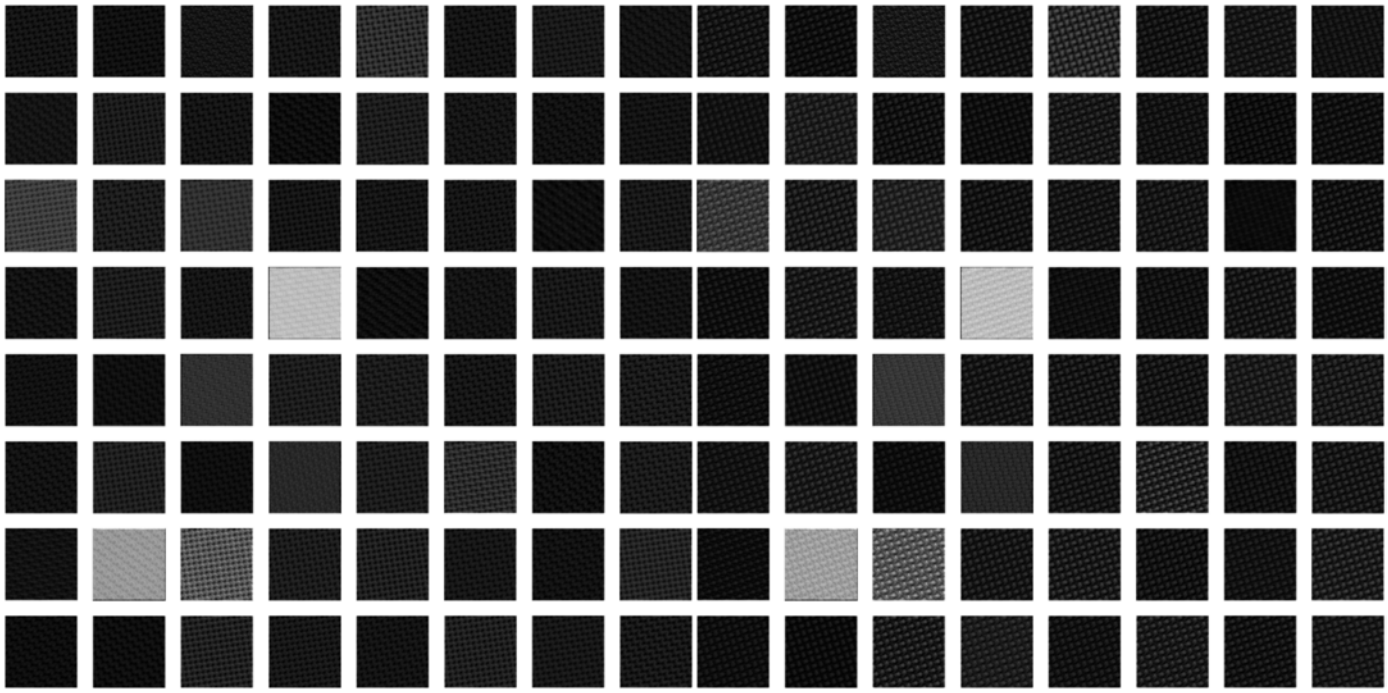
(a) Pattern P4M Last Layer Visualization (b) Pattern CMP6M Last Layer Visualization

Figure 42



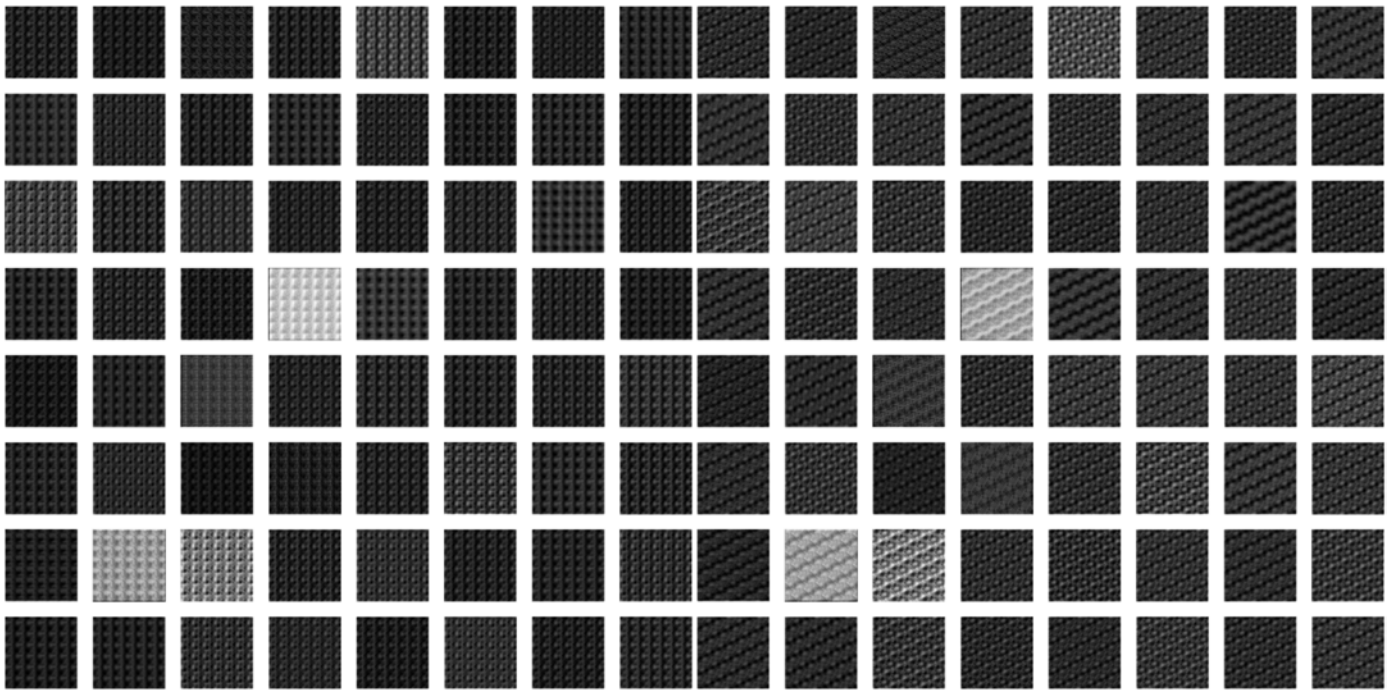
(a) Pattern P6M Last Layer Visualization (b) Pattern P31M Last Layer Visualization

Figure 43



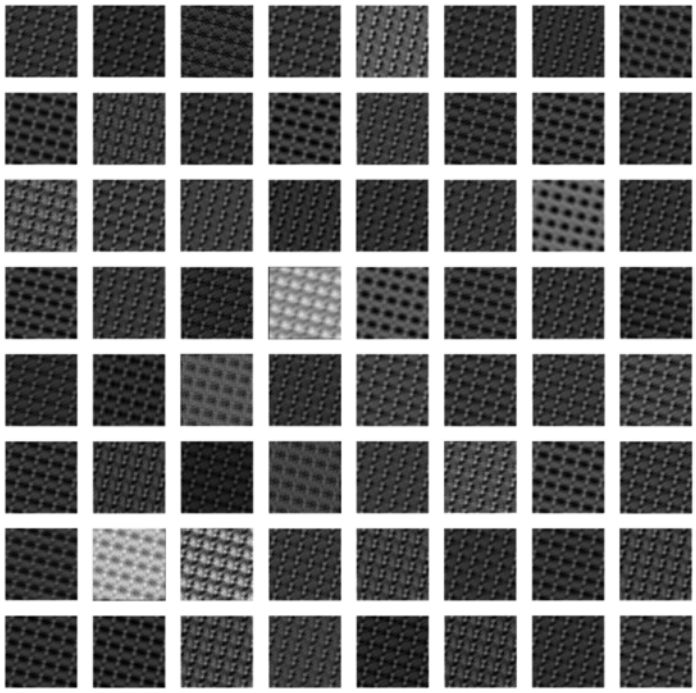
(a) Pattern PG Last Layer Visualization (b) Pattern PGG Last Layer Visualization

Figure 44



(a) Pattern PM Last Layer Visualization (b) Pattern PMG Last Layer Visualization

Figure 45



(a) Pattern PMM Last Layer Visualization

Figure 46

wallpapers in significant amounts (P2, P31M, P4M, PG, PGG, PMM) and all the other types are either rarely or not at all predicted. Looking at the t-SNE graphs we can also see that classes are not seperable.

If the updated model was trained without dropout layers, the model slowly converges and obtains good accuracy in training, but the testing accuracy barely improves. I thought that this was due to overfitting, so I added dropout layers in the model. But this lead to lower train accuracy and still low test accuracy.

I tested with 10 epochs and got an accuracy of 13 percent on the challenge dataset with the Updated Architecture+augment. Looking at the train graph, it seemed that the model did not converge, to I ran the model for 360 epochs. However, I obtained even lower test results. It seems that augmenting the data is making performance worse since the best model is the updated architecture with no augmentation.

Looking at the feature visualizations in the last convolutional layer, it looks like many of the outputs for different models have very similar visualizations - only 3 outputs are white, all others mostly deactivated. It seems that the model cannot distinguish different wallpapers after using augmentation (this can also be observed in the t-SNE graphs). This result was unexpected, since I tried to implement the model in the EscherNet paper as closely as possible, since that model obtained very good results, but it seems that I missed something in my implementation that is leading to these poor results.