Intelligent Field and Robotic Systems Master

Machine Learning

Image Classification Using CNN

Prepared by:

Rihab Laroussi u1100330

Date: December 16, 2024

Table of Contents

Introduction

Machine learning for natural image classification is a significant challenge in modern science and engineering that has garnered much attention. Essentially, machine learning models aim to mimic the remarkable ability of humans and animals to understand and categorize natural images. Hubel and Wiesel, in the 1950s and 1960s, discovered that neurons in the visual cortex of monkeys and cats respond to small regions of the visual field. This finding inspired Convolutional Neural Networks (CNNs), which assign weights to small areas (filters) of an image, rather than to individual pixels as traditional neural networks do. As CNNs train, they adjust these weights using backpropagation to improve classification accuracy.

The term "convolution" is derived from the convolution layer, which performs a dot product on a square grid of pixels (known as a "filter"). This grid moves across the image to create a feature map. By applying different filters during the training process, the network can identify various features, ranging from simple edges to complex shapes.

In this lab, we aim to classify images from a custom multi-class dataset using a CNN as described by Goeddel and Olson in 2016. Our data set consists of approximately 76,500 black and white images, each 100x100 pixels, generated from 2D range scans of different indoor environments. These images are categorized into three classes: corridor, doorway, and room, each representing distinct structural characteristics.

The workflow we will be using for our project is as below:

• Understanding and Pre-processing Data

• Creating the Model

• Training the Model

• Improvising and repeating the process to increase its accuracy

• Testing the Model

Understanding and Pre-processing: Input Data

In this lab, we classify images from a custom multi-class data set using a CNN. The dataset contains 76,500 black-and-white 100x100 pixel images, categorized into three classes: corridor, doorway, and room.

Next, we focused on Data preparation where we do the following processes:

- **Image Transformation**: We apply grayscale conversion, normalization using the mean and standard deviation, and data augmentation for training and testing datasets.
- **Dataset Splitting**: The data is split into 80% for training and 20% for validation.
- **Class Balancing**: To handle class imbalance, we use an *ImbalancedDatasetSampler* that ensures each class has equal sampling probability in each batch.
- **Data Loaders**: We create data loaders for training, validation, and testing using the above transformations and samplers to ensure balanced class representation.

The model is trained with cross-entropy loss, Adam optimizer, and a learning rate scheduler to adjust the learning rate based on validation loss improvements.

Creating the Model

The model consists of two convolutional blocks, each followed by ReLU (Rectified Linear Unit) activations, which is the most commonly used activation function in CNNs. ReLU is linear for positive values and zero for negative values, helping to reduce training time without complex calculations. The model uses average pooling and batch normalization, progressively extracting features from the grayscale input images. The first convolutional layer uses 6 filters of size 9x9, while the second uses 12 filters of the same size. After the convolutional layers, the feature maps are flattened and passed through a fully connected classifier. This classifier includes a dense layer with 100 units, followed by a ReLU activation, batch normalization, and a final softmax layer that outputs class probabilities for the categories: corridor, doorway, and room.

After creating the model, we trained it based on CNN and tested the accuracy and loss percentage.

Training the Model

The goal of this step was to train the LiNet model using different hyperparameter configurations and regularization techniques to determine the best performing setup for our task. I performed the training over multiple epochs, with periodic checkpoints and early stopping to prevent overfitting and save the best model weights. The model training process included logging performance metrics and saving checkpoints for later evaluation.

Train1

For the first training performance of our CNN model, I used the following hyperparameters and settings:

Learning rate: 1e-4

Weight decay: 1e-6

Stepsize: 5

Gamma: 0.1

Batch size: 128

Epochs: 10

Regularization: No dropout, only batch normalization.

Below are the results of the training.

Table1: Training and validation metrics of Train1

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	0.9846	62.72%	0.9271	63.56%
2	0.8432	65.63%	0.8288	63.82%
3	0.7295	72.19%	0.7313	93.80%
4	0.6342	98.29%	0.6502	93.01%
5	0.5525	98.80%	0.5780	93.65%
6	0.5103	99.18%	0.5791	93.10%
7	0.5009	99.32%	0.5745	93.16%

8	0.4934	99.38%	0.5696	92.82%
9	0.4857	99.38%	0.5673	92.21%
10	0.4785	99.41%	0.5606	92.47%

Training Complete in 14m 23s
Best Validation Accuracy: 92.47%

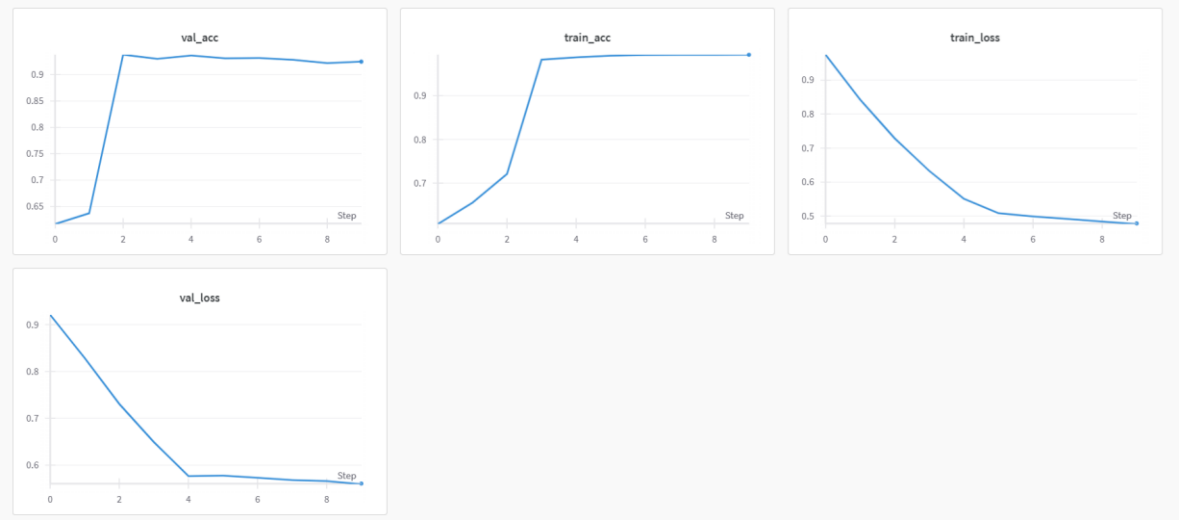

Fig1: Training and Validation metrics output graphs for Train1

We observe that the model performed well overall, achieving its best validation accuracy in the initial few epochs. The relatively high validation accuracy (over 92%) indicates the model's strong capability to generalize. However, the slight decrease towards the end suggests that further refinement could help avoid overfitting. Regarding the losses, both training and validation losses have been decreasing, with a small difference between them, indicating consistent learning and good generalization.

Train2

For the second round of training, I adjusted the hyperparameters to further optimize our model's performance. Specifically, I decreased the learning rate and weight decay. Lowering the learning rate to 5e-5 allows for finer adjustments to the model's weights during training, helping to avoid overshooting the optimal parameters. And reducing the weight decay to 5e-

5 helps in controlling overfitting by applying less regularization, which can be beneficial in learning the distinctive features of our dataset more effectively.

Learning rate: 5e-5

Weight decay: 5e-5

Step size: 5

Gamma: 0.1

Batch size: 128

Epochs: 25

Regularization: No dropout, only batch normalization.

Below are the results of the training.

Table2: Training and validation metrics of Train2

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	0.9110	71.49%	0.8701	85.74%
2	0.8354	92.10%	0.8129	91.39%
3	0.7780	95.86%	0.7692	92.44%
4	0.7265	97.38%	0.7291	92.02%
5	0.6791	98.19%	0.6889	93.06%
6	0.6530	98.64%	0.6845	93.01%
7	0.6480	98.80%	0.6821	92.95%
8	0.6432	98.87%	0.6819	92.35%
9	0.6388	98.86%	0.6749	92.59%
10	0.6341	98.99%	0.6698	92.80%
11	0.6321	98.98%	0.6736	92.51%
12	0.6308	99.02%	0.6697	92.74%
13	**0.6308**	**99.01%**	**0.6704**	**92.71%**
14	0.6297	99.07%	0.6720	92.23%

15	0.6297	99.07%	0.6718	92.42%
16	0.6298	99.03%	0.6730	92.25%
17	0.6296	99.06%	0.6701	92.66%

Early stopping!
Training completed in 24m 40s
Best validation accuracy: 92.74%

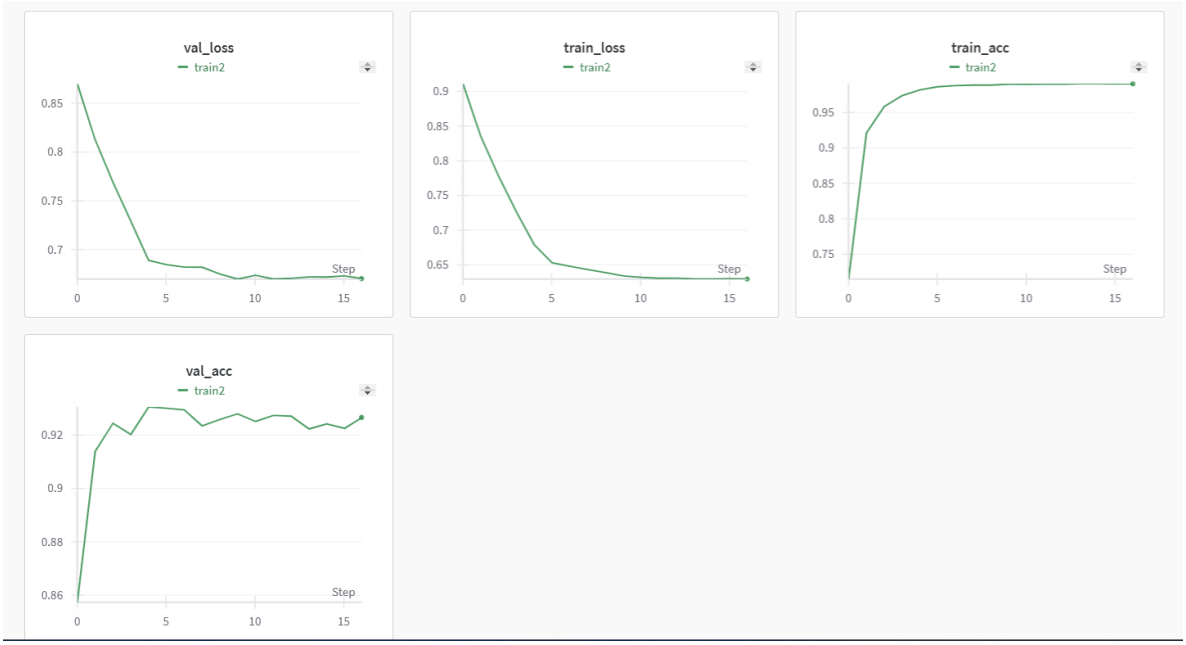

Fig2: Training and Validation metrics output graphs for Train2

From the outputs, it seems like the adjustments to the learning rate and weight decay have benefited the model, with quick learning and high accuracy early in the training, and the validation loss constantly decreases which means stable learning. It seems like the model is learning the task without significant overfitting. Compared to the first training, the validation accuracy shows slight improvements, and this model demonstrates faster convergence, which is crucial. There is still potential for further improvement by employing additional regularization techniques and fine-tuning the mode.

Train3

For the third training, I built upon the pretrained model from the first training session and introduced a dropout rate of 0.3 to reduce overfitting. I applied several data transformations for the training set, such as grayscale conversion, random rotation, color jitter, and normalization to improve generalization and robustness. This time I changed the learning rate scheduler to ReduceLROnPlateau that adjustes the learning rate based on the validation loss.

Learning rate: 1e-5

Weight decay: 1e-5

Step size: 5

Gamma: 0.1

Batch size: 128

Epochs: 25

Regularization: dropout(p=0.3), batch normalization, data transformations

Table3: Training and validation metrics of Train3

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	0.4911	98.23%	0.4784	98.63%
2	0.4785	98.59%	0.4734	98.46%
3	0.4698	98.76%	0.4682	98.41%
4	0.4624	98.85%	0.4606	98.52%
5	0.4551	98.89%	0.4544	98.51%
6	0.4463	99.06%	0.4495	98.39%
7	0.4393	99.09%	0.4410	98.46%
8	0.4327	99.10%	0.4363	98.54%
9	0.4260	99.16%	0.4316	98.39%
10	0.4195	99.18%	0.4259	98.39%
11	0.4130	99.24%	0.4183	98.49%
12	0.4063	99.27%	0.4135	98.43%

13	0.4001	99.30%	0.4097	98.29%
14	0.3937	99.35%	0.4033	98.35%
15	0.3879	99.35%	0.3965	98.51%
16	0.3822	99.35%	0.3935	98.43%
17	0.3767	99.38%	0.3878	98.35%
18	0.3715	99.37%	0.3806	98.43%
19	0.3652	99.41%	0.3775	98.29%
20	0.3599	99.43%	0.3713	98.44%
21	0.3550	99.42%	0.3643	98.56%
22	0.3494	99.47%	0.3632	98.37%
23	0.3437	99.50%	0.3576	98.40%
24	0.3390	99.49%	0.3528	98.37%
25	0.3334	99.55%	0.3499	98.34%

Total Training Time: 28 minutes 52 seconds

Best Validation Accuracy: 98.34% (achieved at epoch 25)

Fig3: Training and Validation metrics output graphs for Train3

We observe that in the first epoch, the validation accuracy is slightly higher than the training accuracy, but as training progresses, the training accuracy increases and becomes consistently higher than the validation accuracy. Throughout the remaining epoch, the validation accuracy remains high but fluctuates slightly, which is normal in training. Towards the end, the validation loss becomes low but still slightly higher than the training loss, which suggests that the model is generalizing well and is not overfitting. Overall, the model is performing well, but further fine-tuning could help to optimize it even more, especially to reduce any small fluctuations in validation accuracy.

Depthwise Seperable Convolution

Next, I replaced the convolutional layers in the model with depthwise separable convolutions. Depthwise separable convolution improves efficiency by breaking the standard convolution process into two parts: depthwise convolution and pointwise convolution.

In depthwise convolution, a separate filter is applied to each input channel. This means each channel is processed independently with its filter. Then, in pointwise convolution, a 1x1 convolution is used to combine the outputs of the depthwise convolution across channels.

The key difference between depthwise separable convolution and traditional convolution is in the number of parameters and computational cost. Traditional convolution uses a single filter that spans all input channels and spatial dimensions, leading to a higher computational cost and more parameters. Depthwise separable convolution, on the other hand, reduces both by splitting the process, which significantly lowers the computational load while maintaining similar performance.

Train1_depth

For the first training session with depthwise separable convolutions, the following hyperparameters were used:

Learning rate: 1e-4

Weight decay: 1e-6

Step size: 5

Gamma: 0.1

Batch size: 128

Epochs: 10

Regularization: Batch normalization, no dropout.

The results are the following:

Table4: Training and validation metrics for Train1_depth

Epoch	Train Loss	Train Accuracy	Val Loss	Val Accuracy
1	0.6766	89.54%	0.6570	90.38%
2	0.6068	95.58%	0.6506	90.39%
3	0.5869	97.31%	0.6503	90.35%
4	0.5775	98.01%	0.6325	92.30%
5	0.5710	98.56%	0.6404	91.67%
6	0.5656	98.89%	0.6343	91.80%
7	0.5638	99.03%	0.6424	90.68%
8	0.5630	99.10%	0.6351	91.60%
9	0.5619	99.18%	0.6356	91.33%

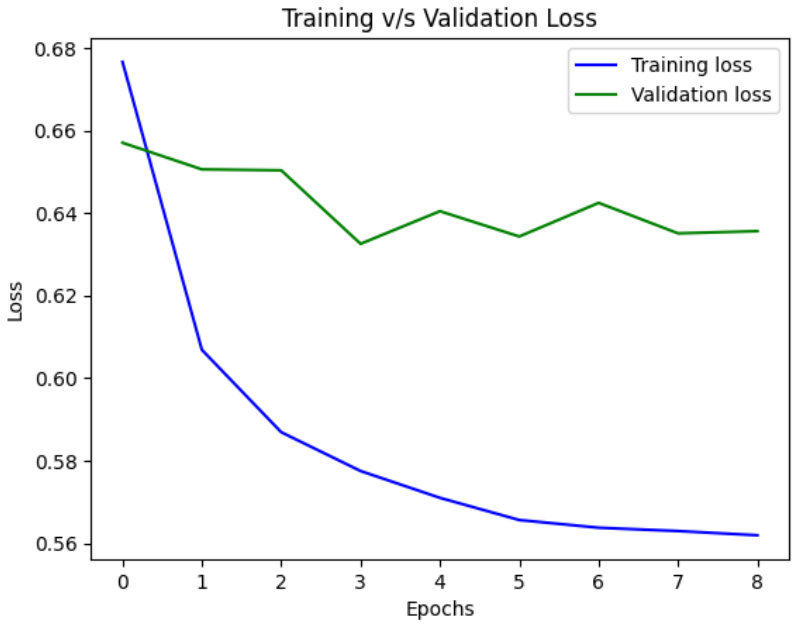

Fig4: Training vs Vlaidation Loss for Train2_Depth

We observe that at the beginning, the validation loss decreases slowly, and the validation accuracy stays around 90.35% to 90.39%. This indicates that the model starts strong but shows only slight improvement in validation metrics. Throughout the rest of the iterations, the validation loss is higher than the training loss and fluctuates. This can mean there is some overfitting, as the model is complex and can easily fit the training data but struggles with the validation set. Early stopping was triggered at epoch 9, indicating no significant improvement beyond the current validation accuracy. We can say that compared to the previous convolutional models using the same hyperparameters, this model performs slightly better as it appears to be learning effectively from the start.

Train2_depth

For the second training session, I introduced some improvements with a dropout rate of 0.5 and data augmentation techniques (random rotation and color jitter) to help prevent overfitting and improve generalization.

Learning rate: 1e-5

Weight decay: 1e-5

Step size: 5

Gamma: 0.1

Batch size: 128

Epochs: 10

Regularization: Dropout (p=0.5), batch normalization, data transformations (random rotation and color jitter).

Table5: Training and validation metrics for Train2_depth

Epoch	Train Loss	Train Accuracy	Val Loss	Val Accuracy
1	0.6107	94.81%	0.6569	89.72%
2	0.6026	95.61%	0.6566	89.57%
3	0.5998	95.91%	0.6474	90.83%
4	0.5968	96.21%	0.6436	91.01%
5	0.5943	96.40%	0.6376	91.87%

6	0.5933	96.56%	0.6403	91.71%
7	0.5907	96.79%	0.6425	91.33%
8	0.5902	96.81%	0.6349	91.93%
9	0.5886	96.99%	0.6424	91.09%
10	0.5868	97.16%	0.6325	92.24%

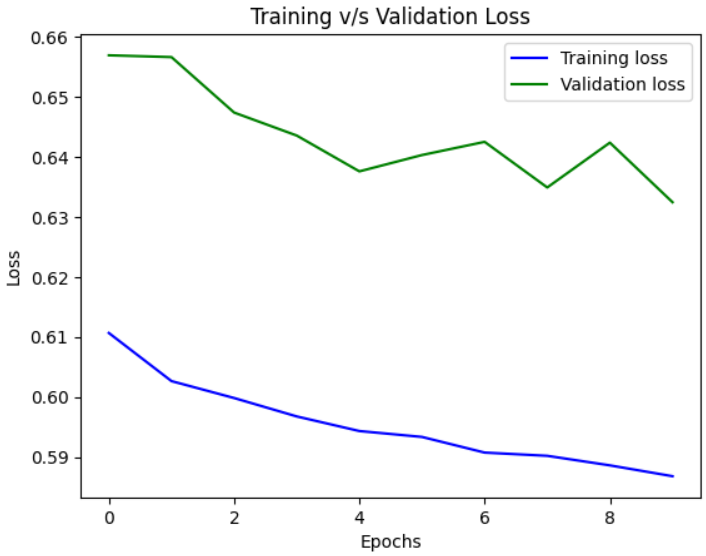

Fig5: Training vs Vlaidation Loss for Train2_Depth

There is some improvements comapared to the previosu training, both traning and validation accuracies have increased steadily. The validation loss is still higher than the traning loss. The model didn't experienvce a significanft imporvoment, it still needs further fine tuning, yet the added dropout (0.5) and data augmentation contributed to better generalization, improving performance on the validation set compared to the first training.

Train3_depth

In the third training session, I trained the model for 25 epochs with the same depthwise separable convolutional structure, but with an adjusted learning rate and weight decay. The goal was to fine-tune the model for optimal performance.

Learning rate: 5e-5

Weight decay: 5e-5

Step size: 5

Gamma: 0.1

Batch size: 128

Epochs: 25

Regularization: Dropout (p=0.5), batch normalization, data transformations.

Table6: Training and validation metrics for Train3_depth

Epoch	Train Loss	Train Accuracy	Val Loss	Val Accuracy
1	0.6049	95.33%	0.6437	90.99%
2	0.5954	96.31%	0.6367	91.90%
3	0.5893	96.90%	0.6347	91.85%
4	0.5846	97.33%	0.6239	92.99%
5	0.5816	97.58%	0.6306	92.41%
6	0.5787	97.88%	0.6219	93.21%
7	0.5769	98.01%	0.6276	92.70%
8	0.5750	98.13%	0.6213	93.15%
9	0.5726	98.37%	0.6165	93.48%
10	0.5715	98.43%	0.6309	92.13%
11	0.5694	98.64%	0.6225	92.88%
12	0.5686	98.68%	0.6188	93.45%
13	0.5662	98.90%	0.6181	93.57%
14	0.5655	98.94%	0.6159	93.66%
15	0.5653	98.96%	0.6193	93.37%
16	0.5648	98.99%	0.6202	93.05%
17	0.5644	99.06%	0.6182	93.48%
18	0.5643	99.06%	0.6193	93.31%

19	0.5642	99.08%	0.6245	92.56%

Best Validation Accuracy: 93.66% (Epoch 14)
Training Time: 23 minutes 6 seconds
Early Stopping Triggered: After Epoch 19

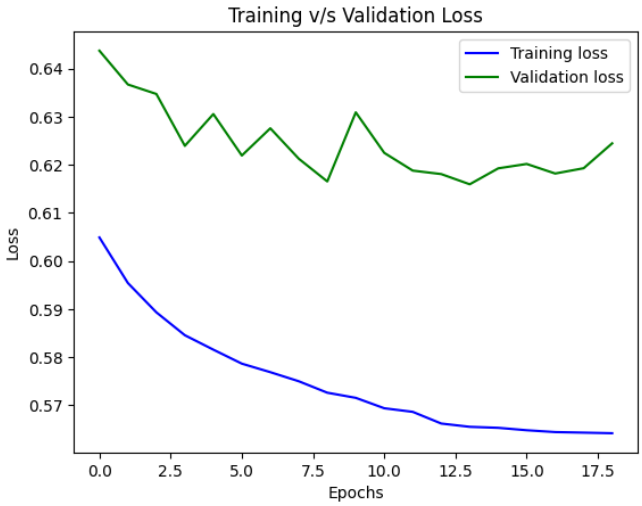

Fig6: Training vs Vlaidation Loss for Train3_Depth

We observe consistent improvements in both training and validation accuracy across the epochs. The training loss continually decreases, while the accuracy increases, peaking at 99.08%. The highest validation accuracy achieved was 93.66%, with noticeable stability in both loss and accuracy after the initial epochs. The validation loss fluctuates slightly but generally shows a decreasing trend, although it remains slightly higher than the training loss. Early stopping was triggered after epoch 19, effectively preventing overfitting by stopping further training once validation performance began to drift. Overall, we can conclude that this update has further optimized the model.

Evaluate

After all these training sessions, I evaluated the model that I believe is the best and assessed its performance on the test set. To determine if the model is overfitting or underfitting, I compared its test performance with the training and validation results. For this evaluation, I used the following metrics:

- Precision: Measures the accuracy of positive predictions, indicating how many of the predicted positives are actually correct.

- Recall: Assesses the model's ability to identify all relevant instances, showing how many actual positives were correctly predicted.

- F1 Score: Balances precision and recall, providing a single metric to gauge overall performance.

I chose the model **Train3** for evaluation because it has several improvements that enhance performance and generalization. This model builds on the pretrained model from the first session, using its previous knowledge to learn better. I added a dropout rate of 0.3 to avoid overfitting and applied various data transformations to make the model more robust to unseen data. I also added the ReduceLROnPlateau learning rate scheduler to adjust the learning rate based on validation loss, helping the model fine-tune its parameters. I also set the learning rate and weight decay to 1e-5 to prevent overfitting while allowing the model to adapt effectively. Throughout the epochs, **Train3** showed stable and consistent performance with high training and validation accuracies, indicating its potential for strong test set performance.

After testing the model, I got the following results:

```
Accuracy of the network on 2362 test images: 68.5013%
        Accuracy of corridor : 0.0000%
        Accuracy of door : 0.5952%
        Accuracy of room : 100.0000%

Kappa Score: 0.0022

              precision    recall  f1-score   support

    corridor       0.00      0.00      0.00       577
        door       1.00      0.01      0.01       168
        room       0.68      1.00      0.81      1617

    accuracy                           0.69      2362
   macro avg       0.56      0.34      0.27      2362
weighted avg       0.54      0.69      0.56      2362
```

Fig6: Metrics results for the test evaluation

The overall accuracy of the model is **68.5%**, indicating somehow moderate success in classifying the test images. However, this accuracy is not uniformly distributed across the different classes. The Kappa score of **0.0022** suggests that the model's performance is only marginally better than random guessing. From the remaining metrics, we conclude that the model performs well on the room class but struggles significantly with the corridor and door classes. This is expected, given that the room class has a dominant presence in the dataset with 46,839 images. To improve the model's performance, adjustments to class balancing and the inclusion of more training data for the underperforming classes would be beneficial.

To further evaluate the performance of the model, we observe the confusion matrix which provides a detailed breakdown of the model's predictions compared to the actual labels.

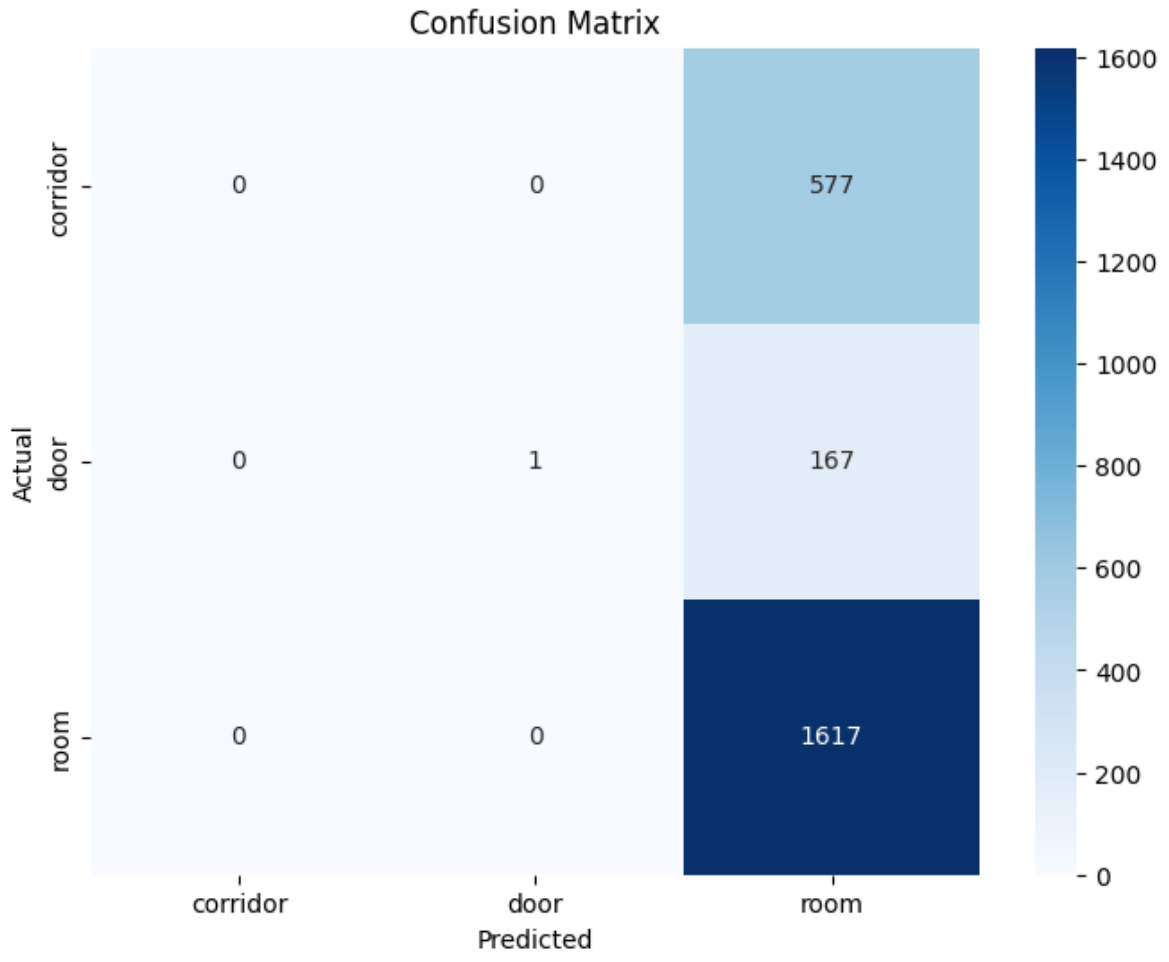

Fig7: The model's confusion matrix

True Positives (TP): This represents the correctly classified instances of each class. For the room class, the model correctly classified all 1617 images. For the door class, it classified only one image. And none for the corridor class.

False Positives (FP): This is where the model incorrectly predicts a class when the actual class is different. For the corridor and door classes, the model incorrectly classified instances as room. There were 577 false positives for the corridor class, and 167 false positives for the door class.

The confusion matrix shows that the model is highly biased toward predicting "room" for most images. While this leads to high accuracy for the room class, it results in a high number of false positives and false negatives for the corridor and door classes. This suggests that the model needs adjustments to overcome the overfitting, and better handle the underperforming classes.

Conclusion

In this lab, the main objective was to enhance a CNN model's performance and generalization. Through the process, I learned how to effectively use pretrained models, implement dropout to prevent overfitting, apply various data transformations, and fine-tune hyperparameters. I also understood the importance of evaluating the model using metrics and confusion matrix to get a comprehensive view of its performance. Overall, the lab provided valuable insights into creating and training convolutional models.