Homework Assignment #1

Adrian Dinu Urse

Task 1

Forward pass implementation MyConvStub:

- Calculates the dimensions of the output tensor based on the input dimensions, kernel size, stride, and dilation
- Initializes an output tensor filled with zeros
- Iterates over the input batches, groups, and output channels to compute the convolution:
    - For each position in the output, it computes the corresponding region of the input tensor by applying the stride and dilation.
    - It multiplies this region by the corresponding weight tensor and sums the results.
    - If bias is enabled, it adds the bias value to the output.

Forward pass implementation MyFilterStub:

Similar to the convolution implementation, this method computes the output by applying the filter to each channel of the input tensor. Iterates over the input batches and channels and computes the output by sliding the filter across the input tensor.

Results:

Managed to pass all the unit tests in 4372 seconds,, ran on CPU.

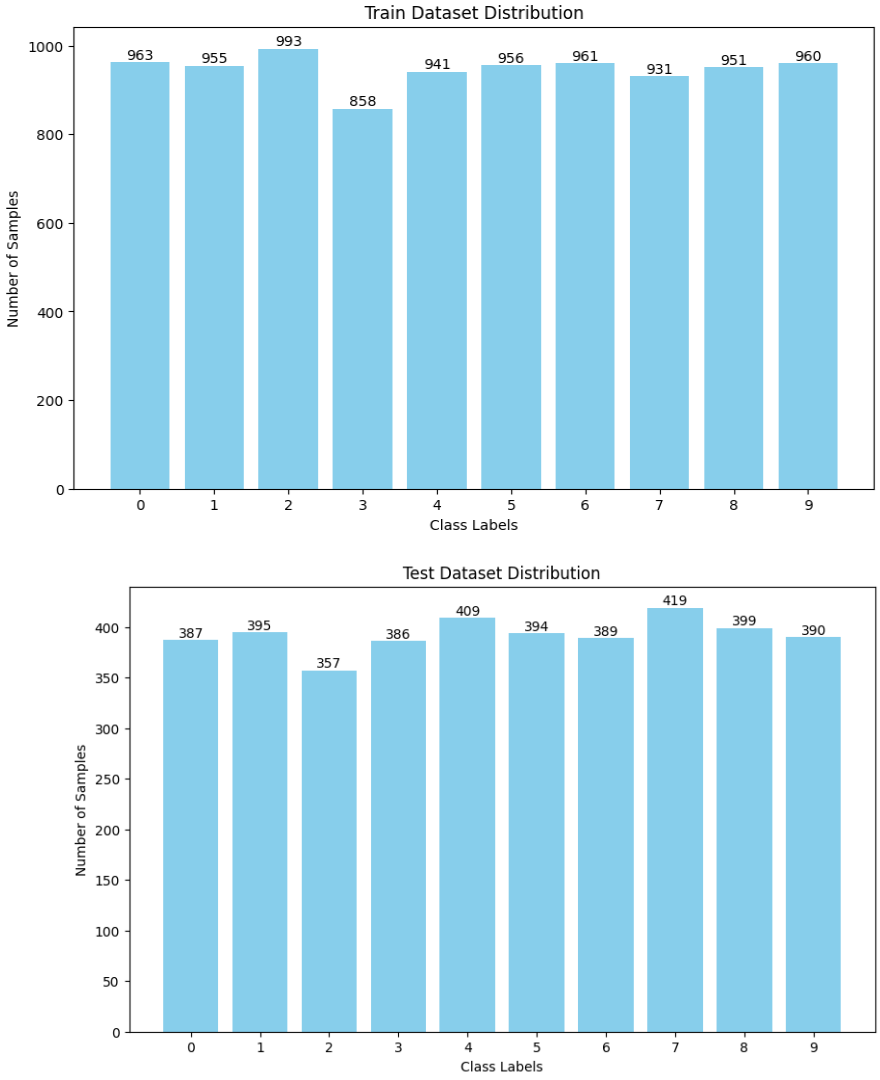
```
C:\Users\bebed\OneDrive\Desktop\DNN\da>python -m unittest test_conv.py
Running test_biased_conv
Test passed for input size: 1 output size: 3
Test passed for input size: 2 output size: 9
Test passed for input size: 4 output size: 27
Test passed for input size: 8 output size: 81
Test passed for input size: 16 output size: 243
.Running test_decreasing_filter_sizes
Test passed for input size: 3 output size: 1
Test passed for input size: 9 output size: 2
Test passed for input size: 27 output size: 4
Test passed for input size: 81 output size: 8
Test passed for input size: 243 output size: 16
.Running test_dilation
Test passed for dilation: 1
Test passed for dilation: 2
Test passed for dilation: 3
.Running test_grouping
Test passed for group size: 4
Test passed for group size: 8
Test passed for group size: 16
Test passed for group size: 32
Test passed for group size: 64
.Running test_horizontal_kernel
Test passed for kernel size: (1, 3)
Test passed for kernel size: (2, 4)
Test passed for kernel size: (3, 5)
Test passed for kernel size: (4, 6)
Test passed for kernel size: (5, 7)
Test passed for kernel size: (6, 8)
Test passed for kernel size: (7, 9)
.Running test_increasing_filter_sizes
Test passed for input size: 1 output size: 3
Test passed for input size: 2 output size: 9
Test passed for input size: 4 output size: 27
Test passed for input size: 8 output size: 81
Test passed for input size: 16 output size: 243
.Running test_same_number_of_filters
Test passed for input size: 1 output size: 1
Test passed for input size: 2 output size: 2
Test passed for input size: 4 output size: 4
Test passed for input size: 8 output size: 8
Test passed for input size: 16 output size: 16
Test passed for input size: 32 output size: 32
Test passed for input size: 64 output size: 64
Test passed for input size: 128 output size: 128
```

```
.Running test_square_kernels
Test passed for kernel size: (1, 1)
Test passed for kernel size: (2, 2)
Test passed for kernel size: (3, 3)
Test passed for kernel size: (4, 4)
Test passed for kernel size: (5, 5)
Test passed for kernel size: (6, 6)
Test passed for kernel size: (7, 7)
.Running test_stride
Test passed for stride: 1
Test passed for stride: 2
Test passed for stride: 3
.Running test_vertical_kernel
Test passed for kernel size: (3, 1)
Test passed for kernel size: (4, 2)
Test passed for kernel size: (5, 3)
Test passed for kernel size: (6, 4)
Test passed for kernel size: (7, 5)
Test passed for kernel size: (8, 6)
Test passed for kernel size: (9, 7)
.Running test_zero_biased_conv
Test passed for input size: 1 output size: 3
Test passed for input size: 2 output size: 9
Test passed for input size: 4 output size: 27
Test passed for input size: 8 output size: 81
Test passed for input size: 16 output size: 243
.Running test_filter
Test passed for blur size: 1
Test passed for blur size: 2
Test passed for blur size: 3
Test passed for blur size: 4
Test passed for blur size: 5
Test passed for blur size: 6
.
----------------------------------------------------------------------
Ran 12 tests in 4372.746s

OK
```

Task 2

Imagenette dataset - 9469 train 3925 test
Label distribution:



The number of samples for each class is balanced in both train and test datasets, helping to prevent bias in the training process.

Base model architecture:

```
SimpleCNN(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (fc1): Linear(in_features=256, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=10, bias=True)
  (relu): ReLU()
  (global_pool): AdaptiveAvgPool2d(output_size=1)
)
```

1. **Convolutional Layers**
   - **Three Convolutional Layers**: The design choice to use three convolutional layers allows for a hierarchical feature extraction, where each layer captures increasingly complex patterns.
   - **Filter Depths (64, 128, 256)**: The increasing filter depth in each successive layer is a typical design in CNNs, as it lets the network capture richer information.

2. Batch Normalization Layers
   - **Optional Batch Normalization after Each Convolution**: Batch normalization stabilizes the learning process and allows for faster convergence.

3. Activation Functions (ReLU)
   - **ReLU (Rectified Linear Unit)**: ReLU is used as the non-linear activation function after each convolutional and fully connected layer. This choice is due to its simplicity and effectiveness, as ReLU helps avoid the vanishing gradient problem common with other activations (e.g., sigmoid or tanh). It allows the network to learn faster and results in sparse activations, which can make the network more efficient in practice.

4. Global Pooling Layer
   - **Adaptive Average Pooling**: The AdaptiveAvgPool2d(1) layer pools each feature map down to a single value, providing a fixed-size input to the fully connected layer, regardless of the input image dimensions. This global pooling approach reduces the risk of overfitting, as it minimizes the number of parameters.

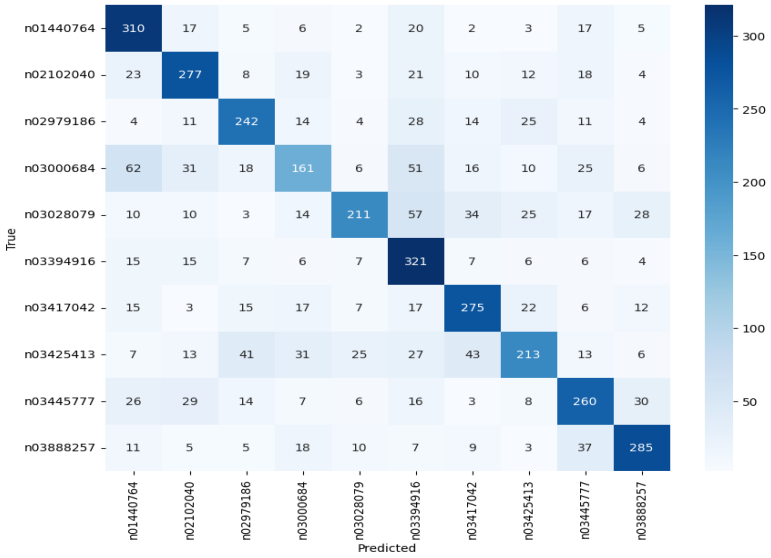
5. Dropout Layer
   - **Optional Dropout after First FC Layer**: Dropout is included to help reduce overfitting by randomly deactivating neurons during training, encouraging the model to learn more robust representations. The dropout rate of 0.5 is commonly used and balances between reducing overfitting and retaining enough information for accurate predictions.

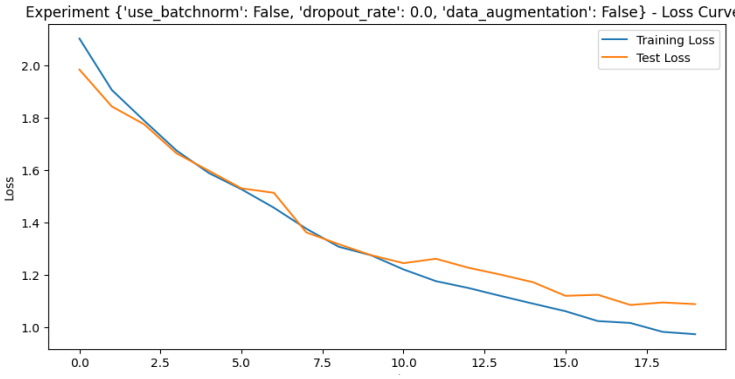
Training setup

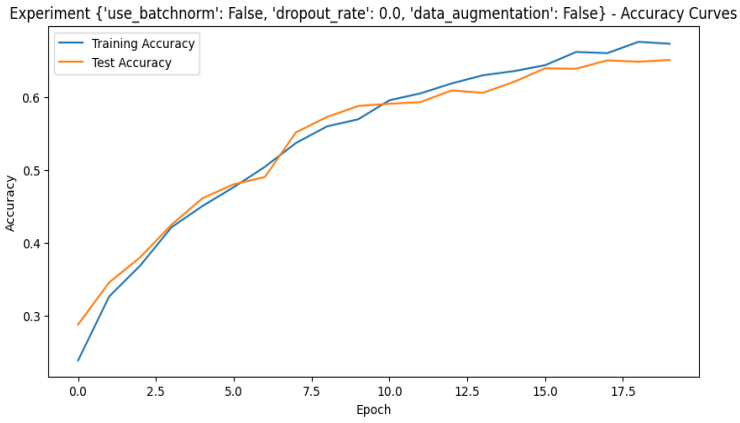
I trained each model for 20 epochs using a learning rate of 0.001 and a weight decay of 1e-4. This setup aims to ensure stable convergence while preventing overfitting through regularization. The chosen learning rate balances the speed of training and accuracy, allowing the models to effectively learn from the data. The weight decay further aids in enhancing generalization by discouraging overly complex models.

Base model

Train Loss: 0.9743, Train Acc: 0.6732, Test Loss: 1.0890, Test Acc: 0.6510



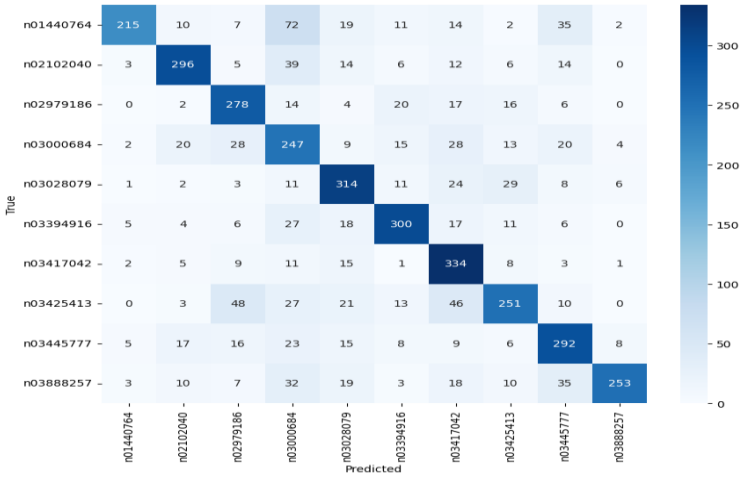


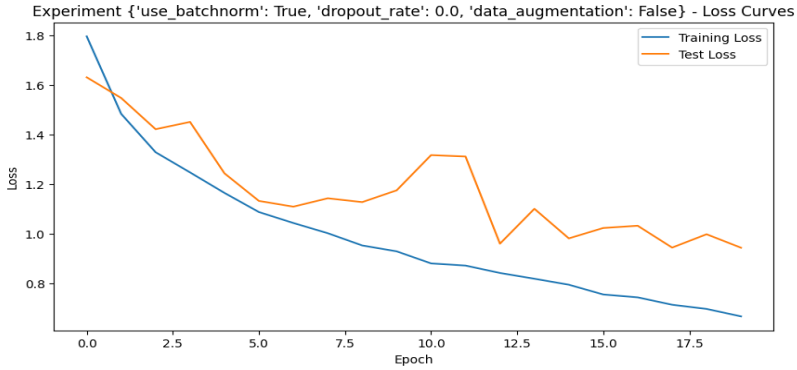


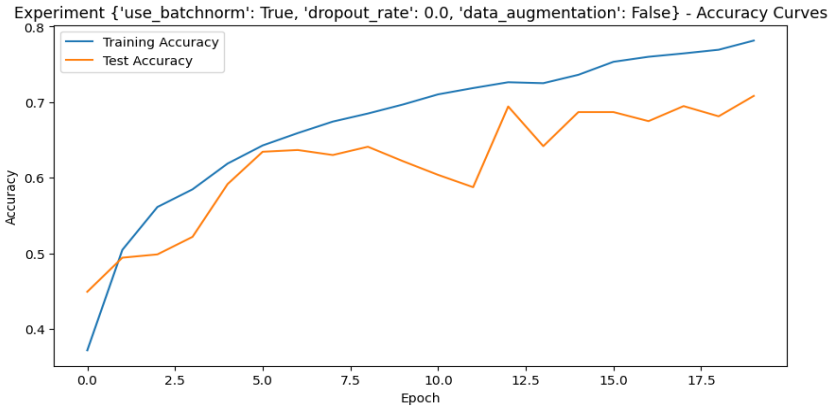
The confusion matrix for the base model reveals significant misclassification errors, particularly in the fourth, fifth, and eighth classes. The model frequently confuses these classes with others, indicating potential weaknesses in distinguishing certain features associated with these categories.

Base model with batch normalization

Train Loss: 0.6674, Train Acc: 0.7815, Test Loss: 0.9440, Test Acc: 0.7083

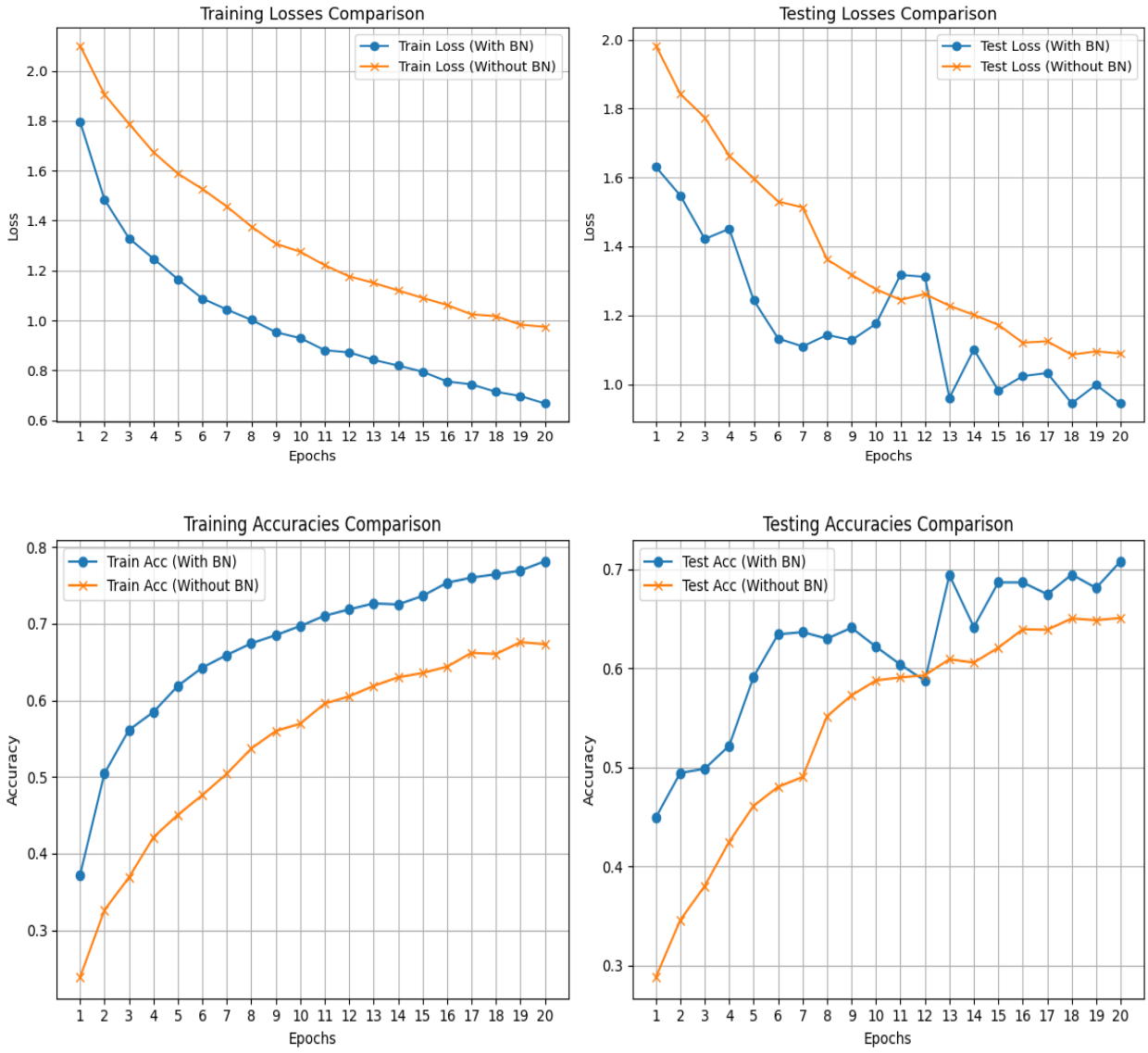






The model with normalization shows improved performance on the fourth, fifth, and eighth classes, but it performs worse on the first class compared to the base model.

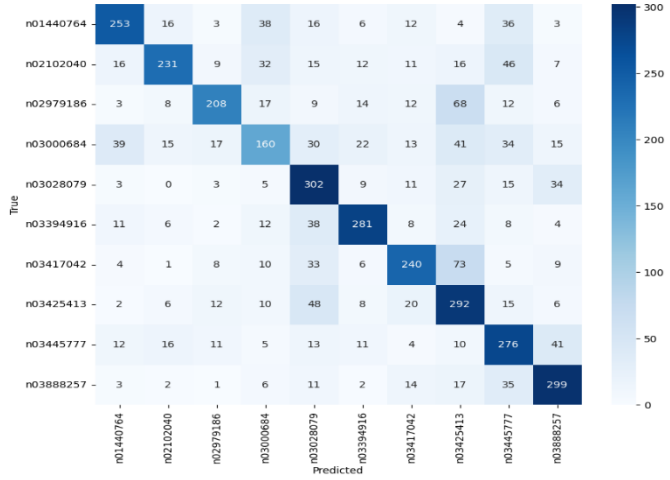
Comparison with base model

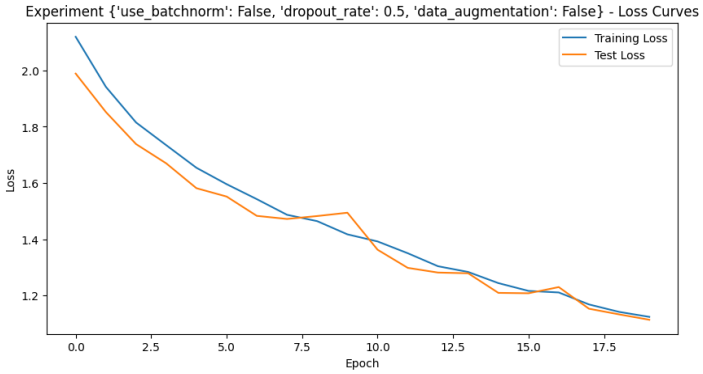


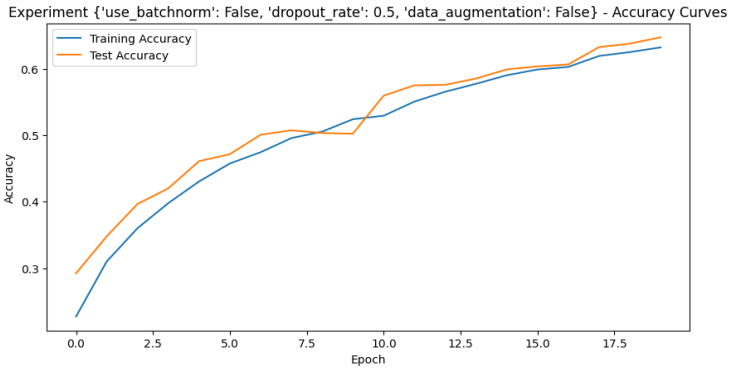
The comparison between the base model and the base model with batch normalization (BN) indicates that the model with BN showed some instability during training, particularly in the test loss and test accuracy. This is due to the way batch normalization adjusts the internal representations dynamically, sometimes causing fluctuations. However, by the end of training, the model with batch normalization achieved lower test loss and higher test accuracy compared to the model without BN, suggesting that it helped the model generalize better and perform more effectively on unseen data.

Base model with dropout

Train Loss: 1.1238, Train Acc: 0.6324, Test Loss: 1.1135, Test Acc: 0.6476

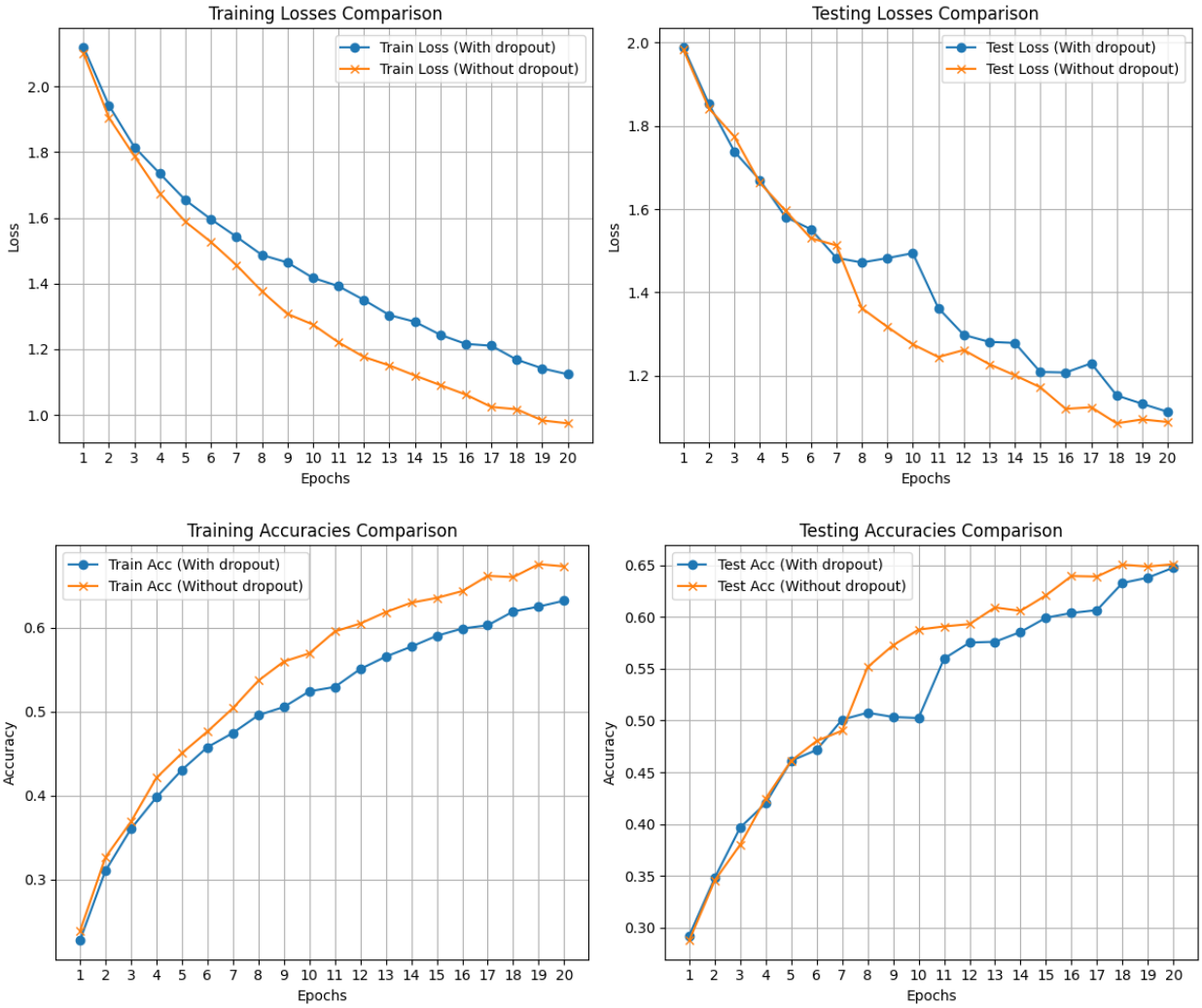






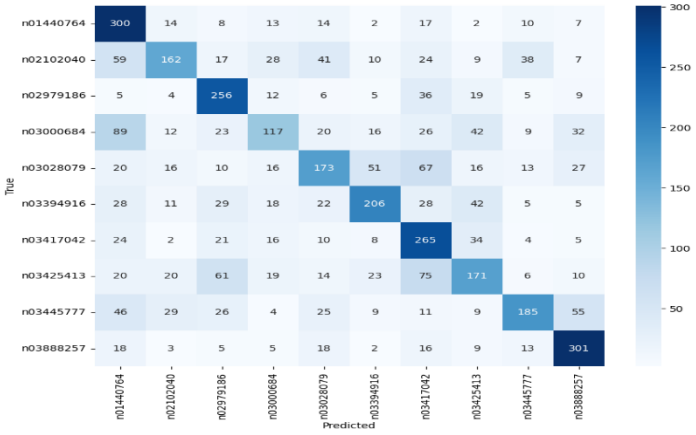
The confusion matrix for the model with dropout reveals significant misclassification errors on the the fourth class.

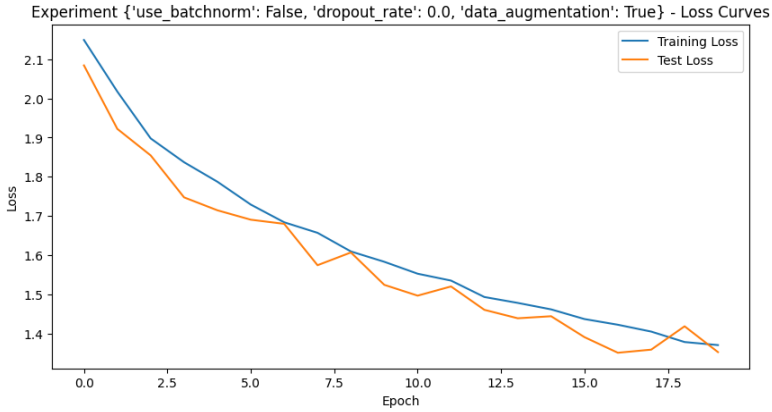
Comparison with base model

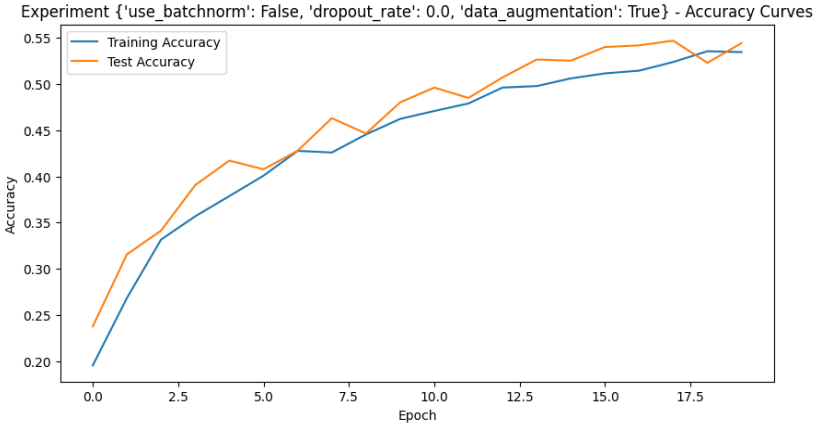


The comparison between the base model and the base model with dropout reveals that the model with dropout exhibited slight instability during training, particularly in the test loss and test accuracy. By the end of training, the model with dropout achieved similar results to the base model on test loss and test accuracy, indicating comparable generalization performance. However, on the training loss and accuracy, the model with dropout showed lower results than the base model, which is expected as dropout acts as a regularization technique that reduces overfitting by making the model less reliant on specific neurons during training.

Base model with data augmentation

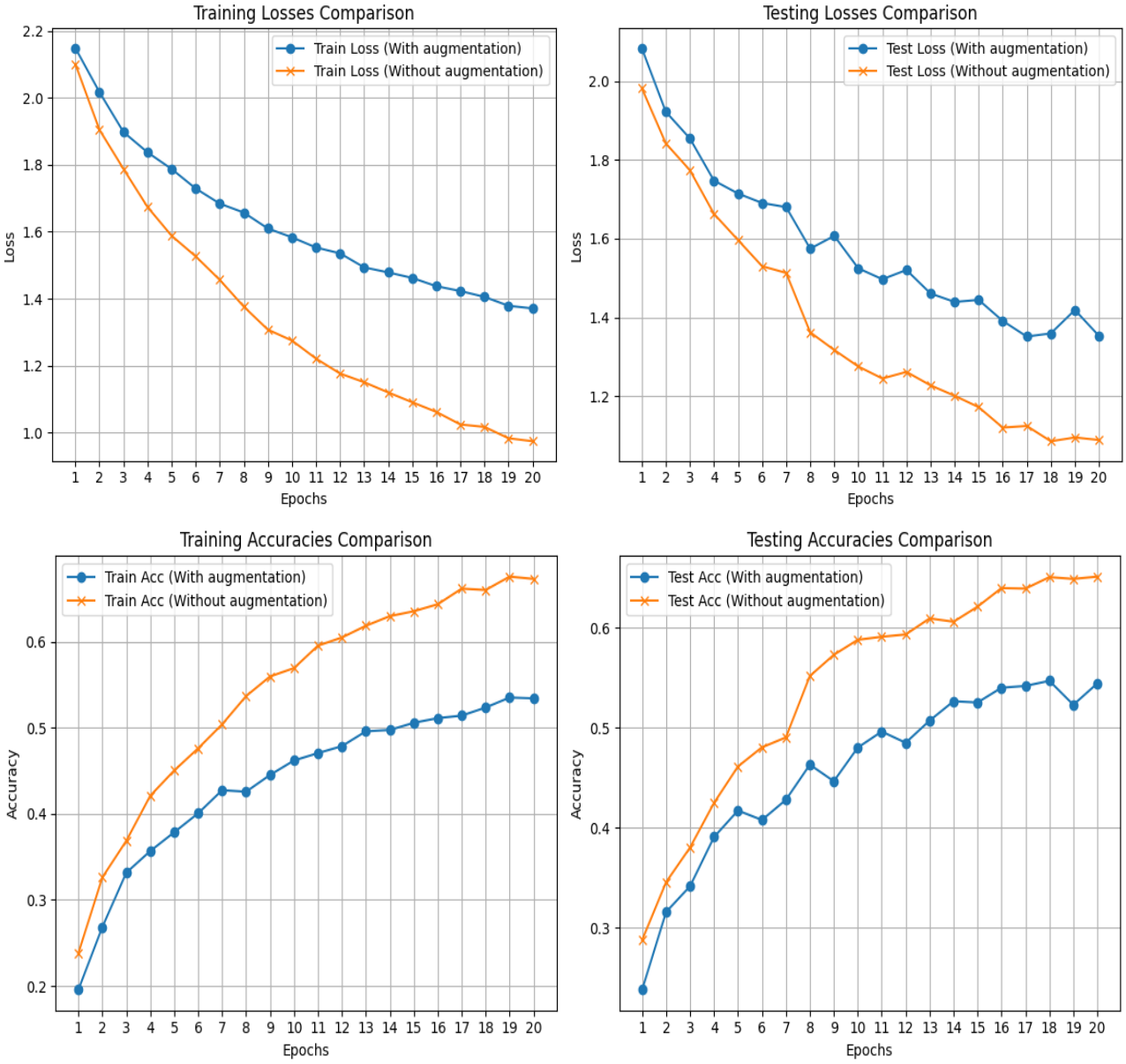






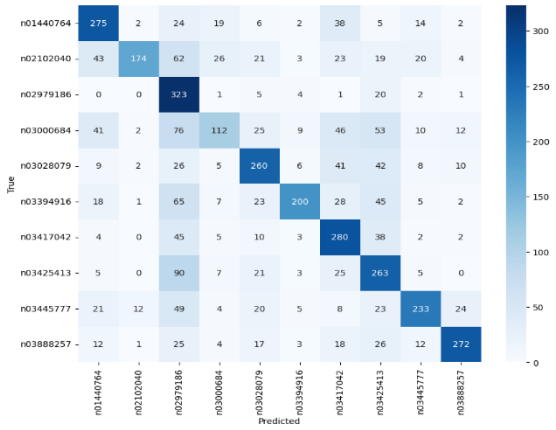
The confusion matrix for the model with data augmentation reveals that significantly stuggles to classify the second class, the fourth class, the fifth class and the eighth class.

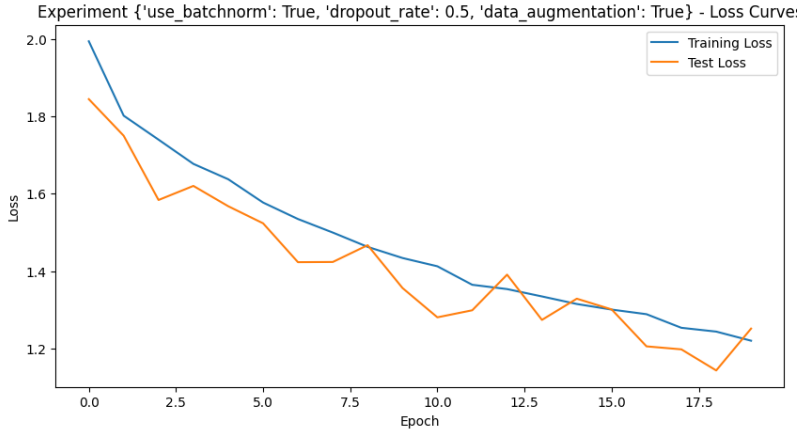
Comparison with base model

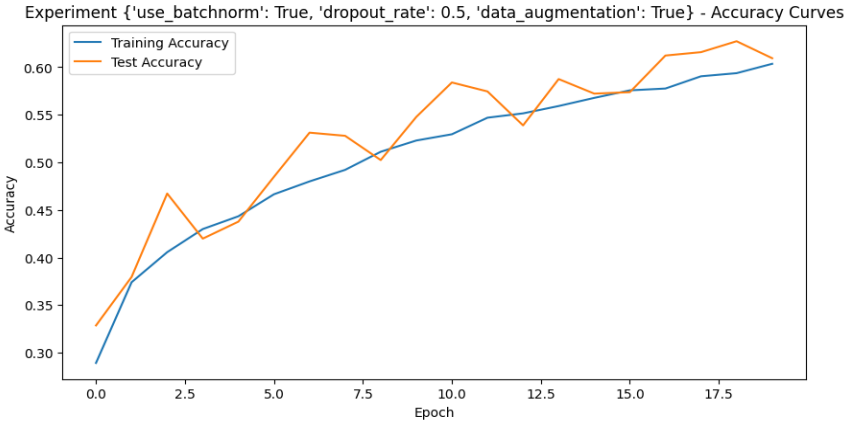


The comparison between the base model and the base model with data augmentation reveals that the latter exhibited instability during training, as evidenced by fluctuations in test loss and test accuracy. Ultimately, the model with data augmentation achieved lower results across all metrics, which are attributed to the alterations introduced by the data augmentation techniques.

Base model with batch normalization, dropout and data augmentation

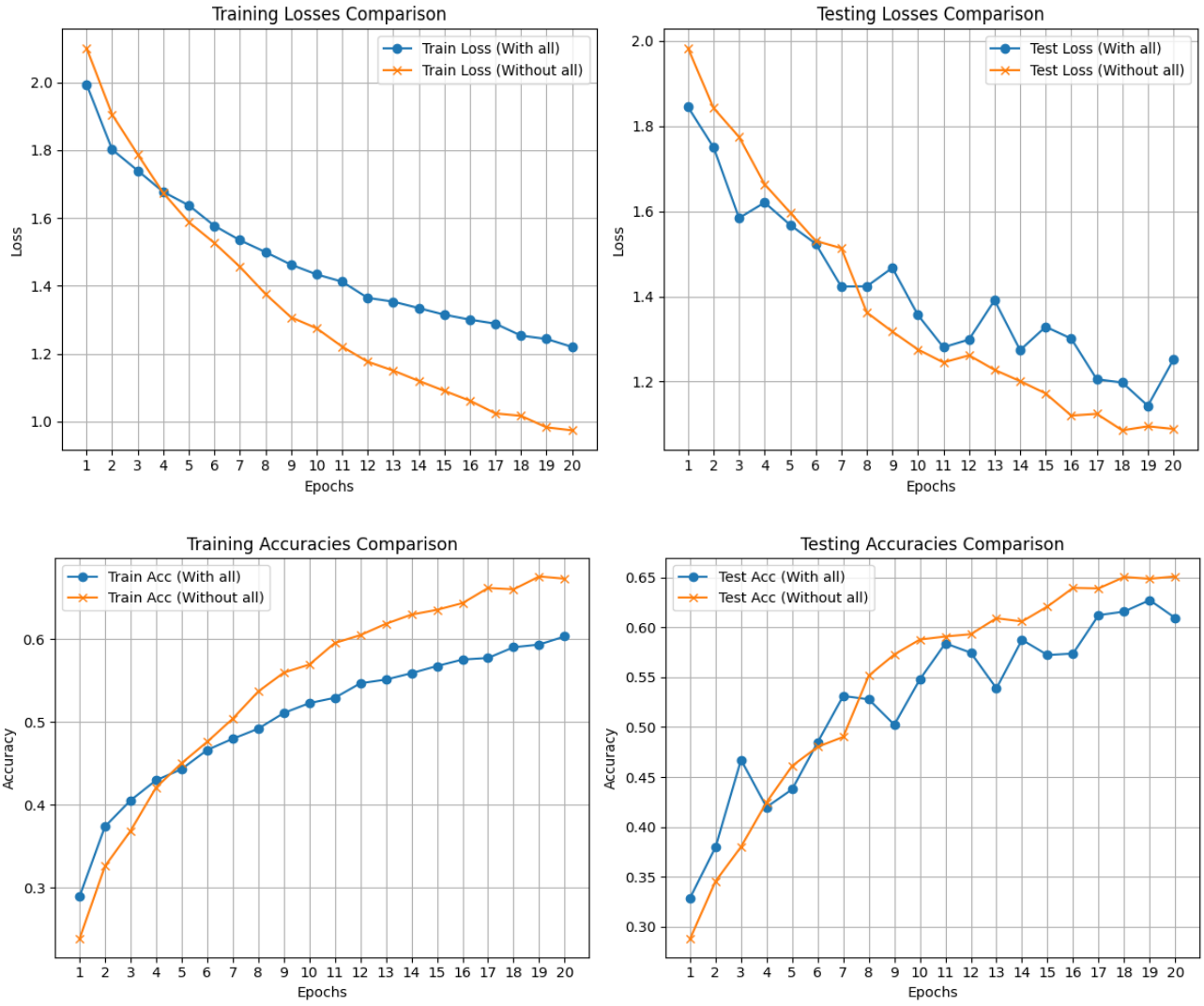






The confusion matrix for the model with batch normalization, dropout and with data augmentation also reveals that model struggle to classify the fourth class.

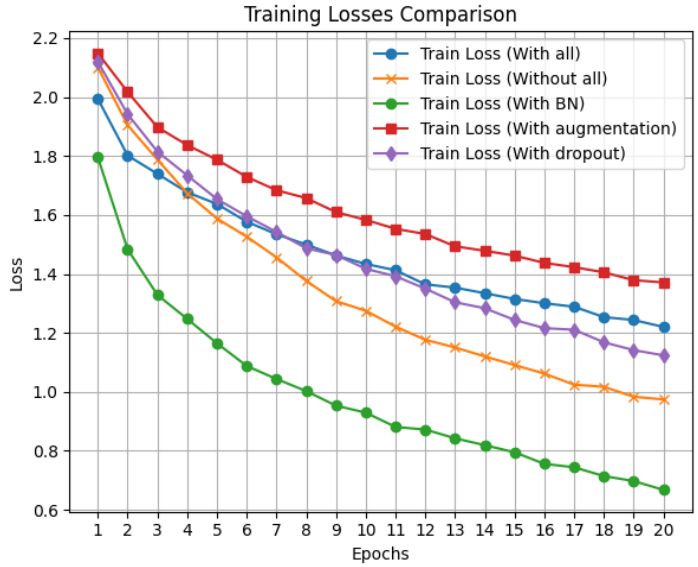
Comparison wuth base model

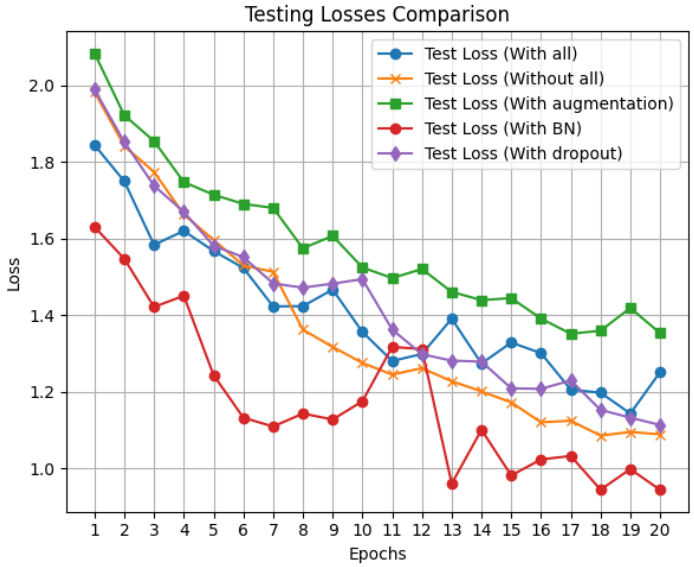


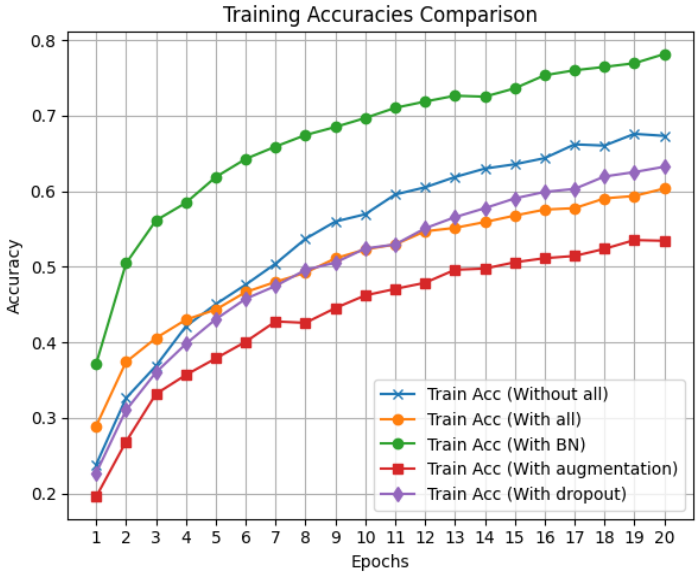
In comparing the base model to the model that incorporates dropout, batch normalization, and data augmentation, we observed that the latter model displayed instability during training. This instability was reflected in its performance, as it achieved lower results across all metrics.

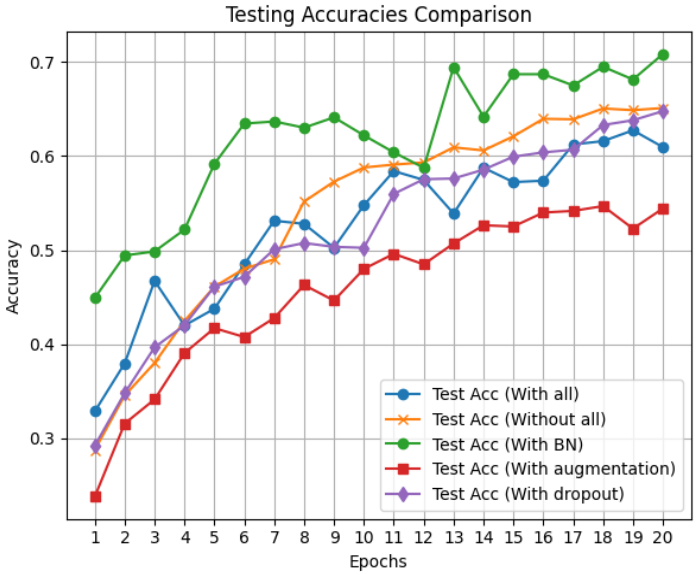
The combination of dropout and batch normalization aimed to enhance model generalization and training stability, while data augmentation sought to introduce variability in the training data. However, these strategies may have introduced conflicting dynamics, leading to difficulties in convergence and ultimately resulting in suboptimal performance.

Comparison of all models





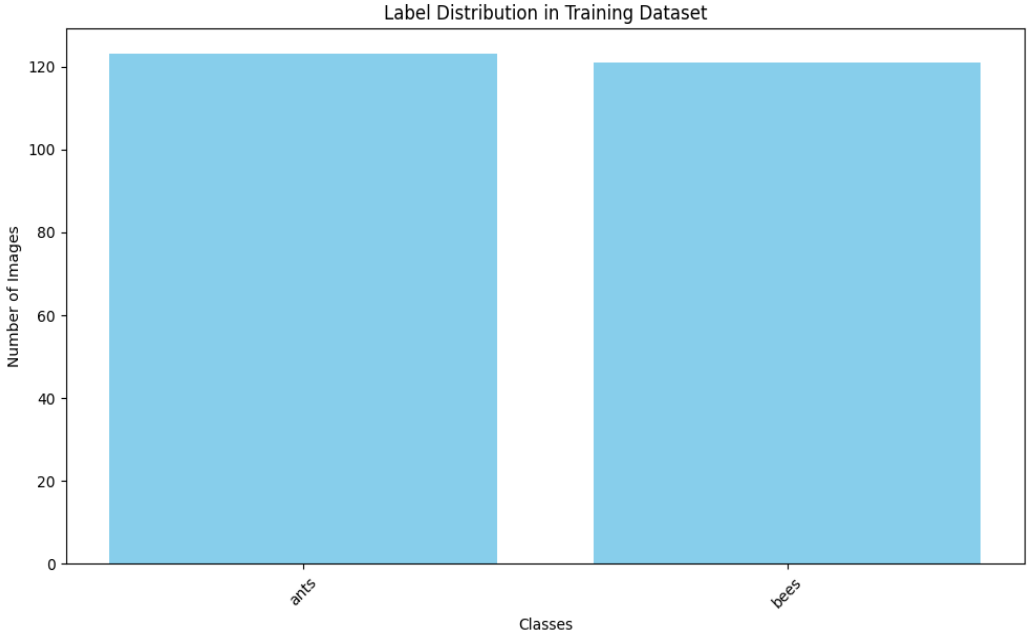


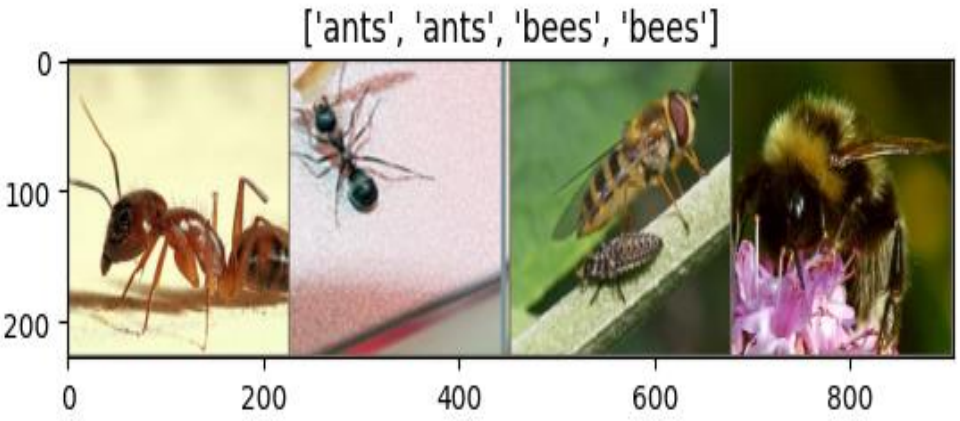


Overall, the model with batch normalization achieved the best results across all metrics.
Train Loss: 0.6674, Train Acc: 0.7815, Test Loss: 0.9440, Test Acc: 0.7083

Task 3

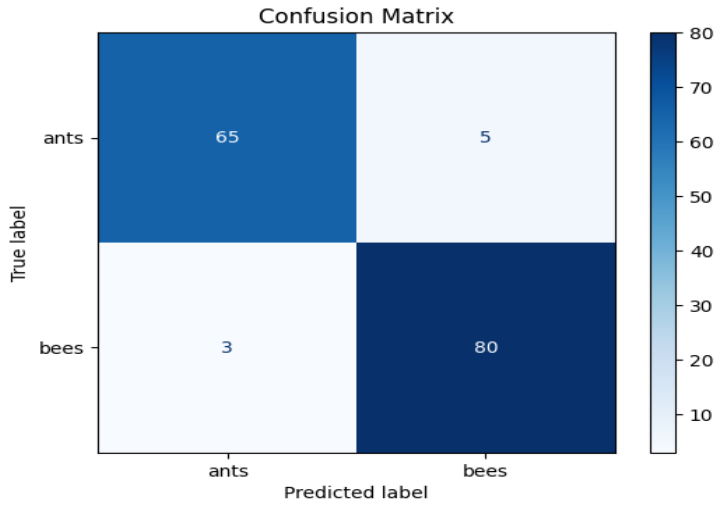
At this task I firstly understood to work with the dataset presented in the transfer learning tutorial with ants and bees, then realized that we have to work with the same dataset used at task 2 to compare the results.
Firstly I will present the results on the dataset from the tutorial, then will present the results on Imagenette.

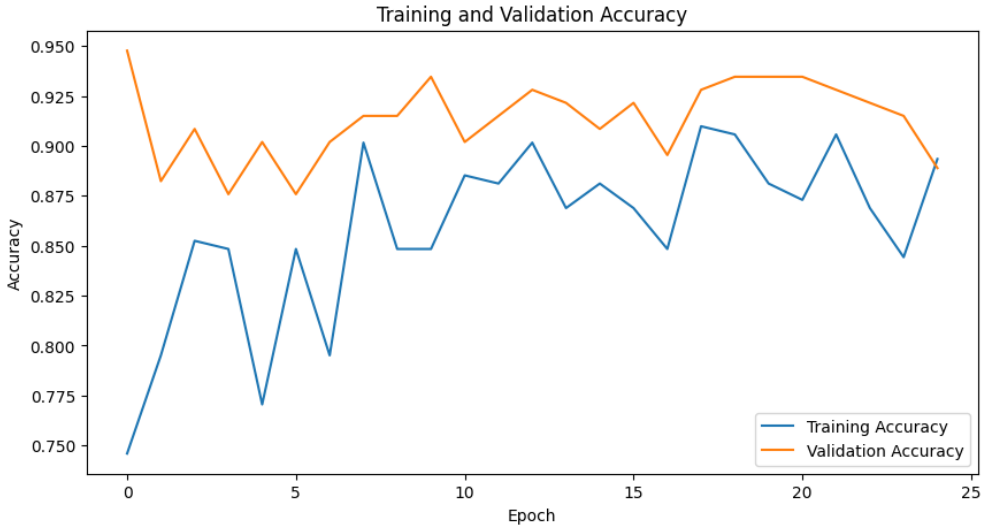




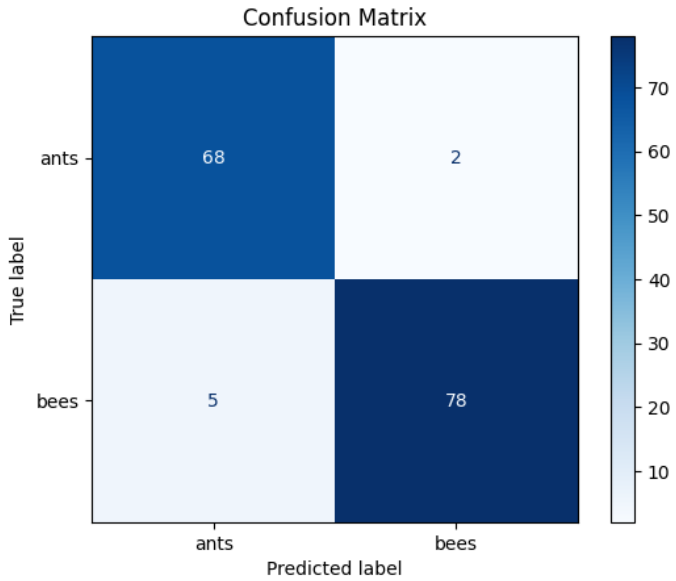
The labels are evenly distributed between the 2 classes in the dataset.

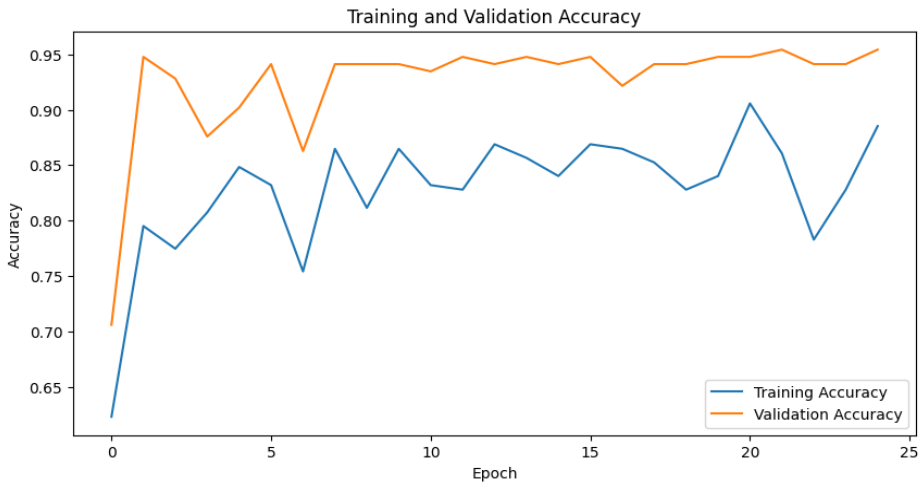
Fine tuned ResNet18



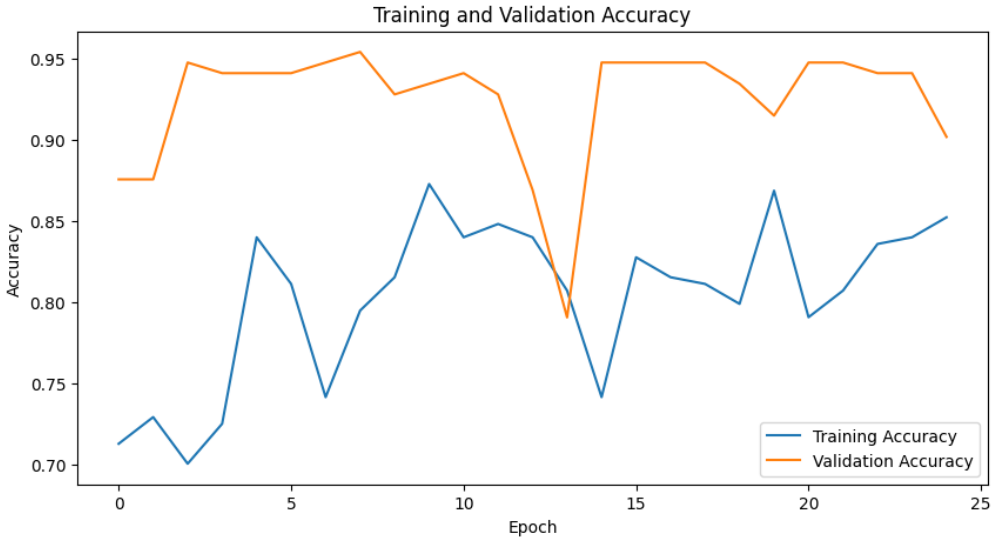


Frozen ResNet18

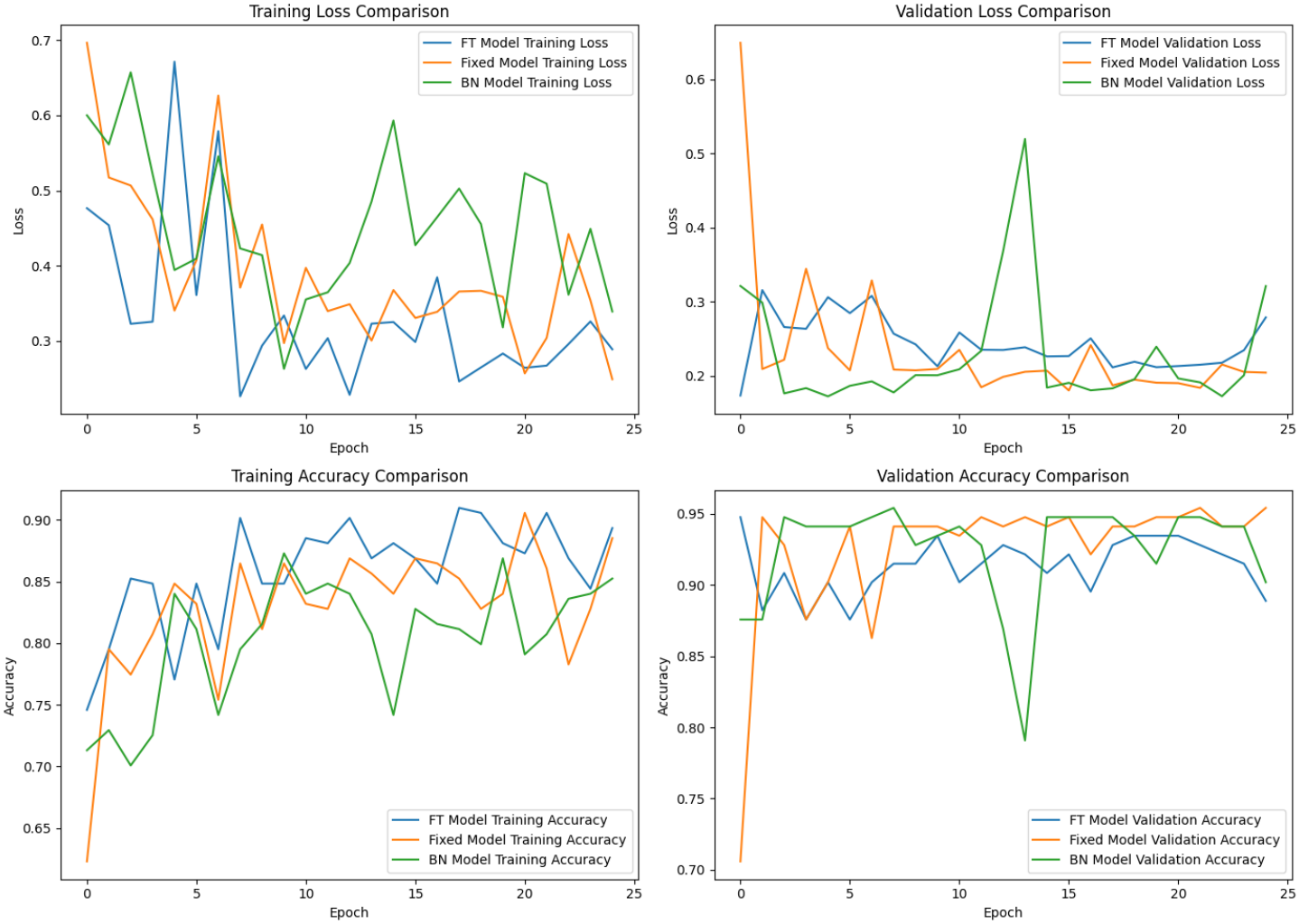




ResNet18 with unfrozen Batch Normalization layers



Comparison of all models

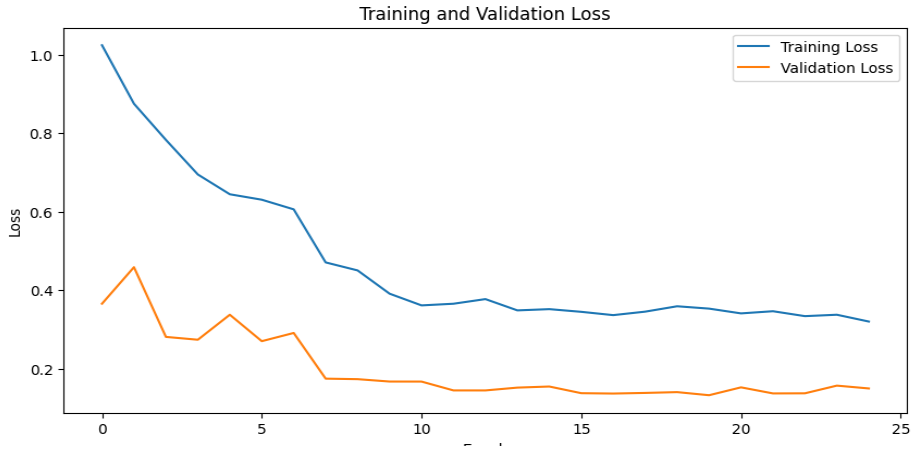


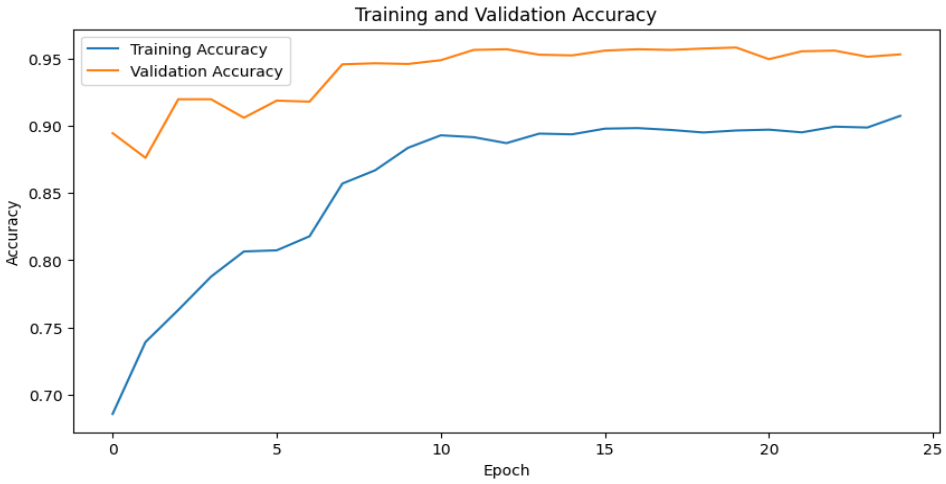
All models exhibited significant instability during training across all metrics. However, in the end, the frozen model outperformed the others, achieving the best validation loss and accuracy.

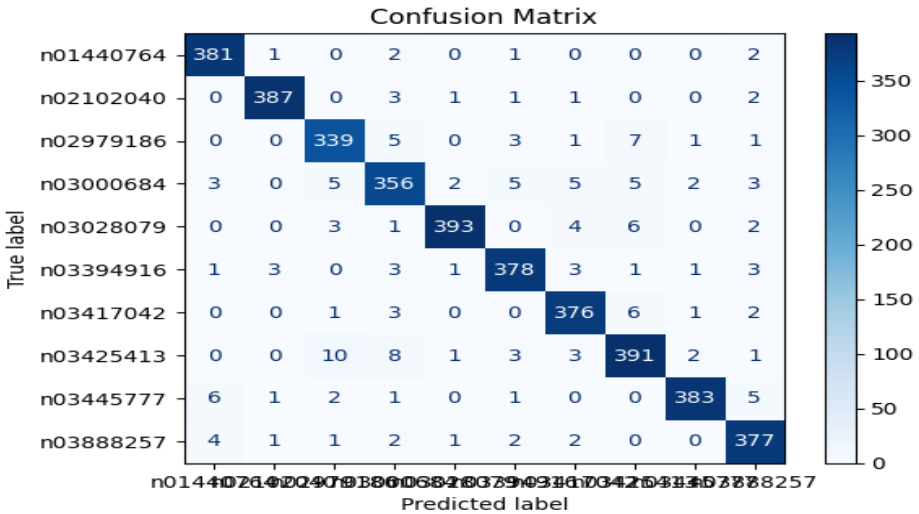
Task 3 on Imagenette

Fine tuned ResNet18

Best val Acc: 0.958217

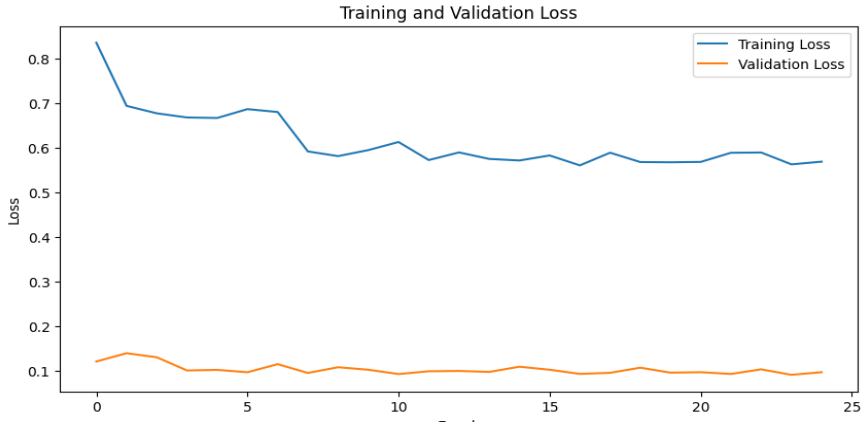


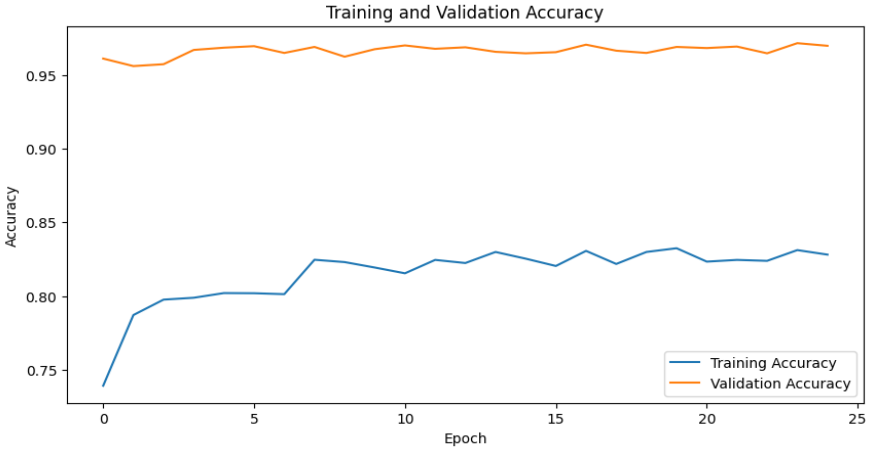


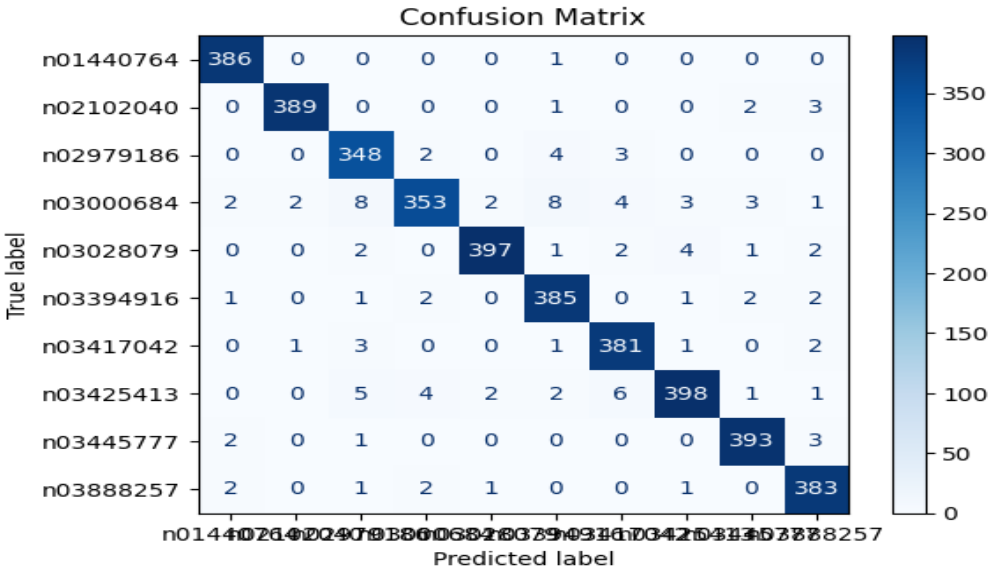


Frozen ResNet18

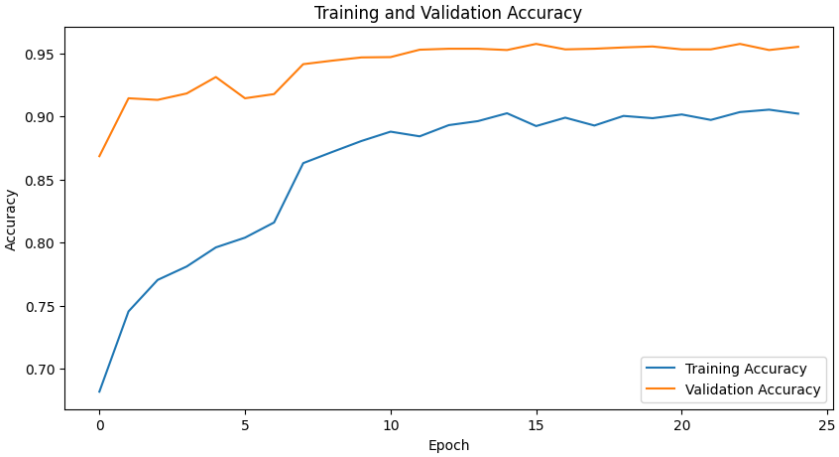
Best val Acc: 0.971465

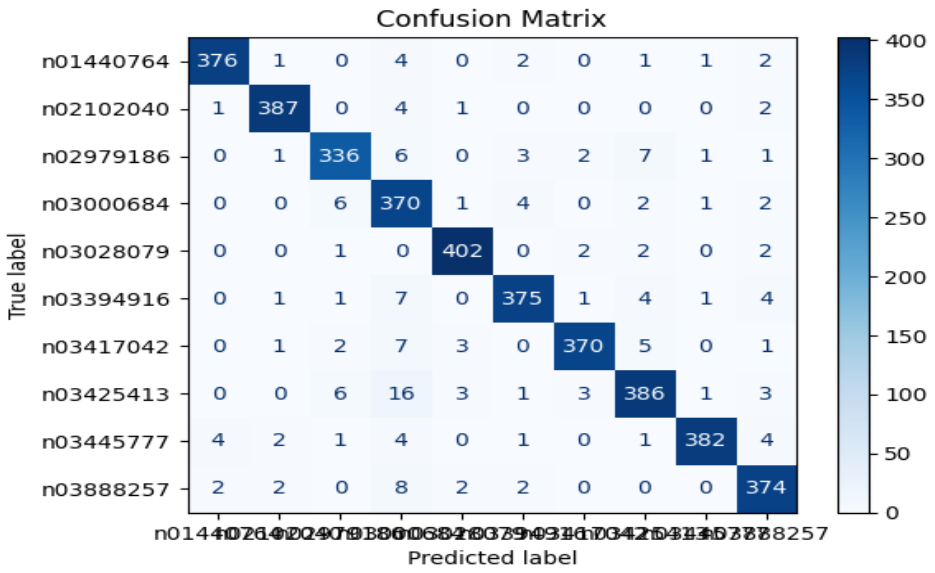




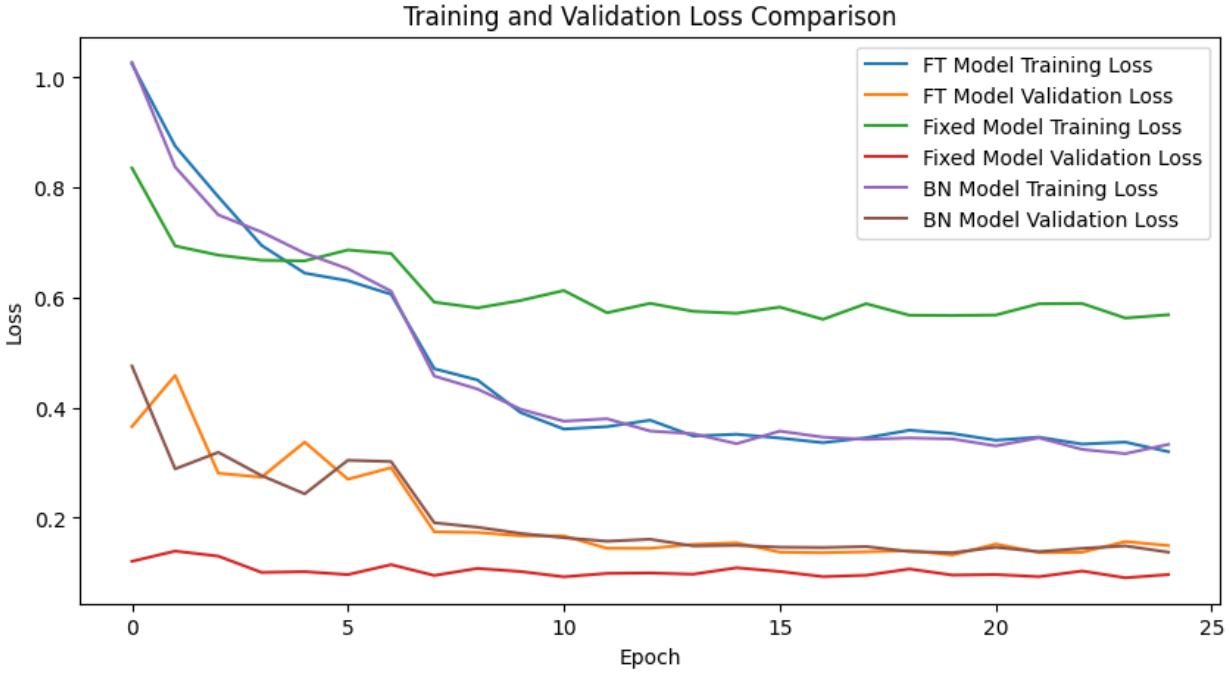


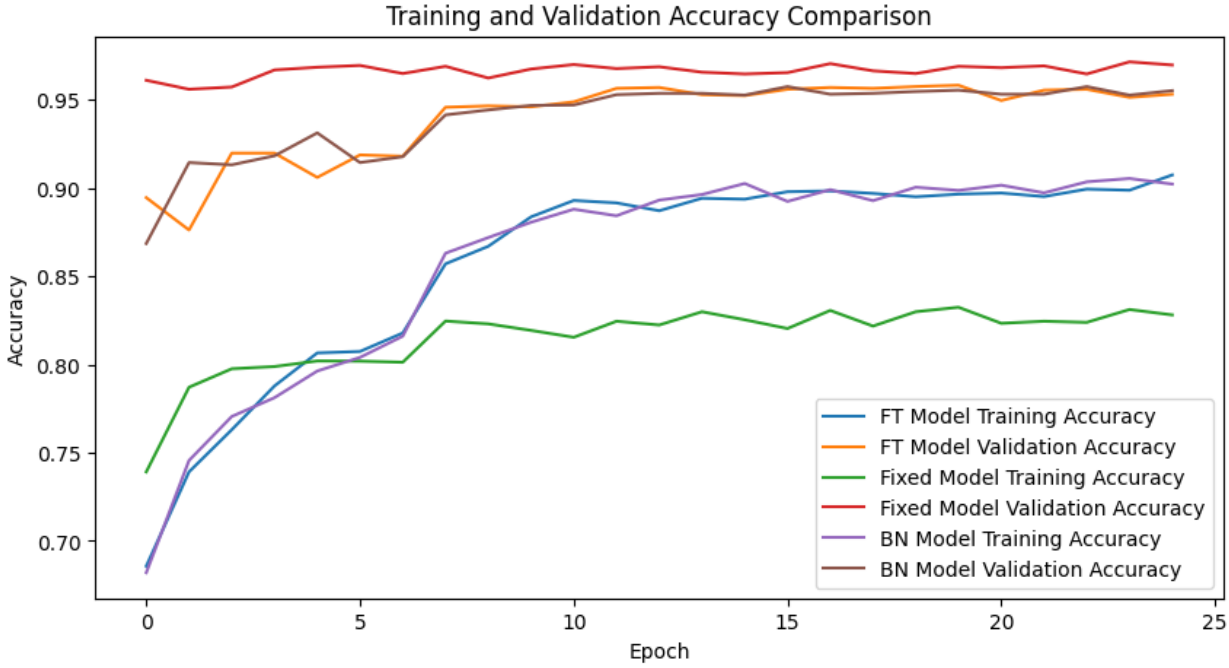
ResNet18 with unfrozen Batch Normalization layers
Best val Acc: 0.957452

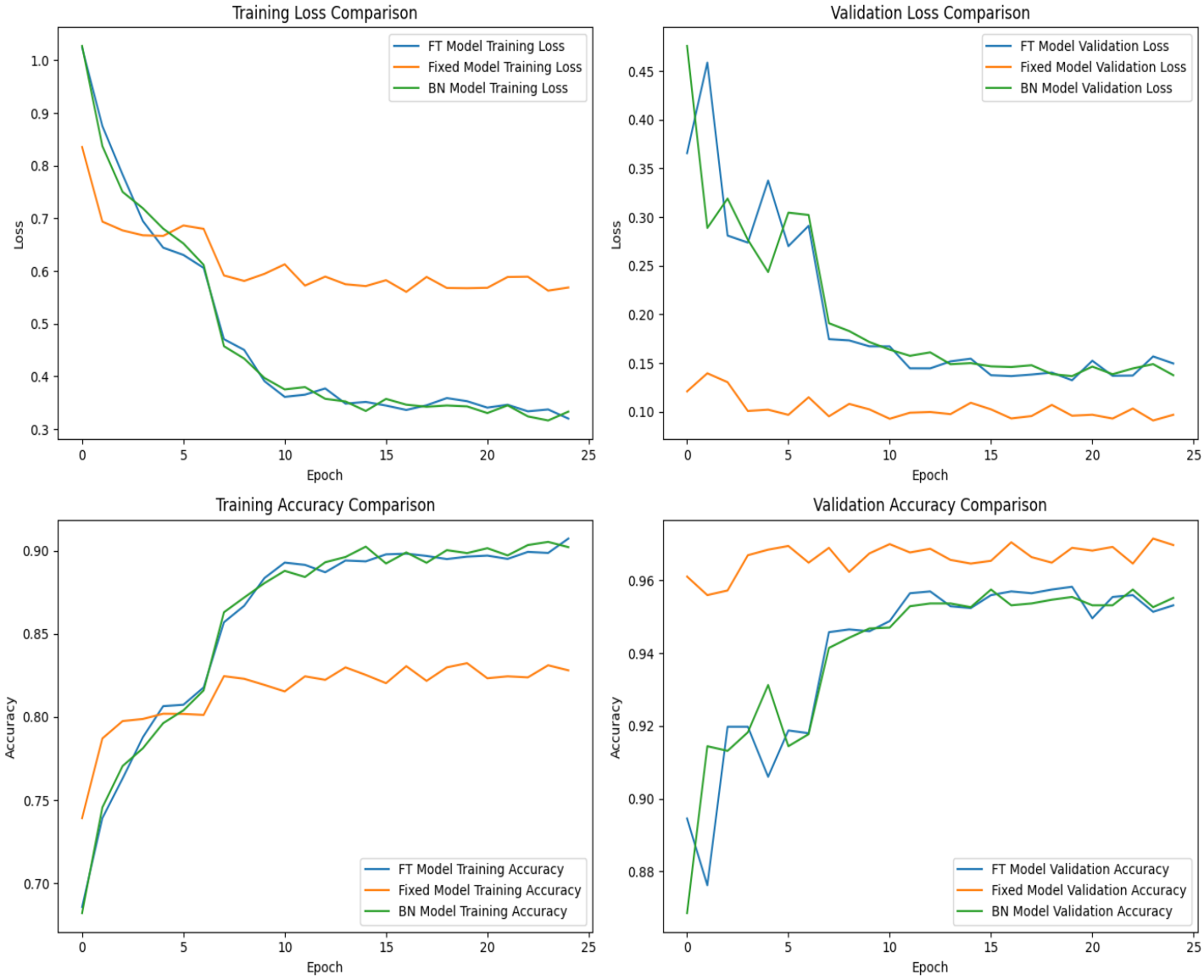




All models comparison







1. Training Loss and Accuracy:

Fine-Tuned (FT) Model: Shows a steady decrease in training loss and an increase in accuracy, which suggests that it is learning effectively from the data. The training accuracy stabilizes around 90%, which indicates a good fit for the training data.

Fixed Model (Frozen Layers): Has a higher training loss compared to the other models and lower accuracy, stabilizing around 80-82%. This implies that freezing the layers limits the model's ability to adapt to the new dataset.

BN Model (Batch Normalization Unfrozen): The training loss curve is similar to the FT model, and its accuracy also approaches 90%. This indicates that unfreezing the BN layers allows the model to adapt better to the training data than the Fixed model.

2. Validation Loss and Accuracy:
FT Model: Validation loss decreases initially but fluctuates, converging to a value similar to the BN model. The validation accuracy increases to around 95%, showing strong generalization to the validation data.
Fixed Model: Has the lowest validation loss and consistently higher validation accuracy (around 96%), suggesting it generalizes well. This is likely because the frozen layers prevent overfitting on the training data, resulting in good performance on the validation set.
BN Model: Similar to the FT model in validation loss and accuracy, though with slightly more fluctuations. The validation accuracy reaches around 95%, indicating decent generalization but a bit more variation compared to the Fixed model.

Effect of Unfreezing Batch Normalization Layers:
Unfreezing the BN layers in the BN Model slightly improves training loss and accuracy, helping the model fit the training data similarly to the FT model. However, in terms of validation performance, unfreezing BN does not outperform the significantly FT model and even introduces slightly more fluctuation in validation accuracy.

Unfreezing the BN layers helps the model adapt better to the training data but does not substantially improve validation performance over the FT model. If the goal is generalization (as seen in the Fixed model's validation results), freezing all layers except the final layer might be beneficial.

Overall Comparison:
Fixed Model has the best validation accuracy and generalization, with the lowest validation loss. However, it is limited in training performance due to the frozen layers.
FT Model performs well overall but is slightly more prone to fluctuations in validation accuracy, indicating a balance between fitting and generalization.
BN Model offers a compromise, with improved training performance but similar generalization to the FT model.

Comparison with CNN model from task 2

The pretrained ResNet18 models not only significantly outperformed the implemented CNN model in Task 2, but they also achieved noticeably higher accuracy. The confusion matrices for these models reflected better classification performance, with fewer misclassifications across classes, indicating stronger model generalization and more precise predictions. Leveraging pretrained weights allowed the ResNet18 models to capture complex features more effectively, which contributed to these improvements across all evaluation metrics.