CS 484 - Introduction to Computer Vision
Project Final Report
Plant Disease Detection and Classification

Deniz Sun
Bilkent University
Email: deniz.sun@ug.bilkent.edu.tr

Mehmet Emre Kantaş
Bilkent University
Email: emre.kantas@ug.bilkent.edu.tr

I. Introduction

This paper addresses the need for effective plant disease detection. The dataset covers 14 crops and 26 distinct diseases, regarding the diversity and complexity of the agricultural challenges. Through computer vision techniques and deep learning through 2 different architectures, we propose a solution for early diagnosis of crop diseases, highlighting the importance of enhanced crop health and sustainable agriculture practices in this report.

The implementation in the research article we follow [1] first preprocess the images in the dataset. There are a total of 31 crop-disease pairs. It starts by fitting the images in the dataset in a standard so that they can be processed easily. Then, the color characteristics of the background are detected and several masks are created. Finally, all masks are used to create a final mask. Applying this final mask separates the image and the background, by making the background black.

After processing the dataset, the research article creates deep convolutional neural network models in AlexNet and GoogleNet architectures [4] [5].

The analysis concludes that the best-performing model achieves a mean F1 score of 0.9934 and an overall accuracy of 99.35% [1]. The model that the best results were obtained from has GoogleNet architecture, uses a transfer learning training mechanism, and color dataset type. Also, This result is achieved in an 80% training set and 20% test set distribution.

II. Proposed Approach

For the preprocessing of the images in the dataset, we are using the implementation in the research article [3]. Using the preprocessed images, we have developed our deep-learning models by following the steps explained in the research article. Lastly, we compared our results and made inferences from them in the upcoming sections.

III. Deep Learning Model Architecture

The research article uses two different architectures, GoogleNet and AlexNet, for the convolutional neural network. Both of these architectures use stacked convolution layers followed by fully connected layers.

AlexNet consists of 8 layers: 5 convolution layers, 3 fully connected layers, and a softMax layer. Convolution layers 1 and 2 (conv1, 2) are each followed by a normalization layer and a pooling layer. The final convolution layer (conv5) is followed by just a pooling layer. All the first 7 layers are associated with ReLu non-linear activation units. The first two fully connected layers (fc6, 7) have dropout layers with a dropout ratio of 0.5.

After the final fully connected layer (fc), there are 31 outputs in this adapted version of AlexNet. This represents the total number of classes in the dataset, which is then fed into the softMax layer. This layer normalizes the input from (fc8) which produces the distribution across 31 different classes. These values add up to 1, and each gives the confidence that an input image is represented by one of the 31 classes. Figure 1 shows the AlexNet CNN architecture.

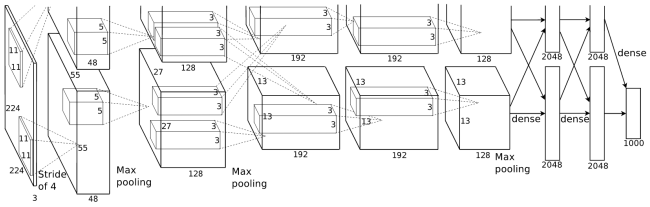

Fig. 1: AlexNet Architecture [4]

One other architecture used in designing the convolutional neural network was GoogleNet. This diverges remarkably from the simpler nature of AlexNet and offers a deeper architecture with 22 layers. Even though it has many more layers, it has 5 million parameters, which is significantly less than AlexNet's 60 million parameters. GoogleNet applies parallel 1x1, 3x3, and 5x5 convolutions and has a max-pooling layer. Thus, it can capture various features at the same time. GoogleNet architecture also offers dimensionality reduction with 1x1

convolutions before and after the larger convolutions and pooling for efficiency reasons. Also, encapsulating all the filters in a single output vector enables GoogleNet to adjust complexity and input size effectively. The article considers all types of hyperparameter changes and their implementation also provides the F1-Scores which can be seen in Figure 1. Figure 2 shows the GoogleNet CNN architecture.

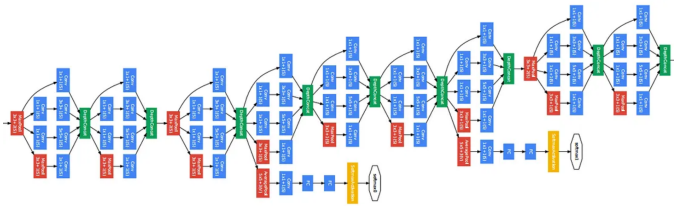

Fig. 2: GoogleNet Architecture [5]

The implementation of the research article also provides confusion matrices for different experiment settings. For example, Figure 2 shows the confusion matrix plot for the architecture AlexNet, with a color dataset type, an 80-20 dataset distribution, and the Train From Scratch training method.

IV. Implementations

A. General Details

There are 42.509 total images in the dataset [2]. The dataset split was chosen as 80-20 with 80% training and 20% testing in our setting, which makes 34.011 images to train and 8.498 to evaluate the models on. Also, we used color dataset type through our implementation. This decision is based on the best overall accuracy result in the original implementation as seen in Figure 1. To finalize dataset preprocessing, we normalized it using normative mean ([0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]) parameters.

As we are implementing what the research article describes, the hyperparameter ranges that we tested are similar to the ones mentioned in the paper. We considered the optimal parameters as a starting point and experimented by making small changes depending on the models.

- Solver type: Stochastic Gradient Descent,
- Base learning rate: 0.0001,
- Learning rate policy: Step (decreases by a factor of 10 every 30/3 epochs),
- Momentum: 0.9,
- Weight decay: 0.0005,
- Gamma: 0.1,
- Batch size(s): 1, 32 (GoogleNet), 100 (AlexNet).

We used DataLoader from the PyTorch library to load our data. Other than that, we created the class ImageDataset to handle finding images in the directories, preprocessing and loading them.

B. AlexNet Implementation

We use a custom AlexNet architecture that has layers adjusted to our needs for this project. First, we resize the input images from 256x256 to 227x227.

1) Feature Extraction: The custom AlexNet architecture starts with feature extraction layers. It consists of 5 convolutional layers, 2 local response normalization layers, 3 max-pooling layers and 1 convolution layer. The convolutional layers have different numbers of in and out channels, kernel sizes, stride numbers and paddings. These configurations for the layers allows the model to extract a wide range of features from the input images, including simple edges and textures that are crucial to classify the crop diseases. There are 5 ReLu activation functions in between the layers.

2) Intermediate Pooling: The intermediate Pooling layer consists of 1 adaptive average pooling 2D layer.

3) Classifier: The classifier layers consist of Dropout layers followed by Linear layers and ReLu activation functions, for 2 iterations. The final layer is another Linear layer.

The final fully connected layer has 31 outputs in the adapted version of AlexNet, which is equal to the number of classes in the dataset.

C. GoogleNet Implementation

There are several different layers and modules to implement training from scratch GoogleNet model.

1) Feature Extraction: The implementation of the GoogleNet model starts with a feature extraction layer. The feature extraction layer consists of 3 convolutional layers, 2 max-pooling layers, and 2 local response normalization layers (not in that order). Convolutional layers have different numbers of in and out channels, kernel sizes, stride numbers and paddings.

This difference between layers enables the model to extract a diverse range of features from the input images, from simple edges and textures to more complex patterns. This variety in layer configurations ensures that each convolutional layer can focus on different aspects of the image at varying scales and complexities to enable the network to build a more robust and comprehensive feature representation necessary for accurate image classification tasks.

2) Inception Modules: After the feature extraction layer, there are 2 inception modules with different parameters. The

Inception Module parameters represent the number of in and out channels.

The Inception Modules consist of 4 different branches. The first branch consists of a convolutional layer, the second and third branches consist of two convolutional layers and the fourth branch consists of a max-pooling layer and a convolutional layer.

3) Classification Pipeline: The 4 layers after the Inception Modules serve as the classification pipeline. The first one is a max-pooling layer, the second one is an adaptive average pooling layer and the third one is a dropout layer. The fourth layer is the fully connected (Linear) layer. This fully connected layer maps the learned high-level features represented by the 480 input features to the number of classes (10 in our case) in the output. It's a linear transformation that learns to weigh the importance of various features for class prediction.

D. Training Details

We calculate the loss using the 'CrossEntropyLoss' function from the PyTorch library. This loss function is well-suited for classification problems, as it combines 'LogSoftmax()' and 'NLLLoss()' in one single class, making it ideal for multi-class classification tasks.

Once the loss is calculated, we use backpropagation to compute the gradients. This is done through the 'backward()' function, which propagates the error backward through the network, allowing us to calculate the gradient of the loss for each weight.

The optimizer used for training the model is the Adam optimizer, also from the PyTorch library. Adam, which stands for Adaptive Moment Estimation, is an efficient optimization algorithm that computes individual adaptive learning rates for different parameters. It combines the advantages of two other extensions of stochastic gradient descent: the adaptive gradient algorithm (AdaGrad) and the root mean square propagation (RMSProp).

Our optimizer uses the 'step()' function to update the weights based on the calculated gradients. This function adjusts the weights in the direction that minimizes the loss, based on the gradients computed during backpropagation.

V. Results

Hyperparameter tuning is a crucial step when working with convolutional neural networks. For our project, we chose 3 different parameters to experiment with: number of epochs, batch size and learning rate.

Choosing a meaningful number of epochs was the first step we took. Too few epochs could lead to underfitting while too many epochs could lead to overfitting in the model. We decided to start with few, and increase it to a value that will not lead to overfitting. We displayed the loss value and the accuracy for each epoch from 1 to 20. The range between 10-20 gave the best results for both models.

The second hyperparameter we chose for tuning was batch size. Since our dataset had 42.509 images, changing the batch size had a trade-off between speed and memory. We could not experiment with a full batch size as our hardware was not capable enough for it. For GoogleNet, we tested batch sizes 1 and 32. For AlexNet, we tested it for 1, 32 and did our hyperparameter tuning for batch sizes 100 and 256.

The final hyperparameter we selected was learning rate in hopes to reach a minimum of the loss function. We tested both models for learning rates 0.001, 0.0005 and 0.0001 and overall achieved good results with these rates.

A. AlexNet Results

For AlexNet, the different hyperparameter configurations are as follows:

- Batch size: [1, 32, 100, 256]
- Epochs: [5, 10, 15, 20]
- Learning Rate: [0.001, 0.0005, 0.0001]

First, we trained the model with batch size 32 and learning rate 0.001. With this configuration, we tested the convergence of the epochs. The accuracy results seemed to converge after around 15 epochs so there was no significant improvement between epochs 15 and 20. After seeing that result we decided to test the other hyperparameters for only 5-10-15 epochs. The best result that we achieved was an F1 score of 0.92 and accuracy of 92.26% for batch size 256, learning rate 0.0005 and 10 epochs as can be seen in Table 1. Overall, the accuracies started to converge towards epoch 15.

LR	Epochs	Batch Size	Test Accuracy
0.001	5	100	84.11%
0.001	10	100	87.80%
0.001	15	100	91.93%
0.005	5	100	84.18%
0.0005	10	100	91.27%
0.0005	15	100	91.97%
0.0001	5	100	82.42%
0.0001	10	100	88.67%
0.0001	15	100	91.89%
0.001	5	256	83.63%
0.001	10	256	88.63%
0.001	15	256	88.49%
0.005	5	256	89.41%
0.0005	10	256	92.36%
0.0005	15	256	91.77%
0.0001	5	256	88.31%
0.0001	10	256	91.55%
0.0001	15	256	91.49%

TABLE I: Summary of AlexNet Runs

B. GoogleNet Results

For GoogleNet, a total of 15 experiments have been conducted, using the hyperparameter configurations below:

- Batch size: [1, 32]
- Epochs: [5, 10, 15, 20]
- Learning Rates: [0.001, 0.0005, 0.0001]

In these experiments, the effect of learning rate, batch size and epoch number on the accuracy of the model have been observed.

Due to computational limitations, the number of experiments conducted with batch size 1 is 3. As experiments conducted selecting batch size 1 generally resulted worse than the experiments conducted selecting batch size 32, we didn't finish all different experiment settings using batch size 1.

Different learning rates used in the experiments are 0.001, 0.0005 and 0.0001. Different epochs used in the experiments are 5, 10, 15 and 20. Different batch sizes used in the experiments are 1 and 32. The test accuracies for different experiment settings are given in the table below. The yellow row is the experiment setting with the best test accuracy, 98.98%.

LR	Epochs	Batch Size	Test Accuracy
0.001	5	1	78.42%
0.001	10	1	89.55%
0.001	15	1	86.42%
0.001	5	32	98.76%
0.001	10	32	98.98%
0.001	15	32	98.78%
0.001	20	32	98.93%
0.0005	5	32	92.22%
0.0005	10	32	92.15%
0.0005	15	32	95.58%
0.0005	20	32	95.56%
0.0001	5	32	96.60%
0.0001	10	32	97.18%
0.0001	15	32	98.06%
0.0001	20	32	94.94%

TABLE II: Summary of GoogleNet Runs

C. Evaluation Metrics

For evaluation of the results, we plotted confusion matrices for these experiments, which can be found in the Appendix in sections VIII-A andVIII-B. As the overall accuracy scores are high, the diagonal pattern can be observed in all the confusion matrices for the hyperparameter testings.

In the confusion matrix of the best experiment setting for GoogleNet, which is Figure 61, one can observe the diagonal pattern due to the high accuracy score.

Also, to observe the difference, Figure 57 (which is the confusion matrix of the worst experiment setting for GoogleNet) can be observed. As can be seen, the diagonal pattern is not as clear as it is in Figure 61, due to the relatively lower accuracy score.

Furthermore, we plotted precision-recall curves for these experiments, which can be found in sections VIII-A and VIII-B. For example, Figure 76 shows the precision-recall curve of the best experiment setting for GoogleNet. It can be observed that the precision-recall curves for each class tend to lean towards the top-right corner, due to the high precision and recall scores.

Also, to observe the difference, Figure 72 (which is the precision-recall curve of the worst experiment setting for GoogleNet) can be observed. As can be seen, the precision-recall curves are more random than they are in Figure 76, due to the relatively lower precision and recall scores.

Lastly, we plotted ROC (receiver operating characteristic) curves for these experiments, which can be found in sections VIII-A and VIII-B. In ROC curve plots, the AUC (area under the curve) value is used. AUC takes values between 0 and 1. Higher AUC values indicate stronger performance for the classification model. In Figure 91 (which is the ROC curve of the best experiment setting for GoogleNet), one can observe the ROC curves for each class tend to the top-left corner, due to the high AUC values.

Also, to observe the difference, Figure 87 (which is the ROC curve of the worst experiment setting for GoogleNet) can be observed. As can be seen, the ROC curves are more random than they are in Figure 91, due to the relatively weaker overall classification performance.

VI. Discussions

As we used training from scratch models, the values we introduced in section V can be considered high. One of the reasons for this can be that we generally followed the implementations of the paper. Also, the optimizers and loss functions from the PyTorch library helped us further improve the performances of our models. There are small differences between the results we obtained and the results from the paper. These differences, their possible reasons and solutions will be discussed in the following subsections elaborately.

One particular issue with our results was caused by the uneven dataset. This can be observed in our Precision-Recall and ROC curves, where one class has a particulary low ratio, which causes the overall F1 score and accuracy to decrease. An example can be seen in Figure 28 where Class 16 has Precision-Recall area under curve (PR AUC) = 0.35 while other classes are almost 1.0, which is the maximum value they can get to achive a perfect score. This class can be observed as having more mismatches and less accurate results all throughout our project. This is due to the fact that this

class only had 152 images while other classes have much more items, usually a few thousand. This made our models prone to have errors when it encounters this class, as there was not much data to train on.

We believe that if this class was excluded, or if it had more images, the overall accuracy of our models could potentially improve.

A. Discussions on AlexNet

AlexNet showed a solid performance in our crop disease identification task. The architecture's simplicity, with its eight layers consisting of convolutional and fully connected layers, allowed it to perform well on our dataset where the features are relatively straightforward to identify. However, our results were slightly below those reported in the reference paper.

To improve the AlexNet performance, regularization techniques such as weight decay can be implemented that could potentially improve generalization while preventing overfitting.

B. Discussions on GoogleNet

Generally, in all of the experiments, our GoogleNet from scratch model performed better than our AlexNet model, due to its relatively more complex nature. However, the introduced results for GoogleNet in V-B are a bit lower than the paper's results, which were given in section I.

There are lots of possible reasons for this difference. First and most significant of all is that we do not have the exact implementation of the paper's GoogleNet model. The general GoogleNet structure was discussed in the paper. However, there isn't any specific detail about its implementation. Also, the optimizer settings or the additional features and methods may be used in the paper's implementation, which may yield a difference.

To close the gap between the paper's and our results, there can be implemented an early stopping mechanism using validation loss. We have to allocate some of the training dataset as the validation set for this. However, this early stopping technique can increase the model performance, regardless of the smaller training dataset size. One other method might be to implement a learning rate scheduler or optimizer hyperparameter tuning to find the best optimizer settings specifically for our model.

VII. Conclusion

In our research, we effectively utilized 2 convolutional neural network models, AlexNet and GoogleNet, for the classification of plant diseases. Our comprehensive experiments emphasized the critical role of hyperparameter adjustments, such as learning rate, batch size, and epoch count, in enhancing the performance of the models. Our results indicated that while AlexNet delivered commendable outcomes, GoogleNet consistently surpassed it due to its intricate architecture and superior feature extraction abilities.

The experiments revealed that appropriate tuning of hyperparameters significantly impacts model accuracy. For instance, the learning rate influenced the convergence speed and final accuracy, with 0.0005 yielding the best results for both models. Batch size affected both computational efficiency and model generalization, with batch size 32 being optimal for GoogleNet. Epochs needed careful balance to avoid underfitting and overfitting, with 15-20 epochs providing the best results.

Despite the hardware limitations and differences in implementation details from the original paper, our models achieved high accuracy, showcasing the promise of deep learning in detecting plant diseases. Future work could focus on incorporating advanced techniques like early stopping, learning rate schedulers, and further optimizer tuning to bridge the gap between our results and those reported in the literature. This research highlights the critical role of hyperparameter tuning in developing effective and reliable CNN models for agricultural applications.

References

[1] P. Mohanty, D. P. Hughes, and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection," *Frontiers in Plant Science*, vol. 7, Sep. 2016. Available: https://doi.org/10.3389/fpls.2016.01419. Accessed: March 15, 2024.

[2] A. Ali, 2019. "PlantVillage Dataset" [Data set]. Kaggle. Available: https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset?select=color. Accessed: March 15, 2024.

[3] digitalepidemiologylab, "plantvillage_deeplearning_paper_dataset," GitHub, Feb. 27, 2024. Available: https://github.com/digitalepidemiologylab/plantvillage_deeplearning_paper_dataset. Accessed: April 24, 2024.

[4] P. Varshney, 2020. "AlexNet Architecture: A Complete Guide," Kaggle. Available: https://www.kaggle.com/code/blurredmachine/alexnet-architecture-a-complete-guide. Accessed: March 15, 2024.

[5] "Understanding GoogleNet Model – CNN Architecture," GeeksForGeeks. Available: https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/. Accessed: March 15, 2024.

VIII. Appendix

A. AlexNet Plots

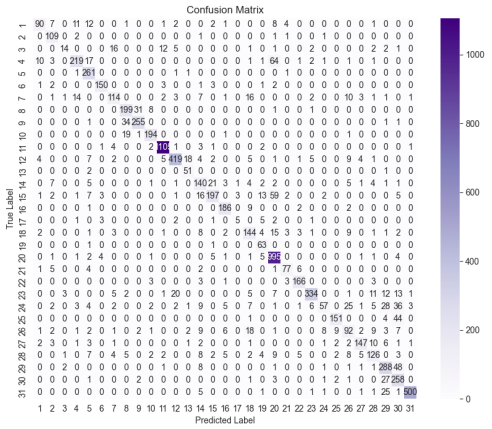

Fig. 3: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.001, Epoch 5

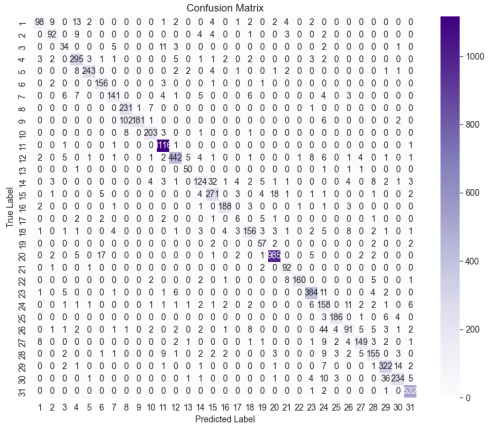

Fig. 4: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.001, Epoch 10

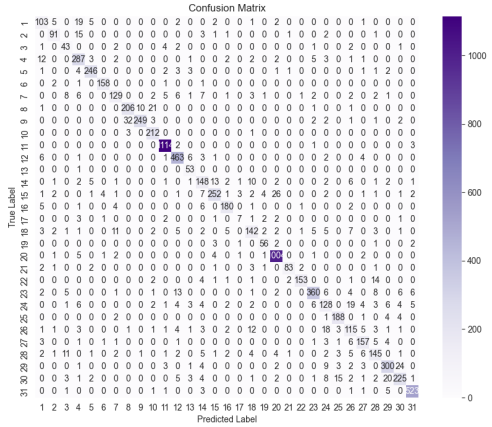

Fig. 5: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.001, Epoch 15

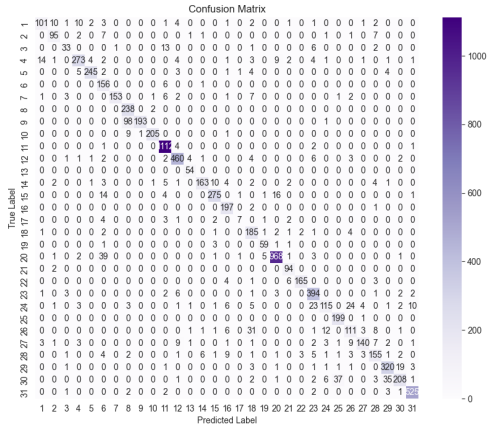

Fig. 6: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.0005, Epoch 5

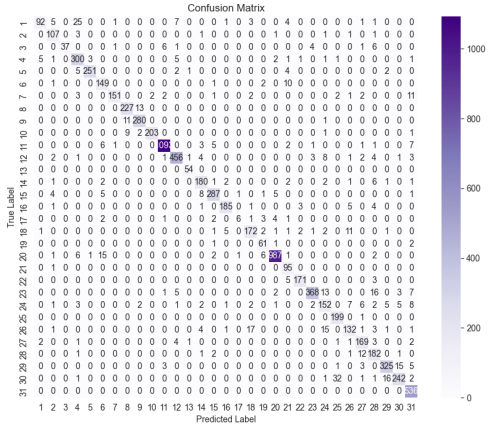

Fig. 7: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.0005, Epoch 10

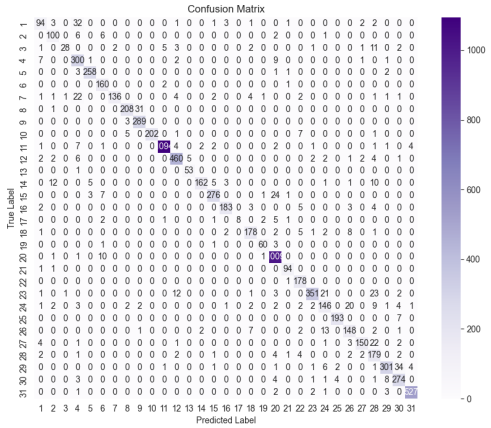

Fig. 8: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.0005, Epoch 15

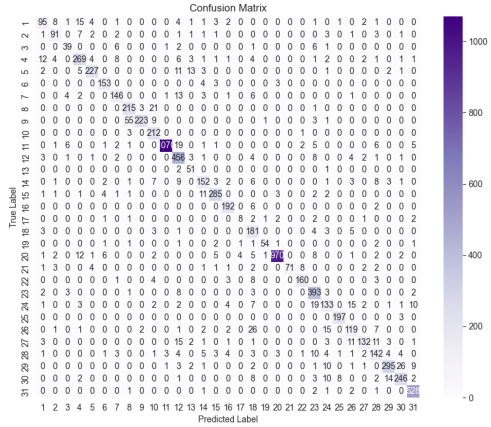

Fig. 9: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.0001, Epoch 5

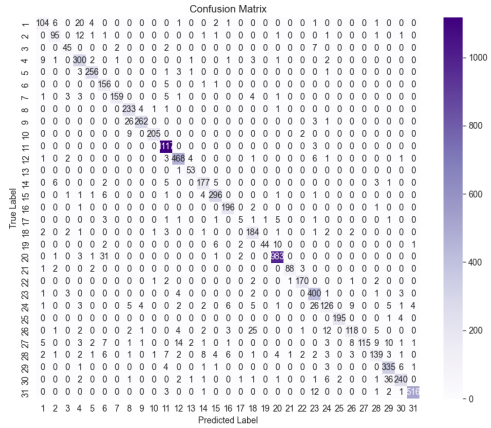

Fig. 10: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.0001, Epoch 10

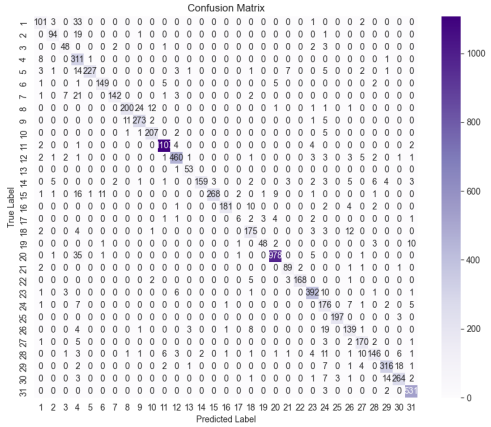

Fig. 11: AlexNet Confusion Matrix for Batch Size 100, Learning Rate 0.0001, Epoch 15

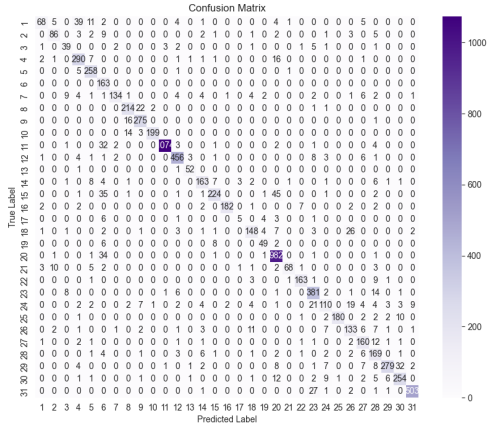

Fig. 13: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.001, Epoch 10

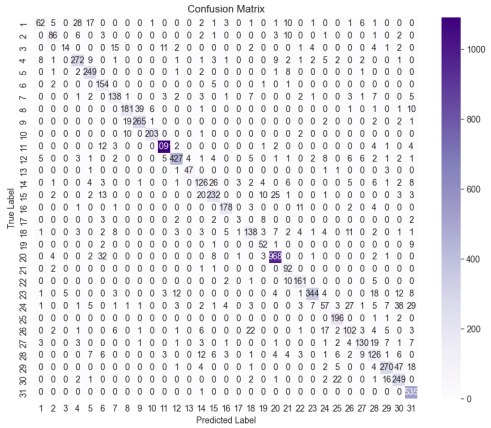

Fig. 12: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.001, Epoch 5

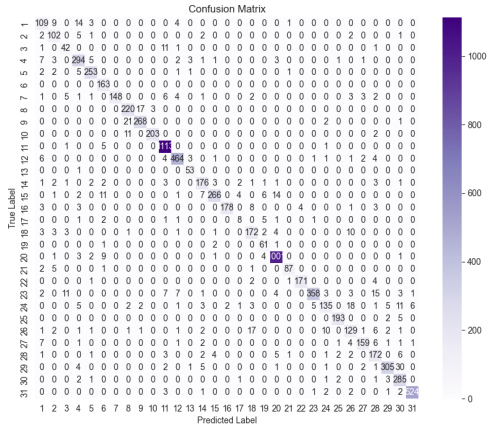

Fig. 14: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.001, Epoch 15

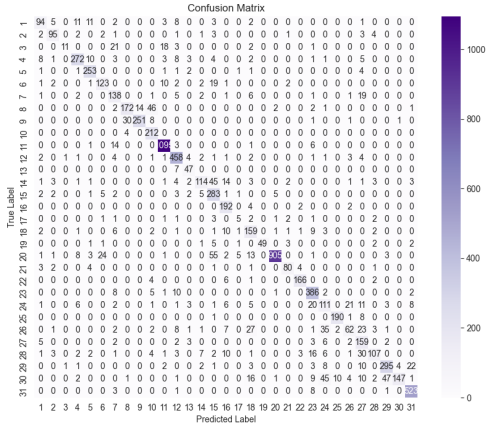

Fig. 15: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.0005, Epoch 5

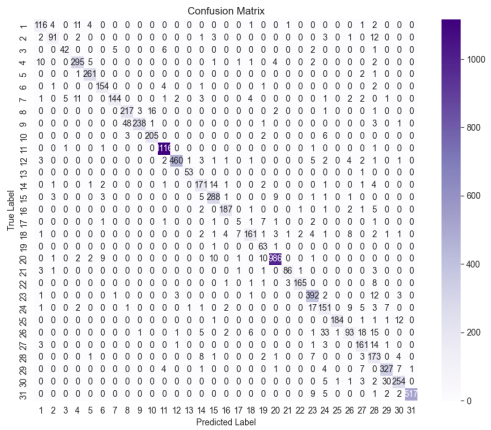

Fig. 16: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.0005, Epoch 10

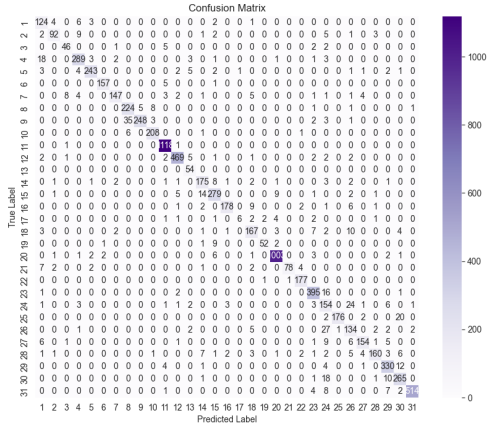

Fig. 17: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.0005, Epoch 15

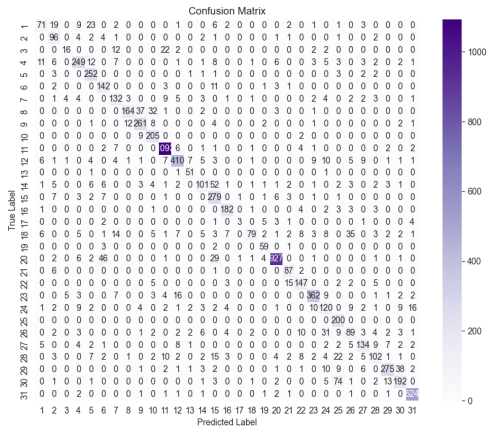

Fig. 18: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.0001, Epoch 5

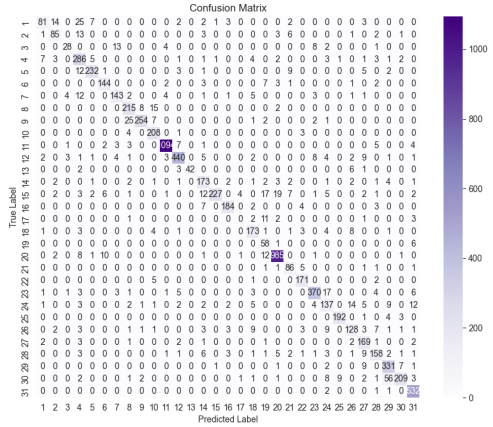

Fig. 19: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.0001, Epoch 10

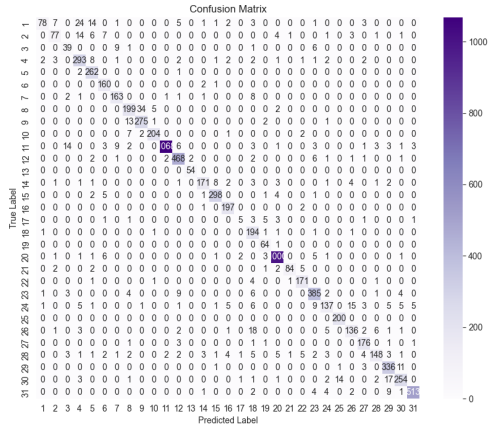

Fig. 20: AlexNet Confusion Matrix for Batch Size 256, Learning Rate 0.0001, Epoch 15

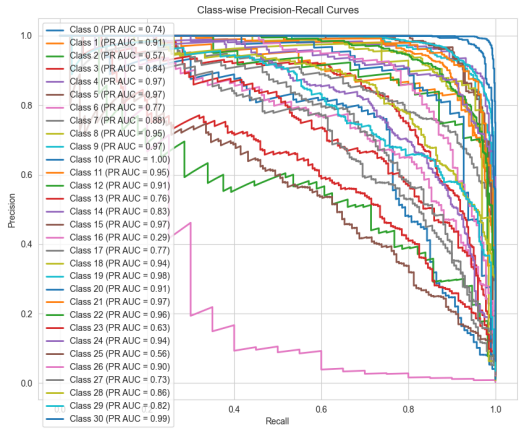

Fig. 21: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.001, Epoch 5

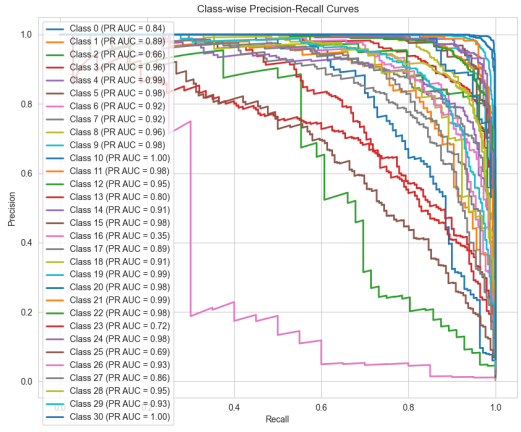

Fig. 22: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.001, Epoch 10

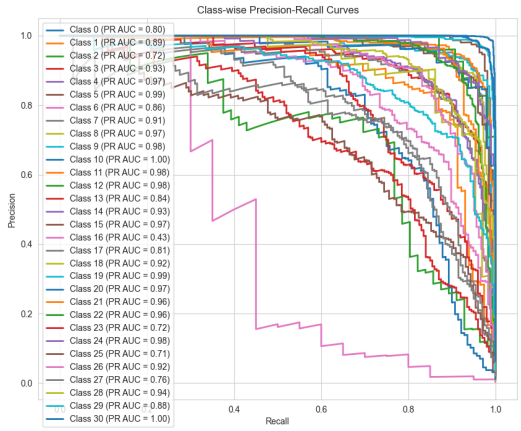

Fig. 23: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.001, Epoch 15

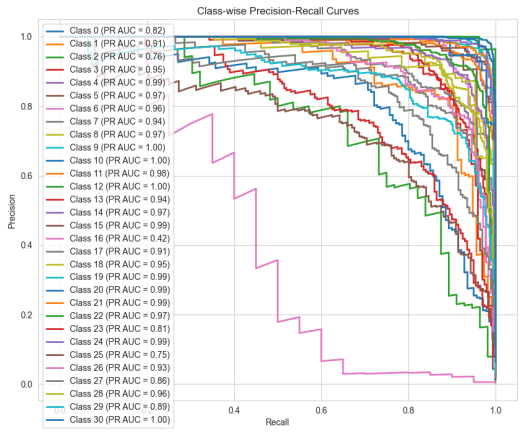

Fig. 24: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.0005, Epoch 5

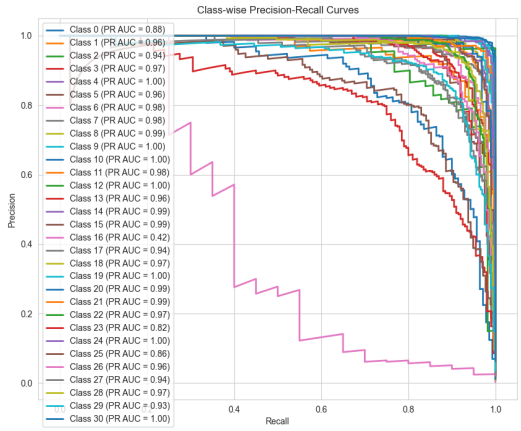

Fig. 25: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.0005, Epoch 10

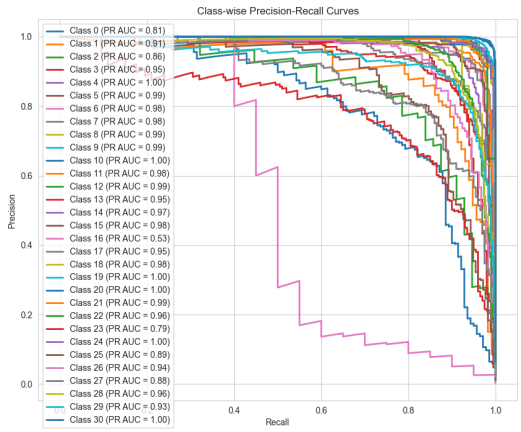

Fig. 26: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.0005, Epoch 15

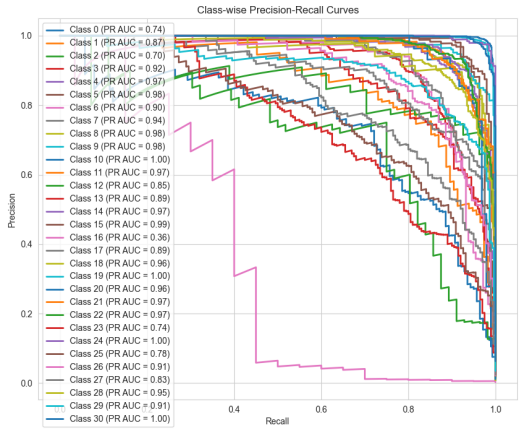

Fig. 27: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.0001, Epoch 5

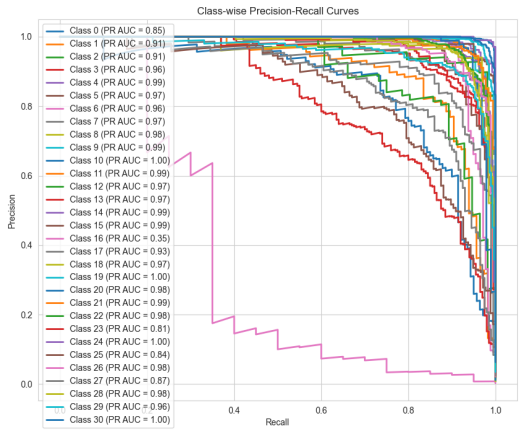

Fig. 28: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.0001, Epoch 10

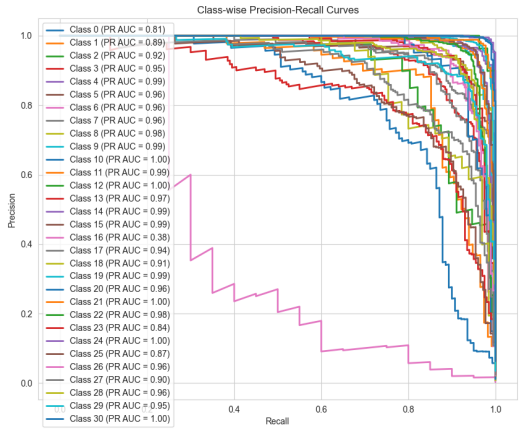

Fig. 29: AlexNet Precision-Recall Curve for Batch Size 100, Learning Rate 0.0001, Epoch 15

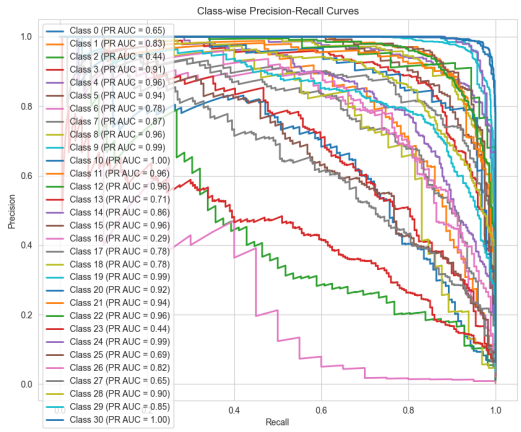

Fig. 30: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.001, Epoch 5

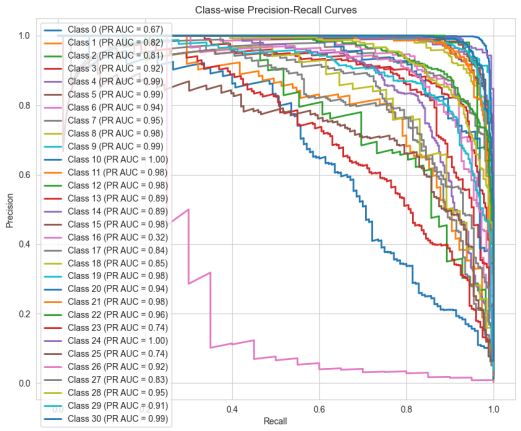

Fig. 31: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.001, Epoch 10

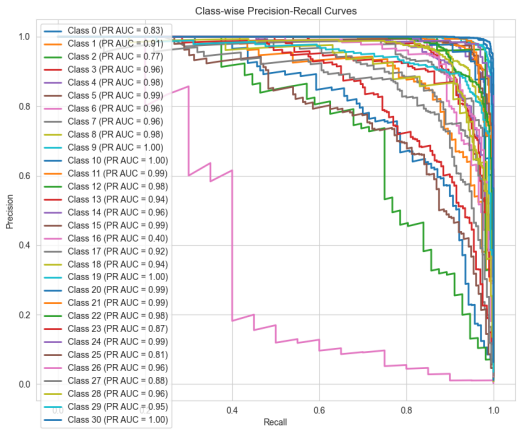

Fig. 32: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.001, Epoch 15

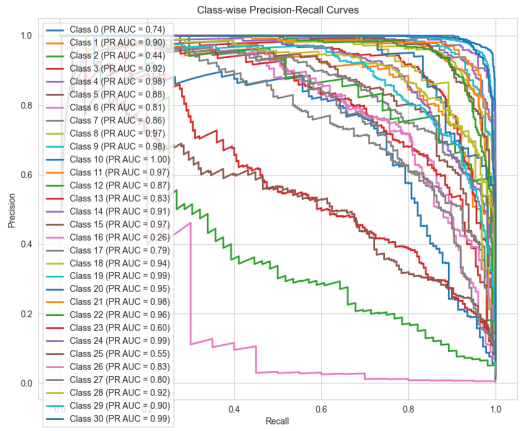

Fig. 33: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.0005, Epoch 5

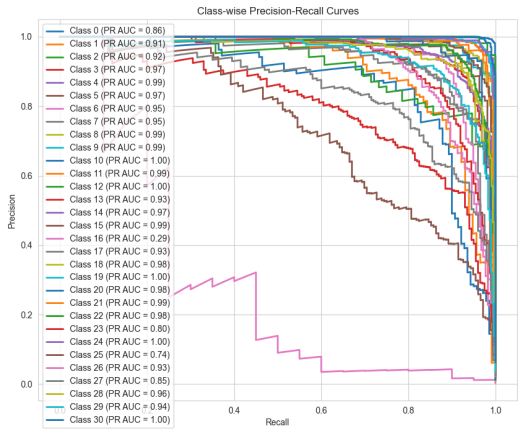

Fig. 34: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.0005, Epoch 10

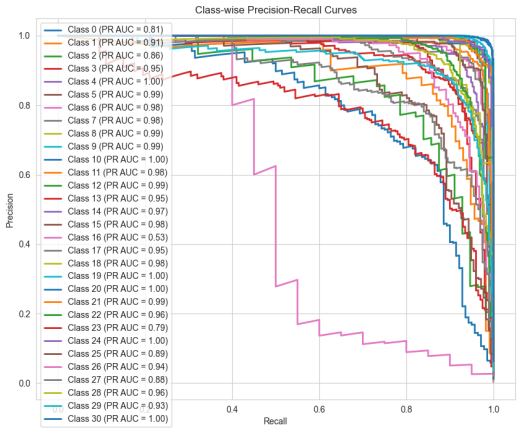

Fig. 35: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.0005, Epoch 15

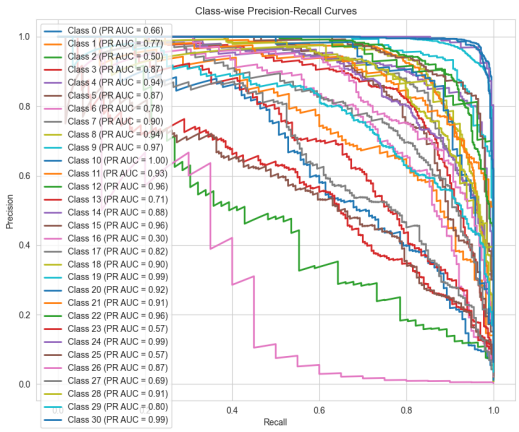

Fig. 36: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.0001, Epoch 5

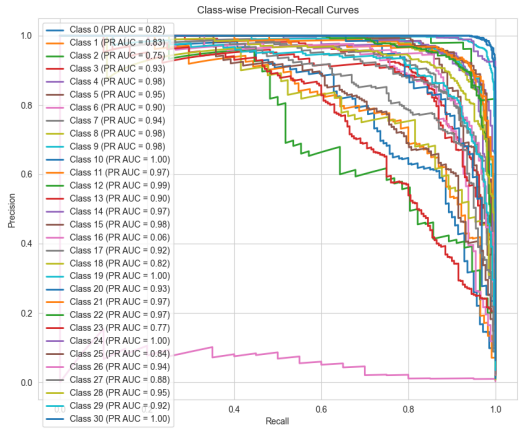

Fig. 37: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.0001, Epoch 10

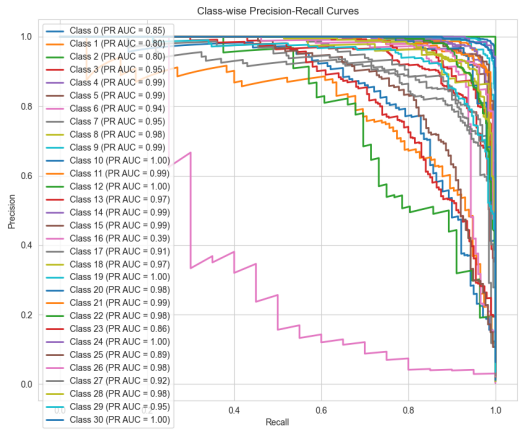

Fig. 38: AlexNet Precision-Recall Curve for Batch Size 256, Learning Rate 0.0001, Epoch 15

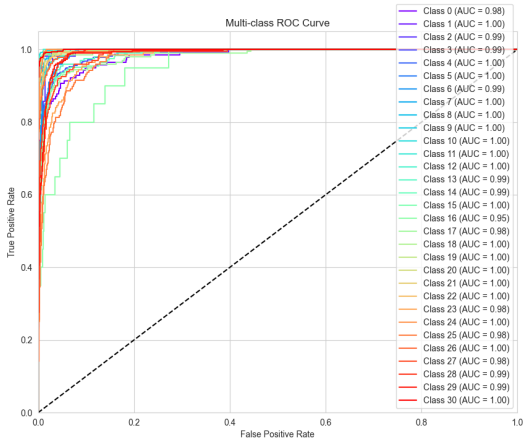

Fig. 39: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.001, Epoch 5

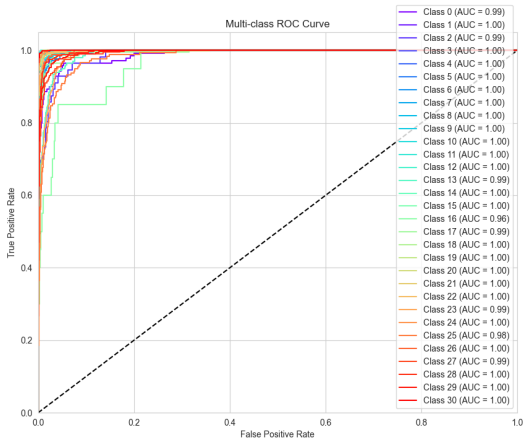

Fig. 40: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.001, Epoch 10

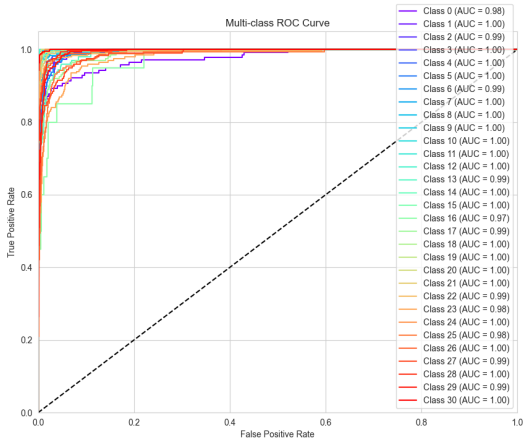

Fig. 41: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.001, Epoch 15

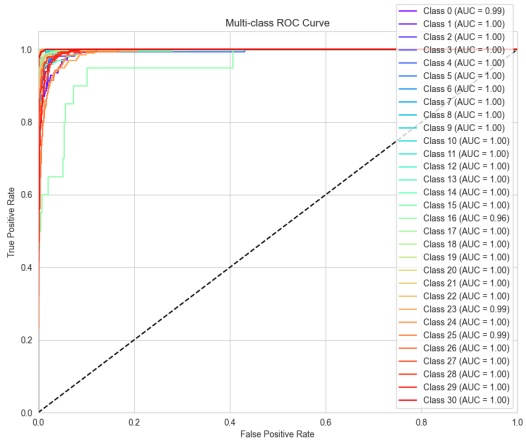

Fig. 42: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.0005, Epoch 5

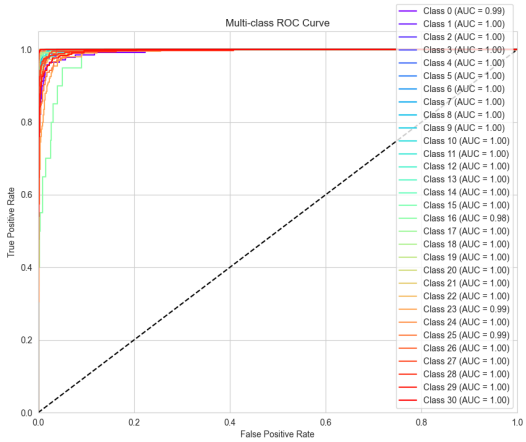

Fig. 43: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.0005, Epoch 10

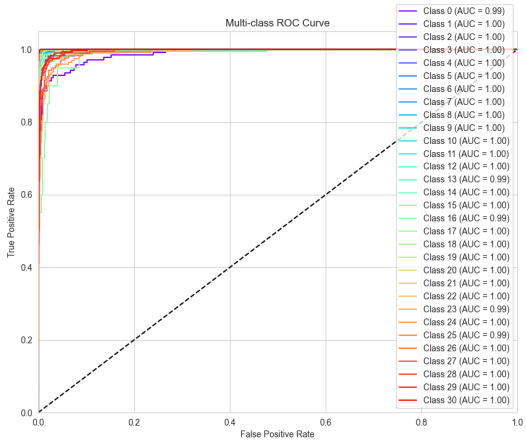

Fig. 44: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.0005, Epoch 15

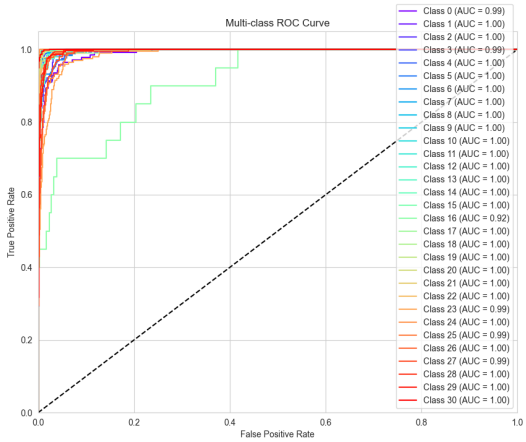

Fig. 45: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.0001, Epoch 5

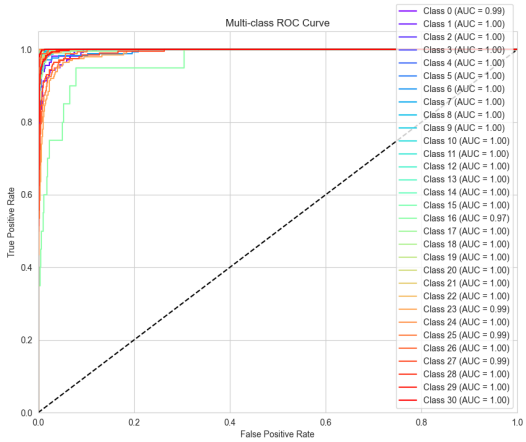

Fig. 46: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.0001, Epoch 10

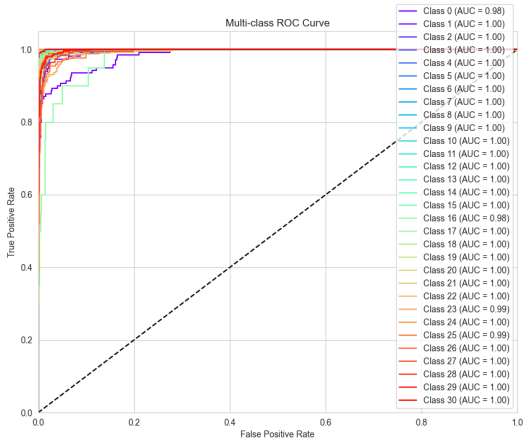

Fig. 47: AlexNet ROC Curve for Batch Size 100, Learning Rate 0.0001, Epoch 15

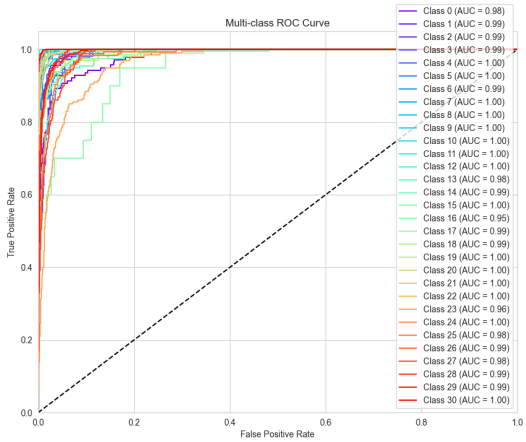

Fig. 48: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.001, Epoch 5

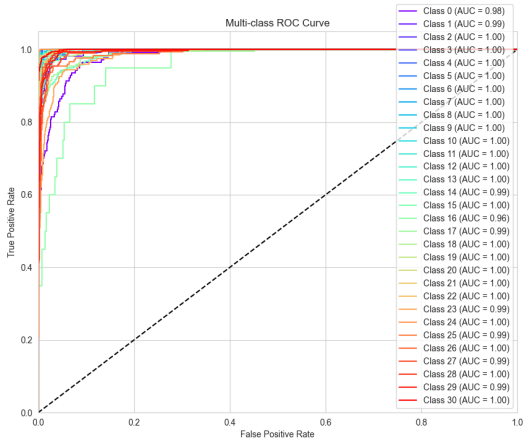

Fig. 49: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.001, Epoch 10

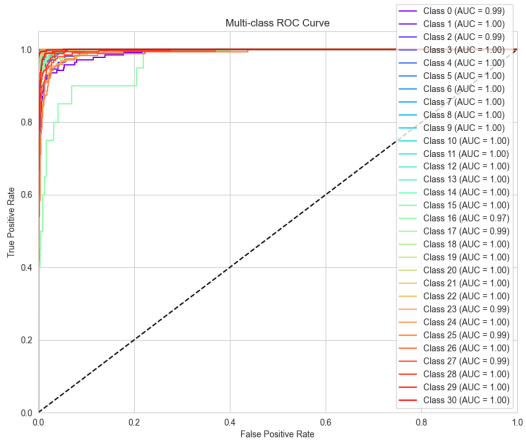

Fig. 50: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.001, Epoch 15

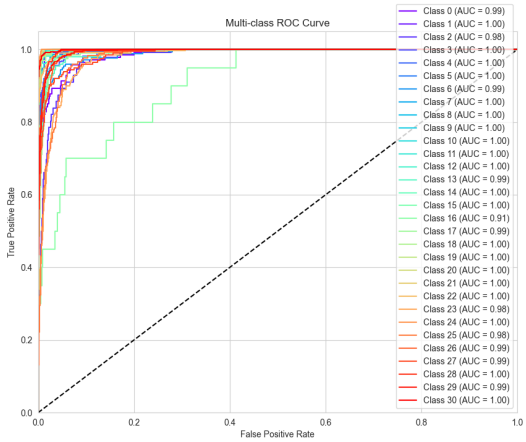

Fig. 51: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.0005, Epoch 5

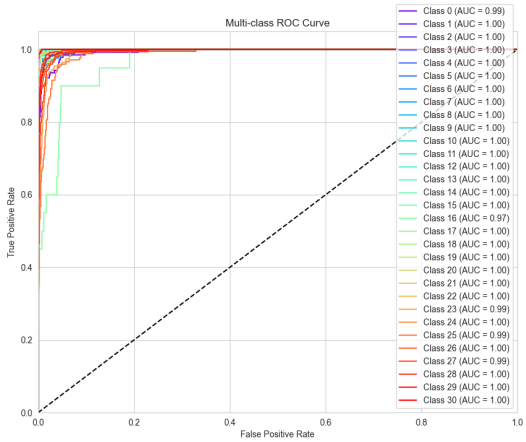

Fig. 52: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.0005, Epoch 10

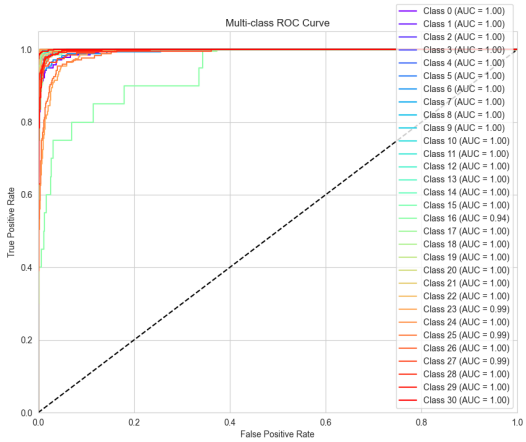

Fig. 53: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.0005, Epoch 15

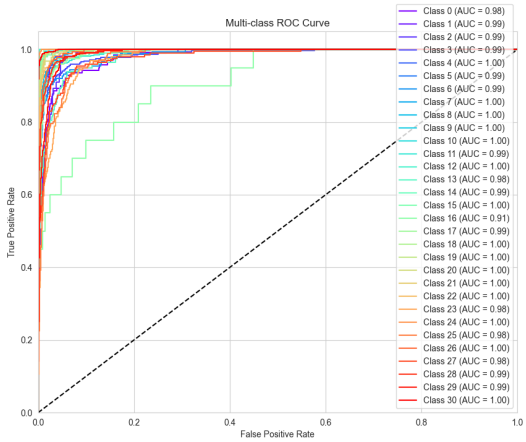

Fig. 54: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.0001, Epoch 5

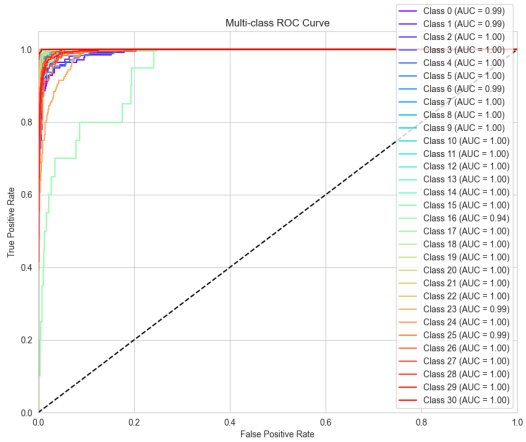

Fig. 55: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.0001, Epoch 10

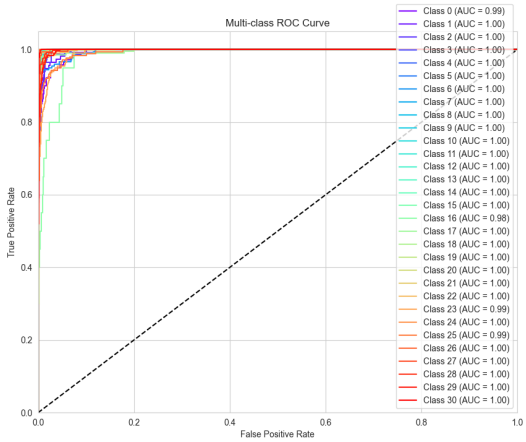

Fig. 56: AlexNet ROC Curve for Batch Size 256, Learning Rate 0.0001, Epoch 15

B. GoogleNet Plots

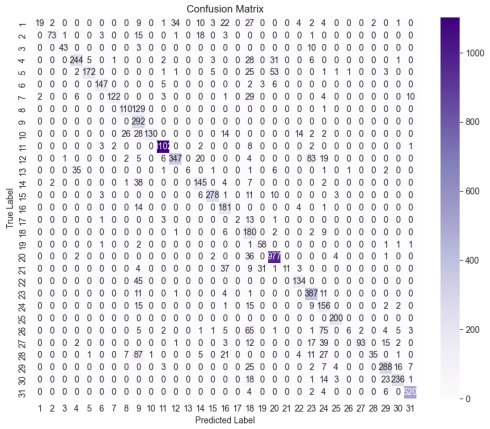

Fig. 57: GoogleNet Confusion Matrix for Batch Size 1, Learning Rate 0.001, Epoch 5

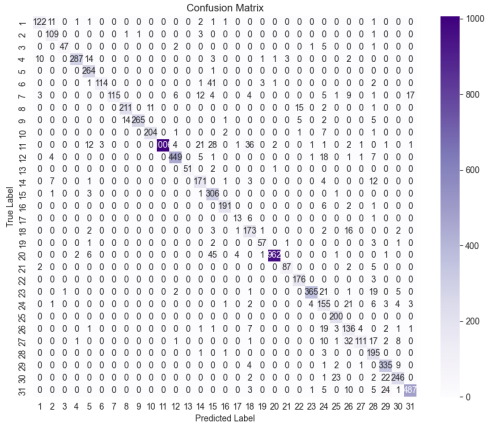

Fig. 58: GoogleNet Confusion Matrix for Batch Size 1, Learning Rate 0.001, Epoch 10

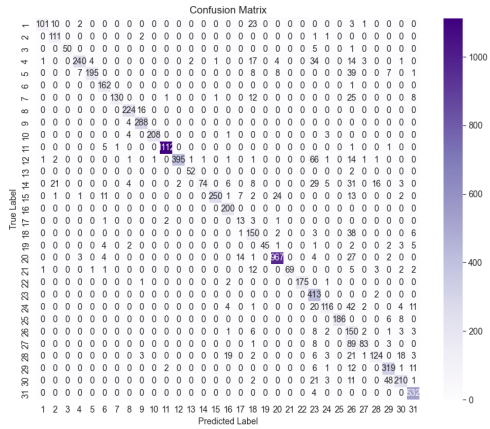

Fig. 59: GoogleNet Confusion Matrix for Batch Size 1, Learning Rate 0.001, Epoch 15

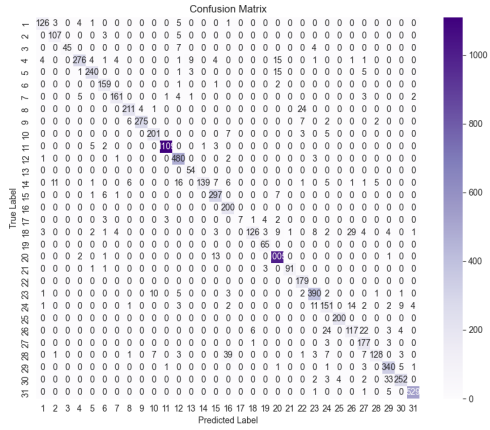

Fig. 60: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.001, Epoch 5

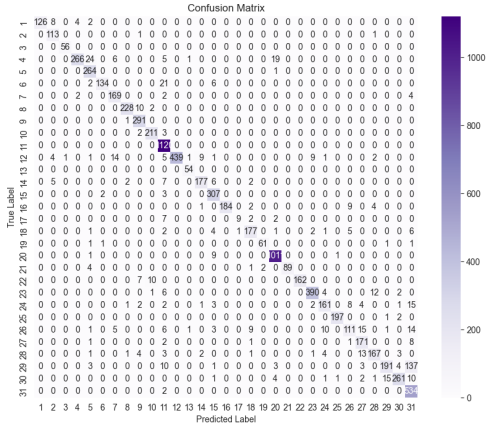

Fig. 61: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.001, Epoch 10

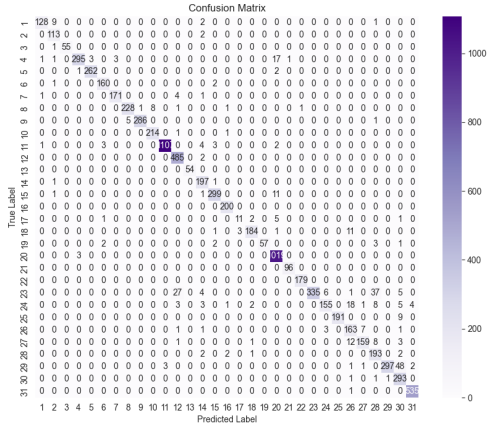

Fig. 63: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.001, Epoch 20

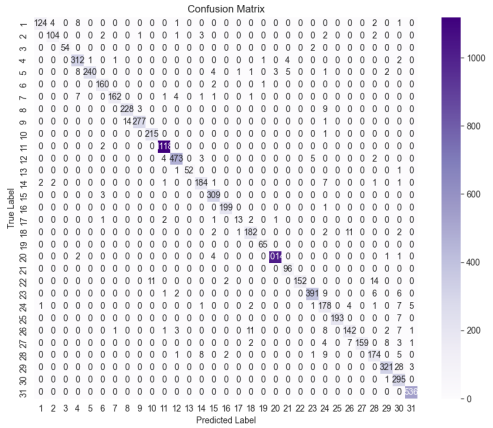

Fig. 62: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.001, Epoch 15

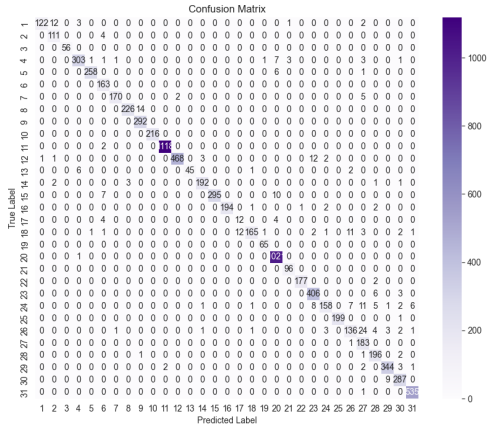

Fig. 64: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0005, Epoch 5

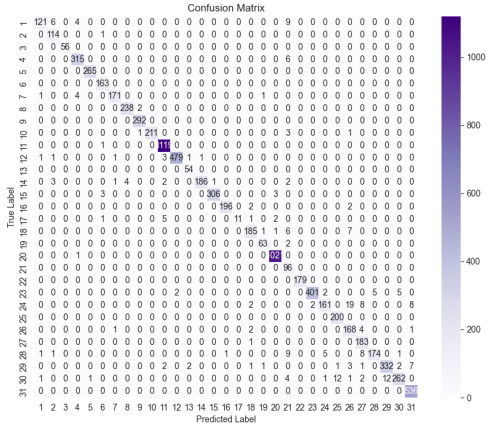

Fig. 65: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0005, Epoch 10

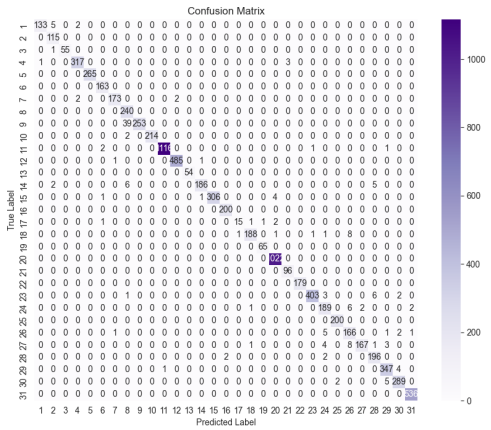

Fig. 66: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0005, Epoch 15

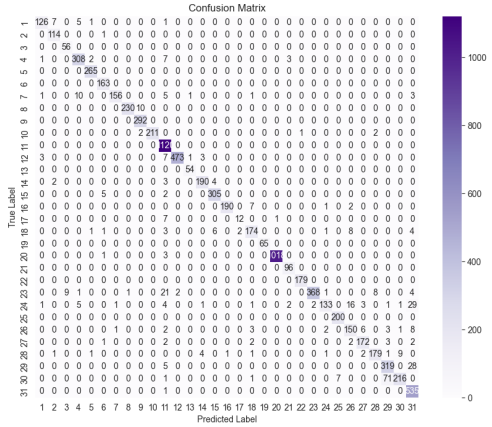

Fig. 67: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0005, Epoch 20

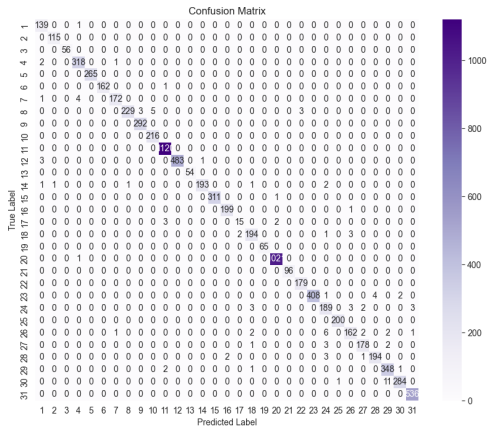

Fig. 68: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0001, Epoch 5

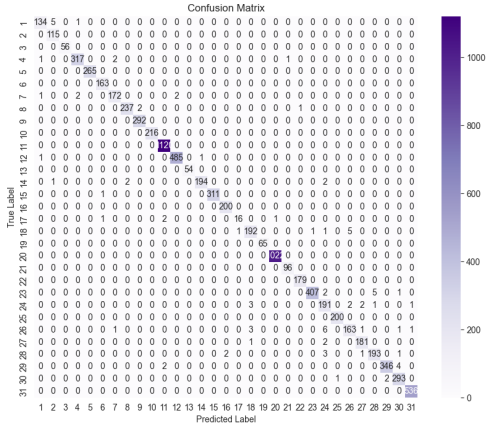

Fig. 69: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0001, Epoch 10

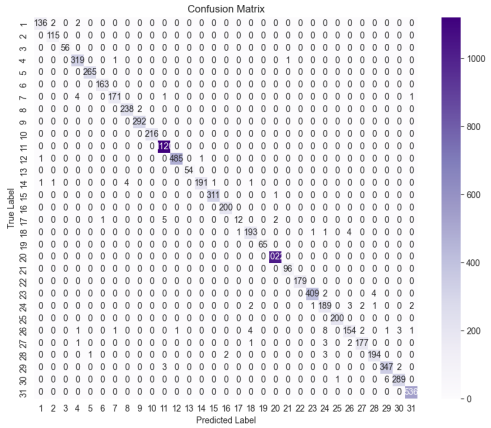

Fig. 70: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0001, Epoch 15

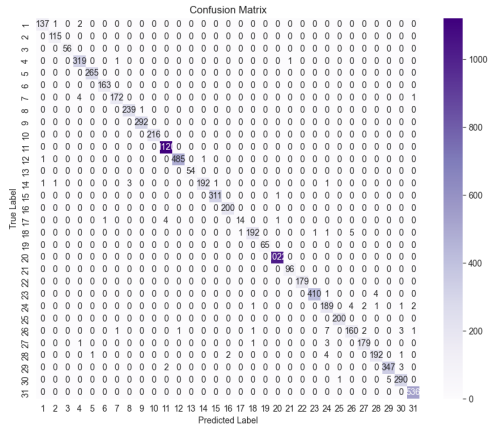

Fig. 71: GoogleNet Confusion Matrix for Batch Size 32, Learning Rate 0.0001, Epoch 20

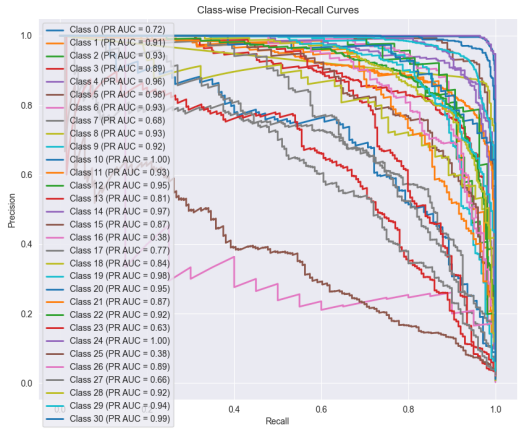

Fig. 72: GoogleNet Precision-Recall Curve for Batch Size 1, Learning Rate 0.001, Epoch 5

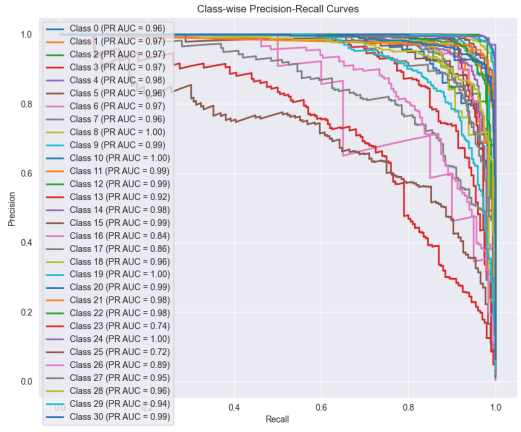

Fig. 73: GoogleNet Precision-Recall Curve for Batch Size 1, Learning Rate 0.001, Epoch 10

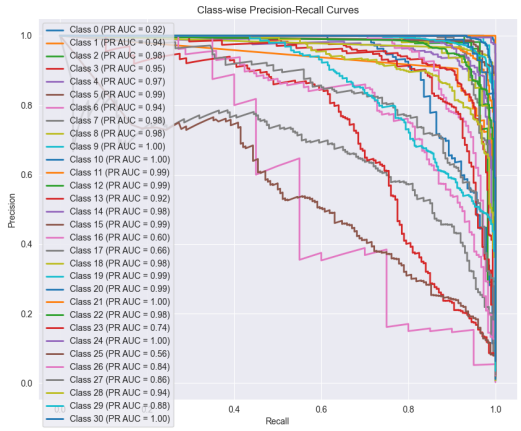

Fig. 74: GoogleNet Precision-Recall Curve for Batch Size 1, Learning Rate 0.001, Epoch 15

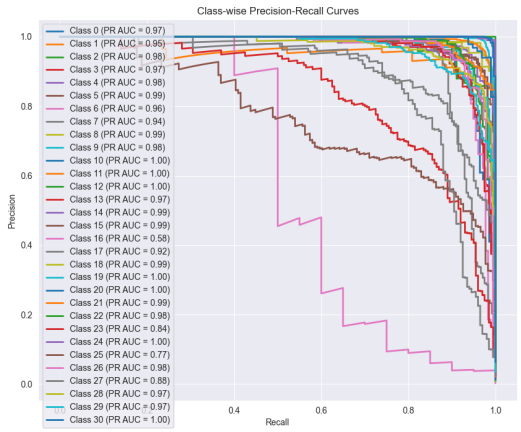

Fig. 75: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.001, Epoch 5

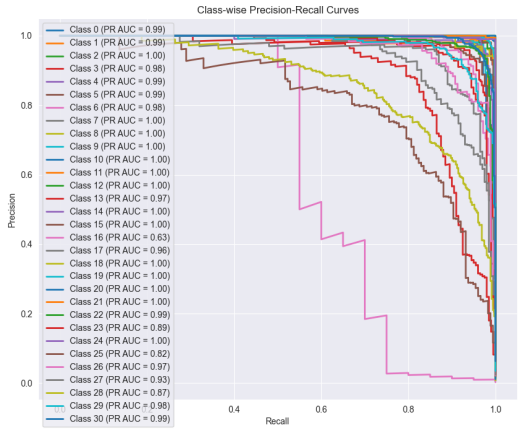

Fig. 76: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.001, Epoch 10

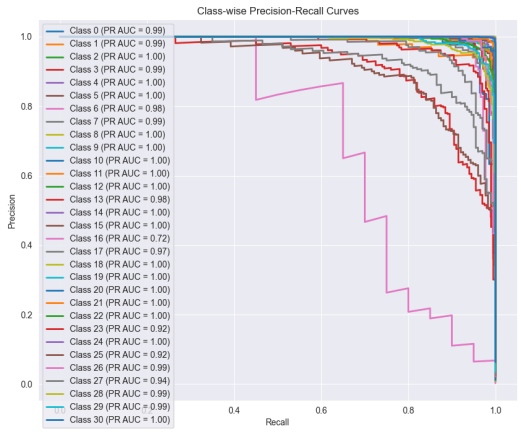

Fig. 77: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.001, Epoch 15

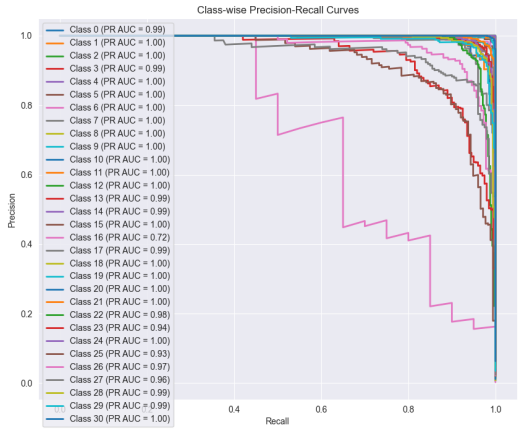

Fig. 78: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.001, Epoch 20

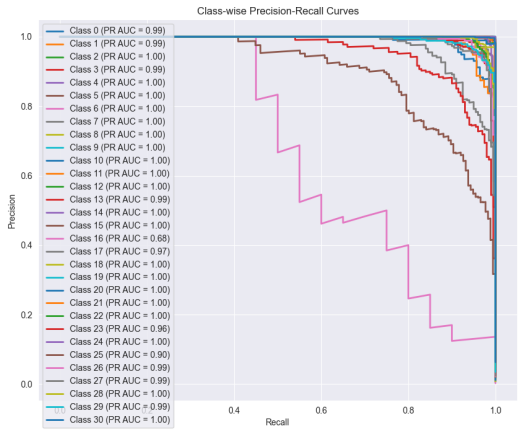

Fig. 79: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0005, Epoch 5

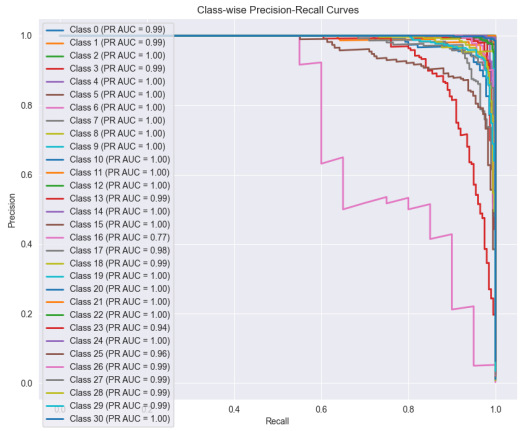

Fig. 80: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0005, Epoch 10

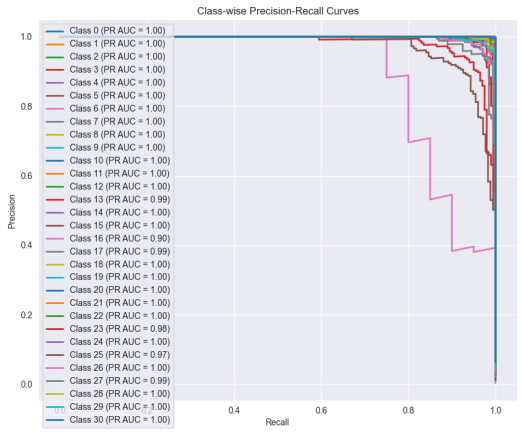

Fig. 81: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0005, Epoch 15

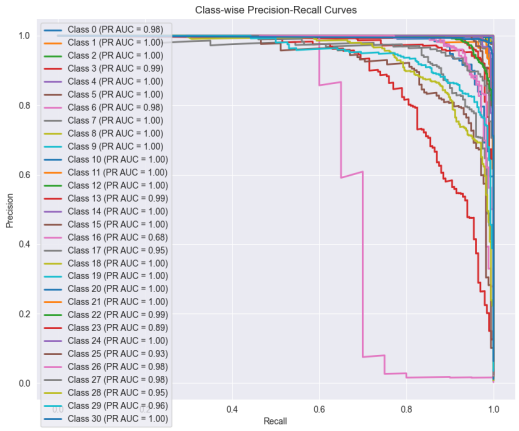

Fig. 82: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0005, Epoch 20

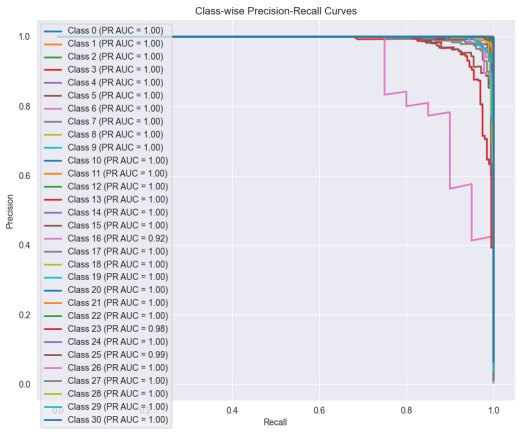

Fig. 83: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0001, Epoch 5

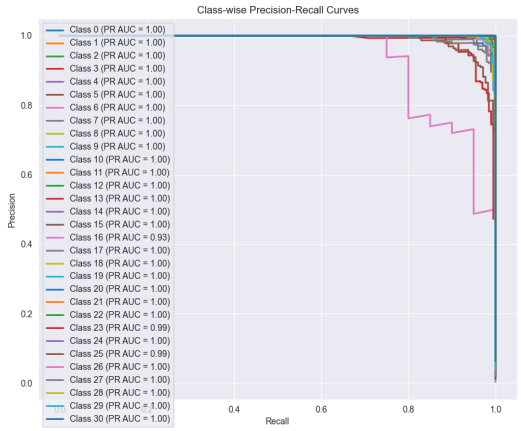

Fig. 84: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0001, Epoch 10

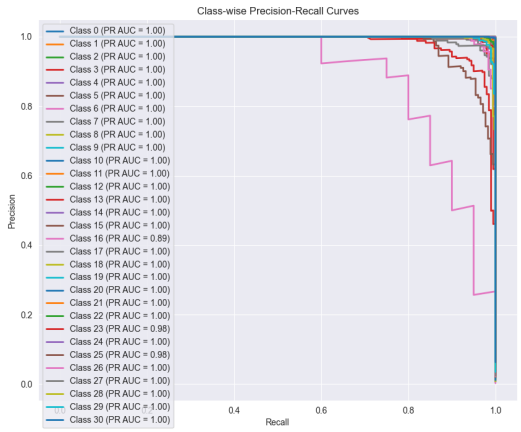

Fig. 85: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0001, Epoch 15

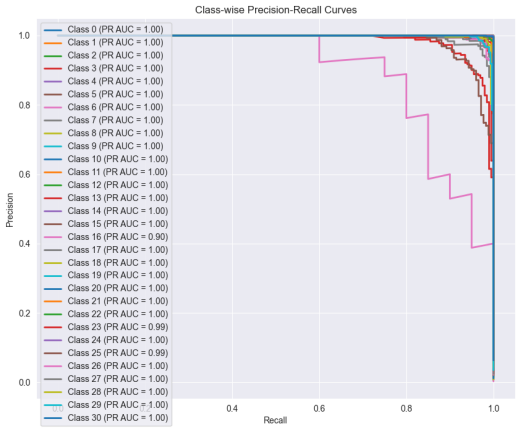

Fig. 86: GoogleNet Precision-Recall Curve for Batch Size 32, Learning Rate 0.0001, Epoch 20

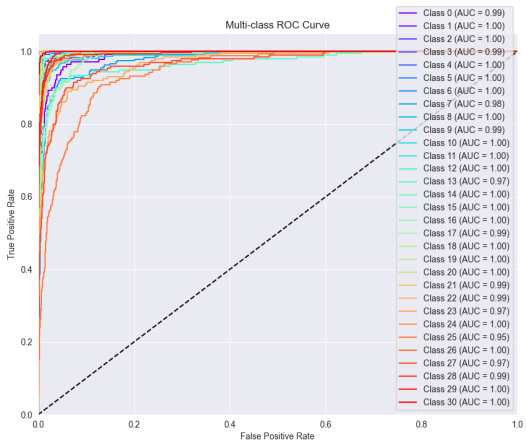

Fig. 87: GoogleNet ROC Curve for Batch Size 1, Learning Rate 0.001, Epoch 5

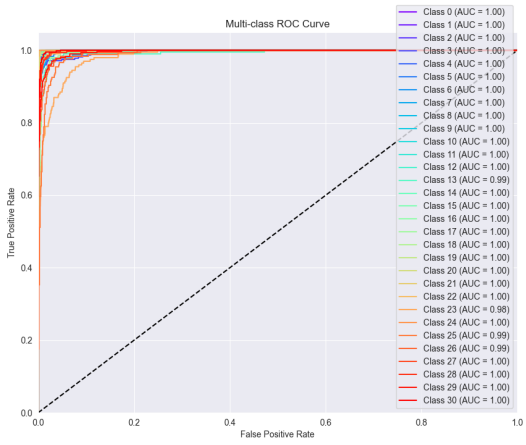

Fig. 88: GoogleNet ROC Curve for Batch Size 1, Learning Rate 0.001, Epoch 10

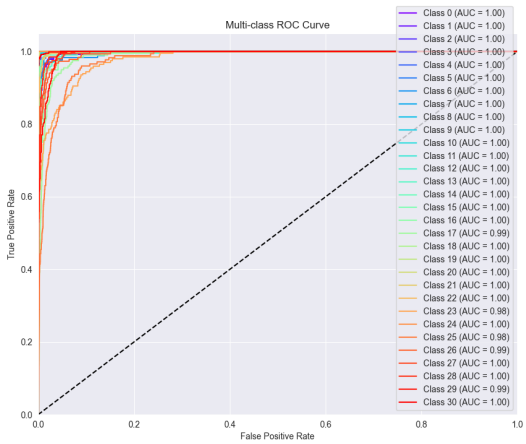

Fig. 89: GoogleNet ROC Curve for Batch Size 1, Learning Rate 0.001, Epoch 15

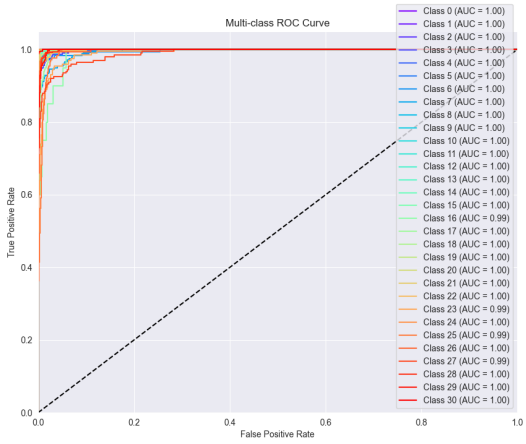

Fig. 90: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.001, Epoch 5

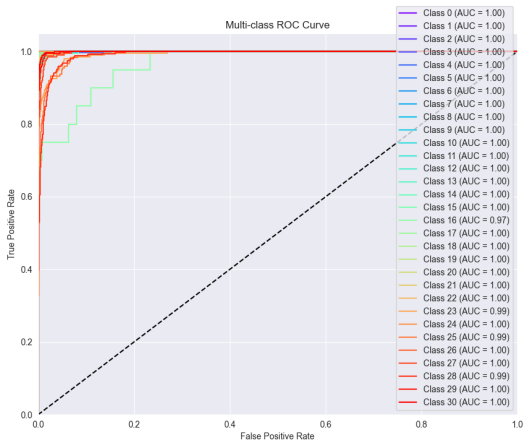

Fig. 91: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.001, Epoch 10

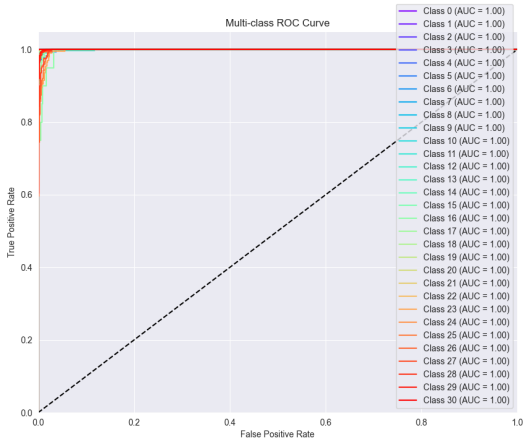

Fig. 92: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.001, Epoch 15

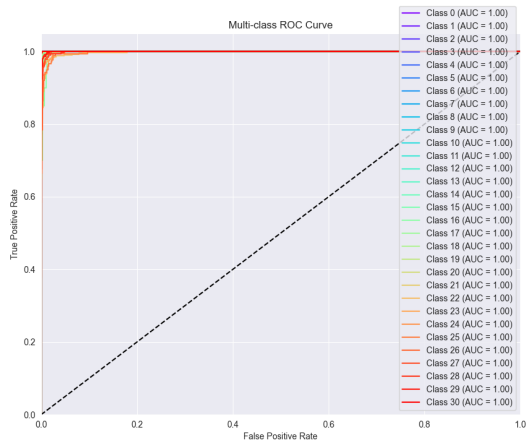

Fig. 93: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.001, Epoch 20

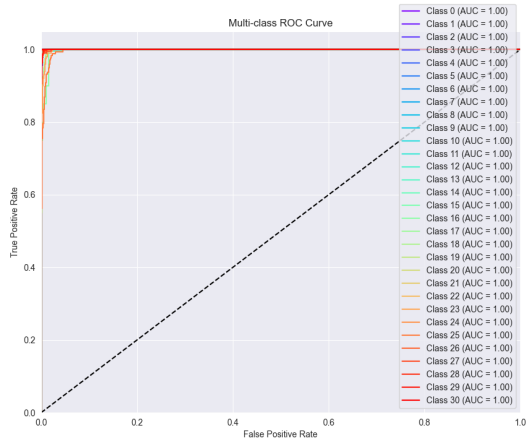

Fig. 94: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0005, Epoch 5

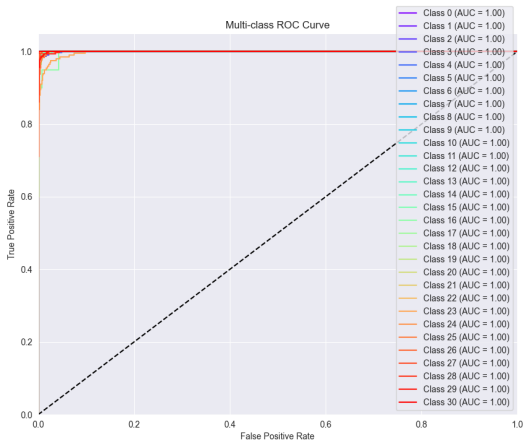

Fig. 95: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0005, Epoch 10

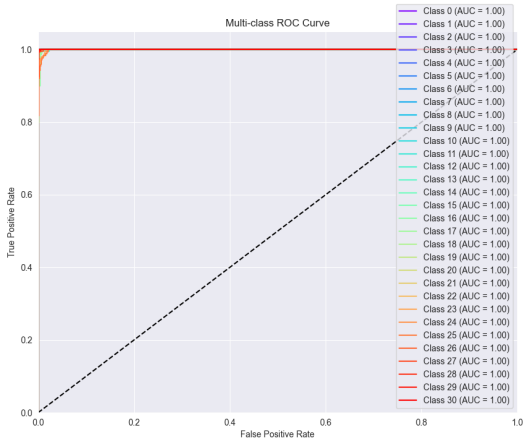

Fig. 96: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0005, Epoch 15

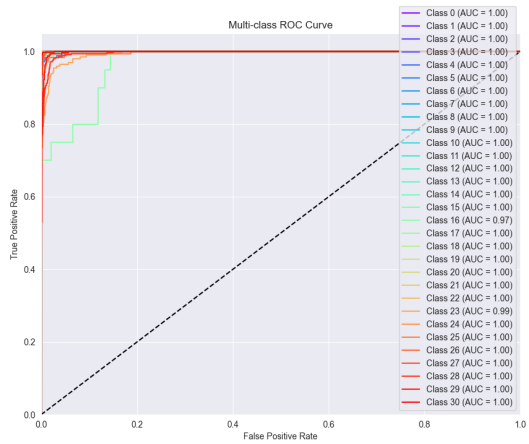

Fig. 97: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0005, Epoch 20

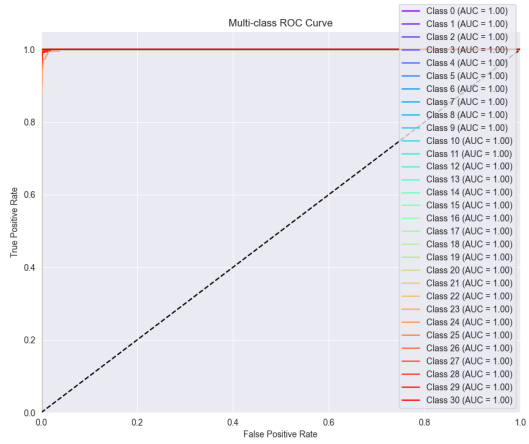

Fig. 98: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0001, Epoch 5

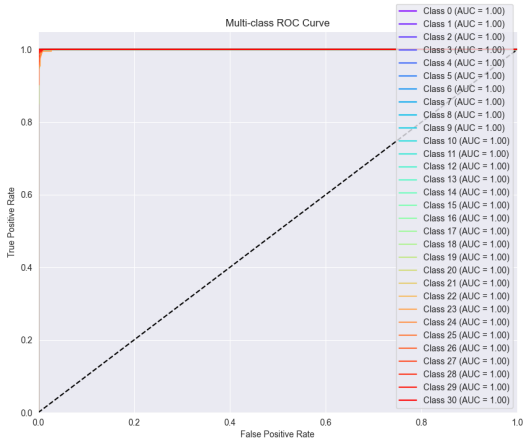

Fig. 99: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0001, Epoch 10

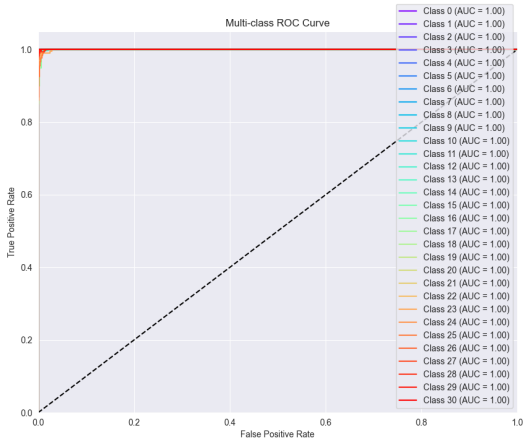

Fig. 100: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0001, Epoch 15

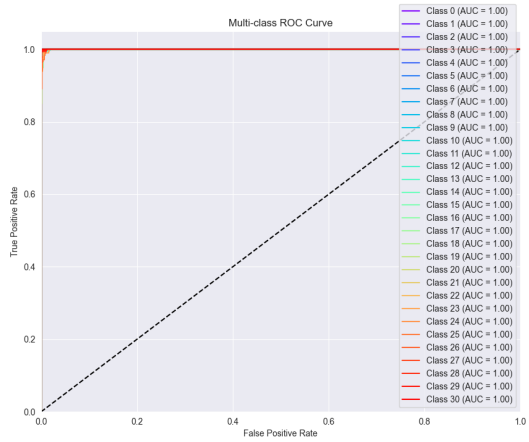

Fig. 101: GoogleNet ROC Curve for Batch Size 32, Learning Rate 0.0001, Epoch 20