Project Report

Xiong Yang 1004876
Gao Fancheng 1004879

April 16, 2023

1 Introduction

Our project is to discover three kind of retinopathy from retinal OCT images (optical coherence tomography). Our task is a classification task. Together with normal retinal OCT images there are 4 classes:

- CNV (Choroidal neovascularization): Abnormal blood vessels that grow beneath the retina in the macular area, which is one of the most common forms of age-related macular degeneration (AMD).
- DME (Diabetic macular edema): A common form of diabetic retinopathy, characterized by fluid accumulation in the macula, which can lead to vision loss.
- DRUSEN: Small yellow deposits that form under the retina, which are an early pathological feature of age-related macular degeneration.
- NORMAL: A normal retinal OCT image without any abnormal features such as CNV, DME, or DRUSEN.

We used following packages in our project:

- Pytorch for model construction, training and evaluation.
- Matplotlib for data and result visualization
- torchsummary for model visualization
- Sklearn to help build confusion matrix

The coding can be found in our Github repository:
https://github.com/Meltryllis628/OCT-images-classification

2 Input and Outputs

The input is rgb images of different scales and the output is a label (CNV, DME, DRUSEN or NORMAL). We are using the Retinal OCT Images (optical coherence tomography) dataset[1] on Kaggle.

3 Data Processing

The original validation set contains only 32 data points and is too small for validation, so we did train-valid split on the train set on the original train set. We transformed the original figures to gray scale pictures and resize them to $64 * 64$ or $128 * 128$ for training.The processed images is shown in Figure 2. The distribution of the 4 classes is shown in Figure 3

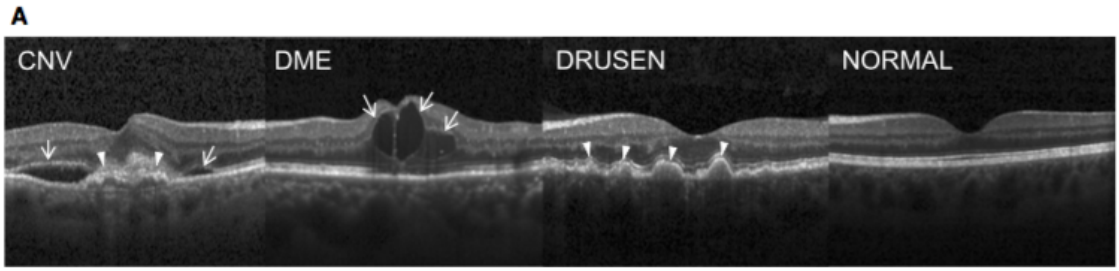

Figure 1: samples of different classes and major differences between them.

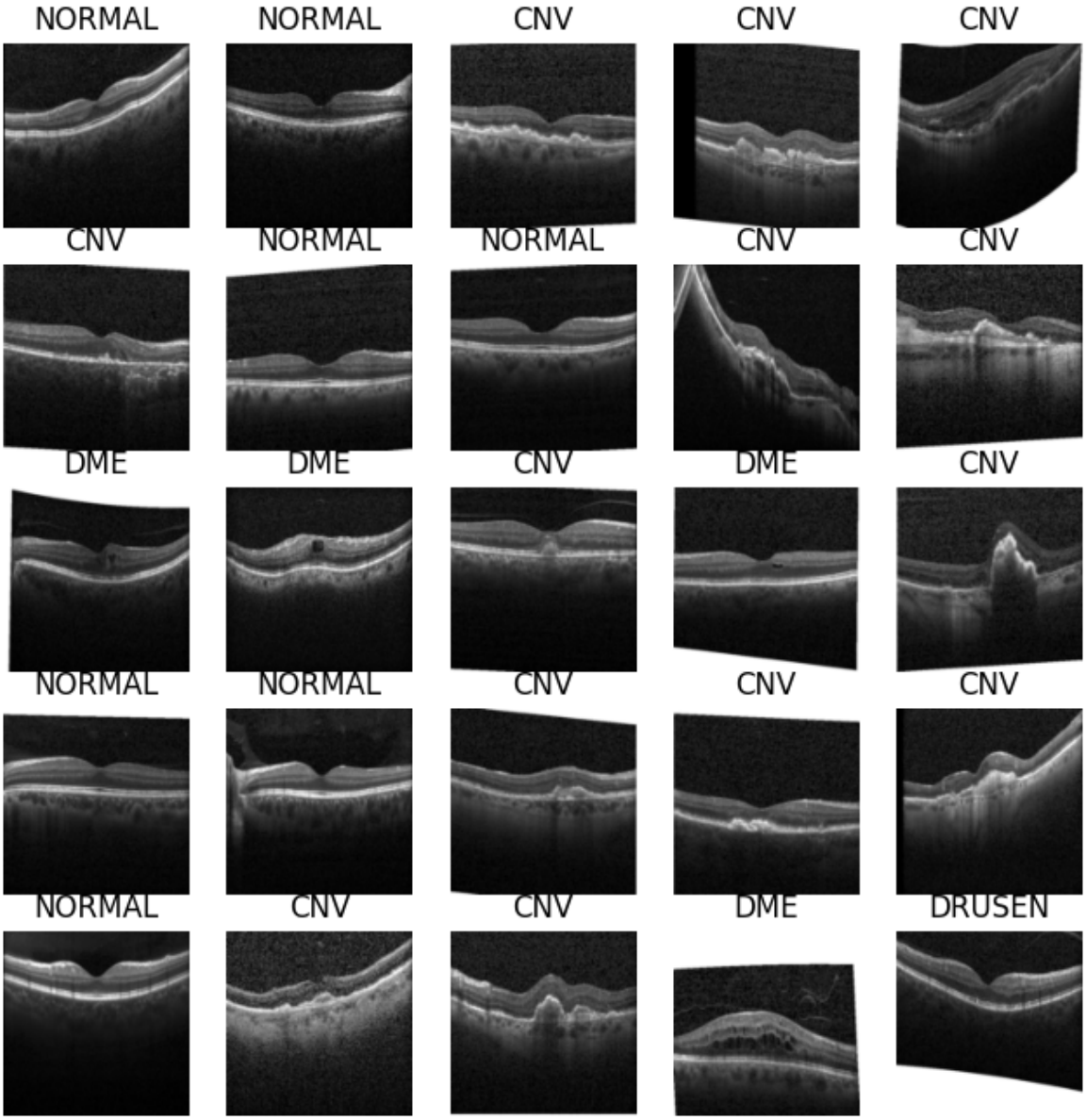

Figure 2: processed figures.

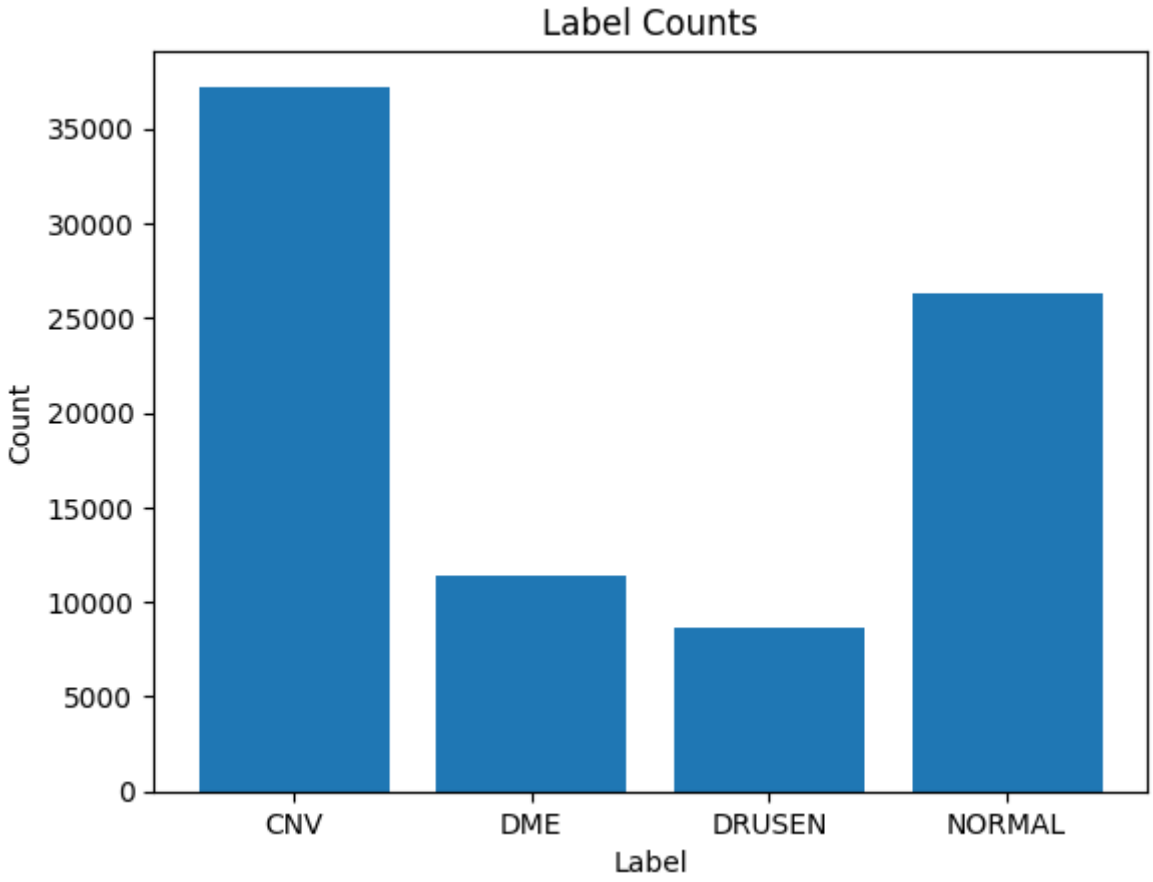

Figure 3: label counts.

4 Model Architecture

4.1 First trail

The model we used is a simple Convolutional Neural Network which consists of two convolutional layers and two linear layers (Figure 2). The first convolutional layer has 1 input channel and 32 output channels. The second convolutional layer has 32 input channels and 64 output channels. Both convolutional layers have kernel size equal to 3, stride equal to 1, and paddings equal to 1. The first linear layer has an input size of $64 * image\_size * image\_size$ (262144 for $image\_size = 64$) and an output size of 128. The second linear layer has an input size of 128 and an output size of 4 which is the output of our model. The activation function we used is ReLu.

4.2 Improvement

The early model tends to suffer from overfitting. In order to improve the generalization of the model, we tried doing some hyperparameter tuning, adding Batch-Normalization layers and dropout layers. We added the BN layer before the activation of each convolutional layer. The momentum we set for the BN layer is 0.99. We also added the dropout layer after the activation of each convolutional layer. The dropout ratio we set is 0.25. However, the result appears to be only slightly better than the early model and still suffers from overfitting. We think this could be because although the structure of the model is simple, there are too many trainable parameters in our model (33,574,084 params). We then decided to try using another structure.

```
================================================================
        Layer (type)               Output Shape          Param #
================================================================
            Conv2d-1          [-1, 32, 64, 64]             320
       BatchNorm2d-2          [-1, 32, 64, 64]              64
         Dropout2d-3          [-1, 32, 64, 64]               0
            Conv2d-4          [-1, 64, 64, 64]          18,496
       BatchNorm2d-5          [-1, 64, 64, 64]             128
         Dropout2d-6          [-1, 64, 64, 64]               0
            Linear-7                [-1, 128]      33,554,560
            Linear-8                  [-1, 4]             516
================================================================
Total params: 33,574,084
Trainable params: 33,574,084
Non-trainable params: 0
================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 9.00
Params size (MB): 128.07
Estimated Total Size (MB): 137.09
================================================================
```

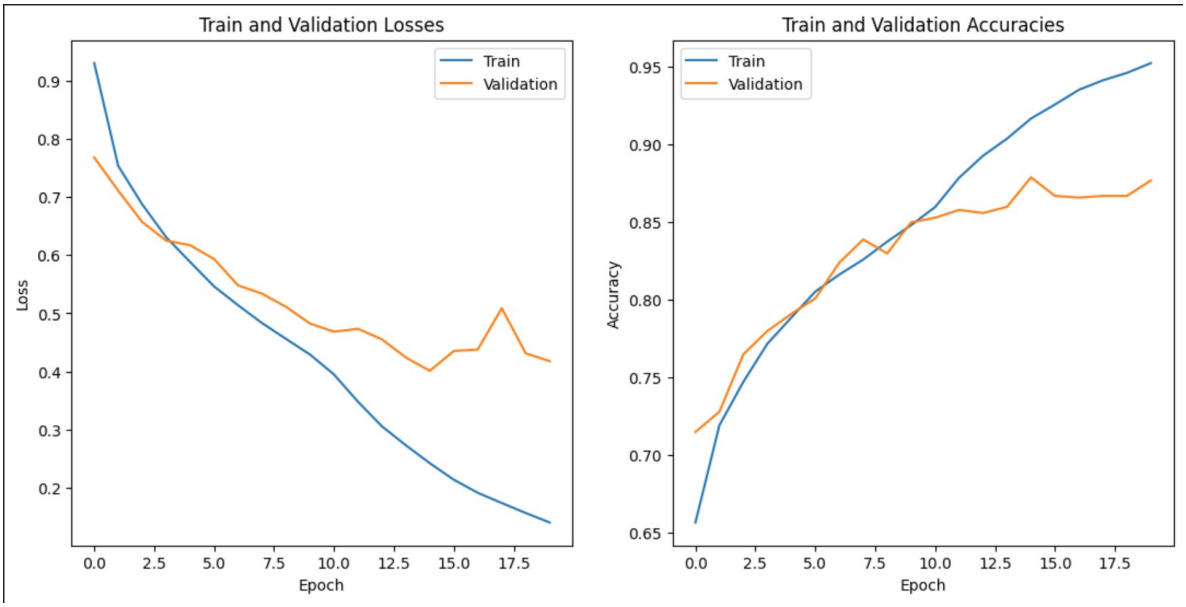

Figure 4: loss and accuracy plot

4.3 Residual Network

The new structure we used is the Resnet18[2] structure. The summary of model is shown below. Although it has much more complex structure than the original one, it has much less trainable parameters.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
|-Conv2d: 1-1                            [-1, 64, 64, 64]          3,136
|-BatchNorm2d: 1-2                       [-1, 64, 64, 64]          128
|-ReLU: 1-3                              [-1, 64, 64, 64]          --
|-Sequential: 1-4                        [-1, 64, 64, 64]          --
|    |-ResidualBlock: 2-1                [-1, 64, 64, 64]          --
|    |    |-Conv2d: 3-1                  [-1, 64, 64, 64]          36,864
|    |    |-BatchNorm2d: 3-2             [-1, 64, 64, 64]          128
|    |    |-ReLU: 3-3                    [-1, 64, 64, 64]          --
|    |    |-Conv2d: 3-4                  [-1, 64, 64, 64]          36,864
|    |    |-BatchNorm2d: 3-5             [-1, 64, 64, 64]          128
|    |    |-ReLU: 3-6                    [-1, 64, 64, 64]          --
|    |-ResidualBlock: 2-2                [-1, 64, 64, 64]          --
|    |    |-Conv2d: 3-7                  [-1, 64, 64, 64]          36,864
|    |    |-BatchNorm2d: 3-8             [-1, 64, 64, 64]          128
|    |    |-ReLU: 3-9                    [-1, 64, 64, 64]          --
|    |    |-Conv2d: 3-10                 [-1, 64, 64, 64]          36,864
|    |    |-BatchNorm2d: 3-11            [-1, 64, 64, 64]          128
|    |    |-ReLU: 3-12                   [-1, 64, 64, 64]          --
|-Sequential: 1-5                        [-1, 128, 32, 32]         --
|    |-ResidualBlock: 2-3                [-1, 128, 32, 32]         --
|    |    |-Conv2d: 3-13                 [-1, 128, 32, 32]         73,728
|    |    |-BatchNorm2d: 3-14            [-1, 128, 32, 32]         256
|    |    |-ReLU: 3-15                   [-1, 128, 32, 32]         --
|    |    |-Conv2d: 3-16                 [-1, 128, 32, 32]         147,456
|    |    |-BatchNorm2d: 3-17            [-1, 128, 32, 32]         256
|    |    |-Sequential: 3-18             [-1, 128, 32, 32]         8,448
|    |    |-ReLU: 3-19                   [-1, 128, 32, 32]         --
|    |-ResidualBlock: 2-4                [-1, 128, 32, 32]         --
|    |    |-Conv2d: 3-20                 [-1, 128, 32, 32]         147,456
```

```
|    |    |-BatchNorm2d: 3-21            [-1, 128, 32, 32]         256
|    |    |-ReLU: 3-22                   [-1, 128, 32, 32]         --
|    |    |-Conv2d: 3-23                 [-1, 128, 32, 32]         147,456
|    |    |-BatchNorm2d: 3-24            [-1, 128, 32, 32]         256
|    |    |-ReLU: 3-25                   [-1, 128, 32, 32]         --
|-Sequential: 1-6                        [-1, 256, 16, 16]         --
|    |-ResidualBlock: 2-5                [-1, 256, 16, 16]         --
|    |    |-Conv2d: 3-26                 [-1, 256, 16, 16]         294,912
|    |    |-BatchNorm2d: 3-27            [-1, 256, 16, 16]         512
|    |    |-ReLU: 3-28                   [-1, 256, 16, 16]         --
|    |    |-Conv2d: 3-29                 [-1, 256, 16, 16]         589,824
|    |    |-BatchNorm2d: 3-30            [-1, 256, 16, 16]         512
|    |    |-Sequential: 3-31             [-1, 256, 16, 16]         33,280
|    |    |-ReLU: 3-32                   [-1, 256, 16, 16]         --
|    |-ResidualBlock: 2-6                [-1, 256, 16, 16]         --
|    |    |-Conv2d: 3-33                 [-1, 256, 16, 16]         589,824
|    |    |-BatchNorm2d: 3-34            [-1, 256, 16, 16]         512
|    |    |-ReLU: 3-35                   [-1, 256, 16, 16]         --
|    |    |-Conv2d: 3-36                 [-1, 256, 16, 16]         589,824
|    |    |-BatchNorm2d: 3-37            [-1, 256, 16, 16]         512
|    |    |-ReLU: 3-38                   [-1, 256, 16, 16]         --
|-Sequential: 1-7                        [-1, 512, 8, 8]           --
|    |-ResidualBlock: 2-7                [-1, 512, 8, 8]           --
|    |    |-Conv2d: 3-39                 [-1, 512, 8, 8]           1,179,648
|    |    |-BatchNorm2d: 3-40            [-1, 512, 8, 8]           1,024
|    |    |-ReLU: 3-41                   [-1, 512, 8, 8]           --
|    |    |-Conv2d: 3-42                 [-1, 512, 8, 8]           2,359,296
|    |    |-BatchNorm2d: 3-43            [-1, 512, 8, 8]           1,024
|    |    |-Sequential: 3-44             [-1, 512, 8, 8]           132,096
|    |    |-ReLU: 3-45                   [-1, 512, 8, 8]           --
|    |-ResidualBlock: 2-8                [-1, 512, 8, 8]           --
|    |    |-Conv2d: 3-46                 [-1, 512, 8, 8]           2,359,296
|    |    |-BatchNorm2d: 3-47            [-1, 512, 8, 8]           1,024
|    |    |-ReLU: 3-48                   [-1, 512, 8, 8]           --
|    |    |-Conv2d: 3-49                 [-1, 512, 8, 8]           2,359,296
|    |    |-BatchNorm2d: 3-50            [-1, 512, 8, 8]           1,024
|    |    |-ReLU: 3-51                   [-1, 512, 8, 8]           --
|-AdaptiveAvgPool2d: 1-8                 [-1, 512, 1, 1]           --
|-Linear: 1-9                            [-1, 4]                   2,052
==========================================================================
Total params: 11,172,292
Trainable params: 11,172,292
Non-trainable params: 0
Total mult-adds (G): 2.25
==========================================================================
Input size (MB): 0.06
Forward/backward pass size (MB): 37.50
Params size (MB): 42.62
Estimated Total Size (MB): 80.18
==========================================================================
```

The training takes 113m 2.4s for 32 epoches.

4.4 Further Improvement

Inspired by the structure of Resnet18, we tried adding an average pooling layer to the original model to reduce the number of parameters that require training. The pooling layer is added before the first linear layer. After adding the pooling layer, the original CNN model is much simpler and is expected to generalize better.

```
==========================================================
        Layer (type)               Output Shape         Param #
```

```
================================================================
            Conv2d-1          [-1, 32, 64, 64]             320
       BatchNorm2d-2          [-1, 32, 64, 64]              64
         Dropout2d-3          [-1, 32, 64, 64]               0
            Conv2d-4          [-1, 64, 64, 64]          18,496
       BatchNorm2d-5          [-1, 64, 64, 64]             128
         Dropout2d-6          [-1, 64, 64, 64]               0
 AdaptiveAvgPool2d-7            [-1, 64, 4, 4]               0
            Linear-8                 [-1, 128]         131,200
            Linear-9                   [-1, 4]             516
================================================================
Total params: 150,724
Trainable params: 150,724
Non-trainable params: 0
================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 9.01
Params size (MB): 0.57
Estimated Total Size (MB): 9.60
================================================================
```

5 Training Process and Training Results

The losses and accuracies on the train set and validation set for the original CNN model without pooling layer is shown in the plot Figure 5. And the losses and accuracies on the train set and validation set for the Resnet18 model we trained is shown in the plot Figure 6. We can see from the plot that the Resnet model performs much better than the original CNN model. However, it still shows a trend of getting overfitting if the training keeps going. This may be because of the model is too complex for the problem we are trying to solve and it may study the noises and unimportant features in the input data.

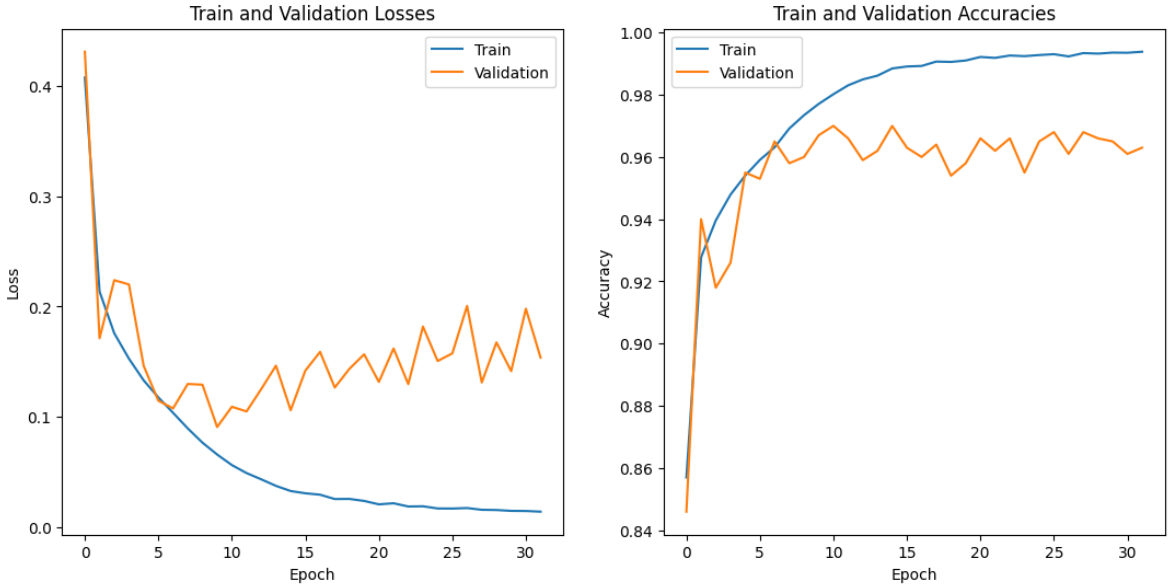

Figure 5: loss and accuracy plot for Resnet

We get a good result on the test set loading the Resnet model on epoch 7. The test loss is 0.0085 and test accuracy is 0.9990.

The losses and accuracies on the train set and validation set for the original CNN model with pooling layer is shown in the plot Figure 7. We can observe from the result that, although the CNN

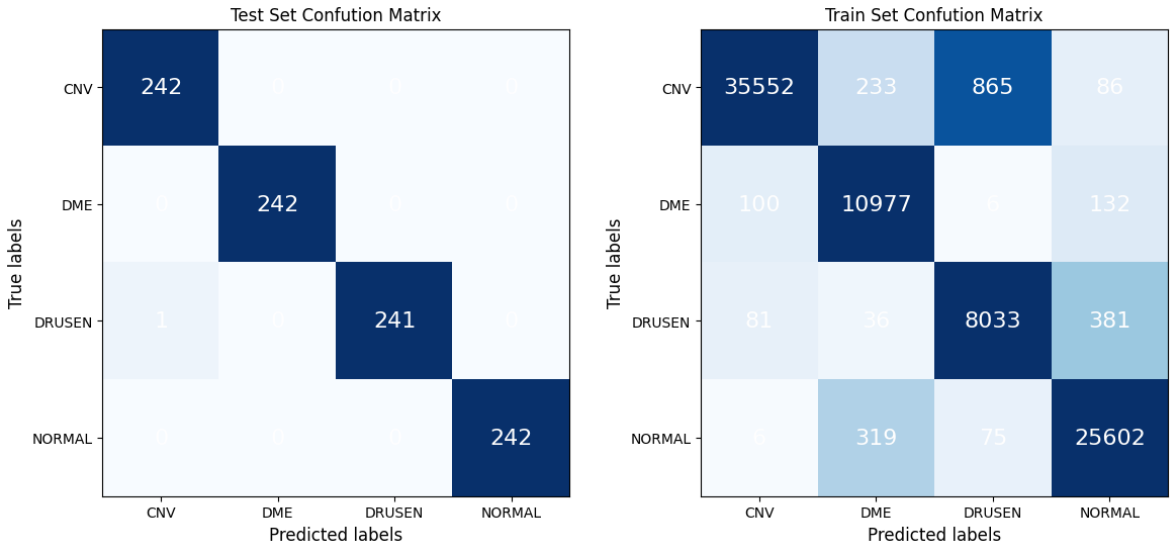

Figure 6: confusion matrix for Resnet

model with a pooling layer isn't able to reach as high accuracy and low loss as the Resnet model, it is able to generalize better.

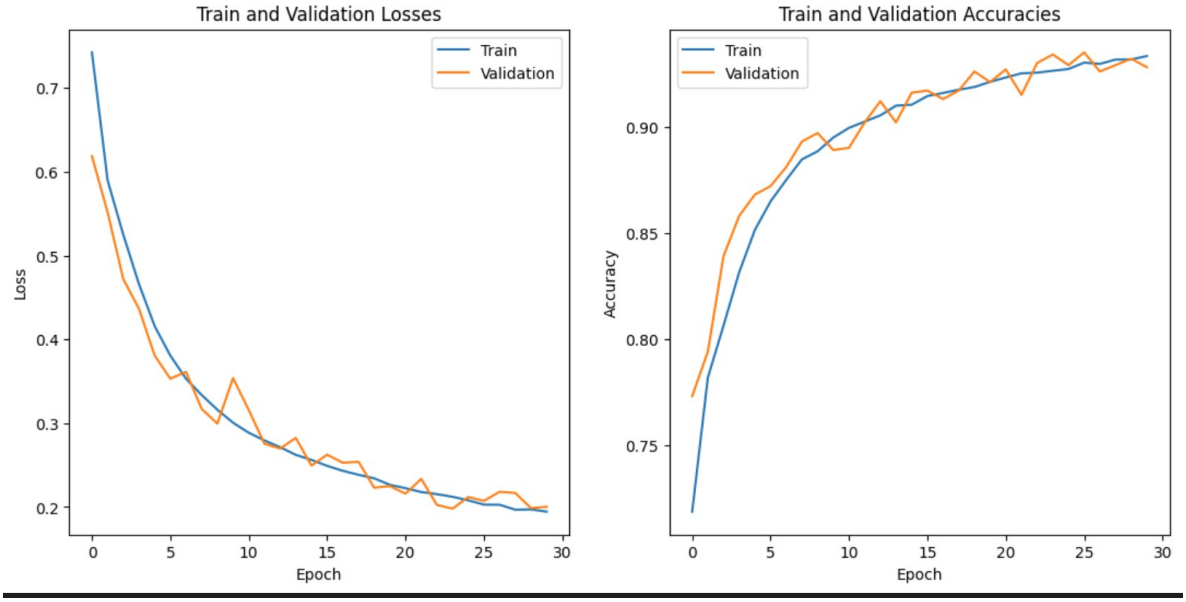

Figure 7: loss and accuracy plot

We get a good result on the test set loading the CNN model on epoch 29. The test loss is 0.0629 and test accuracy is 0.9855.

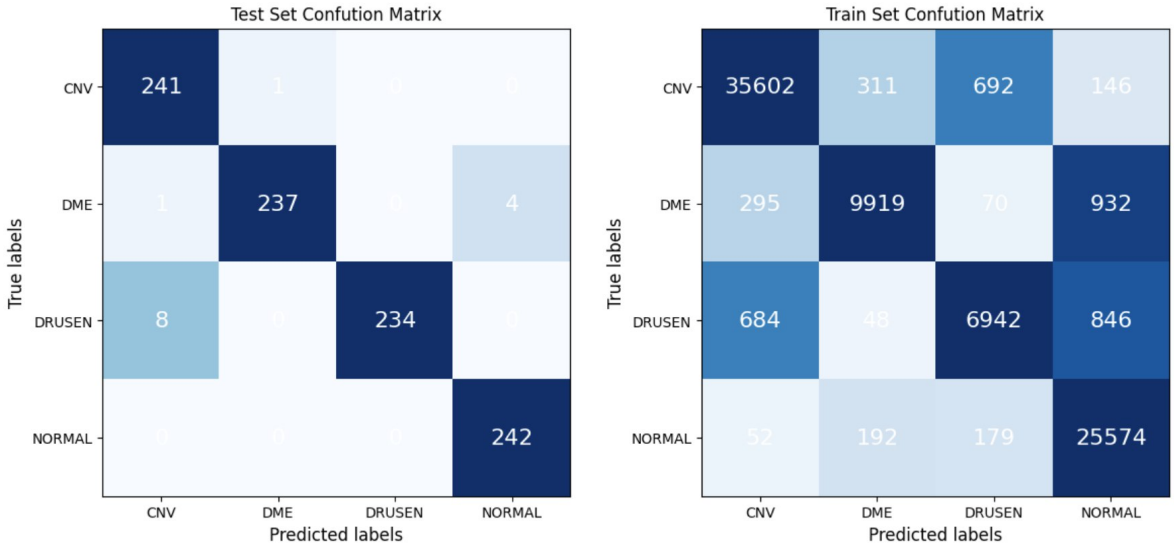

Figure 8: confusion matrix

References

[1] Daniel Kermany, Kang Zhang, and Michael Goldbaum. Labeled optical coherence tomography (oct) and chest x-ray images for classification. Mendeley Data, 2018.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.