CSCE 421 Final Project: AUC Maximization by Deep Learning for Imbalanced Medical Image Classification

Jonathan Zhao, David-Tyler Ighedosa , Kyumin Lee

Abstract

During the course of this class we've used different techniques and methods in the hopes of better optimizing machine learning models. In this project we combine these techniques in order to maximize the Area under the Curve which measures the performance of the machine learning model. These techniques were used on the Imbalanced Medical Image Classification that was provided.

Introduction

Using ResNet18, LibAUC, MedMNIST, and PneumoniaMNIST different techniques were implemented in order to optimize our models. The three main methodologies we used were data augmentation, adjustment of the learning rate, controlling overfitting, and changing the optimizer. With these techniques we trained a deep learning model to maximize the area under the curve for imbalanced medical images.

Methods

In this study, we explore several methods for enhancing the performance of machine learning models:

Data Augmentation
We performed many data augmentation strategies before including L1 and L2 regularization. Data has been provided with no augmentation, some augmentation, and the final augmentation we tried. The first 10 epochs and their respective AUC values were provided that demonstrate the effectiveness of the data augmentation technique used. From the following data we can see that the AUC value does not increase with any change, some of the changes we made provided

lower AUC values. Changes that were performed include flipping, random rotation, color jitter, and blur.

Adjusting the Learning Rate
The learning rate affects how fast the model converges during training. Having a learning rate that's too low will slow down the convergence, while having a learning rate that's too high will cause the model to overshoot, and become unstable. We adjusted the learning rate from the default of 0.1 to 0.2, which helped speed up the convergence at the lower epochs, while also not being too high that it will cause the model to overshoot.

Controlling Overfitting

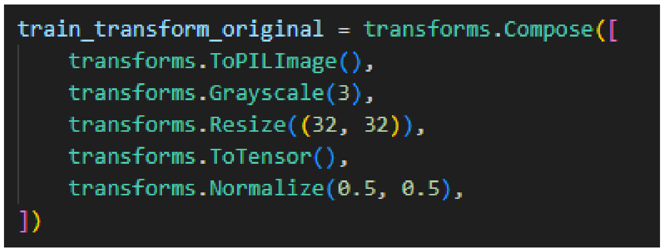
- L1 Regularization
    - L1 regularization adds a penalty proportional to the absolute value of the coefficients to the loss function. It encourages sparsity in the model, meaning it tends to push the weights of less important features to exactly zero. L1 Regularization can simplify the model and make it more interpretable by selecting only the most important features. It can also help prevent overfitting by reducing the model's complexity.
- L2 Regularization
    - L2 Regularization, also known as weight decay, adds a penalty proportional to the square of the coefficients to the loss function. It tends to shrink the weights of the model overall without necessarily setting them to zero. L2 Regularization is used to prevent overfitting by discouraging overly complex models. It smooths the model's learned parameters and can improve generalization to unseen data.

Changing Optimizers
We tried changing the optimizer to using the Adam optimizer. The Adam optimizer is an adaptive learning rate optimization algorithm that combines the advantages of two other popular optimization algorithms, AdaGrad and RMSProp. Going into it we did not know how much it would affect the results but from our experiments we could see that it had a decent impact compared to other optimization methods.

Experiments and Data

No Data Augmentation without Optimizers:

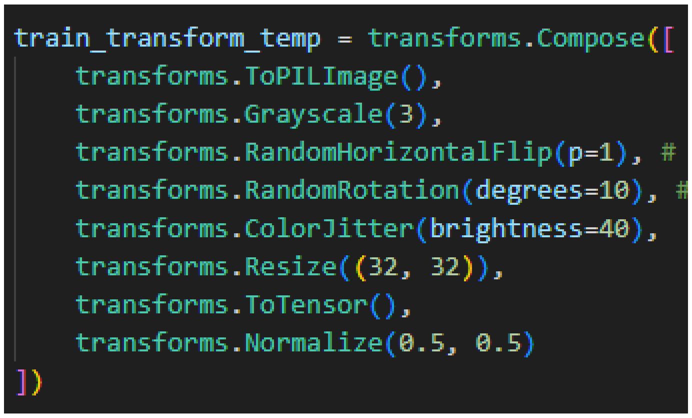
```
Epoch: 0; Test AUC: 0.38784461152882205
Epoch: 1; Test AUC: 0.5352965747702589
Epoch: 2; Test AUC: 0.7717209690893903
Epoch: 3; Test AUC: 0.47556390977443613
Epoch: 4; Test AUC: 0.7017543859649124
Epoch: 5; Test AUC: 0.5639097744360902
Epoch: 6; Test AUC: 0.7758980785296575
Epoch: 7; Test AUC: 0.7926065162907268
Epoch: 8; Test AUC: 0.7731829573934836
Epoch: 9; Test AUC: 0.7909356725146199
```



```
train_transform_original = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Grayscale(3),
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize(0.5, 0.5),
])
```

Some Data Augmentation without Optimizers:

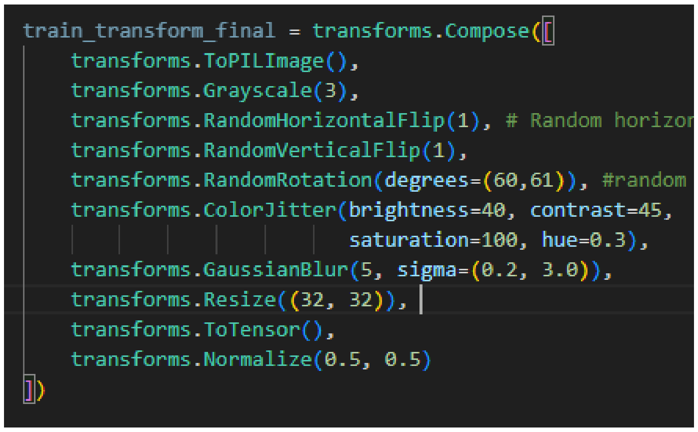
```
Epoch: 0; Test AUC: 0.3504594820384294
Epoch: 1; Test AUC: 0.39598997493734334
Epoch: 2; Test AUC: 0.3834586466165414
Epoch: 3; Test AUC: 0.38345864661654133
Epoch: 4; Test AUC: 0.31370091896407687
Epoch: 5; Test AUC: 0.37259816207184626
Epoch: 6; Test AUC: 0.5348788638262322
Epoch: 7; Test AUC: 0.7224310776942356
Epoch: 8; Test AUC: 0.7428989139515456
Epoch: 9; Test AUC: 0.6917293233082707
```



```
train_transform_temp = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Grayscale(3),
    transforms.RandomHorizontalFlip(p=1), #
    transforms.RandomRotation(degrees=10), #
    transforms.ColorJitter(brightness=40),
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize(0.5, 0.5)
])
```

Final data Augmentation without Optimizers:

```
Epoch: 0; Test AUC: 0.3767752715121136
Epoch: 1; Test AUC: 0.5421888053467001
Epoch: 2; Test AUC: 0.5954469507101086
Epoch: 3; Test AUC: 0.6351294903926483
Epoch: 4; Test AUC: 0.7007101086048455
Epoch: 5; Test AUC: 0.72890559732665
Epoch: 6; Test AUC: 0.6313700918964077
Epoch: 7; Test AUC: 0.6349206349206349
Epoch: 8; Test AUC: 0.6449456975772766
Epoch: 9; Test AUC: 0.3845029239766082
```

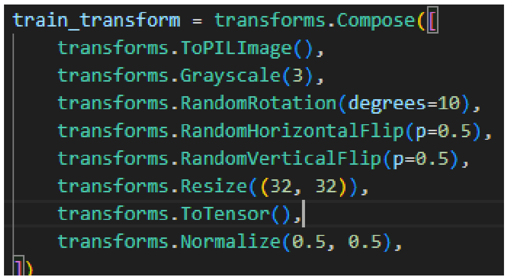


```
train_transform_final = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Grayscale(3),
    transforms.RandomHorizontalFlip(1), # Random horizon
    transforms.RandomVerticalFlip(1),
    transforms.RandomRotation(degrees=(60,61)), #random
    transforms.ColorJitter(brightness=40, contrast=45,
                           saturation=100, hue=0.3),
    transforms.GaussianBlur(5, sigma=(0.2, 3.0)),
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize(0.5, 0.5)
])
```

These Optimization Tests were done with the following data augmentations:

No Optimizations:

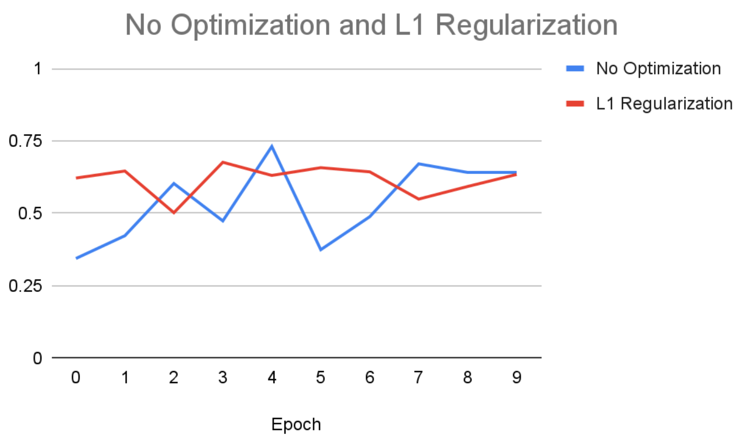
```
Epoch: 0; Test AUC: 0.34398496240601506
Epoch: 1; Test AUC: 0.42251461988304095
Epoch: 2; Test AUC: 0.6027568922305765
Epoch: 3; Test AUC: 0.47368421052631576
Epoch: 4; Test AUC: 0.730576441102757
Epoch: 5; Test AUC: 0.3742690058479532
Epoch: 6; Test AUC: 0.4880952380952381
Epoch: 7; Test AUC: 0.6706349206349206
Epoch: 8; Test AUC: 0.6409774436090225
Epoch: 9; Test AUC: 0.6409774436090226
```



```
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Grayscale(3),
    transforms.RandomRotation(degrees=10),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize(0.5, 0.5),
])
```

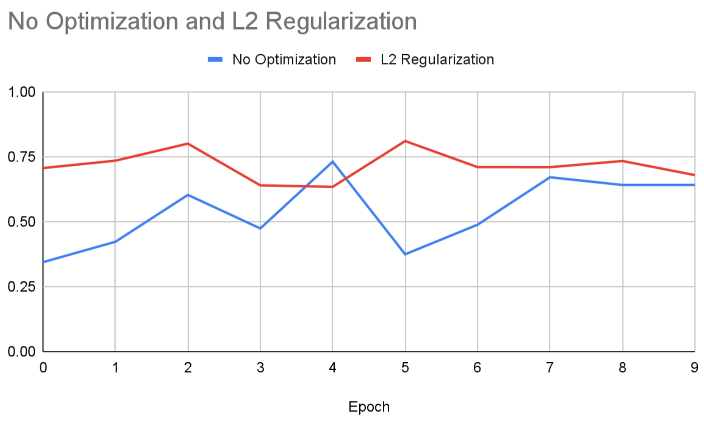
Controlling Overfitting with L1 Regularization Using K fold Cross Validation:

```
Optimizing with L1 Regularization
Epoch: 0; Test AUC: 0.6216791979949875
Epoch: 1; Test AUC: 0.6458020050125314
Epoch: 2; Test AUC: 0.5020050125313282
Epoch: 3; Test AUC: 0.6760651629072683
Epoch: 4; Test AUC: 0.6307226399331662
Epoch: 5; Test AUC: 0.6573934837092732
Epoch: 6; Test AUC: 0.643107769423559
Epoch: 7; Test AUC: 0.5489348370927318
Epoch: 8; Test AUC: 0.5925020885547202
Epoch: 9; Test AUC: 0.6340434419381789
```



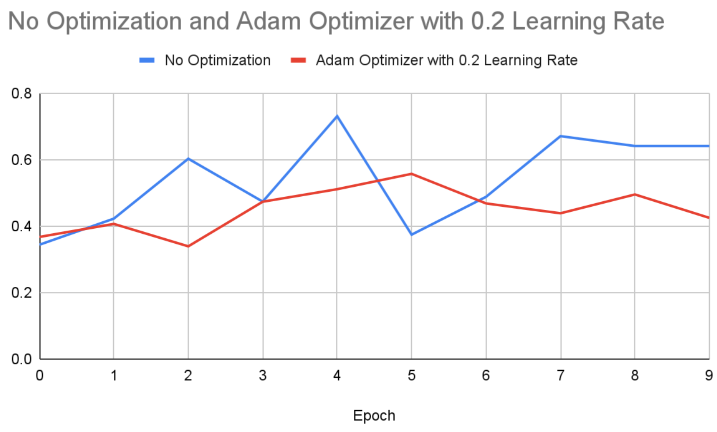
Controlling Overfitting with L2 Regularization Using K fold Cross Validation:

```
Epoch: 0; Test AUC: 0.7060985797827902
Epoch: 1; Test AUC: 0.7344611528822055
Epoch: 2; Test AUC: 0.8000417710944028
Epoch: 3; Test AUC: 0.639390142021721
Epoch: 4; Test AUC: 0.6338345864661654
Epoch: 5; Test AUC: 0.8100250626566415
Epoch: 6; Test AUC: 0.7096700083542189
Epoch: 7; Test AUC: 0.7094820384294069
Epoch: 8; Test AUC: 0.7331662489557227
Epoch: 9; Test AUC: 0.679030910609858
```



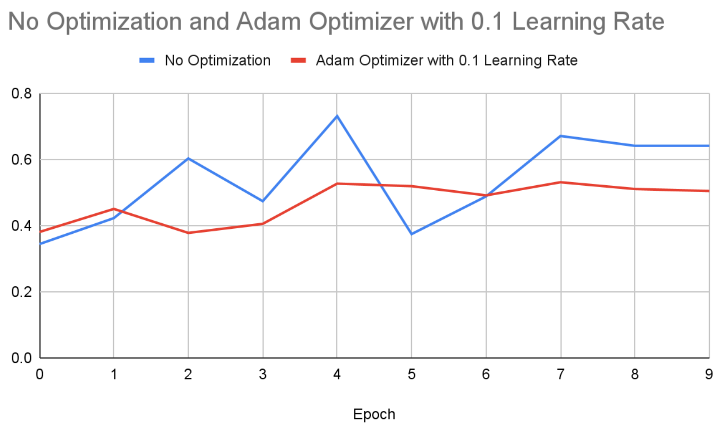
Using the Adam Optimizer with 0.2 learning rate:

```
Epoch: 0; Test AUC: 0.36737677527151213
Epoch: 1; Test AUC: 0.4066416040100251
Epoch: 2; Test AUC: 0.3391812865497076
Epoch: 3; Test AUC: 0.4733709273182957
Epoch: 4; Test AUC: 0.5110693400167083
Epoch: 5; Test AUC: 0.5572263993316624
Epoch: 6; Test AUC: 0.46825396825396826
Epoch: 7; Test AUC: 0.43859649122807015
Epoch: 8; Test AUC: 0.4951963241436925
Epoch: 9; Test AUC: 0.424812030075188
```



Using the Adam Optimizer with 0.1 learning rate:

```
Epoch: 0; Test AUC: 0.38053467000835417
Epoch: 1; Test AUC: 0.4502923976608187
Epoch: 2; Test AUC: 0.3778195488721805
Epoch: 3; Test AUC: 0.40517961570593153
Epoch: 4; Test AUC: 0.526733500417711
Epoch: 5; Test AUC: 0.5190058479532164
Epoch: 6; Test AUC: 0.49122807017543857
Epoch: 7; Test AUC: 0.5309106098579782
Epoch: 8; Test AUC: 0.5104427736006684
Epoch: 9; Test AUC: 0.5043859649122807
```



Conclusions

With L1 Regularization, we can see that the results can vary from epoch to epoch to some performing better with L1 Regularization compared to the original data. But on average, we do see a slight increase in performance due to the optimization. On the other hand with L2 Regularization we see a general increase in performance for almost every epoch showing an increase in performance due to the optimization on average. For changing the optimizer with the two different learning rates we see a general decrease in results as compared to the PESG optimizer that was originally used.

With the learning rate, the base model uses a value of 0.1, which provides adequate AUC values. Increasing the learning rate to 0.5 caused the AUC values of the smaller value epochs to be better, but the higher value epochs had overall worse AUC values. Overall, the best AUC values were obtained with a learning rate of 0.2.

With Data Augmentation, we can conclude that often some changes to the data might lower the performance of the model. Some of the epochs saw noticeable improvements with augmentation but as a whole some of the changes that were made to the data led to lower AUC performance. Further tests with varying levels of brightness, rotation, cropping, etc would provide a higher AUC.