CS633 Final Project: Improving Generalization of Deep AUC Maximization for Medical Image Classification

by

Siddhanth Reddy
Saishrawan Vutharkar

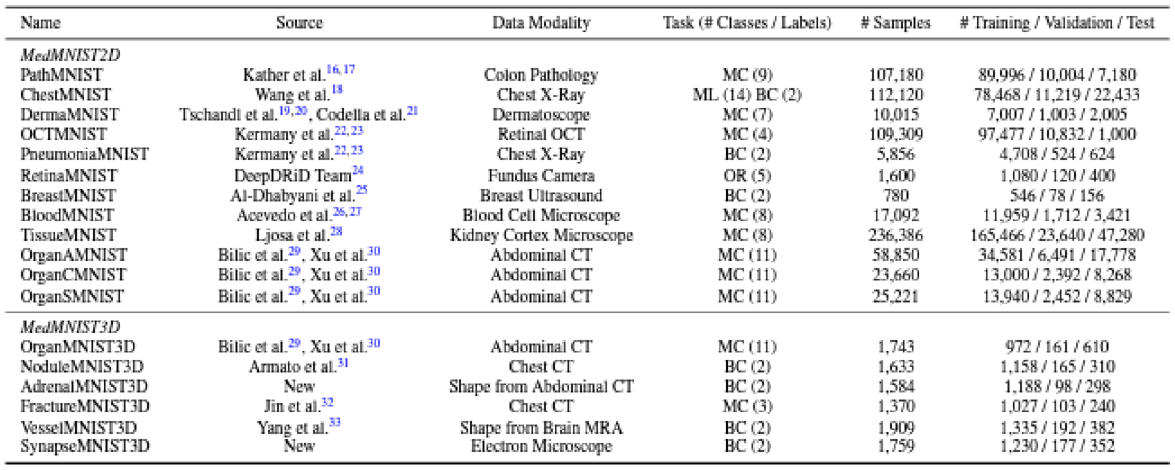
TABLE OF CONTENTS

1. Introduction

There are many techniques available for learning a deep neural network. Deep AUC Maximization (DAM) is a new paradigm for learning a deep neural network by maximizing the AUC score of the model on a dataset. This technique has achieved great success in solving real-world problems. Although DAM has achieved great success on large-scale datasets, it might easily overfit on small training data. In this project we work on the medical image dataset MedMNIST to develop strategies to improve the generalization ability of DAM for medical image classification tasks,

2. Dataset

MedMNIST is a dataset of standardized biomedical images. It has a collection of both 2D and 3D images.MedMNIST has diverse datasets available as can be seen in the table below:



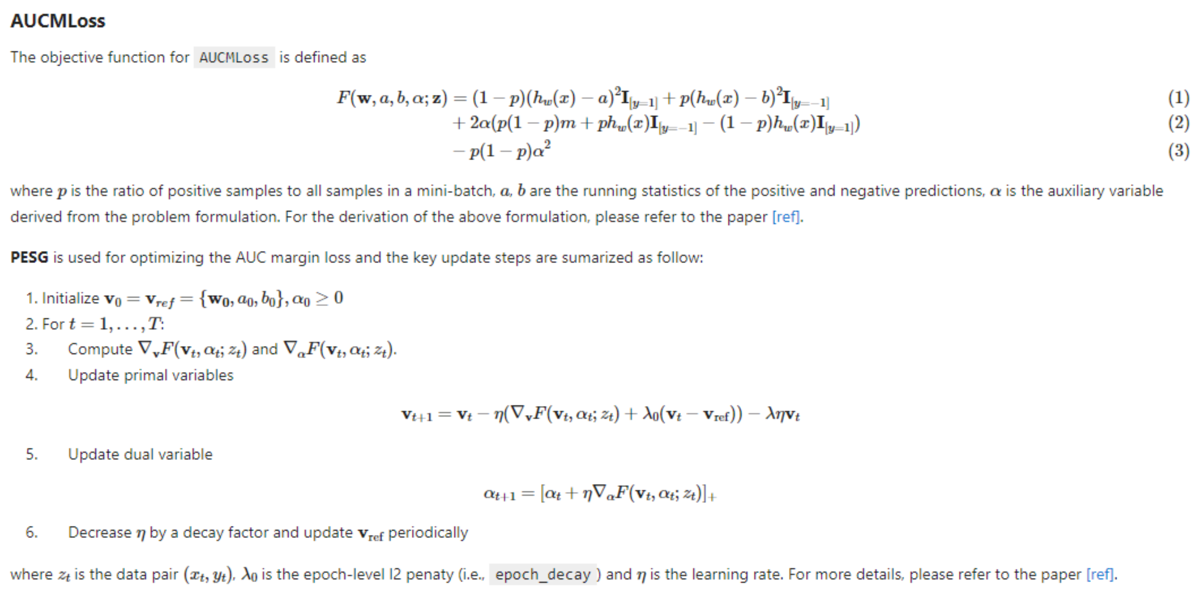
Name	Source	Data Modality	Task (# Classes / Labels)	# Samples	# Training / Validation / Test
MedMNIST2D					
PathMNIST	Kather et al.[16,17]	Colon Pathology	MC (9)	107,180	89,996 / 10,004 / 7,180
ChestMNIST	Wang et al.[18]	Chest X-Ray	ML (14) BC (2)	112,120	78,468 / 11,219 / 22,433
DermaMNIST	Tschandl et al.[19,20], Codella et al.[21]	Dermatoscope	MC (7)	10,015	7,007 / 1,003 / 2,005
OCTMNIST	Kermany et al.[22,23]	Retinal OCT	MC (4)	109,309	97,477 / 10,832 / 1,000
PneumoniaMNIST	Kermany et al.[22,23]	Chest X-Ray	BC (2)	5,856	4,708 / 524 / 624
RetinaMNIST	DeepDRiD Team[24]	Fundus Camera	OR (5)	1,600	1,080 / 120 / 400
BreastMNIST	Al-Dhabyani et al.[25]	Breast Ultrasound	BC (2)	780	546 / 78 / 156
BloodMNIST	Acevedo et al.[26,27]	Blood Cell Microscope	MC (8)	17,092	11,959 / 1,712 / 3,421
TissueMNIST	Ljosa et al.[28]	Kidney Cortex Microscope	MC (8)	236,386	165,466 / 23,640 / 47,280
OrganAMNIST	Bilic et al.[29], Xu et al.[30]	Abdominal CT	MC (11)	58,850	34,581 / 6,491 / 17,778
OrganCMNIST	Bilic et al.[29], Xu et al.[30]	Abdominal CT	MC (11)	23,660	13,000 / 2,392 / 8,268
OrganSMNIST	Bilic et al.[29], Xu et al.[30]	Abdominal CT	MC (11)	25,221	13,940 / 2,452 / 8,829
MedMNIST3D					
OrganMNIST3D	Bilic et al.[29], Xu et al.[30]	Abdominal CT	MC (11)	1,743	972 / 161 / 610
NoduleMNIST3D	Armato et al.[31]	Chest CT	BC (2)	1,633	1,158 / 165 / 310
AdrenalMNIST3D	New	Shape from Abdominal CT	BC (2)	1,584	1,188 / 98 / 298
FractureMNIST3D	Jin et al.[32]	Chest CT	MC (3)	1,370	1,027 / 103 / 240
VesselMNIST3D	Yang et al.[33]	Shape from Brain MRA	BC (2)	1,909	1,335 / 192 / 382
SynapseMNIST3D	New	Electron Microscope	BC (2)	1,759	1,230 / 177 / 352

In this project we work on BreastMNIST, PneumoniaMNIST, ChestMNIST, NoduleMNIST3D, AdrenalMNIST3D, VesselMNIST3D, SynapseMNIST3D. We can observe from the table that except for ChestMNIST, the other datasets are relatively small which make them suitable for our project which aims to improve the generalization ability of DAM for medical image classification tasks.

3. Models and methods

We initially started with a basic model using AUCMLoss() and PESG optimizer from libauc package to optimize AUROC. We also used ResNet18 as our convolutional neural network and changed its first layer for our model. We then trained our model for multiple epochs till we saw no improvements in the errors.In each iteration of the epoch we trained the model and noted the training and validation errors.

The function and algorithm for AUCMLoss() and PESG is shown below:



AUCMLoss

The objective function for `AUCMLoss` is defined as

$$\begin{aligned} F(\mathbf{w}, a, b, \alpha; \mathbf{z}) = &(1-p)(h_w(x)-a)^2 \mathbf{I}_{[y=1]} + p(h_w(x)-b)^2 \mathbf{I}_{[y=-1]} && (1)\\ &+ 2\alpha(p(1-p)m + p h_w(x)\mathbf{I}_{[y=-1]} - (1-p)h_w(x)\mathbf{I}_{[y=1]}) && (2)\\ &- p(1-p)\alpha^2 && (3) \end{aligned}$$

where p is the ratio of positive samples to all samples in a mini-batch, a, b are the running statistics of the positive and negative predictions, α is the auxiliary variable derived from the problem formulation. For the derivation of the above formulation, please refer to the paper [ref].

PESG is used for optimizing the AUC margin loss and the key update steps are sumarized as follow:

1. Initialize $\mathbf{v}_0 = \mathbf{v}_{ref} = \{\mathbf{w}_0, a_0, b_0\}, \alpha_0 \geq 0$
2. For $t = 1, \ldots, T$:
3. Compute $\nabla_{\mathbf{v}} F(\mathbf{v}_t, \alpha_t; z_t)$ and $\nabla_{\alpha} F(\mathbf{v}_t, \alpha_t; z_t)$.
4. Update primal variables

$$\mathbf{v}_{t+1} = \mathbf{v}_t - \eta(\nabla_{\mathbf{v}} F(\mathbf{v}_t, \alpha_t; z_t) + \lambda_0(\mathbf{v}_t - \mathbf{v}_{\text{ref}})) - \lambda\eta\mathbf{v}_t$$

5. Update dual variable

$$\alpha_{t+1} = [\alpha_t + \eta\nabla_{\alpha} F(\mathbf{v}_t, \alpha_t; z_t)]_+$$

6. Decrease η by a decay factor and update $\mathbf{v}_{\text{ref}}$ periodically

where z_t is the data pair (x_t, y_t), λ_0 is the epoch-level l2 penaty (i.e., `epoch_decay`) and η is the learning rate. For more details, please refer to the paper [ref].

We then tried different methods which can be used to improve the accuracy and generalizability like hyperparameter tuning of the learning rate, dropout, changing the class imbalance to check for and improve model accuracy and learning rate model which decreases the learning rate after each decay epoch.
Hyperparameter tuning on learning rate can also help us identify if the learning rate has an impact on the error and in choosing the best learning rate.Class imbalance can also cause inaccurate predictions and we can use ImbalancedDataGenerator to check if it can improve the accuracy.Dropout is a technique that can be used to improve the generalizability of a model. We can also use decay epochs to decrease the learning rate for better model learning.
Showing the key features of the models below:

Basic:
lr = 0.01
epoch_decay = 2e-3
weight_decay = 0.01
margin = 1.0

Hyperparameter tuning:
lr = [0.001, 0.005, 0.01, 0.05, 0.1]
epoch_decay = 2e-3
weight_decay = 0.0001
margin = 1.0
For i in range(5):
    ….

optimizer = PESG(net, loss_fn=loss_fn, lr=lr[i], momentum=0.8, margin=margin, epoch_decay=epoch_decay,weight_decay=weight_decay)

Train data for optimizer
val_auc = auc_roc_score(val_true, val_pred)
val_log.append(val_auc)

final_log.append(val_auc)

fl = np.argmax(final_log) gives the index for best learning rate

Learning rate change model:
lr = 0.1 which we decrease after decay_epochs
epoch_decay = 2e-3
weight_decay = 0.0001
decay_epochs = [20, 25]
margin = 1.0

Updating learning rate as shown below:

```
if epoch in decay_epochs:
  optimizer.update_regularizer(decay_factor=10)
```

Class imbalance model:
lr = 0.1
epoch_decay = 2e-3
weight_decay = 0.0001
decay_epochs = [20, 25]
margin = 1.0
Changing data balance to have equal balance of classes

```
from libauc.utils import ImbalancedDataGenerator
generator = ImbalancedDataGenerator(verbose=True, random_seed=1)
(train_datav, train_labelsv) = generator.transform(train_data, train_labels, imratio=0.49)
(val_datav, val_labelsv) = generator.transform(val_data, val_labels, imratio=0.49)
```

Dropout model:
lr = 0.1
epoch_decay = 2e-3
weight_decay = 0.00001
decay_epochs = [20, 25]
margin = 1.0
Has dropout layer shown below:

```
net.dropout = torch.nn.Dropout(0.5)
```

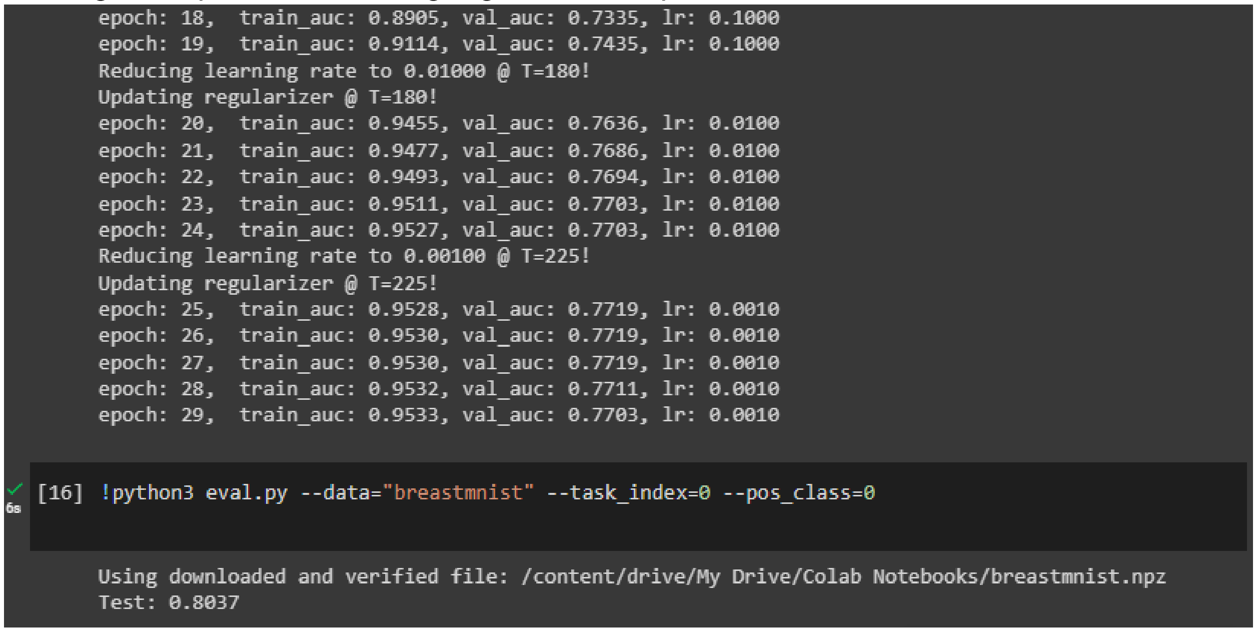
4. Results

Approach 1:

The first model we implemented has a learning rate of 0.01 and epochs =10. The results from the test set showed that we obtained AUC close to the benchmark for a few models but that the model can be improved. We then implemented a model with hyperparameter tuning to check for the best learning rate. We trained the model for learning rates [0.001, 0.005, 0.01, 0.05, 0.1] and selected the best learning rate based on the validation data. We observed that hyperparameter tuning improved the accuracy of a few datasets but decreased the accuracy of a few other datasets.

We then implemented a new model where the learning rate decreases by a factor after a few epochs. The initial learning rate of 0.1 decreases to 0.01 after 20 epochs and to 0.001 after 25 epochs. We further implemented two more models, one with dropout of 0.5 and one with equal class balance.
Sharing sample output from the learning rate decrease model for breastMNIST:

Sharing example results from google colab output for breastmnist:



```
epoch: 18,  train_auc: 0.8905, val_auc: 0.7335, lr: 0.1000
epoch: 19,  train_auc: 0.9114, val_auc: 0.7435, lr: 0.1000
Reducing learning rate to 0.01000 @ T=180!
Updating regularizer @ T=180!
epoch: 20,  train_auc: 0.9455, val_auc: 0.7636, lr: 0.0100
epoch: 21,  train_auc: 0.9477, val_auc: 0.7686, lr: 0.0100
epoch: 22,  train_auc: 0.9493, val_auc: 0.7694, lr: 0.0100
epoch: 23,  train_auc: 0.9511, val_auc: 0.7703, lr: 0.0100
epoch: 24,  train_auc: 0.9527, val_auc: 0.7703, lr: 0.0100
Reducing learning rate to 0.00100 @ T=225!
Updating regularizer @ T=225!
epoch: 25,  train_auc: 0.9528, val_auc: 0.7719, lr: 0.0010
epoch: 26,  train_auc: 0.9530, val_auc: 0.7719, lr: 0.0010
epoch: 27,  train_auc: 0.9530, val_auc: 0.7719, lr: 0.0010
epoch: 28,  train_auc: 0.9532, val_auc: 0.7711, lr: 0.0010
epoch: 29,  train_auc: 0.9533, val_auc: 0.7703, lr: 0.0010

[16] !python3 eval.py --data="breastmnist" --task_index=0 --pos_class=0

Using downloaded and verified file: /content/drive/My Drive/Colab Notebooks/breastmnist.npz
Test: 0.8037
```

Please find the results of the models for each class below:

Model	Basic	Hyperparameter tuning	Learning rate change	Class imbalance	Dropout
Breast	0.7586	0.8853	0.8124	0.5447	0.8037

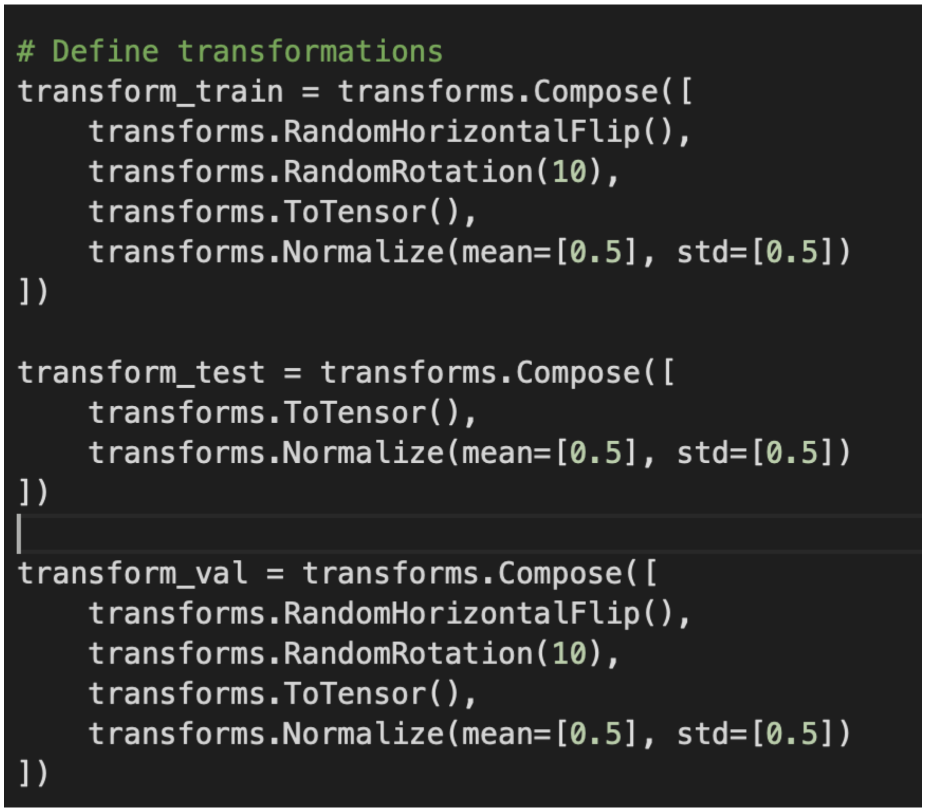
Pneumonia	0.9183	0.8963	0.9185	0.9312	0.9117
Adrenal	0.7246	0.7814	0.7846	0.7876	0.7846
Vessel	0.6833	0.8039	0.6191	0.8554	0.6191
Synapse	0.4956	0.4734	0.5531	0.6533	0.5531
Nodule	0.8289	0.7901	0.8133	0.7743	0.8133

We can see from the results that some models are better for some datasets and worse for some other datasets. Hyperparameter tuning seems to improve the error in general but it is terrible for the Synapse dataset. The validation data for Synapse may be imbalanced causing bad performance. Data Augmentation can help class imbalance problems. We can also see that the Learning rate change model also in general improves the accuracy.

Changing the class imbalance can show really good improvement (Synapse,Vessel) or it can also really decrease the accuracy(Breast). Adding a dropout layer also does not seem to significantly change the model performance.

Approach 2:

Using the following approach, we found out that the test_auc does not seem to be so close to the benchmark, so we tried another approach, where we changed the hyper parameters and used data augmentation and used early stopping. We defined the transformations as such:



```python
# Define transformations
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])

transform_val = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

These lines of code are defining three different data augmentation and preprocessing pipelines for training, testing, and validation datasets using the PyTorch transforms module.

transform_train pipeline is defined for the training dataset, which includes the following operations:
transforms.RandomHorizontalFlip(): randomly flips the input image horizontally with a probability of 0.5.
transforms.RandomRotation(10): randomly rotates the input image by a maximum of 10 degrees.
transforms.ToTensor(): converts the input image to a PyTorch tensor.
transforms.Normalize(mean=[0.5], std=[0.5]): normalizes the tensor by subtracting 0.5 from each element and dividing by 0.5.
The purpose of data augmentation is to increase the variability of the training dataset, which helps the model to generalize better to new unseen data.

transform_test pipeline is defined for the testing dataset, which includes the following operations:
transforms.ToTensor(): converts the input image to a PyTorch tensor.
transforms.Normalize(mean=[0.5], std=[0.5]): normalizes the tensor by subtracting 0.5 from each element and dividing by 0.5.

The purpose of normalization is to make the pixel values of the input image to have zero mean and unit variance, which helps the model to learn more effectively.

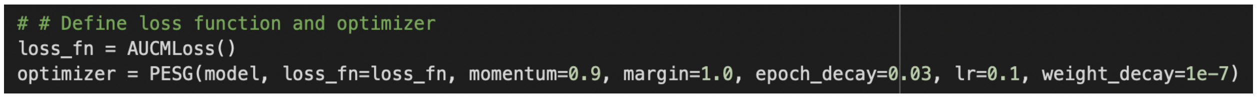
transform_val pipeline is defined for the validation dataset, which includes the following operations:
transforms.RandomHorizontalFlip(): randomly flips the input image horizontally with a probability of 0.5.
transforms.RandomRotation(10): randomly rotates the input image by a maximum of 10 degrees.
transforms.ToTensor(): converts the input image to a PyTorch tensor.
transforms.Normalize(mean=[0.5], std=[0.5]): normalizes the tensor by subtracting 0.5 from each element and dividing by 0.5.

The purpose of data augmentation in validation dataset is to make it more representative of the test dataset, which helps to get a more accurate estimate of the model's performance.

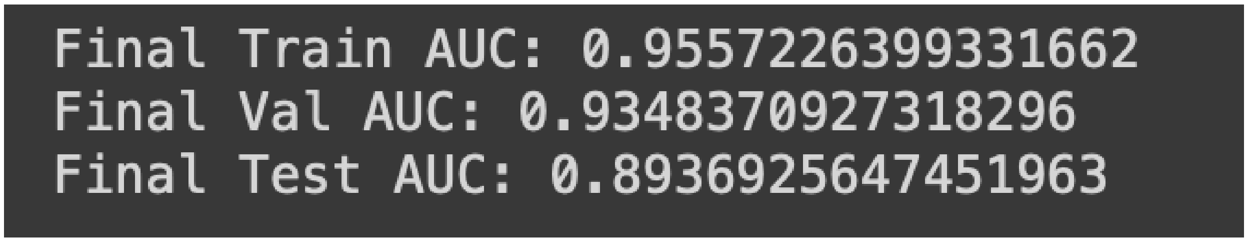
BreastMNIST:

For this dataset, we did the data augmentation as shown above, changed the hyper parameters to a lower learning rate and weight decay, and achieved a test_auc of about 89.3, which is very close to the benchmark.

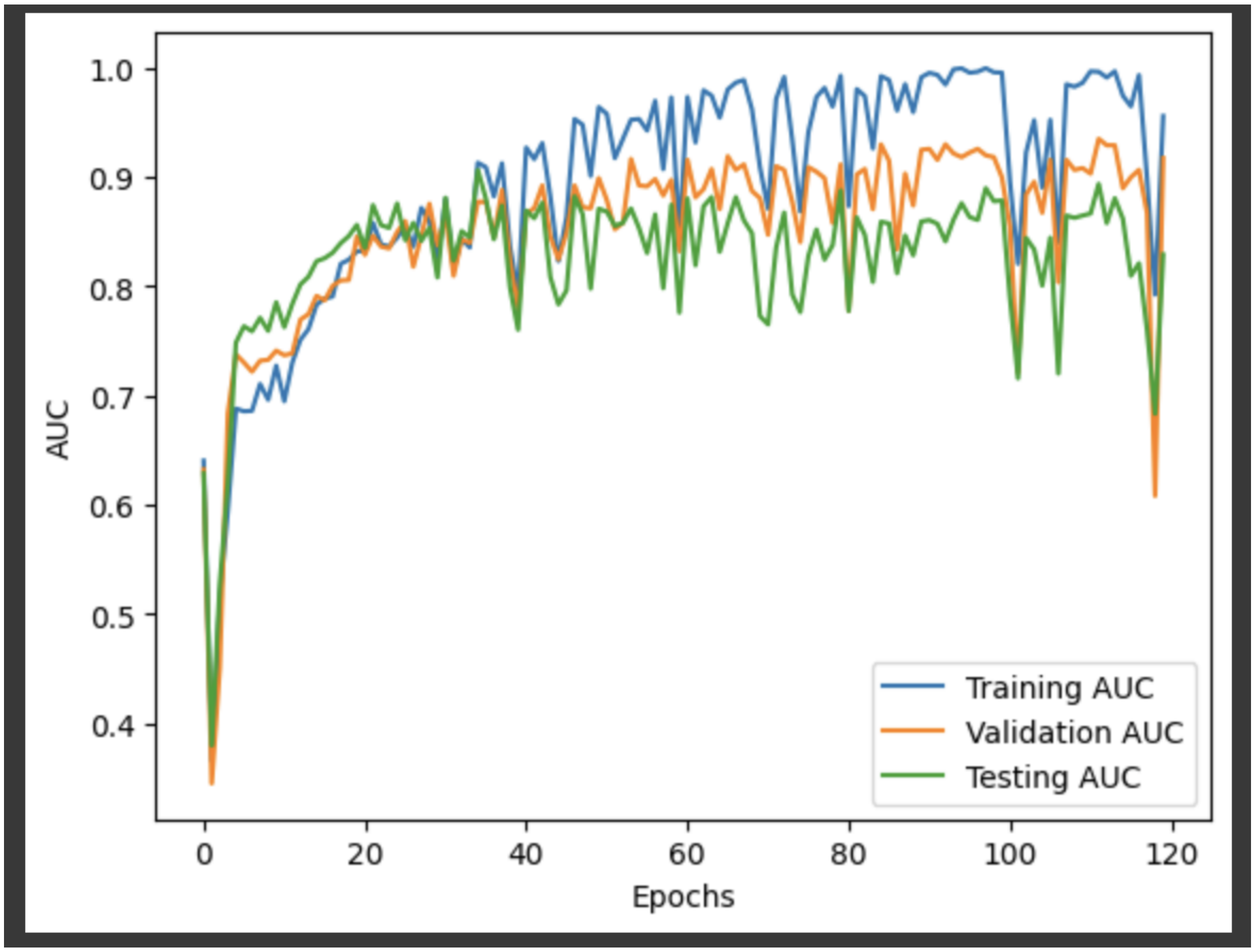
We used BATCH_SIZE of 64, total_epochs as 100 and the other hyper parameters as such.



```
# # Define loss function and optimizer
loss_fn = AUCMLoss()
optimizer = PESG(model, loss_fn=loss_fn, momentum=0.9, margin=1.0, epoch_decay=0.03, lr=0.1, weight_decay=1e-7)
```

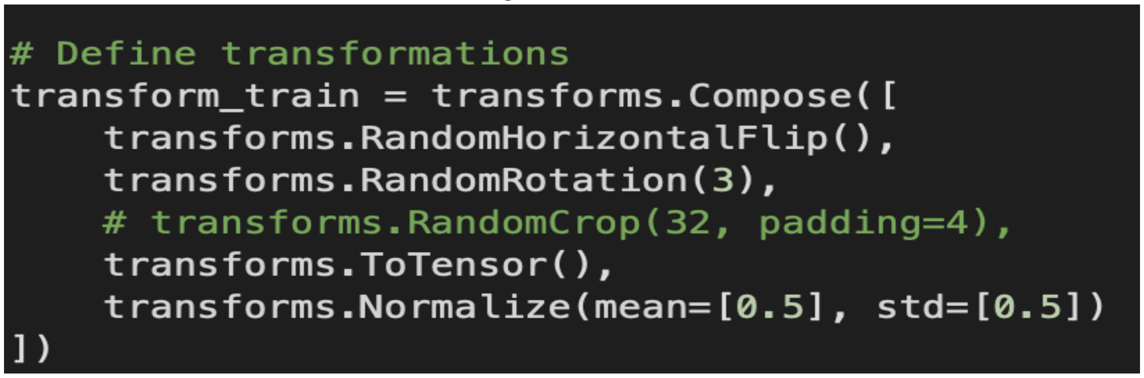


```
Final Train AUC: 0.9557226399331662
Final Val AUC: 0.9348370927318296
Final Test AUC: 0.8936925647451963
```



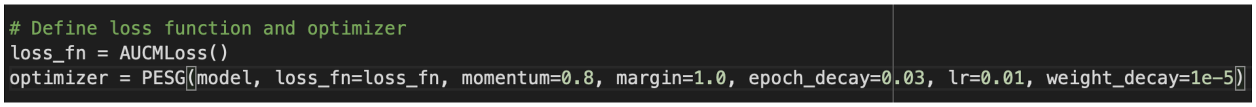
PneumoniaMNIST:

For this dataset, we did the data augmentation as such:



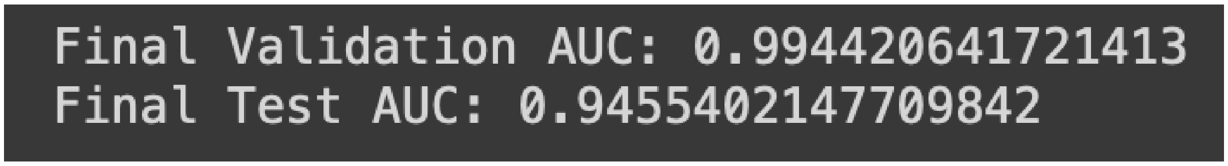
```
# Define transformations
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(3),
    # transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

Here the RandomCrop method was omitted, and the following hyper parameters were used, keeping the batch size as 64 and total_epochs to be 120.

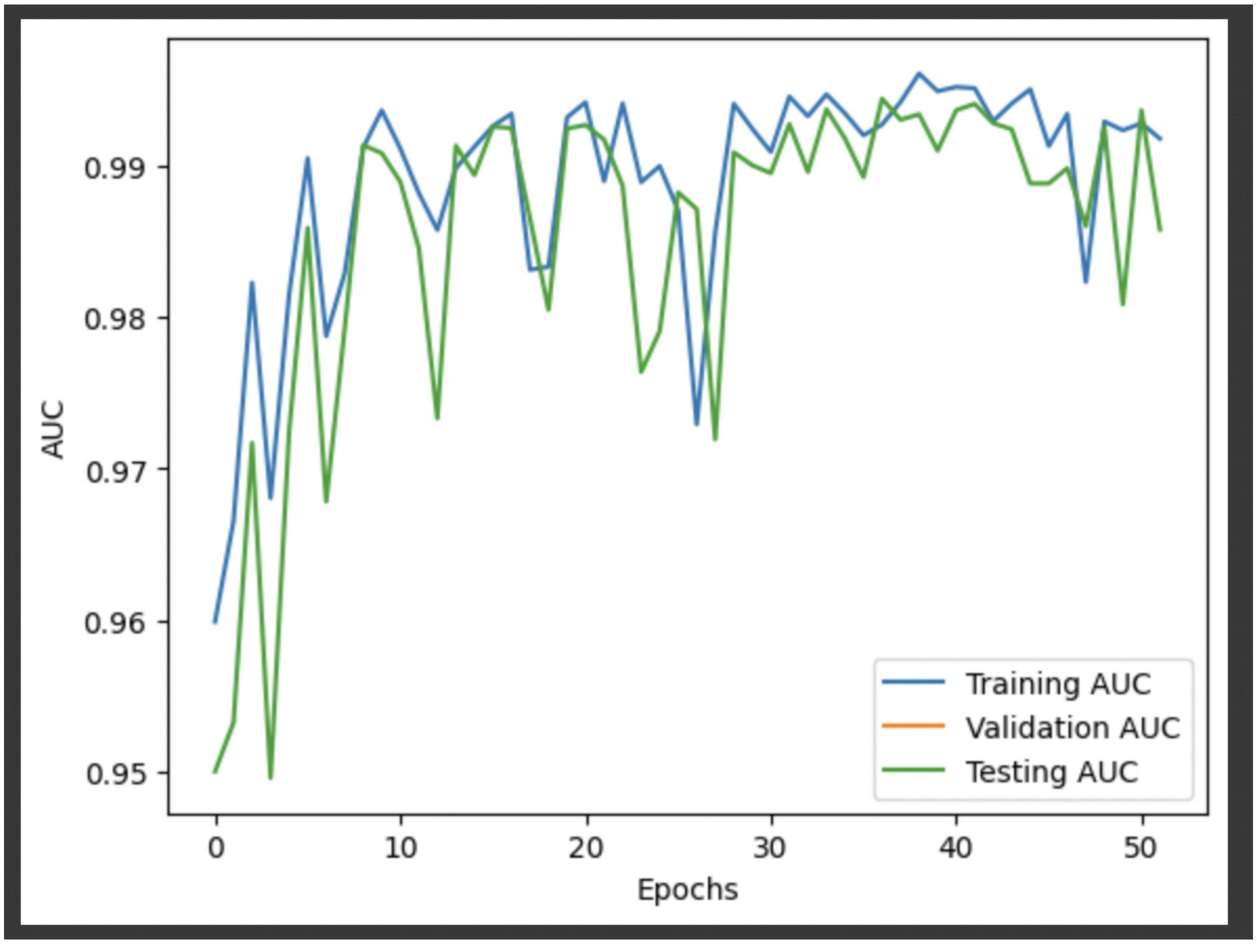


```
# Define loss function and optimizer
loss_fn = AUCMLoss()
optimizer = PESG(model, loss_fn=loss_fn, momentum=0.8, margin=1.0, epoch_decay=0.03, lr=0.01, weight_decay=1e-5)
```

Here we were able to beat the benchmark test_auc and we obtained the following results.

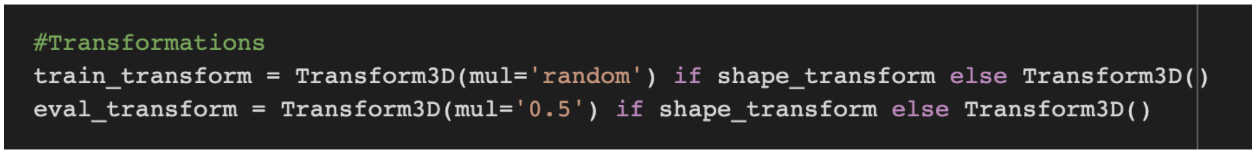


```
Final Validation AUC: 0.994420641721413
Final Test AUC: 0.9455402147709842
```



NoduleMNIST3D:

This is a 3D dataset and the data augmentation and transformation was done differently compared to the 2D model. We performed the transformation as such in our 3D models.

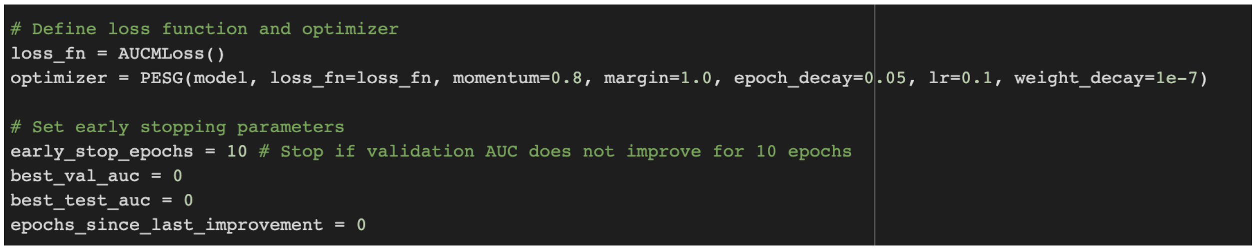


```
#Transformations
train_transform = Transform3D(mul='random') if shape_transform else Transform3D()
eval_transform = Transform3D(mul='0.5') if shape_transform else Transform3D()
```

This code is defining two 3D transformations, train_transform and eval_transform, that can be applied to the input data before feeding it to a 3D deep learning model for training and evaluation.

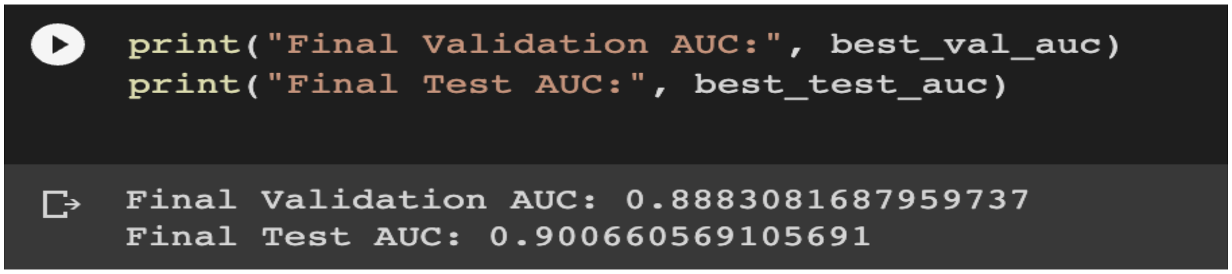
train_transform: This transformation is applied to the training data and includes a random scaling operation, which is controlled by the mul parameter. If shape_transform is true, then mul is set to 'random', which means that the input data will be randomly scaled by a factor between 0.5 and 2.0. If shape_transform is false, then mul is not provided and the input data will not be scaled.
eval_transform: This transformation is applied to the evaluation data (validation or test data) and includes a scaling operation, which is controlled by the mul parameter. If shape_transform is true, then mul is set to '0.5', which means that the input data will be scaled by a factor of 0.5. This scaling is intended to match the scaling applied to the training data during random augmentation. If shape_transform is false, then mul is not provided and the input data will not be scaled.
Overall, these transformations are intended to increase the variability of the training data and to ensure that the evaluation data is consistent with the training data in terms of scaling. This can help to improve the accuracy and generalization of the 3D deep learning model.

The hyper parameters we used are below, including batch size of 64 and total epochs as 120.



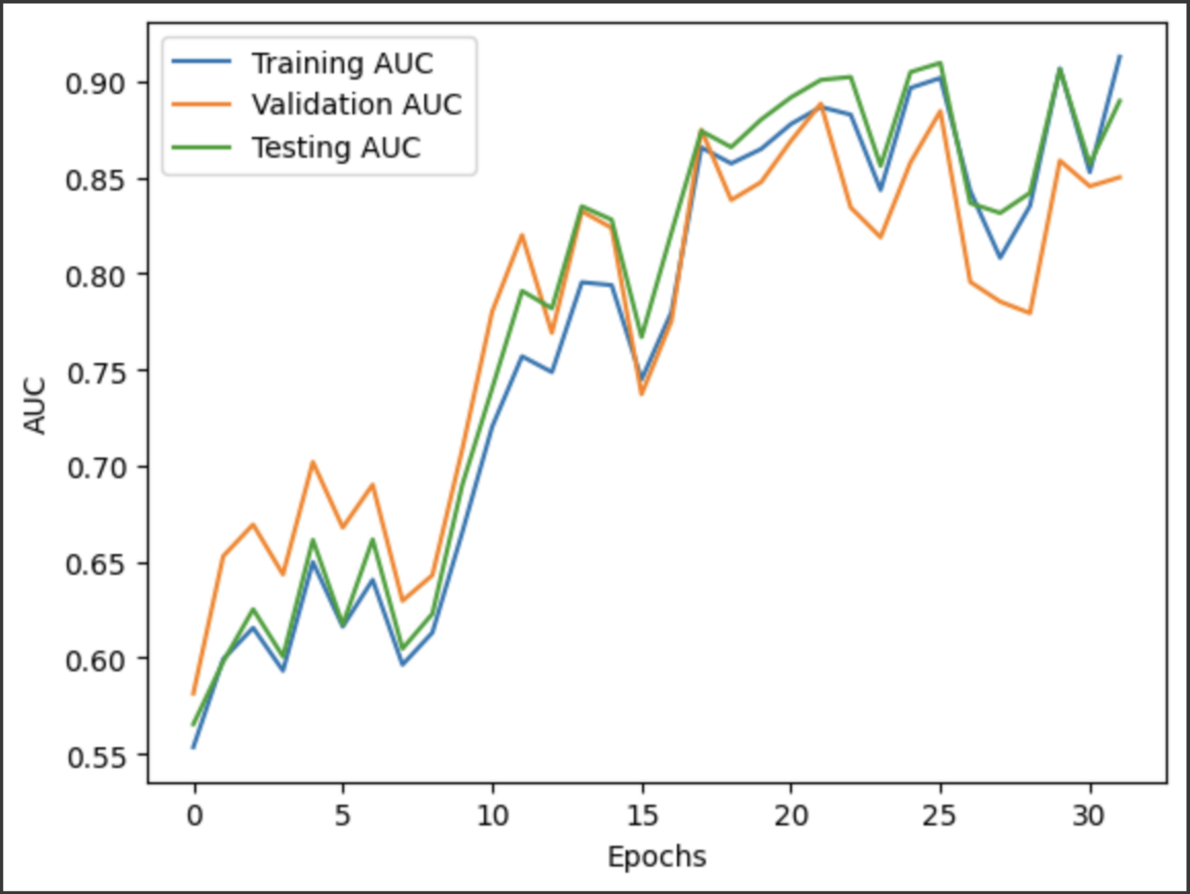
```
# Define loss function and optimizer
loss_fn = AUCMLoss()
optimizer = PESG(model, loss_fn=loss_fn, momentum=0.8, margin=1.0, epoch_decay=0.05, lr=0.1, weight_decay=1e-7)

# Set early stopping parameters
early_stop_epochs = 10 # Stop if validation AUC does not improve for 10 epochs
best_val_auc = 0
best_test_auc = 0
epochs_since_last_improvement = 0
```

Using these hyper parameters, we achieved the test_auc of 0.90 and we beat the benchmark test_auc of 0.863.



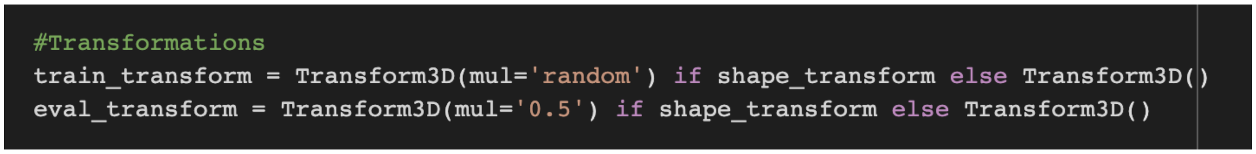
```
print("Final Validation AUC:", best_val_auc)
print("Final Test AUC:", best_test_auc)
```

```
Final Validation AUC: 0.8883081687959737
Final Test AUC: 0.900660569105691
```



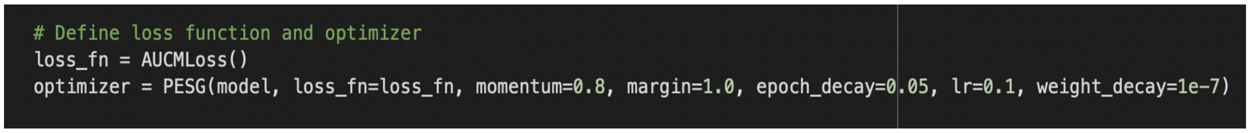
VesselMNIST3D:

We defined the transformations as such:



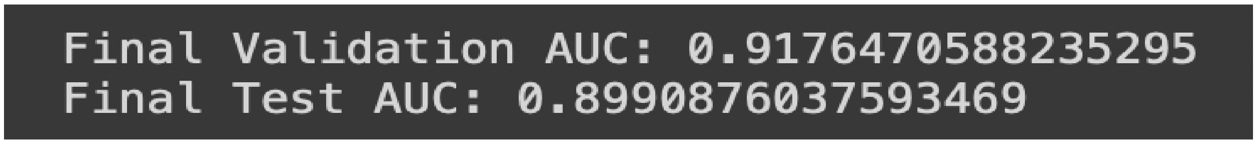
```
#Transformations
train_transform = Transform3D(mul='random') if shape_transform else Transform3D()
eval_transform = Transform3D(mul='0.5') if shape_transform else Transform3D()
```

The hyper parameters we used are:

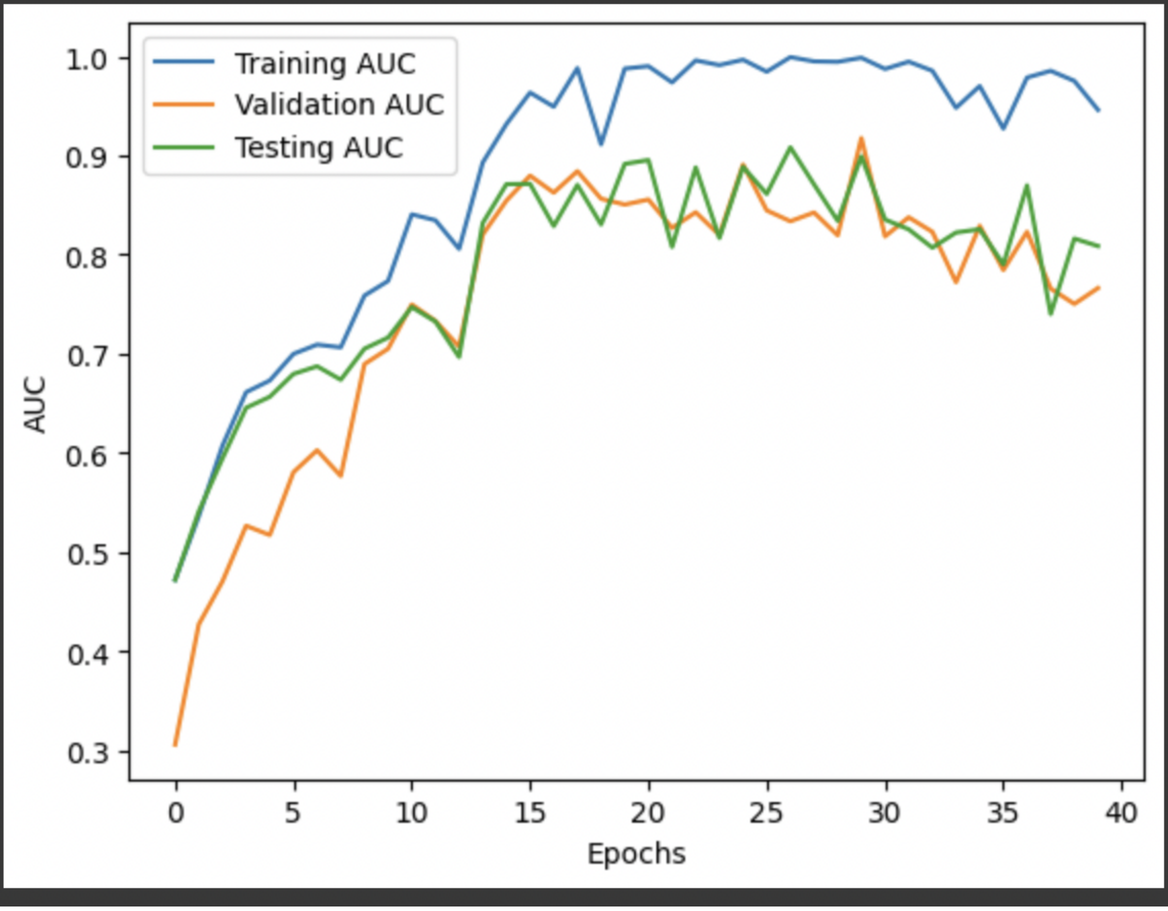


```
# Define loss function and optimizer
loss_fn = AUCMLoss()
optimizer = PESG(model, loss_fn=loss_fn, momentum=0.8, margin=1.0, epoch_decay=0.05, lr=0.1, weight_decay=1e-7)
```

Decreasing the weight decay to 1e-7 gave us better results of test_auc and we were able to beat the benchmark test_auc of 0.874.

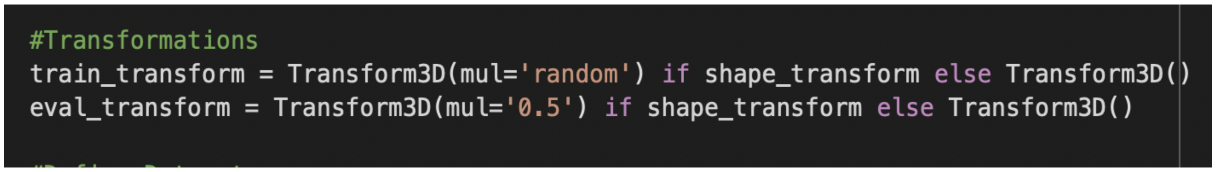


```
Final Validation AUC: 0.9176470588235295
Final Test AUC: 0.8990876037593469
```



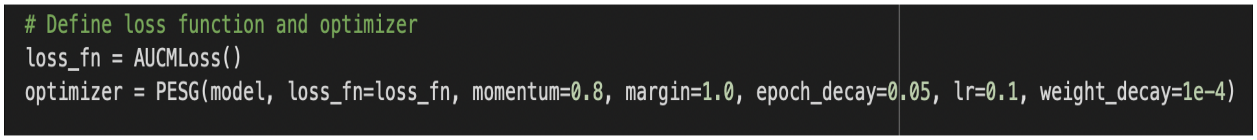
AdrenalMNIST3D:

Transformations were defined as such.



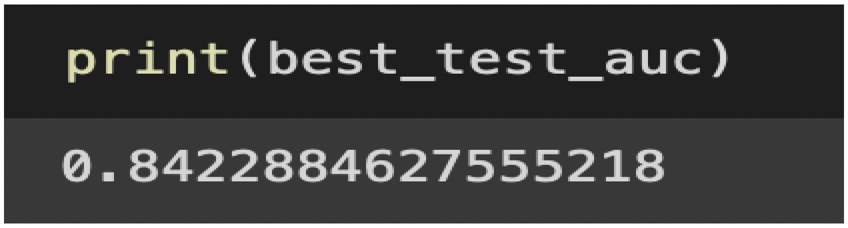
```
#Transformations
train_transform = Transform3D(mul='random') if shape_transform else Transform3D()
eval_transform = Transform3D(mul='0.5') if shape_transform else Transform3D()
```

The following hyper parameters were used and increasing the weight_decay to 1e-7 gave us the best results.



```
# Define loss function and optimizer
loss_fn = AUCMLoss()
optimizer = PESG(model, loss_fn=loss_fn, momentum=0.8, margin=1.0, epoch_decay=0.05, lr=0.1, weight_decay=1e-4)
```

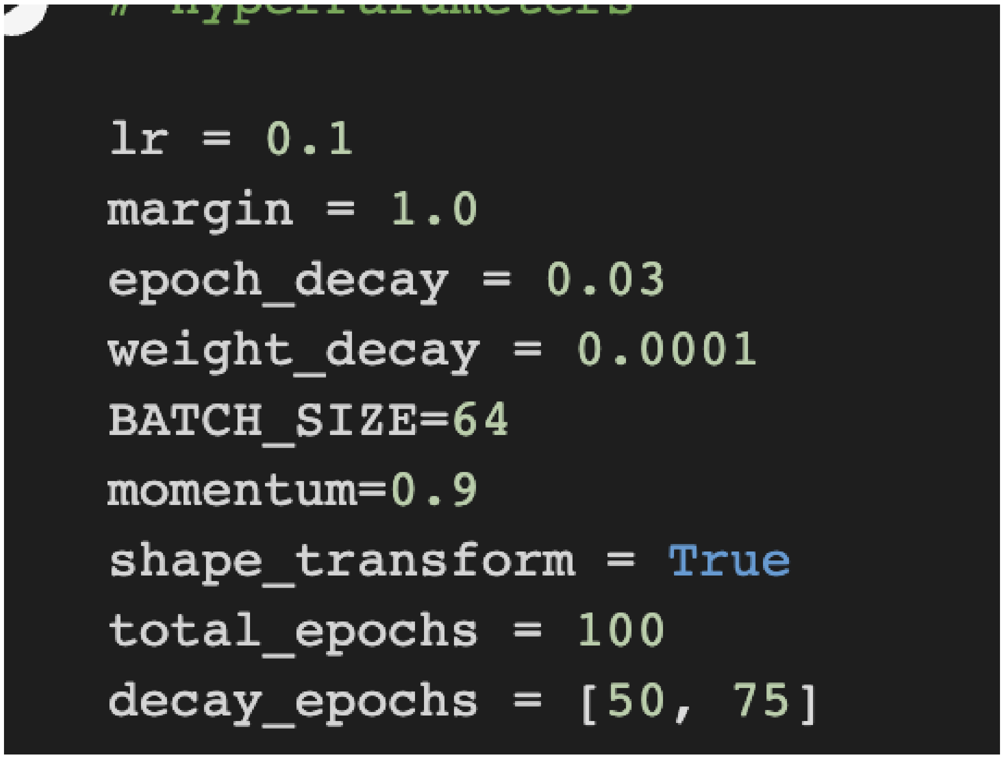
We were able to beat the benchmark test_auc of 0.8270.



```
print(best_test_auc)
0.8422884627555218
```

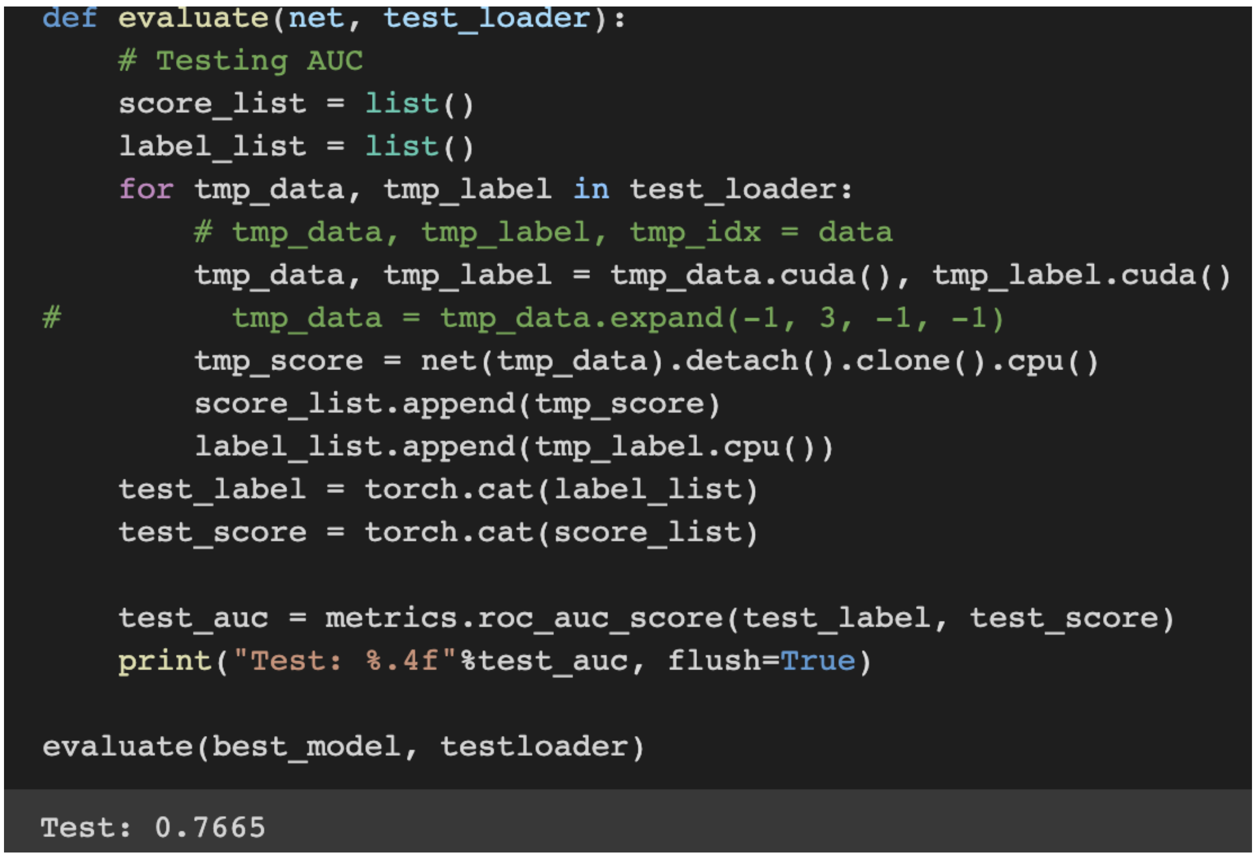
SynapseMNIST3D:

The same transformations as above were done and the following hyperparameters were used.



```
lr = 0.1
margin = 1.0
epoch_decay = 0.03
weight_decay = 0.0001
BATCH_SIZE=64
momentum=0.9
shape_transform = True
total_epochs = 100
decay_epochs = [50, 75]
```

Using these hyper parameters we were not able to beat the benchmark test auc, but we were able to achieve a test auc of about 0.76.

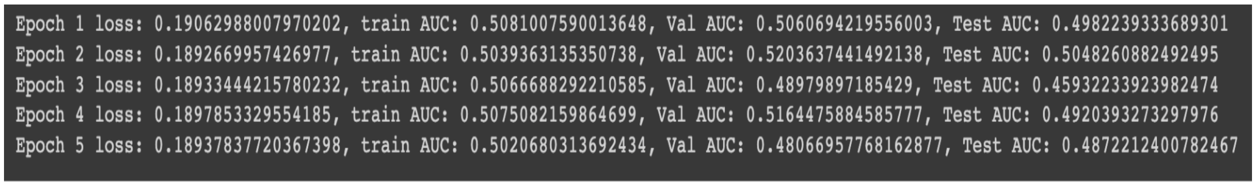


```
def evaluate(net, test_loader):
    # Testing AUC
    score_list = list()
    label_list = list()
    for tmp_data, tmp_label in test_loader:
        # tmp_data, tmp_label, tmp_idx = data
        tmp_data, tmp_label = tmp_data.cuda(), tmp_label.cuda()
#         tmp_data = tmp_data.expand(-1, 3, -1, -1)
        tmp_score = net(tmp_data).detach().clone().cpu()
        score_list.append(tmp_score)
        label_list.append(tmp_label.cpu())
    test_label = torch.cat(label_list)
    test_score = torch.cat(score_list)

    test_auc = metrics.roc_auc_score(test_label, test_score)
    print("Test: %.4f"%test_auc, flush=True)

evaluate(best_model, testloader)
```

```
Test: 0.7665
```

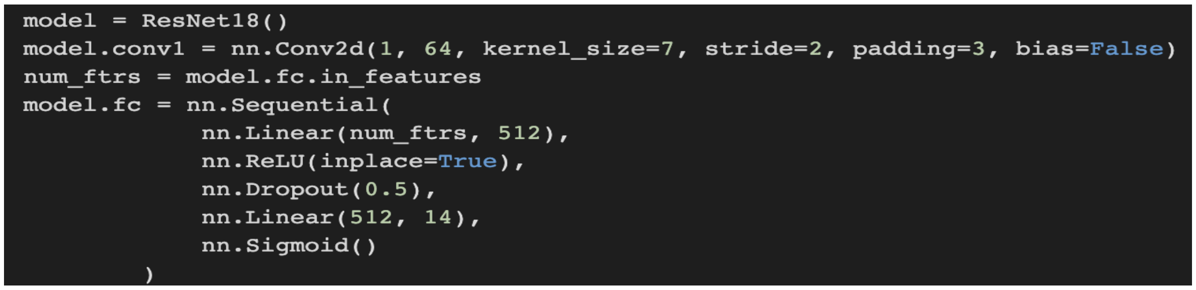
ChestMNIST:

Due to the large dataset of chestmnist we were able to perform only a few epochs the results we obtained are such.



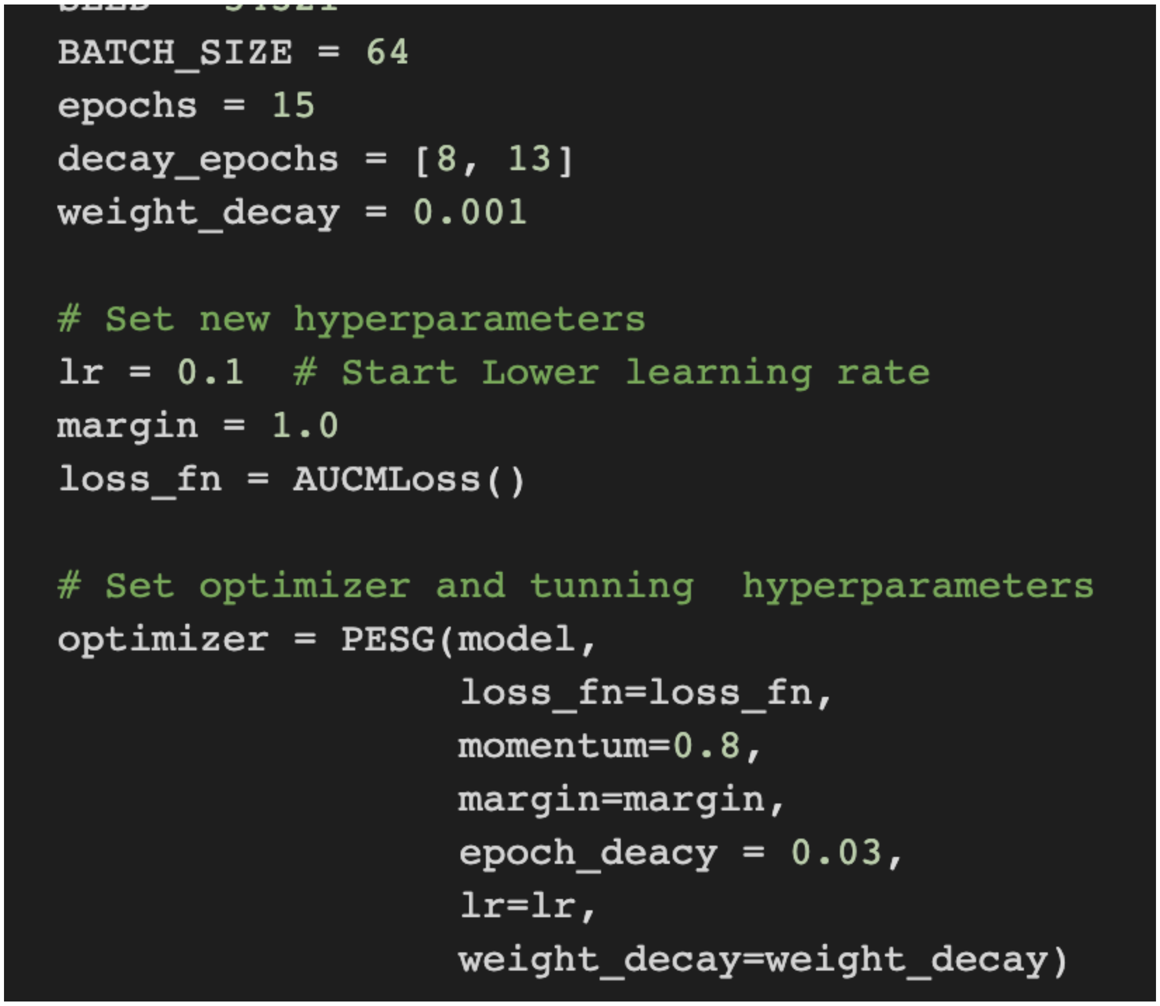
```
Epoch 1 loss: 0.19062988007970202, train AUC: 0.5081007590013648, Val AUC: 0.5060694219556003, Test AUC: 0.4982239333689301
Epoch 2 loss: 0.1892669957426977, train AUC: 0.5039363135350738, Val AUC: 0.5203637441492138, Test AUC: 0.5048260882492495
Epoch 3 loss: 0.18933444215780232, train AUC: 0.5066688292210585, Val AUC: 0.48979897185429, Test AUC: 0.45932233923982474
Epoch 4 loss: 0.1897853329554185, train AUC: 0.5075082159864699, Val AUC: 0.5164475884585777, Test AUC: 0.4920393273297976
Epoch 5 loss: 0.18937837720367398, train AUC: 0.5020680313692434, Val AUC: 0.48066957768162877, Test AUC: 0.4872212400782467
```

The model was designed as such, with a dropout layer being added with Sigmoid as the activation function and an test_auc of 0.55 was achieved.



```python
model = ResNet18()
model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
num_ftrs = model.fc.in_features
model.fc = nn.Sequential(
            nn.Linear(num_ftrs, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, 14),
            nn.Sigmoid()
        )
```

The following hyper parameters were used, and we can further tune these hyper parameters to get better results.



```python
BATCH_SIZE = 64
epochs = 15
decay_epochs = [8, 13]
weight_decay = 0.001

# Set new hyperparameters
lr = 0.1  # Start Lower learning rate
margin = 1.0
loss_fn = AUCMLoss()

# Set optimizer and tunning  hyperparameters
optimizer = PESG(model,
                 loss_fn=loss_fn,
                 momentum=0.8,
                 margin=margin,
                 epoch_deacy = 0.03,
                 lr=lr,
                 weight_decay=weight_decay)
```

5. Conclusions

We conclude that we were able to beat the benchmark for the following datasets: PneumoniaMNIST, NoduleMNIST3D, AdrenalMNIST3D and Vessel MNIST3D. We used several data augmentation techniques and prevented the overfitting of the model using low learning rates and low weight decay, which helped in achieving a good test auc in most of our models.

6. References

1. Algorithmic Foundation of Deep X-Risk Optimization
2. https://libauc.org/
3. MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification
4. https://medmnist.com/
5. https://github.com/MedMNIST/experiments
6. https://github.com/Optimization-AI/LibAUC
7. Large-scale Robust Deep AUC Maximization: A New Surrogate Loss and Empirical Studies on Medical Image Classification
8. Auc maximization in the era of big data and ai: A survey.