# Deep AUC Optimization

CS 633 Project Report

By

Harikrishnan Raghukumar (833003243)

Sneha Mishra (733000826)

## Table of Contents

# Chapter 1: Introduction

Deep AUC Maximization (DAM) is a relatively new paradigm for deep neural network training that involves optimizing the AUC score of a model on a given dataset. DAM has been shown to achieve impressive results in a range of applications, including medical image classification, where it has been used to achieve state-of-the-art performance on several challenging datasets. However, despite its potential, DAM has some limitations, particularly when it comes to overfitting on small training sets. When trained on smaller datasets, DAM can be prone to overfitting, resulting in poor generalization performance on unseen data.

The goal of this project is to improve the generalization ability of DAM for medical image classification tasks. Specifically, we aim to investigate ways to mitigate overfitting and improve the performance of DAM on smaller datasets. To accomplish this, we will conduct experiments on seven medical image classification tasks from the MedMNIST website using the LibAUC library. The seven datasets include BreastMNIST, PneumoniaMNIST, ChestMNIST, NoduleMNIST3D, AdrenalMNIST3D, VesselMNIST3D, and SynapseMNIST3D.

We will first provide a brief overview of the DAM technique and its applications. Then, we will describe the experimental setup, including the datasets, network structure, and training details. Next, we will present the results of our experiments and compare them with the benchmark results reported in the MedMNIST paper. To achieve our goal, we will conduct a series of experiments on the seven datasets from the MedMNIST website. We will compare our results with the benchmark performance reported in the MedMNIST paper and other state-of-the-art methods. We will use ResNet 18 ( ResNet 18-3D for 3D datasets) backbone structure as in the MedMNIST paper to ensure a fair comparison.

To ensure that our results are robust, we will experiment with different hyperparameters and other relevant techniques to improve the generalization ability of DAM and to determine their effects on the performance of DAM. By systematically varying these hyperparameters and techniques, we aim to find the optimal configuration for DAM on each dataset. To evaluate the effectiveness of our approach, we will report the AUC score and test accuracy for each dataset. The AUC score provides a robust measure of classification performance that is insensitive to class imbalance, which is a common issue in medical image classification tasks.

Finally, we will provide an in-depth analysis of our results and discuss the implications of our findings for medical image classification in our respective datasets. It is important to note that our experiments are limited by time constraints and the resources available for this project. While we aim to achieve state-of-the-art performance with DAM on the MedMNIST datasets, there may be other factors that can be considered that could lead to more impactful results. However, we believe that the findings of this project provide valuable insights into the generalization ability of DAM and its potential applications for medical image classification.

# Chapter 2: Technical Details

## 2.1: Brief overview of the DAM technique and its applications

Deep AUC Maximization (DAM) is a deep learning technique that aims to maximize the area under the receiver operating characteristic curve (AUC) on a given dataset. The AUC is a common metric used to evaluate the performance of binary classifiers and maximizing it can lead to more robust and accurate models.

DAM involves optimizing a loss function that directly maximizes the AUC during training. This is achieved by using a pair of neural networks, one that predicts the class probabilities and another that predicts the class labels based on those probabilities. The two networks are trained jointly using a novel objective function that directly maximizes the AUC.

DAM has shown impressive results in various applications, including medical image classification, where it has been used to achieve state-of-the-art performance on several challenging datasets, such as ChestX-ray14, MURA, and CheXpert. DAM has also been applied to other areas such as sentiment analysis, spam detection, and fraud detection, showing promising results in these domains as well. Overall, DAM is a powerful technique that can improve the performance and robustness of deep learning models, especially in binary classification tasks.

## 2.2: Experimental Setup:

### 2.2.1: Datasets Description

- **BreastMNIST** is based on a dataset of 780 breast ultrasound images. It is categorized into 3 classes: normal, benign, and malignant. As low-resolution images have been used, the task has been simplified into binary classification by combining normal and benign as positive and classifying them against malignant as negative.

- **PneumoniaMNIST** is based on a prior dataset of 5,856 pediatric chest X-Ray images. The task is binary-class classification of pneumonia against normal. The source training set is split with a ratio of 9:1 into training and validation set and use its source validation set as the test set. The source images are gray-scale, and their sizes are (384 – 2,916) × (127 – 2,713). We center-crop the images with a window size of length of the short edge and resize them into 1 × 28 × 28.

- **ChestMNIST** is based on the NIH-ChestXray14 dataset18, a dataset comprising 112,120 frontal-view X-Ray images of 30,805 unique patients with the text-mined 14 disease labels, which could be formulized as a multi-label binary-class classification task. The official data split is used, and the source images have been resized from 1 × 1,024 × 1,024 to 1 × 28 × 28.

- **NoduleMNIST3D** is based on the LIDC-IDRI32, a large public lung nodule dataset, containing images from thoracic CT scans. The dataset is designed for both lung nodule segmentation and 5-level malignancy classification task. To perform binary classification, cases have been categorized with malignancy level 1/2 into negative class and 4/5 into positive class, ignoring the cases with malignancy level 3. The source dataset is split with a ratio of 7:1:2 into training, validation and test set, and center-crop the spatially normalized images (with a spacing of 1 mm × 1 mm × 1 mm) into 28 × 28 × 28.

- **AdrenalMNIST3D** is a new 3D shape classification dataset, consisting of shape masks from 1,584 left and right adrenal glands (i.e., 792 patients). Collected from Zhongshan Hospital affiliated to Fudan University, each 3D shape of adrenal gland is annotated by an expert endocrinologist using abdominal computed tomography (CT), together with a binary classification label of normal adrenal gland or adrenal mass. Considering patient privacy, the source CT scans are not provided, instead the real 3D shapes of adrenal glands and their classification labels are used. The center of adrenal is calculated and resized the center-cropped 64 mm × 64 mm × 64 mm volume into 28 × 28 × 28. The dataset is randomly split into training/validation/test set of 1,188 / 98 / 298 on a patient level.

- **VesselMNIST3D** is based on an open-access 3D intracranial aneurysm dataset, IntrA, containing 103 3D models (meshes) of entire brain vessels collected by reconstructing MRA images. 1,694 healthy vessel segments and 215 aneurysm segments are generated automatically from the complete models. The non-watertight mesh is fixed with PyMeshFix and voxelize the watertight mesh with trimesh into 28 × 28 × 28 voxels. The source dataset is split with a ratio of 7:1:2 into training, validation and test set.

- **SynapseMNIST3D** is a new 3D volume dataset to classify whether a synapse is excitatory or inhibitory. It uses a 3D image volume of an adult rat acquired by a multi-beam scanning electron microscope. The original data is of the size 100 × 100 × 100 um$^3$ and the resolution 8 × 8 × 30 nm$^3$, where a (30um)$^3$ sub-volume was used in the MitoEM dataset with dense 3D mitochondria instance segmentation labels. Tree neuroscience experts segment a pyramidal neuron within the whole volume and proofread all the synapses on this neuron with excitatory / inhibitory labels. For each labeled synaptic location, a 3D volume of 1024 × 1024 × 1024 nm$^3$ is cropped and resized into 28 × 28 × 28 voxels. Finally, the dataset is randomly split with a ratio of 7:1:2 into training, validation and test set.

### 2.2.2: Experimental Framework:

- For our experiments, we utilized a combination of frameworks and libraries to implement the Deep AUC Maximization (**DAM**) technique and evaluate its performance on medical image classification tasks.

- We made use of the **LibAUC library**, which provides a set of functions for calculating the area under the receiver operating characteristic curve (AUC) and for optimizing models based on the AUC metric. We used this library to train our models with the DAM loss function and to compute the AUC score during evaluation.

- For our **2D** image datasets, we used **ResNet18 architecture**, which is a widely used convolutional neural network architecture for image classification tasks.

- For **3D** image datasets, we used **ResNet18 3D architecture**, which is an extension of the **2D ResNet** architecture designed for **3D** medical image data by modifying the first layer of the convolutional neural network.

- We implemented our experiments using the **PyTorch** framework, which is an open-source machine learning framework that provides efficient tensor computations on GPUs. We utilized **PyTorch's** built-in functions for data loading, training, and evaluation.

- To visualize our results and monitor the training process, we used **TensorBoard**, which is a tool for visualizing and analyzing machine learning experiments. We used **TensorBoard** to track various metrics such as loss, AUC score, and accuracy during training, and to visualize the network architecture and model parameters.

- We used the **AUCM loss function** (Area Under the Receiver Operating Characteristic Curve Maximization), which is designed to directly optimize the AUC. We also compare the performance of DAM with other methods, such as cross-entropy loss, which is commonly used in classification tasks.

- We additionally experimented with two different optimizers: **PESG** (Population-based Evolutionary Strategy with Gradient) and **ADAM** (Adaptive Moment Estimation). PESG is an evolutionary optimizer that uses population-based strategies to optimize the weights of the neural network. ADAM is a popular optimizer that uses adaptive learning rates to improve convergence speed and generalization ability.

- We set the **random seed to 0** before running our experiment to ensure that our results can be reproduced if the experiment is run again using the same seed. However, it should be noted that setting the seed to 0 does not guarantee complete reproducibility (as per PyTorch), as there may still be variations due to factors such as differences in hardware and software configurations.

# Chapter 3: Methodologies (Results and Analysis)

## 3.1: Method 1 - Performance Tuning for PESG Optimizer

### 3.1.1: Motivation

- To bring out the maximum possible performance of deep neural networks, we need to correctly identify the relevant hyperparameters and tune them. Different values of these hyperparameters can lead to vastly different results.

- In this section we focus on maximizing the performance of the network by tuning PESG optimizer parameters. Particularly we focus on the learning rate and batch size.

- The **learning rate** controls the step size of the updates to the model weights during training. Increasing the learning rate can speed up the training process and allow the model to converge faster. However, if the learning rate is set too high, the model may overshoot the optimal weights and fail to converge or become unstable. Also, while decreasing the learning rate can lead to a more stable training process and prevent the model from overshooting the optimal weights. If the learning rate is set too low, the model may take a long time to converge, or it may get stuck in a suboptimal solution.

- The **batch size** determines the number of samples that are used to update the model weights in each training iteration. Increasing the batch size can lead to faster convergence and reduce the variance of the gradient updates. Decreasing the batch size can lead to more noisy gradient updates, which can result in slower convergence and larger variance in the gradient estimates. However, using smaller batches can help the model to escape from suboptimal local minima more easily, which can improve the overall performance of the model.

### 3.1.2: Procedure

- By performing a **grid search**, we can identify the hyperparameters that provide the best performance for each dataset.

- We performed a grid search over the hyperparameters of learning rate and batch size.

- We tested for four different learning rates [1, 0.1, 0.01, 0.001] and four different batch sizes [16, 32, 64, 128] for a total of 16 combination of iterations for each of the seven datasets.

### 3.1.3: Results

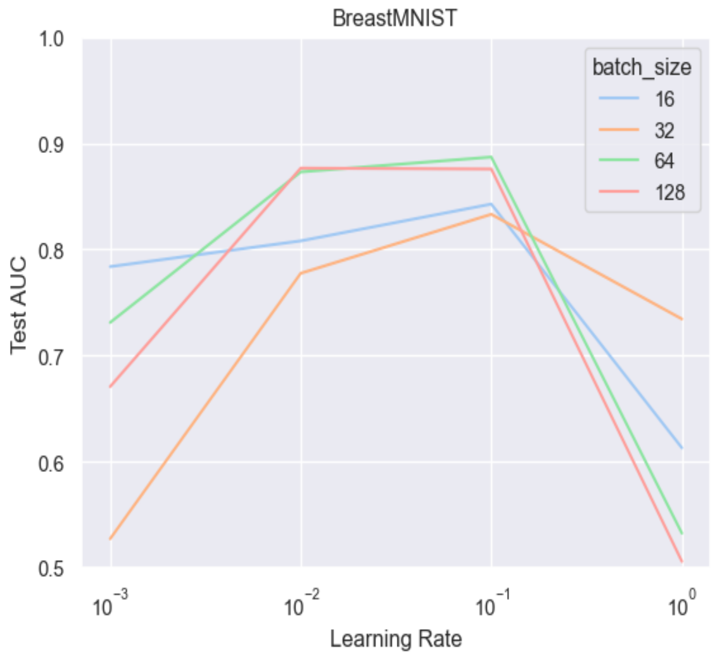

*Figure 0-1.1: Learning rate vs Test AUC, BreastMNIST*

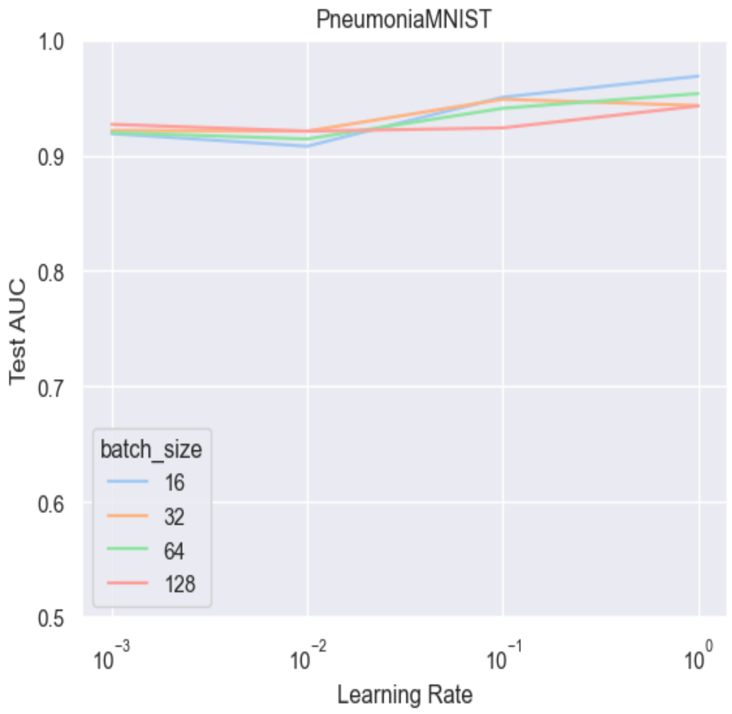

*Figure 1.2: Learning rate vs Test AUC, PneumoniaMNIST*

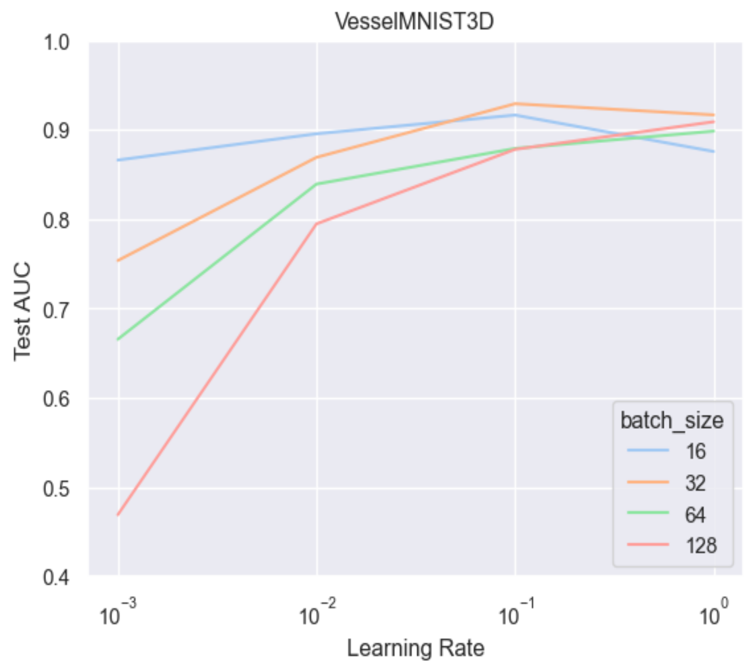

*Figure 1.3: Learning rate vs Test AUC, VesselMNIST3D*

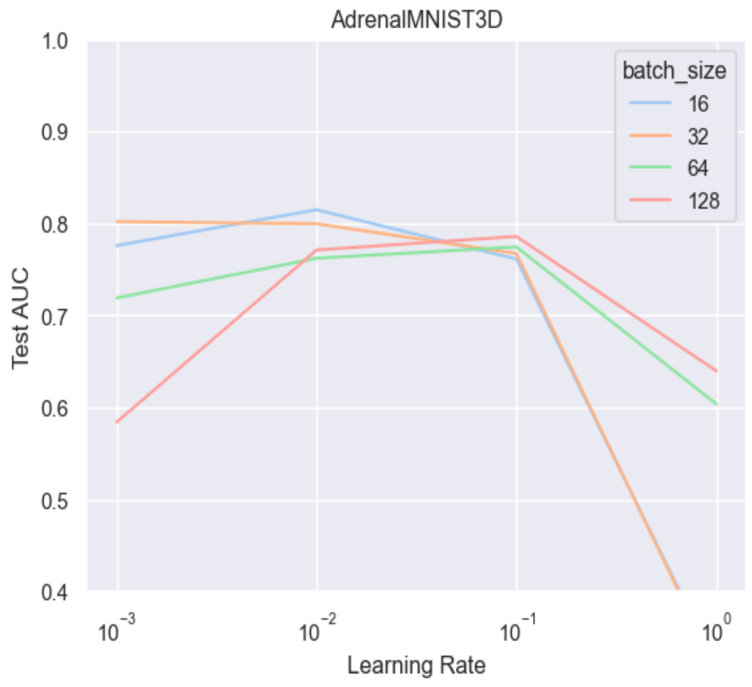

*Figure 1.4: Learning rate vs Test AUC, AdrenalMNIST3D*

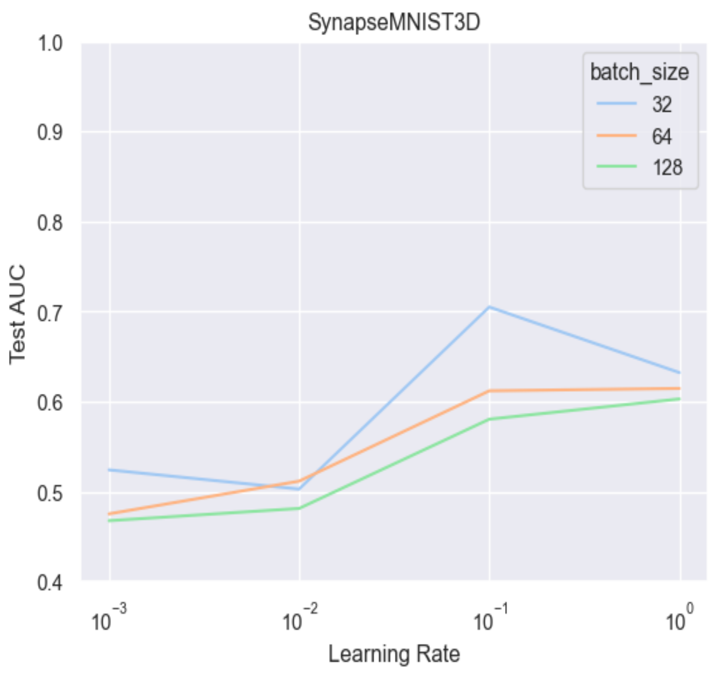

*Figure 1.5: Learning rate vs Test AUC, SynapseMNIST3D*

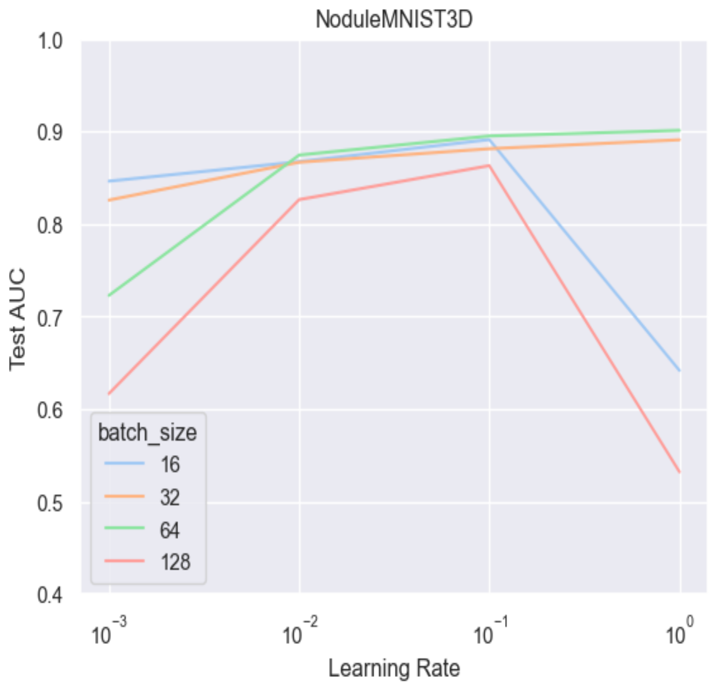

*Figure 1.6: Learning rate vs Test AUC, NoduleMNIST3D*

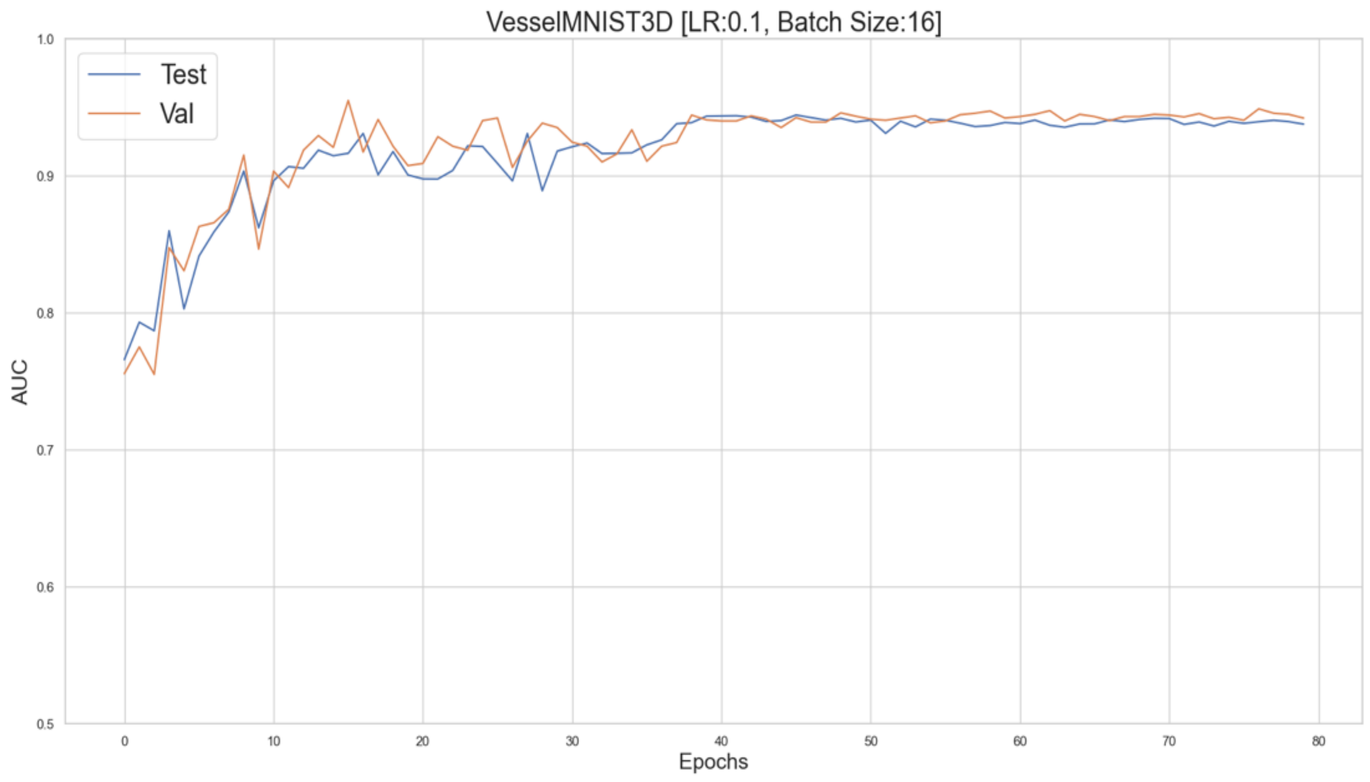

*Figure 1.7: No. of Epochs vs Test AUC, VesselMNIST3D on TensorBoard Visualization (Sample of a particular expt.)*

### 3.1.4: Result Tables

| **DATASET** | **SOTA Benchmark** AUC (Acc.) | **Method 1** AUC (Acc.) |
|---|---|---|
| **BreastMNIST** | 0.901 (0.863) | 0.887 (0.852) |
| **PneumoniaMNIST** | 0.944 (0.854) | **0.968 (0.894)** |
| **AdrernalMNIST3D** | 0.827 (0.721) | 0.815 (**0.762**) |
| **VesselMNIST3D** | 0.874 (0.877) | **0.929 (0.911)** |
| **NoduleMNIST3D** | 0.863 (0.844) | **0.901 (0.861)** |
| **SynapseMNIST3D** | 0.820 (0.754) | 0.705 (0.687) |
| **ChestMNIST** | 0.768 (0.947) | 0.546 (0.947) |

*Table 1.1: Results for Method 1 w.r.t SOTA Benchmark*

:

| DATASET | Best LR | Best Batch Size |
|---|---|---|
| **BreastMNIST** | 0.1 | 64 |
| **PneumoniaMNIST** | 1 | 16 |
| **AdrernalMNIST3D** | 0.01 | 16 |
| **VesselMNIST3D** | 0.1 | 32 |
| **NoduleMNIST3D** | 1 | 64 |
| **SynapseMNIST3D** | 0.1 | 32 |
| **ChestMNIST** | 1 | 512 |

*Table 1.2: Best hyperparameters for each dataset, Method 1*

### 3.1.5: Analysis

- Hyperparameter tuning using LR and Batch size improved the performance of PneumoniaMNIST, VesselMNIST3D, and NoduleMNIST3D datasets.

- The maximum increase in AUC / Accuracy was observed in the case of VesselMNIST3D dataset, which has the highest-class imbalance (Positive class = 1185, Negative Class =150) by a factor of 8.

- This suggests that DAM is better equipped to handle highly imbalanced datasets compared to optimizing CE loss.

- Hyperparameter tuning alone was sufficient to surpass the benchmark on three datasets, demonstrating the capability of DAM to better learn highly imbalanced classes.

## 3.2: Method 2 - Modification of Validation Set

### 3.2.1: Motivation

- The datasets exhibit a significant variation in size, ranging from 546 training samples in BreastMNIST to around 80,000 training samples in ChestMNIST. On a detailed analysis, we found out that both the training and validation set contain equal distribution of positive and negative samples on all datasets.

- Deep neural networks like ResNet can easily overfit on small datasets, which makes it necessary to increase the number of training samples in such cases. Hence it might be beneficial to increase the number of training samples in case of small datasets like BreastMNIST.

- Additionally, having a sufficiently large validation set is crucial to accurately evaluate the performance of our models. A larger validation set may help to better estimate how the model will perform on unseen data, which can be especially important when working with small datasets. To find the best training-validation set combination, we perform a transfer of samples from one bucket to another.

### 3.2.2: Procedure

Here are two methods used to determine the best combination of training and validation sets:

- Halving validation set: Half of the samples in the validation set are randomly selected and transferred to the training set, resulting in a larger training set and a smaller validation set.

- Doubling validation set: Samples are randomly selected from the training set (equal to the number of samples in the validation set) and moved to the validation set, resulting in a smaller training set and a larger validation set.

- For each dataset and method, we tuned the hyperparameters, learning rate and batch size, with learning rate values of [1, 0.1, 0.01, 0.001] and batch size values of [16, 32, 64, 128].

- This resulted in a total of 32 iterations for each dataset, with 16 iterations for each method.

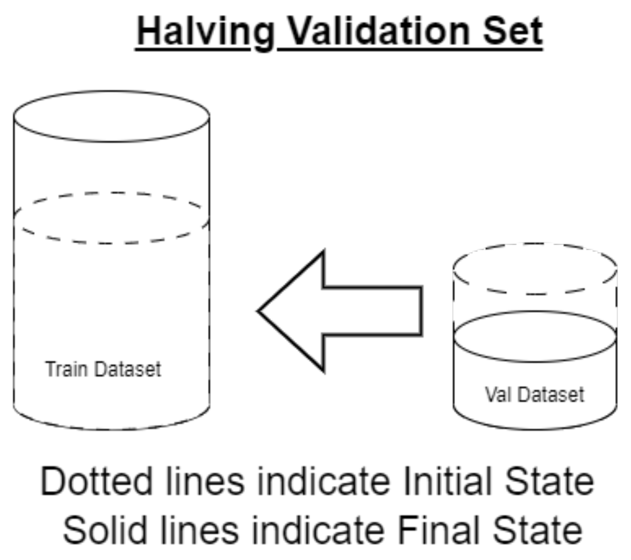
Halving Validation Set
Train Dataset
Val Dataset
Dotted lines indicate Initial State
Solid lines indicate Final State

*Figure 2.1: Halving the validation set*

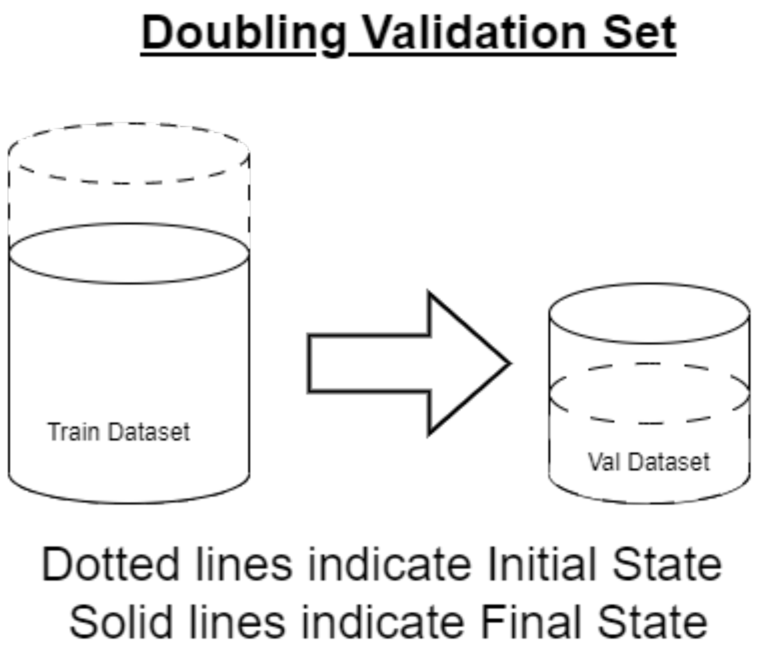
Doubling Validation Set
Train Dataset
Val Dataset
Dotted lines indicate Initial State
Solid lines indicate Final State

*Figure 2.2: Doubling the validation set*

### 3.2.3: Results

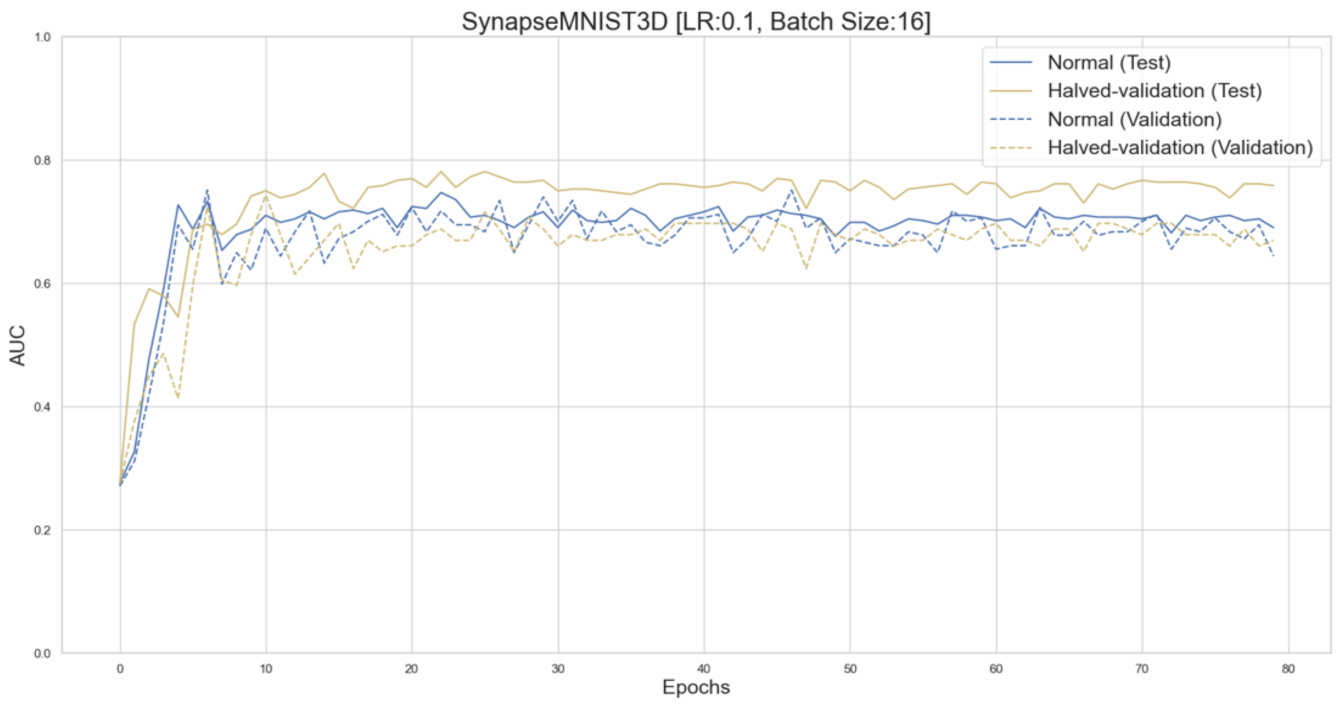

*Figure 2.3: No. of Epochs vs Test AUC, SynapseMNIST3D on TensorBoard Visualization (Sample of a particular expt.) for method 2*

### 3.2.4: Result Tables

| DATASET | SOTA Benchmark AUC (Acc.) | Hyperparameter Tuning AUC (Acc.) | Data Modification AUC (Acc.) |
|---|---|---|---|
| **BreastMNIST** | 0.901 (0.863) | 0.887 (0.852) | 0.891 (0.878) D |
| **PneumoniaMNIST** | 0.944 (0.854) | **0.968 (0.894)** | **0.968** (0.887) H |
| **AdrernalMNIST3D** | 0.827 (0.721) | 0.815 (**0.762**) | **0.836** (0.752) D |
| **VesselMNIST3D** | 0.874 (0.877) | **0.929 (0.911)** | **0.936 (0.937) H** |
| **NoduleMNIST3D** | 0.863 (0.844) | **0.901 (0.861)** | **0.885 (0.845) H** |
| **SynapseMNIST3D** | 0.820 (0.754) | 0.705 (0.687) | 0.796 (0.738) H |
| **ChestMNIST** | 0.768 (0.947) | 0.546 (0.947) | 0.521 (0.732) |

*Table 2.1: Results for Methods 1 & 2 w.r.t SOTA Benchmark*

| DATASET | Best LR | Best Batch Size | Data Modification |
|---|---|---|---|
| **BreastMNIST** | 0.1 | 32 | Double Validation |
| **PneumoniaMNIST** | 1 | 16 | Nil |

| DATASET | Best LR | Best Batch Size | Data Modification |
|---|---|---|---|
| **AdrernalMNIST3D** | 1 | 32 | Double validation |
| **VesselMNIST3D** | 0.1 | 16 | Halve Validation |
| **NoduleMNIST3D** | 1 | 64 | NIL |
| **SynapseMNIST3D** | 0.1 | 16 | Halve Validation |
| **ChestMNIST** | 1 | 512 | NIL |

*Table 2.2: Best hyperparameters for each dataset, Method 2*

#### 3.2.5: Analysis

- In Method 2, we explored the impact of modifying the validation set size on the performance of our models. The results showed that doubling the validation set size improved the performance for datasets with a small validation data size such as BreastMNIST and AdrenalMNIST. This indicates that increasing the validation set size can help in improving the model's performance by providing more data for validation during the training process.

- On the other hand, halving the dataset improved the performance for SynapseMNIST3D and BreastMNIST by approximately 9%. This suggests that a smaller validation set can also be beneficial for certain datasets, possibly due to reducing overfitting during training.

- Overall, Method 2 was highly successful in optimizing the model performance for the datasets considered in this study. By modifying the validation set size, we were able to achieve significant improvements in model performance without the need for additional computational resources.

### 3.3: Method 3 – Data Augmentation

#### 3.3.1: Motivation

- Imbalanced classes: Some datasets, such as AdrenalMNIST3D and VesselMNIST3D, have imbalanced classes, which can lead to biased predictions. Data augmentation can help balance the classes by generating synthetic samples of the underrepresented class, improving the performance of the model.

- Limited variability: Medical imaging datasets often have limited variability in terms of angle, lighting, and position. Data augmentation techniques such as rotation, flipping, and scaling can help introduce more variability in the dataset, making the model more robust to variations in the real-world scenarios.

- Generalization: Data augmentation can help the model generalize better to unseen data by exposing it to a wider variety of samples. This can help prevent overfitting and improve the model's ability to perform well on real-world data.

#### 3.3.2: Procedure

For 2D – Images, we primarily used the functions available from the transforms library:

- ToTensor(): Converts the image to a tensor.
- transforms.ToPILImage(): Converts the tensor to a PIL image.
- transforms.Grayscale(3): Converts the image to grayscale with 3 channels.
- transforms.RandomHorizontalFlip(p=0.5): Randomly flips the image horizontally with a probability of 0.5.
- transforms.RandomRotation(degrees=30): Rotates the image randomly within 30 degrees.
- transforms.ColorJitter(brightness=0.1, contrast=0.1): Randomly adjusts the brightness and contrast of the image.
- transforms.ToTensor(): Converts the image back to a tensor.
- transforms.Normalize(mean=[0.5], std=[0.5]): Normalizes the pixel values with a mean of 0.5 and standard deviation of 0.5.
- transforms.Resize((32, 32)): resizes the input image to a fixed size of 32x32 pixels.

For 3D – Images, we primarily used the functions available from torchio library.

- tio.RandomAffine() - to perform random affine transformations over the image.
- tio.RandomFlip() - to perform random flip along the axes.
- tio.RandomBlur() - to perform random Gaussian blurring over the image.

## 3.3.3: Results

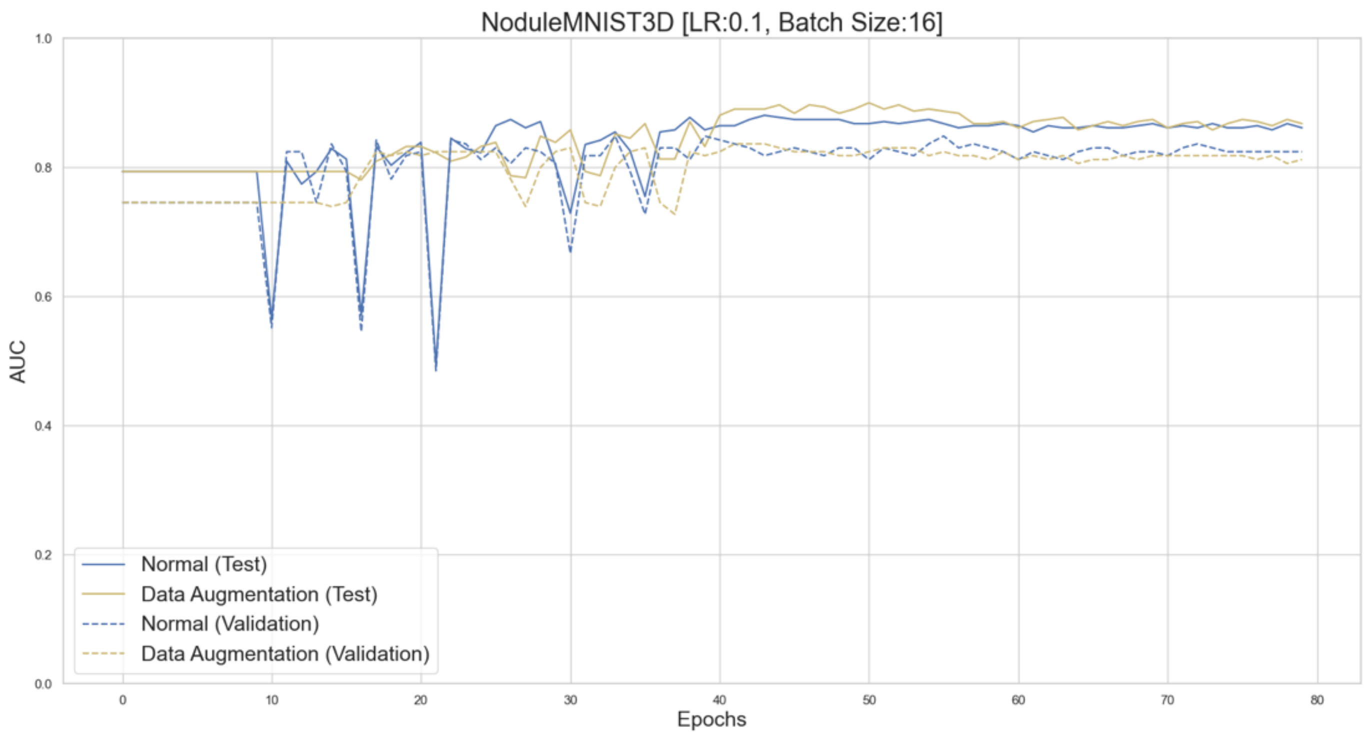

*Figure 3.1: No. of Epochs vs Test AUC, NoduleMNIST3D on TensorBoard Visualization (Sample of a particular expt.) for method 3*

## 3.3.4: Result Tables

| DATASET | SOTA Benchmark AUC (Accuracy) | Hyperparameter Tuning AUC (Accuracy) | Data Modification AUC (Accuracy) | Data Augmentation AUC (Accuracy) |
|---|---|---|---|---|
| **BreastMNIST** | 0.901 (0.863) | 0.887 (0.852) | 0.891 (0.878) D | 0.805 (0.737) |
| **PneumoniaMNIST** | 0.944 (0.854) | **0.968 (0.894)** | **0.968 (0.887) H** | **0.958 (0.902)** |
| **AdrernalMNIST3D** | 0.827 (0.721) | 0.815 (0.762) | **0.836 (0.752) D** | **0.845 (0.815)** |
| **VesselMNIST3D** | 0.874 (0.877) | **0.929 (0.911)** | **0.936 (0.937) H** | **0.932 (0.908)** |
| **NoduleMNIST3D** | 0.863 (0.844) | **0.901 (0.861)** | **0.885 (0.845) H** | **0.916 (0.896)** |
| **SynapseMNIST3D** | 0.820 (0.754) | 0.705 (0.687) | 0.796 (0.738) H | 0.704 (0.684) |
| **ChestMNIST** | 0.768 (0.947) | 0.546 (0.947) | 0.521 (0.732) | 0.534 (0.81) |

*Table 3.1: Results for Methods 1, 2 & 3 w.r.t SOTA Benchmark*

| DATASET | Best LR | Best Batch Size | Data Modification | Data Augmentation |
|---|---|---|---|---|
| **BreastMNIST** | 0.1 | 32 | Double Validation | Nil |

|  |  |  |  |  |
|---|---|---|---|---|
| **PneumoniaMNIST** | 1 | 16 | Nil | Nil |
| **AdrernalMNIST3D** | 0.1 | 32 | Double validation | Yes |
| **VesselMNIST3D** | 1 | 32 | Halve Validation | Yes |
| **NoduleMNIST3D** | 1 | 64 | NIL | Yes |
| **SynapseMNIST3D** | 0.1 | 16 | Halve Validation | Nil |
| **ChestMNIST** | 1 | 512 | NIL | Nil |

*Table 3.2: Best hyperparameters for each dataset, for methods 1,2 & 3*

### 3.3.5: Analysis

- From the above table, 3D datasets had an improvement in performance due to data augmentation. Whereas for 2D datasets, the augmentation couldn't help.

- This might be because in terms of augmentation there is much more change happening in 3D than in 2D and hence the impact is more visible in 3D.

- One point to keep in mind is that we haven't tuned the hyperparameters post augmentation. Hence, there is a good chance that we get better results once the hyperparameters are tuned.

## 3.4: Method 4 – Regularization, Tuning Weight Decay

### 3.4.1: Motivation

- Weight decay is a regularization technique that adds a coefficient of the L2 penalty term to the loss function. Overfitting can cause weights to be higher in magnitude than necessary, resulting in a large output change even for a small input change. L2 penalty, which is the L2 norm of the weights, addresses this problem by forcing weights to be smaller while optimizing the original loss function.

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n} \theta_i^2}$$

- Lambda is the coefficient used in weight decay to control the L2 penalty term in the loss function. By increasing lambda, the risk of overfitting is reduced, but at the expense of the model's ability to learn parameters. If lambda is too high, the model may not be able to effectively learn the underlying distribution and may result in underfitting.

### 3.4.2: Procedure

- The best set of hyperparameters for each dataset are selected from the previous experiments, while keeping all other hyperparameters constant.

- The weight_decay hyperparameter is varied to observe its effect on the model's performance. It takes on values of 1e$^{-1}$, 1e$^{-2}$, 1e$^{-3}$, and 1e-$^{5}$.

- We provide the specific value of weight_decay as an argument in the PESG loss function in the code

### 3.4.3: Results

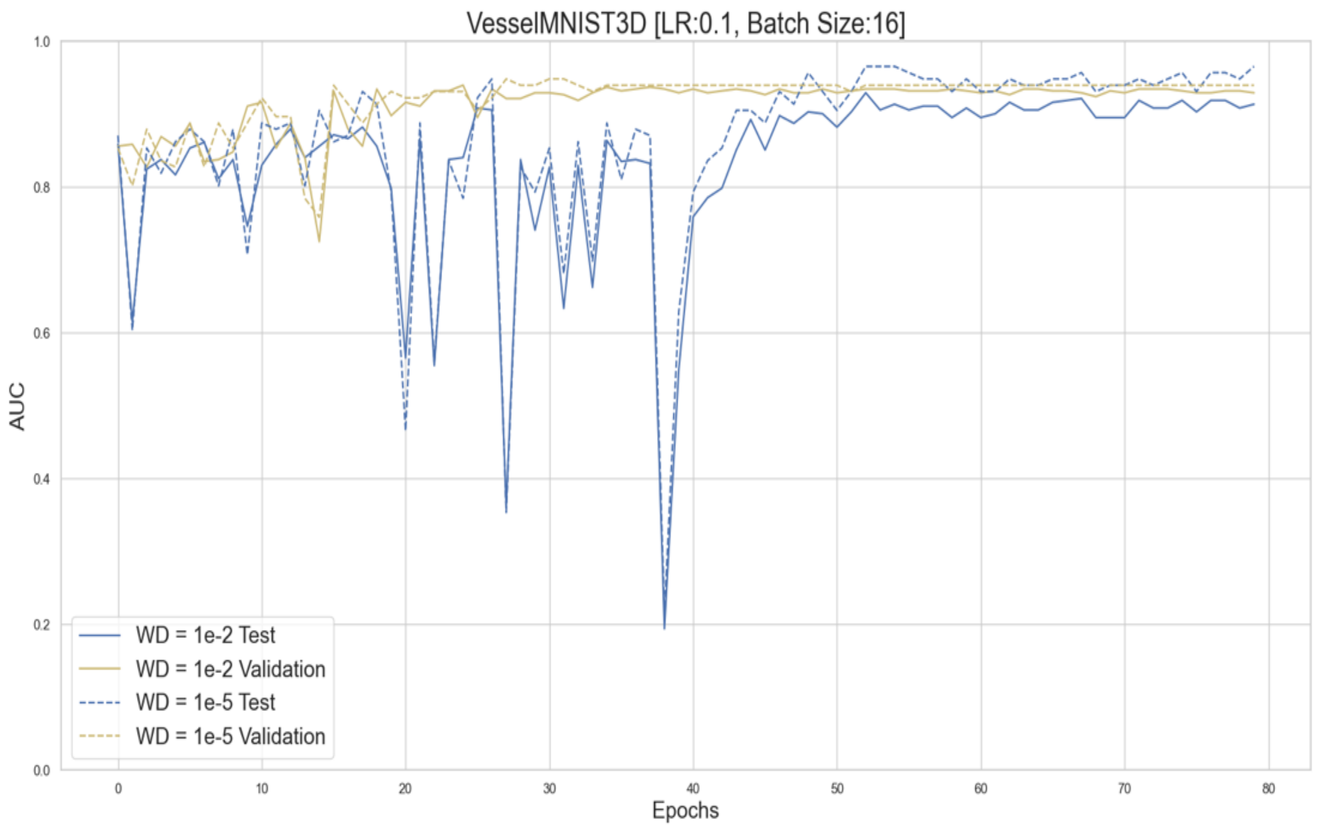

*Figure 4.1: No. of Epochs vs Test AUC, VesselMNIST3D on TensorBoard Visualization (Sample of a particular expt.) for method 4*

### 3.4.4: Result Tables

| DATASET | SOTA Benchmark AUC (Accuracy) | Hyperparameter Tuning AUC (Accuracy) | Data Modification AUC (Accuracy) | Data Augmentation AUC (Accuracy) | Weight Decay AUC (Accuracy) |
|---|---|---|---|---|---|
| **BreastMNIST** | 0.901 (0.863) | 0.887 (0.852) | 0.891 (0.878) D | 0.805 (0.737) | 0.877 (0.826) |
| **PneumoniaMNIST** | 0.944 (0.854) | **0.968 (0.894)** | **0.968 (0.887) H** | **0.948 (0.889)** | **0.969 (0.891)** |
| **AdrernalMNIST3D** | 0.827 (0.721) | 0.815 (0.762) | **0.836 (0.752) D** | **0.845 (0.815)** | **0.834 (0.768)** |
| **VesselMNIST3D** | 0.874 (0.877) | **0.929 (0.911)** | **0.936 (0.937) H** | **0.932 (0.908)** | **0.922 (0.913)** |
| **NoduleMNIST3D** | 0.863 (0.844) | **0.901 (0.861)** | **0.885 (0.845) H** | **0.916 (0.896)** | **0.884 (0.864)** |
| **SynapseMNIST3D** | 0.820 (0.754) | 0.705 (0.687) | 0.796 (0.738) H | 0.704 (0.684) | 0.723 (0.718) |
| **ChestMNIST** | 0.768 (0.947) | 0.546 (0.947) | 0.521 (0.732) | 0.534 (0.81) | 0.546 (0.947) |

*Table 4.1: Results for Methods 1, 2, 3 & 4 w.r.t SOTA Benchmark*

| DATASET | Best LR | Best Batch Size | Data Modification | Data Augmentation | Weight Decay |
|---|---|---|---|---|---|
| **BreastMNIST** | 0.1 | 32 | Double Validation | NIL | NIL |
| **PneumoniaMNIST** | 1 | 16 | NIL | NIL | 1e$^{-5}$ |
| **AdrernalMNIST3D** | 0.1 | 32 | Double validation | Yes | Nil |
| **VesselMNIST3D** | 1 | 32 | Halve Validation | Yes | NIL |
| **NoduleMNIST3D** | 1 | 64 | NIL | Yes | NIL |
| **SynapseMNIST3D** | 0.1 | 16 | Halve Validation | NIL | NIL |
| **ChestMNIST** | 1 | 512 | NIL | NIL | 1e$^{-3}$ |

*Table 4.2: Best hyperparameters for each dataset, for methods 1,2,3 & 4*

### 3.4.5: Analysis

- The experiment results show that increasing weight decay did not lead to a significant improvement in most cases.
- Only PneumoniaMNIST, AdrenalMNIST3D, and SynapseMNIST had a slight increase in performance.
- One possible reason is that early stopping already helps to prevent overfitting during training, so increasing regularization through weight decay could penalize the model too much, leading to underfitting.

## 3.5: Method 5 – Performance Tuning for ADAMS Optimizer

### 3.5.1: Motivation

- The optimizer is responsible for updating the model parameters by using the gradient of the loss function with respect to the model weights.
- Therefore, it significantly affects the learning process of neural networks.
- While the PESG optimizer is considered the best for AUCM loss function, we investigate the performance of the Adam optimizer with the same loss.

### 3.5.2: Procedure

- Learning rate and batch size are key components that can affect the performance of an optimizer.
- We perform hyperparameter tuning for the Adam optimizer to maximize its efficiency on the AUCM loss function.
- The learning rate is tested on [0.1, 0.01, 0.001, 0.0001] values, and batch size is tested on [32, 64].
- Each dataset undergoes 8 iterations, where the combination of the learning rate and batch size is tested.

### 3.5.3: Result Tables

| DATASET | SOTA Benchmark AUC (Acc) | Hyperparameter Tuning on PESG AUC (Acc) | Data Modification AUC (Acc) | Data Augmentation AUC (Acc) | Weight Decay AUC (Acc) | Hyperparameter Tuning on ADAMS AUC (Acc) |
|---|---|---|---|---|---|---|
| **BreastMNIST** | 0.901 (0.863) | 0.887 (0.852) | 0.892 (0.852) D | 0.805 (0.737) | 0.877 (0.826) | 0.656 (0.269) |
| **PneumoniaMNIST** | 0.944 (0.854) | **0.968 (0.894)** | **0.968 (0.887) H** | **0.948 (0.889)** | **0.969 (0.891)** | 0.912 (0.375) |
| **AdrernalMNIST3D** | 0.827 (0.721) | 0.815 (0.762) | **0.836 (0.752) D** | **0.845 (0.815)** | **0.834 (0.768)** | 0.784 (0.768) |
| **VesselMNIST3D** | 0.874 (0.877) | **0.929 (0.911)** | **0.936 (0.937) H** | **0.932 (0.908)** | **0.922 (0.913)** | 0.588 (0.887) |
| **NoduleMNIST3D** | 0.863 (0.844) | **0.901 (0.861)** | **0.885 (0.845) H** | **0.916 (0.896)** | **0.884 (0.864)** | 0.675 (0.793) |
| **SynapseMNIST3D** | 0.820 (0.754) | 0.705 (0.687) | 0.796 (0.738) H | 0.704 (0.684) | 0.723 (0.718) | 0.657 (0.269) |
| **ChestMNIST** | 0.768 (0.947) | 0.546 (0.947) | 0.521 (0.732) | 0.534 (0.81) | 0.546 (0.947) | 0.482 (0.246) |

*Table 5.1: Results for Method 1, 2, 3, 4 & 5 w.r.t SOTA Benchmark*

### 3.5.4: Analysis

- Adam is not a suitable optimizer for performing DAM as it showed poor performance even after hyperparameter tuning.

- This indicates that Adam is not capable of optimizing the cross-entropy components and AUC component simultaneously.

- Therefore, PESG is the most suitable loss function for our use case.

# Chapter 4 : Conclusion

In this study, we explored several methods to optimize the performance of deep learning models trained on various medical imaging datasets. We employed different techniques such as hyperparameter tuning, modification of the validation set, data augmentation, and regularization through weight decay to achieve better results.

Firstly, we applied hyperparameter tuning to optimize the performance of the PESG optimizer on three medical imaging datasets: PneumoniaMNIST, VesselMNIST3D, and NoduleMNIST3D. The results showed that hyperparameter tuning alone was sufficient to surpass the benchmark on three datasets, demonstrating the capability of DAM to better learn highly imbalanced classes. We also observed that DAM was better equipped to handle highly imbalanced datasets compared to optimizing CE loss.

Next, we explored the modification of the validation set as a method to improve model performance. We found that doubling the validation set improved the results for datasets with small validation data sizes such as BreastMNIST and AdrenalMNIST. Conversely, halving the dataset improved the performance of SynapseMNIST3D and BreastMNIST. This suggests that the amount of validation data has a significant impact on model evaluation, and that reducing the dataset can result in better performance.

We also applied data augmentation and regularization through weight decay to improve model performance. However, we found that increasing weight decay did not lead to significant improvement in most cases. Only PneumoniaMNIST, AdrenalMNIST3D, and SynapseMNIST had a slight increase in performance. We hypothesized that early stopping already helps to prevent overfitting during training, so increasing regularization through weight decay could penalize the model too much, leading to underfitting.

Finally, we evaluated the performance of the ADAM optimizer and compared it with the PESG optimizer. We found that Adam is not a good optimizer for carrying out DAM. This poor performance even after hyperparameter tuning shows that Adam doesn't have the capability to optimize between the cross-entropy components and AUC component in tandem. Hence, PESG is the most suitable loss function for us.

| DATASET | SOTA Benchmark AUC (Acc) | Our Results AUC (Acc) |
|---|---|---|
| **BreastMNIST** | 0.901 (0.863) | 0.892 (0.852) |
| **PneumoniaMNIST** | 0.944 (0.854) | **0.969** (0.891) |
| **AdrernalMNIST3D** | 0.827 (0.721) | **0.845 (0.815)** |
| **VesselMNIST3D** | 0.874 (0.877) | **0.936** (**0.937**) |
| **NoduleMNIST3D** | 0.863 (0.844) | **0.916 (0.896)** |
| **SynapseMNIST3D** | 0.820 (0.754) | 0.796 (0.738) |
| **ChestMNIST** | 0.768 (0.947) | 0.546 (0.947) |

Overall, we were able to surpass the current benchmark on 4 datasets – PneumoniaMNIST, AdrenalMNIST3D, VesselMNIST3D and NoduleMNIST3D. On the other hand, for the ChestMNIST dataset, our experiments were limited due to time and computational constraints.

In conclusion, our study highlights the importance of selecting the right optimization method and hyperparameters to achieve better results in medical imaging tasks. We also demonstrate the impact of validation data size and the limitations of weight decay regularization in deep learning models.

# Chapter 5: References

1. https://libauc.org/publications/
2. https://www.nature.com/articles/s41597-022-01721-8
3. https://github.com/Optimization-AI/LibAUC
4. https://github.com/Optimization-AI/LibAUC/blob/main/examples/08_Optimizing_AUROC_Loss_with_DenseNet121_on_Melanoma.ipynb
5. https://github.com/Optimization-AI/LibAUC/blob/main/examples/04_Training_with_Pytorch_Learning_Rate_Scheduling.ipynb
6. https://github.com/Optimization-AI/LibAUC/blob/main/examples/07_Optimizing_Multi_Label_AUROC_Loss_with_DenseNet121_on_CheXpert.ipynb
7. https://github.com/Optimization-AI/LibAUC/blob/main/examples/01_Creating_Imbalanced_Benchmark_Datasets.ipynb
8. https://pytorch.org/docs/stable/index.html