Anushka Garg : 532008694
Shilpa Mukhopadhyay : 433003777

1 The Problem Statement

This project aims to improve the generalization ability of DAM for medical image classification tasks.

1.1 Goal

To improve the benchmark performance reported in the MedMNIST paper . For fair comparison, we used the same network structure as in the MedMNIST paper i.e. Resnet-18(28).

NOTE: We are participating in the Incentive Program.

2 Experiments

2.1 2-D Data

Datasets used:

1. BreastMNIST
2. PneumoniaMNIST
3. ChestMNIST

2.1.1 Techniques Used to reduce Overfitting in 2-D datasets provided

2.1.1.1 Data Augmentation Techniques Used The code uses various image augmentation techniques to generate new images from the original ones, which can help to increase the diversity of the training set and improve the robustness of the model.

The augmentation techniques used in this code are as follows:

1. **RandomAffine** Applies random affine transformation to the image, which includes rotation, translation, scale changes, and shearing.
2. **RandomApply**: Randomly applies the given transformations to the image with a certain probability.
3. **ColorJitter**: Randomly adjusts the brightness, contrast, saturation, and hue of the image.
4. **RandomHorizontalFlip**: Randomly flips the image horizontally.
5. **RandomVerticalFlip**: Randomly flips the image vertically.
6. **RandomApply**: Randomly applies Gaussian blur with kernel size 3 to the image with a certain probability.

Overall, these augmentation techniques help to increase the diversity of the training set and improve the generalization performance of the model.

2.1.1.2 Controlling Overfitting The code is using various techniques to control overfitting. Here are some of the techniques used in the code:

1. **Regularization**: The code is using weight decay as a regularization technique to prevent overfitting. It is adding an L2 regularization term to the cost function to penalize the model's weights for being too large, which helps to avoid overfitting. The weight_decay parameter is set to 1e-5, which is the strength of the L2 penalty.
2. **Dropout**: Dropout is a popular technique for preventing overfitting in neural networks.
3. **Learning Rate Scheduling**: The code is using a StepLR scheduler, which reduces the learning rate by a factor of 0.1 after every 15 epochs. This technique helps the model to avoid overfitting by slowing down the learning rate as the training progresses, making the model less likely to overfit. The code is using a ReduceLROnPlateau scheduler as the primary scheduler, which monitors the validation AUC, and if the validation AUC does not improve(maximize) after a certain number of epochs (patience), the learning rate is reduced by a factor of 0.5. This technique helps the model to avoid getting stuck at saddle points.
4. **AUCM and CompositionalAUC Loss Functions**: The code is using the AUCM loss function for binary classes, which is designed to optimize the AUC (area under the receiver operating characteristic curve) directly, rather than indirectly by optimizing the mean squared error or binary cross-entropy loss. For the multiclass chestMNIST dataset, we used the CompositionalAUCLoss function as AUCM was not an ideal choice. These techniques help the model to avoid overfitting by directly optimizing the performance metric that is most relevant to the problem.
5. **Data Augmentation**: As discussed above, we have performed multiple data augmentation techniques which help in incorporating diversity in training data and make model more generalized which in turn results in reduced over-fitting.
6. **Optimizer tuning**: We performed extensive grid search on learning rate, batch size and weight decay on PESG/ADAM/SGD/AdamW/RMSProp q optimizers to get best parameters for every model which would reduce over-fitting. We will discuss in detail in next section.

2.1.1.3 Data Imbalance Handling The **DualSampler** is designed to handle imbalanced datasets by oversampling the minority class to balance the dataset. The sampling_rate parameter is used to control the oversampling rate of the minority class.

In this case, the DualSampler is used to create batches of size batch_size and the sampling_rate parameter is set to 0.5, which means that the minority class will be oversampled to be equal to the size of the majority class. The oversampling is done on the fly during training, which means that the data loader will generate new examples of the minority class to match the size of the majority class for each batch.

Therefore, this code handles imbalanced data by oversampling the minority class during training, which helps to ensure that the model is exposed to enough samples from the minority class to learn its characteristics properly.

2.1.1.4 Hyper Tuning

1. **Batch Sizes**: 16, 32, 64 - for binary classification and 256, 512, 1024 for multi-class classification
2. **Learning Rates** : 0.05, 0.04, 0.01, 0.001, 0.1, 0.06, 0.07, 0.08, 0.09,
3. **Epoch Decay**: 2e-3, 1e-4

4. **Weight Decay**: 1e-5, 1e-4
5. **Optimizers**: PESG, Adam, AdamW, RMS Prop, SGD
6. **Learning Rate Schedulers**: StepLR, ReduceLROnPlateau, Exponential Decay LR, LR Scheduler with Warmup Period
7. **Networks**:
   (a) model.fc = nn.Sequential(
   nn.Linear(512, 1),
   nn.Sigmoid()
)

   (b) model.fc = nn.Sequential(nn.Linear(input_channel, 128),
   nn.ReLU(),
   nn.Dropout(p=0.2),
   nn.Linear(128, 32),
   nn.ReLU(),
   nn.Dropout(p=0.1),
   nn.Linear(32, num_classes)
   nn.Sigmoid()
)

   (c) (fc): Sequential(
   (0): Linear(in_features=512, out_features=256, bias=True)
   (1): ReLU()
   (2): Linear(in_features=256, out_features=128, bias=True)
   (3): ReLU()
   (4): Linear(in_features=128, out_features=14, bias=True)
)

2.1.1.5 Co-training : Transfer learning involves using pre-trained models that have already been trained on large amounts of data. These pre-trained models have learned to generalize well, and so they can serve as a form of regularization that helps prevent overfitting to the small dataset being used for the specific task at hand. However, we have followed the restrictions as per the problem. We have co-trained using only project mentioned data with consistent data format.

2.2 3-D Data

2.2.1 Techniques Used to reduce Overfitting in 3-D datasets provided

1. NoduleMNIST3D
2. AdrenalMNIST3D
3. VesselMNIST3D
4. SynapseMNIST3D

2.2.1.1 Data Augmentation Techniques Used

The augmentation techniques used in this code are as follows:

1. **RandomAffine**: Applies a random affine transformation to the image, which includes rotation, translation, scaling, and shearing. This can help to simulate different viewing angles of the object in the image. Using scales=(0.9, 1.2) zoomed out the image, making the objects inside look small while preserving the physical size and position of the image bounds with probability= 0.5. Rotation range is set to 15 degrees with probability= 0.5.

2. **RandomGamma**: In dataset, we observed we can change contrast of images randomly for better training. We randomly changed contrast of an image by raising its values to the power γ.

3. **RandomFlip**: Flips the image horizontally or vertically with a given probability, which can help to account for different orientations of the object in the image. Reverse the order of elements in an image along the given axes.

4. **RandomNoise**: Add noise sampled from a normal distribution with random parameters.

   Overall, these augmentation techniques help to increase the diversity of the training set and improve the generalization performance of the model.

2.2.1.2 Controlling Overfitting

The code is using various techniques to control overfitting. Here are some of the techniques used in the code:

1. **Regularization**: The code is using weight decay as a regularization technique to prevent overfitting. It is adding an L2 regularization term to the cost function to penalize the model's weights for being too large, which helps to avoid overfitting. The weight_decay parameter is set to 1e-5, which is the strength of the L2 penalty.

2. **Dropout**: Dropout is a popular technique for preventing overfitting in neural networks. For some of the models introducing dropout gave better results so we added for fully connected layers of Resnet. While for some it didn't work well so we skipped there. Here, we have put dropout only for vesselmnist3d dataset.

3. **Learning Rate Scheduling**: The code is using a StepLR scheduler, which reduces the learning rate by a factor of 0.1 after every 5/10(depending on data) epochs. This technique helps the model to avoid overfitting by slowing down the learning rate as the training progresses, making the model less likely to overfit. The code is using a ReduceLROnPlateau scheduler as the primary scheduler, which monitors the validation AUC, and if the validation AUC does not improve(maximize) after a certain number of epochs (patience), the learning rate is reduced by a factor of 0.5. This technique helps the model to avoid getting stuck at saddle points. We incorporate a min_lr check of $1e-08$ to stop the training when learning rate falls below the threshold.

4. **AUCM Loss Function**: The code is using the AUCM loss function, which is designed to optimize the AUC (area under the receiver operating characteristic curve) directly, rather than indirectly by optimizing the mean squared error or binary cross-entropy loss. This technique helps the model to avoid overfitting by directly optimizing the performance metric that is most relevant to the problem.

5. **Data Augmentation**: As discussed above, we have performed multiple data augmentation techniques which help in incorporating diversity in training data and make model more generalized which in turn results in reduced over-fitting.

6. **Optimizer tuning**: We performed extensive grid search on learning rate, batch size and weight decay on PESG/ADAM/SGD optimizers to get best parameters for every model which would reduce over-fitting. We will discuss in detail in next section.

7. **Sampling**: We observed that most of the datasets were imbalanced by being heavily skewed towards one of the classes. So, we used DualSampler with sampling rate 0.5 for training data to reduce intrinsic data bias and over-fitting.

2.2.1.3 Data Imbalance Handling

The **DualSampler** is designed to handle imbalanced datasets by oversampling the minority class to balance the dataset. The sampling_rate parameter is used to control the oversampling rate of the minority class. We have set sampling rate to 0.5 since it is best to set same amount of training data for both classes.

In this case, the DualSampler is used to create batches of size batch_size and the sampling_rate parameter is set to 0.5, which means that the minority class will be oversampled to be equal to the size of the majority class. The oversampling is done on the fly during training, which means that the data loader will generate new examples of the minority class to match the size of the majority class for each batch.

Therefore, this code handles imbalanced data by oversampling the minority class during training, which helps to ensure that the model is exposed to enough samples from the minority class to learn its characteristics properly.

2.2.1.4 Hyper-parameters Tuning

1. **Batch Sizes**: 16, 32, 64: since most of the datasets had training data of around ∼1200 images only.

2. **Learning Rates** : 0.1, 0.05, 0.001, 0.005, 1e-03, 1e-04, 3e-04.

3. **Weight Decay**: 1e-5, 1e-3.

4. **Optimizers**: PESG, Adam, RMSProp, SGD: PESG and ADAM were taken from LibAUC and for other 2 we found this uppon research that these mostly work well for image data.

5. **Learning Rate Schedulers**: StepLR, ReduceLROnPlateau, CosineAnnealing, CosineAnnealingWarmRestarts: since all optimize differently.

6. **Networks**:

   (a) Modified the FC layer of ResNet18-3d to give output for binary classification. This was our final selected model architechture.
   (fc): Sequential(
   (0): Linear(in_features=512, out_features=128, bias=True)
   (1): ReLU()
   (2): Dropout(p=0.2, inplace=False)
   (3): Linear(in_features=128, out_features=32, bias=True)
   (4): ReLU()
   (5): Dropout(p=0.1, inplace=False)
   (6): Linear(in_features=32, out_features=1, bias=True)
   (7): Sigmoid()
)

   (b) Initially we tried to train the 3-D data(1 x 28 x 28 x 28) using ResNet18(Conv2D) by changing the input layer to accept 28 channels instead of 3 channels and converted the input data to 28 x 28 x 28. However, we were trying to use 2D convolution on a 3D image data which led to

loss of important features during the initial steps resulting in low AUC. Hence, we opted for the previously mentioned architecture.
ResNet(
(conv1): Conv2d(28, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
(relu): ReLU(inplace=True)
(maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
(layer1): Sequential(
.
.
.
(fc): Sequential(
(0): Linear(in_features=512, out_features=128, bias=True)
(1): ReLU()
(2): Linear(in_features=128, out_features=1, bias=True)
(3): Sigmoid()
)

2.2.1.5 Co-training : Transfer learning involves using pre-trained models that have already been trained on large amounts of data. These pre-trained models have learned to generalize well, and so they can serve as a form of regularization that helps prevent overfitting to the small dataset being used for the specific task at hand. However, we have followed the restrictions as per the problem. We have co-trained using only project mentioned data with consistent data format.

3 Result Comparison with Baseline Estimate

Table 1: Comparison Table with Baseline Estimate

Datasets	Ours	Baseline
BreastMNIST	**0.925**	0.901
PneomoniaMNIST	**0.973**	0.944
ChestMNIST	0.727	**0.768**
NoduleMNIST3D	**0.926**	0.915
AdrenalMNIST3D	**0.868**	0.827
VesselMNIST3D	**0.965**	0.874
SynapseMNIST3D	**0.862**	0.820

4 Result Analysis

1. **BreastMNIST**

   There were around 546 train images with 73% of data belonging to one class. Hence, we performed sampling of training data with sampling rate = 0.5. Given the dataset was small, we tried with smaller batch sizes of [16, 32] and found batch size= 32 to be optimal. As we were using StepLR and ReduceLROnPlateau as mentioned in section 2.1.1.2.3, so we started with a high learning rate of 0.01 as it helped in converging faster for initial epochs. We tried with weight decay of 2e-03 and 1e-05 and

found the later to be performing better. We also experimented with introducing dropout in the FC layer. However, we found the architecture in 2.1.1.4.7(b) with the dropout layers to be performing the best. We performed co-training for 30 epochs to improve test AUC from 0.901 to 0.925. The model could be found at saved_models/aucm_trained_model_breastmnist.pth.

2. **PneumoniaMNIST**

   There were around 4708 train images with 74% of data belonging to one class. Hence, we performed sampling of training data with sampling rate = 0.5. Given the dataset was small, we tried with smaller batch sizes of [16, 32] and found batch size= 32 to be optimal. As we were using StepLR and ReduceLROnPlateau as mentioned in section 2.1.1.2.3, so we started with a high learning rate of 0.01 as it helped in converging faster for initial epochs. We tried with weight decay of 2e-03 and 1e-05 and found the later to be performing better. We also experimented with introducing dropout in the FC layer. However, we found the architecture in 2.1.1.4.7(b) with the dropout layers to be performing the best. We performed co-training for 30 epochs o improve test AUC from 0.944 to 0.973. The model could be found at saved_models/aucm_trained_model_pneumoniamnist.pth.

3. **ChestMNIST**

   There were around 78,468 train images. As there were 14 classes with class imbalance, we performed sampling of training data with sampling rate = 0.5. Given the dataset was quite large than any of other datasets, we tried with batch sizes of [256, 512, 1024] and found batch size = 512 to be optimal. We used StepLR and ReduceLROnPlateau as mentioned in section 2.2.1.2.3. Given the data was much complex than any other datasets, we started with a relatively small learning rate of 0.05 as it helped in learning the complex decision boundary for multiple classes. We tried with weight decay of 1e-03 and 1e-05 and found the later to be performing better. We also experimented with introducing dropout in the FC layer. We used CompisitionalAUCLoss as the loss function as the problem is a multi-class problem. We found the architecture in 2.1.1.4.7(c) without the dropout layers to be performing the best. We performed co-training for 20 epochs to get a test AUC of 0.727. The model could be found at saved_models/chestmnist_compositionalauc_512bs_0.05_SLR_RLRP_7275.pth. However, we couldn't improve the model AUC from what was mentioned in the paper due to time constraints. We believe we could get a better model if we could do some more hyperparameter tuning as the data is really large and being multiclass, its a complex task. Also the traditional AUCMLoss didn't do better due to which we had to go to CompositionalAUCLoss after going through the LibAUC code base.

4. **NoduleMNIST3D**

   There were around 1100 train images with 74% of data belonging to one class. Hence, we performed sampling of training data with sampling rate = 0.5. Given the dataset was small, we tried with smaller batch sizes of [16, 32, 64] and found batch size= 32 to be optimal. As we were using StepLR and ReduceLROnPlateau as mentioned in section 2.2.1.2.3, so we started with a high learning rate of 0.1 as it helped in converging faster for initial epochs. We tried with weight decay of 1e-03 and 1e-05 and found the later to be performing better. We also experimented with introducing dropout in the FC layer. However, we found the architecture in 2.2.1.4.6(a) without the dropout layers to be performing the best. We performed co-training for 20 epochs to get a test AUC of 0.9263. The model could be found at saved_models/nodule3d_resnet3d_pesg_0.1_SLR_RLRP_noDP_9263.pth. Hence, we improved the model AUC from 0.915 to 0.9263.

5. **AdrenalMNIST3D**

   There were around 1200 train images with 78% of data belonging to one class. Hence, we performed sampling of training data with sampling rate = 0.5. Given the dataset was small, we tried with smaller batch sizes of [16, 32, 64] and found batch size= 32 to be optimal. As we were using StepLR and ReduceLROnPlateau as mentioned in section 2.2.1.2.3, so we started with a high learning rate of 0.1 as it helped in converging faster for initial epochs. We tried with weight decay of 1e-03 and 1e-05 and

found the later to be performing better. We also experimented with introducing dropout in the FC layer. However, we found the architecture in 2.2.1.4.6(a) without the dropout layers to be performing the best. We performed co-training for 20 epochs to get a test AUC of 0.8676. The model could be found at saved_models/adrenal3d_resnet3d_pesg_0.1_SLR_RLRP_noDP_8676.pth. Hence, we improved the model AUC from 0.827 to 0.8676.

6. **VesselMNIST3D**

   There were around 1300 train images with ∼89% of data belonging to one class. Hence, we performed sampling of training data with sampling rate = 0.5. Given the dataset was small, we tried with smaller batch sizes of [16, 32, 64] and found batch size= 32 to be optimal. As we were using StepLR and ReduceLROnPlateau as mentioned in section 2.2.1.2.3, so we started with a high learning rate of 0.1 as it helped in converging faster for initial epochs. We tried with weight decay of 1e-03 and 1e-05 and found the later to be performing better. We also experimented with introducing dropout in the FC layer. However, we found the architecture in 2.2.1.4.6(a) to be performing the best.We performed co-training for 20 epochs to get a test AUC of 0.9652. The model could be found at saved_models/vessel3d_resnet3d_pesg_0.1_SLR_RLRP_9652.pth. Hence, we improved the model AUC from 0.874 to 0.9652.

7. **SynapseMNIST3D**

   There were around 1200 train images with ∼77% of data belonging to one class. Hence, we performed sampling of training data with sampling rate = 0.5. Given the dataset was small, we tried with smaller batch sizes of [16, 32, 64] and found batch size= 16 to be optimal. Even though we were using StepLR and ReduceLROnPlateau as mentioned in section 2.2.1.2.3, we started with a learning rate of 0.05 as it balanced the convergence as well as gave a overall better validation AUC. We tried with weight decay of 1e-03 and 1e-05 and found the later to be performing better. We also experimented with introducing dropout in the FC layer. However, we found the architecture in 2.2.1.4.6(a) without the dropout layers to be performing the best.We performed co-training for 30 epochs to get a test AUC of 0.8623. The model could be found at saved_models/synapse3d_resnet3d_pesg_0.05_SLR_RLRP_16batchsize_8632.pth. Hence, we improved the model AUC from 0.820 to 0.8623.