Classification Report

Srinivas Natarajan
Arizona State University
snatar28@asu.edu

1. Project 1 (Colonoscopy)

1.1. Colonoscopy Video Quality Assessment (Classification)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can assess frames of a colonoscopy procedure and determine its quality. This is important as good quality frames are needed for the localization and segmentation of polyps.

1.1.1 Dataset Exploration

The dataset used is from the University of Toronto in 2015. It consists of 1,062 images with an equal class distribution. The sample images of the clear and burry images of the colonoscopy video are given below.

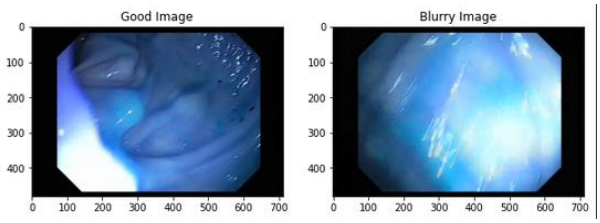

Fig 1: Class Images

The images are of the shape 480x712x3 which is larger than we need. The preprocessing I do is explained below.

1.1.2 Data Processing

This situation can be seen as a binary classification problem where each frame is classified as 'clear' (label 0) or 'blurry' (label 1). To process the images, I first create a custom python class to represent it. Each instance stores the label as well as a processed image using OpenCV and PyVison's transforms. The images are resized to a 256x379 dimension and gray scaled to improve performance and converted to Tensors. This data is split as 80-20 for the training and testing sets

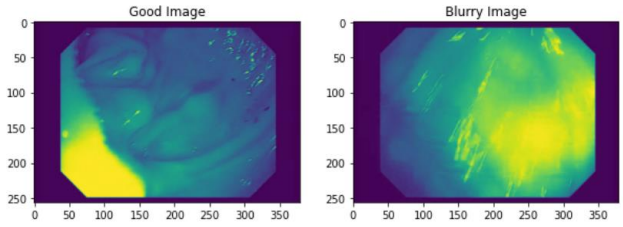

Fig 2: Processed Image

1.1.3 Model

The model I chose for this problem is the ResNet-18. I chose this over larger models as when running video computation, speed is also a critical factor. It is modified for this resolution with two output nodes. In addition, I added dropout layers (p=0.2) in between the dense layers to prevent any overfitting.

```
   | Name  | Type             | Params
-------------------------------------------
0  | model | ResNet           | 11.7 M
1  | out1  | Linear           | 200 K
2  | out2  | Dropout          | 0
3  | out   | Linear           | 402
4  | loss  | CrossEntropyLoss | 0
-------------------------------------------
11.9 M    Trainable params
0         Non-trainable params
11.9 M    Total params
47.535    Total estimated model params size (MB)
```

Fig 3: Model Summary

The Adam optimizer is used with a learning rate of 0.01. The loss function used is the standard cross entropy loss.

1.1.4 Results

The model is trained for 23 epochs. Any more resulted in overfitting. The training is accelerated with the help of a GPU.

```
Epoch 22: 100% 54/54 [00:14<00:00, 3.68it/s, loss=0.197, v_num=1
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=2
```

Fig 4: Model training

Of the multiple runs done, the best result achieved is an accuracy of 91.080% and an F-score of 0.911. As possible improvements for this problem, we could finetune the

model a little more by altering the learning rate over time.

	precision	recall	f1-score	support
0	0.94	0.95	0.94	168
1	0.80	0.78	0.79	45
accuracy			0.91	213
macro avg	0.87	0.86	0.87	213
weighted avg	0.91	0.91	0.91	213

Fig 5: Classification report of the model

The overall metrics of the model can be seen in the included graph with an average AUC score of 0.862. The class predictions are visualized through a heatmap as shown below.

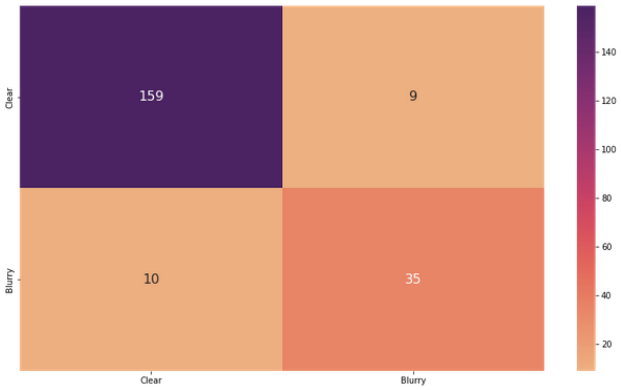

Fig 6: Confusion Matrix of the predictions

1.2. Polyp Segmentation (Segmentation)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can accurately segment the boundaries of polyps in the gastrointestinal tract (GI)

1.2.1 Dataset Exploration

The Clinic-CVC dataset is used for this exercise. It consists of 612 images of frames from a colonoscopy procedure, each consisting of an image an image and a mask indicating the boundaries of the polyp. The representation can be seen below.

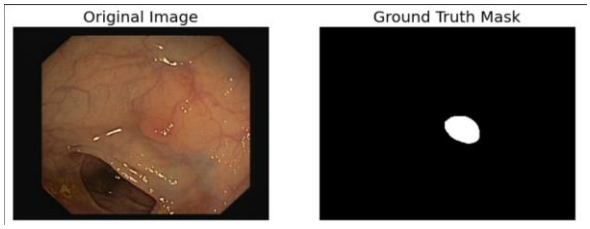

Fig 7: Data set images provided

1.2.2 Data Processing

The data is resized to a 288x384 dimension and converted to gray scale using OpenCV. Due to the small size of the data set, I decided to augment the training data using Horizontal flips, vertical flips and zooms.

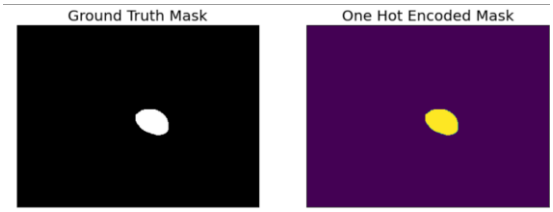

Fig 8: Processed masks

I create a custom python class to represent it. Each instance stores the label as well as a processed image using OpenCV and PyVison's transforms. Augmentation is handled with PyTorch's Albumentations subpackage. This data is then split as 80-10-10 for the training, validation and testing sets.

1.2.3 Model

I chose the UNet, UNet++ and the DeepLabv3 models for this task. First for the UNet model, I used the ResNet50 structure for the encoder architecture with the ImageNet weights. I used Dice Loss for the loss function and IoU for the score. The Adam Optimizer is used with a learning rate of 0.0001 and a learning rate scheduler to reduce the learning rate once it plateaus.

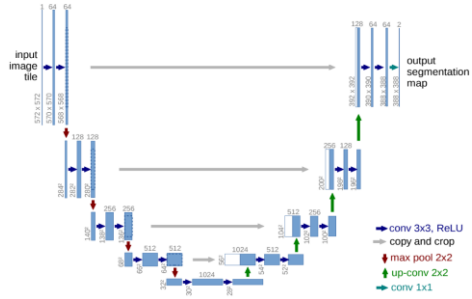

Fig 9: UNet Model Architecture

For the UNet ++ model and DeepLabv3, I use similar parameters to make sure I can understand the difference between these models.

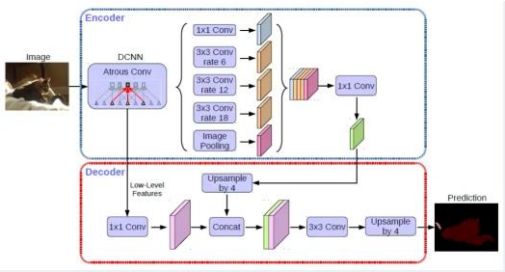

Fig 10: DeepLabv3 Architecture

1.2.4 Results

The models are each trained for 10 epochs with a batch size of 16. Any more resulted in overfitting. The training is accelerated with the help of a GPU.

```
Epoch: 2
train: 100%|██████████| 31/31 [00:21<00:00,  1.44it/s, dice_loss - 0.0516, iou_score - 0.9783]
valid: 100%|██████████| 4/4 [00:02<00:00,  1.98it/s, dice_loss - 0.05094, iou_score - 0.9735]
Model saved!

Epoch: 3
train: 100%|██████████| 31/31 [00:21<00:00,  1.45it/s, dice_loss - 0.04427, iou_score - 0.9816]
valid: 100%|██████████| 4/4 [00:02<00:00,  1.89it/s, dice_loss - 0.04748, iou_score - 0.9684]
```

Fig 11: UNet Model training

Of the runs done, the best result that the UNet model achieved is an IoU score of 0.982 and a Dice loss of 0.044. The variations of the IoU score and Dice loss can be seen below.

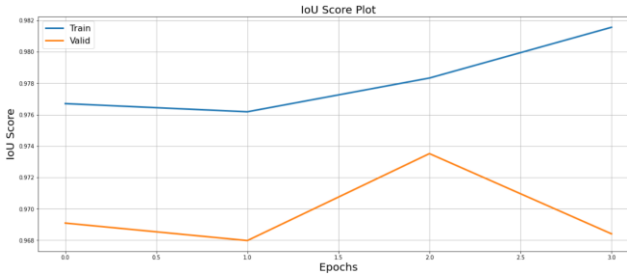

Fig 12: IoU scores of the UNet model

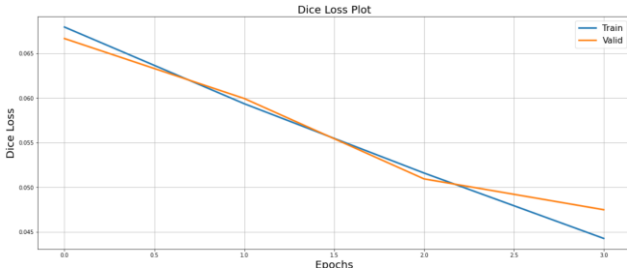

Fig 13: Dice loss of the UNet model

For the UNet++ model, a similar approach was taken and the training was halted after 9 epochs to prevent overfitting.

```
Epoch: 8
train: 100%|██████████| 31/31 [00:55<00:00,  1.78s/it, dice_loss - 0.1439, iou_score - 0.9764]
valid: 100%|██████████| 4/4 [00:03<00:00,  1.14it/s, dice_loss - 0.1351, iou_score - 0.9785]

Epoch: 9
train: 100%|██████████| 31/31 [00:55<00:00,  1.79s/it, dice_loss - 0.1312, iou_score - 0.9743]
valid: 100%|██████████| 4/4 [00:03<00:00,  1.14it/s, dice_loss - 0.125, iou_score - 0.9756]
```

Fig 14: UNet++ Model training

The best results achieved by the UNet++ model is an IoU score of 0.9756 and a Dice loss of 0.1250. The variations of both can be seen below.

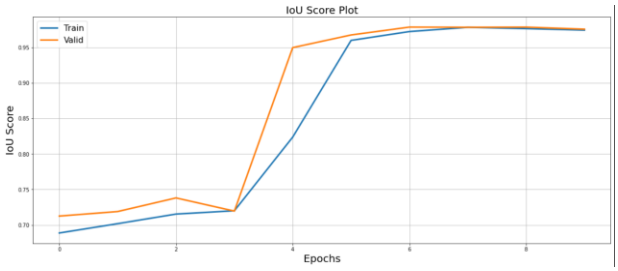

Fig 15: IoU scores of the UNet++ model

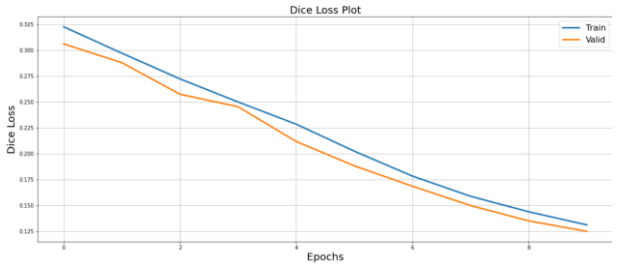

Fig 16: Dice loss of the UNet++ model

For the DeepLabV3 model, the best results achieved were an IoU score of 0.9716 and a dice loss of 0.05.

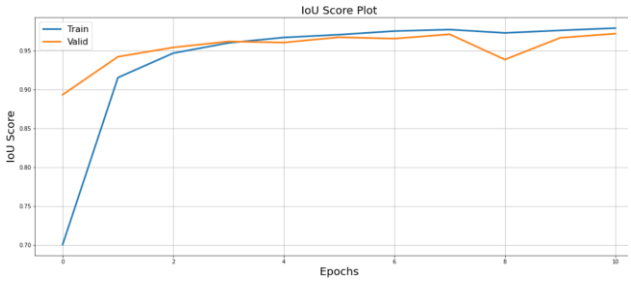

Fig 17: IoU scores of the DeepLabv3 model

The best performing model is the UNet++ model according to my work.

1.3. Polyp Bounding Boxes (Localization)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can assess frames of a colonoscopy procedure and locate polyps in the GI. This is important as in aiding doctors in the identification process.

1.3.1 Dataset Exploration

The dataset used is from the University of Toronto in 2015. It consists of 1,062 images with an equal class distribution. The sample images of the clear and burry images of the colonoscopy video are given below. The images are of the shape 480x712x3 which is larger than we need. The preprocessing I do is explained below.

1.3.2 Data Processing

This situation can be seen as a localization task with coordinates for the bounding box for each frame. Each instance stores the label as well as a processed image using OpenCV and PyVison's transforms. The images are resized to a 256x379 dimension and gray scaled to improve performance and converted to Tensors. This data is split as 80-20 for the training and testing sets

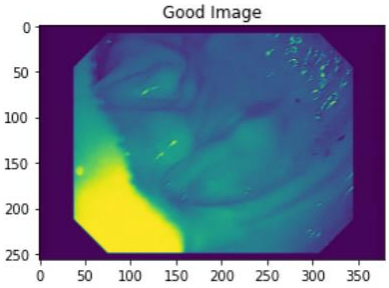

Fig 18: Processed Image

1.3.3 Model

The model I chose for this problem is the YOLOv5. I chose this as it is one of the best approaches to the localization tasks. Leveraging transfer learning, this model is already pretrained with weights, greatly speeding up our task.

The Adam optimizer is used with a learning rate of 0.01. The loss function used is the standard cross entropy loss.

1.3.4 Results

The model is trained for 23 epochs. Any more resulted in overfitting. The training is accelerated with the help of a GPU.

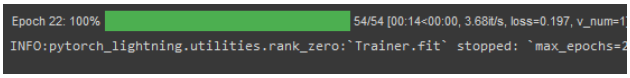

Fig 19: Model training

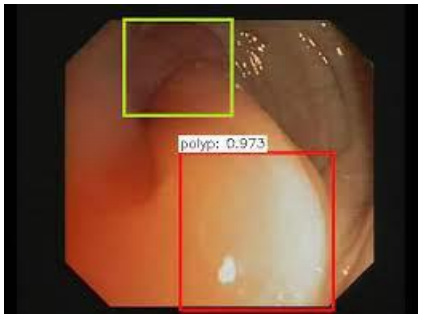

Fig 20: Model prediction on the image

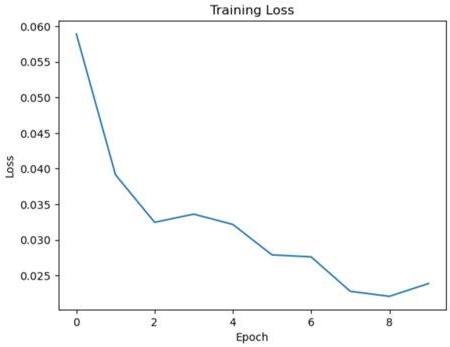

Fig 21: Training loss

In the end, the model achieves an IoU score of 0.78 when tested against the ground truth bounding boxes.

2. Chest X-rays

2.1. Chest X-ray Classification (Classification)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can analyze chest X-rays to identify the presence of thoracic diseases in a patient.

2.1.1 Dataset Exploration

The dataset used is the NIH Chest X-ray dataset consisting of 112,120 X-ray instances. It consists of the following classes: 'Atelectasis', 'Consolidation', 'Infiltration', 'Pneumothorax', 'Edema', 'Emphysema', 'Fibrosis', 'Effusion', 'Pneumonia', 'Pleural Thickening', 'Cardiomegaly', 'Nodule', 'Mass', 'Hernia' and 'No Findings'. We first look into the distribution of classes to make sure there isn't any major imbalances.

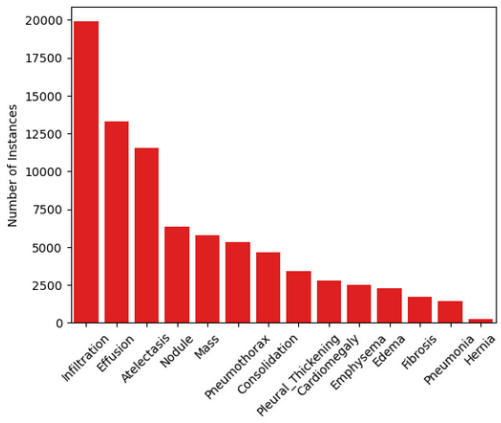

Fig 22: Class distribution of the diseases

As can be seen, the classes are skewed with diseases like 'Hernia' having little too few instances. To balance the data, we assign weights to each class for redistribution and choose 40,000 X-rays that do not have any missing data.

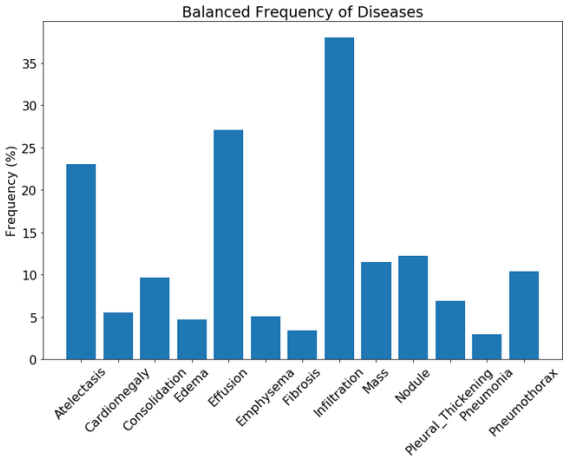

Fig 23: Redistributed Class frequencies

2.1.2 Data Processing

This situation can be seen as a multi-label classification consisting of 15 classes. The first step is to create a target vector of all the classes for each instance in the data set. This is done in a manner similar to one hot encoding where we have an array of length 15 consisting of 1s indicating the presence of a disease and 0s indicating the absence.

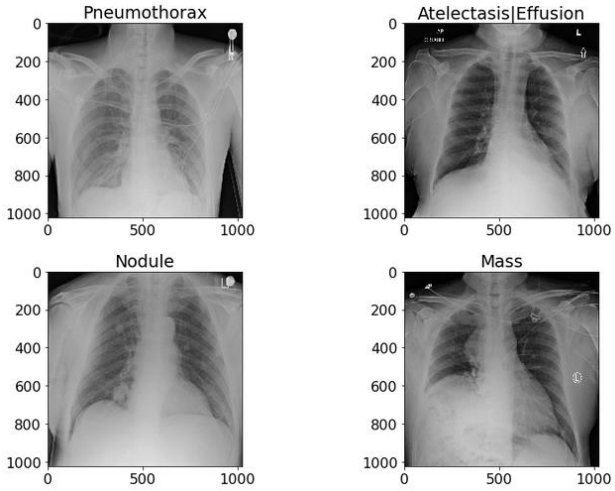

Fig 24: Sample images of classes

We use a 75-15-10 split on the dataset for the training, validation and testing sets. This means that we have 30000, 6000 and 4000 instances respectively. The training data is also augmented using height shift ranges, width shift ranges, zooms and horizontal flips. This will help us gain more cases of classes that are scarce in samples. The images are resized to 128x128 and gray scaled to aid in computation

2.1.3 Model

Two models are compared in this exercise: the ResNet50 and Mobile Net. Both of the base models are initialized and layers of Global average pooling, dense layers and dropouts are added before the final output layer with a sigmoid activation function.

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 4, 4, 2048)	23581440
global_average_pooling2d_2 (	(None, 2048)	0
dropout_3 (Dropout)	(None, 2048)	0
dense_3 (Dense)	(None, 512)	1049088
dropout_4 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 13)	6669

Total params: 24,637,197
Trainable params: 24,584,077
Non-trainable params: 53,120

Fig 25: ResNet50 Model Summary

The Adam optimizer is used with a learning rate of 0.01. The loss function used is the standard cross entropy loss.

Layer (type)	Output Shape	Param #
mobilenet_1.00_128 (Model)	(None, 4, 4, 1024)	3228288
global_average_pooling2d_1 (	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_2 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 13)	6669

Total params: 3,759,757
Trainable params: 3,737,869
Non-trainable params: 21,888

Fig 26: Mobile Net Model Summary

2.1.4 Results

The model is trained for 5 epochs with 100 steps per epoch. Any more resulted in overfitting. The training is accelerated with the help of a NVIDIA P100 GPU.

```
Epoch 00004: val_loss improved from 0.36379 to 0.33196, saving model
to xray_class_weights.best.hdf5
Epoch 5/5
100/100 [==============================] - 88s 883ms/step - loss: 0.3
355 - binary_accuracy: 0.8740 - mean_absolute_error: 0.1950 - val_los
s: 0.3363 - val_binary_accuracy: 0.8760 - val_mean_absolute_error: 0.
1707
```

Fig 27: Model training

Multiple runs were done and the AUC scores were recorded for both models. They are detailed in the table and graphs below.

Disease / Model	Mobile Net	ResNet50
Atelectasis	0.75	0.834
Cardiomegaly	0.85	0.913
Consolidation	0.70	0.818
Edema	0.84	0.914
Effusion	0.82	0.905
Emphysema	0.87	0.936
Fibrosis	0.79	0.786
Infiltration	0.67	0.665
Mass	0.76	0.886
Nodule	0.68	0.790
Pleural Thickening	0.72	0.813
Pneumonia	0.66	0.807
Pneumothorax	0.83	0.906

Table 1: AUC scores of the models

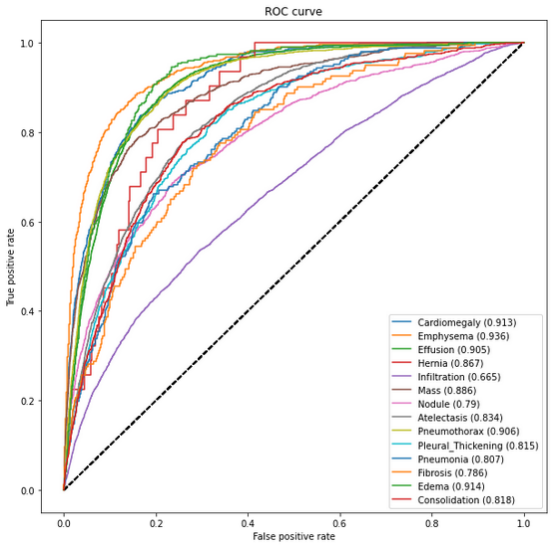

Fig 28: ROC curve for the ResNet50 model

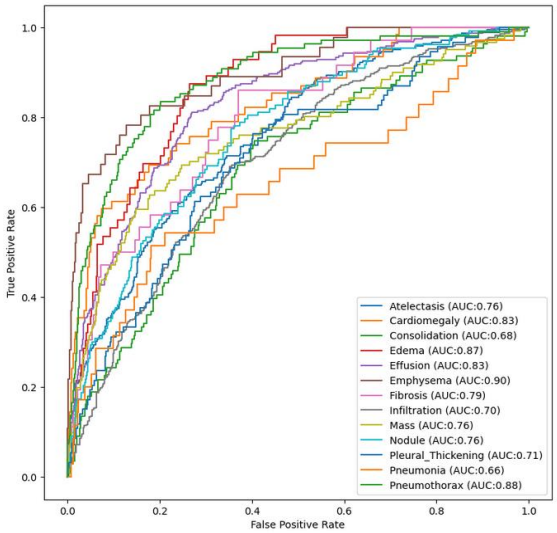

Fig 29: ROC curve for the Mobile Net model

We can see that Resnet is clearly the best performing model as seen from the accuracy and AUC scores for each class, achieving an average on 0.81. Further improvements can be made with equal balancing of classes, implementing architectures like the UNet which are designed for medical classification tasks and by possibly including patient data as a second set of parameters for the decision.

2.2. Lungs, Heart and Clavicles (Segmentation)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can analyze chest X-rays to accurately segment the Lungs, Heart and clavicles.

2.2.1 Dataset Exploration

Data used is from the JSRT database consisting of 154 X-ray images of resolution 2048x2048. These are accompanied by individual masks for each of the 5 classes mentioned above, all stored in their separate folders.

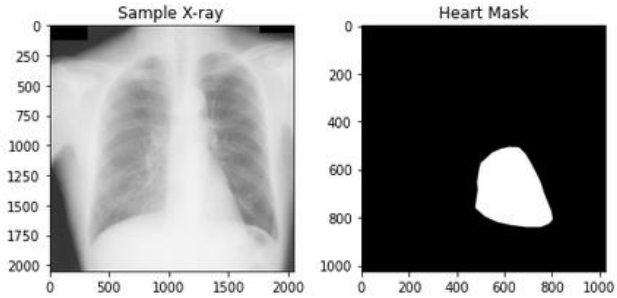

Fig 30.1: Representation of X-ray and heart mask

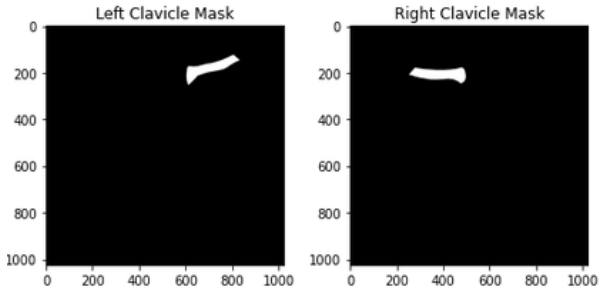

Fig 30.2: Representation of clavicle masks

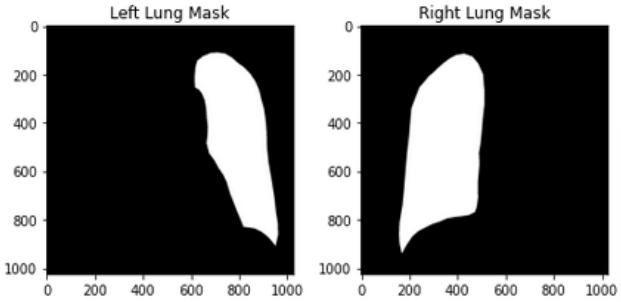

Fig 30.3: Representation of lung masks

2.2.2 Data Processing

This is a segmentation task where we have a limited amount of data. The raw image size is too big to fit the batch in memory and hence need to be resized. A data class is created mapping each image with its respective 5 masks. Each image is resized to 256x26, gray scaled and converted to tensors. It is then split in an 80-20% ratio for training and testing. Due to the lack of data, I opted to forgo the validation set. This data is then loaded using PyTorch Data Loaders and is displayed below.

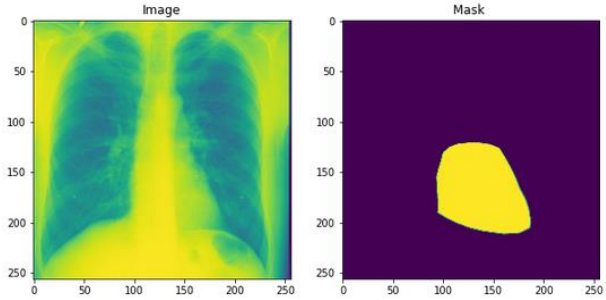

Fig 31.1: Processed Image and heart mask

We use a 75-15-10 split on the dataset for the training, validation and testing sets. This means that we have 30000, 6000 and 4000 instances respectively. The training data is also augmented using height shift ranges, width shift ranges, zooms and horizontal flips. This will help us gain more cases of classes that are scarce in samples. The images are resized to 128x128 and gray scaled to aid in computation

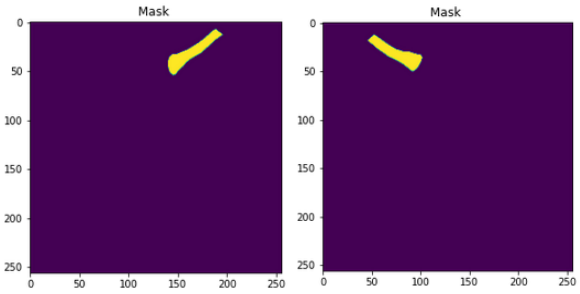

Fig 31.2: Processed clavicle masks

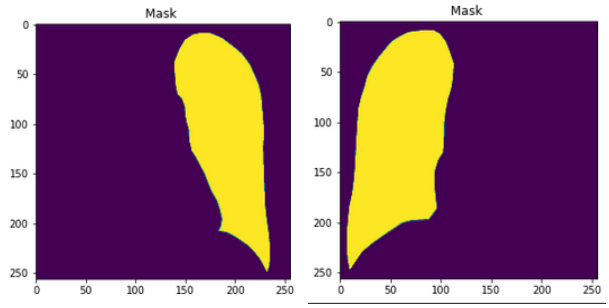

Fig 31.3: Processed lung masks

2.2.3 Model

For the segmentation process, I used the UNet architecture. To compare the performance of the models trained from scratch versus a pretrained network, I used the ImageNet weights and the ResNet18 encoder.

In the selection for loss function, I use Dice Loss as we are handling a segmentation task. The IoU metric is used to determine the score after every epoch. The Adam optimizer is used with an initial learning rate of 0.0003 The learning rate is also reduced when it plateaus using the 'Step learning Rate' function by a factor of 0.5 after 3 iterations of plateauing. The model is run for 25 epochs.

```
ch 22/25      LR:9.375e-06    train_loss:0.16462188959121704  train_score:0.9592372179031372
ch 23/25      LR:9.375e-06    train_loss:0.16322574019432068  train_score:0.9610909223556519
ch 24/25      LR:9.375e-06    train_loss:0.1614832580089569   train_score:0.9635406732559204
ch 25/25      LR:9.375e-06    train_loss:0.16043561697006226  train_score:0.9648317098617554
ch 00025: reducing learning rate of group 0 to 4.6875e-06.
```

Fig 32: Model Training

The next section will detail the collected data from 10 runs of the model from scratch and 10 more on the pretrained version.

2.2.4 Results

Class	Pre- Trained	Scratch
Heart	0.9544	0.6595
Left Clavicle	0.7905	0.5668
Right Clavicle	0.8795	0.5377
Left Lung	0.9748	0.9465
Right Lung	0.9692	0.9568

Table 2 Averaged IoU scores over 10 runs

To see if there is significant statistical difference between the two results, I set the following hypothesis:

1. Null Hypothesis (H_0): There is no significant difference between the results
2. Alternate Hypothesis (H_1): There is significant difference between the results

I do a two-tail student t-test to test the hypotheses I put forth.

```
from scipy import stats

pretrained_dice = [0.9544246196746826, 0.8794725847244263, 0.7904697775840759, \
                   0.9691820240020752, 0.9748317098617554]
scratch_dice = [0.6594725847244263, 0.5377006530761719, 0.5668262243270874, \
                0.9465135931968689,0.9567821621894836]

print(stats.ttest_ind(pretrained_dice, scratch_dice))
```

```
Ttest_indResult(statistic=1.8409718205271965, pvalue=0.10289192336880103)
```

Fig 33: Student t-test values

Given that the p-values does not exceed the 95% threshold of 0.10 for the two-tail test, I reject the null hypothesis and conclude that using the pretrained weights gives a significant difference. This also logically makes sense as the model is trained on extremely large datasets which generalize very well rather than just on a single dataset. This helps the model learn more than it ever could learning purely from the JSRT dataset.

2.3. Chest Disease Localization (Localization)

The goal of this exercise is to create a State-of-the-art (SoTA) model based that can localize the presence of lung diseases from an x-ray image.

2.3.1 Dataset Exploration

Data used is from the VinDr CXR database consisting of 3000 X-ray images of varying resolutions. These are accompanied by multilabel class strings for each of the 14 classes.

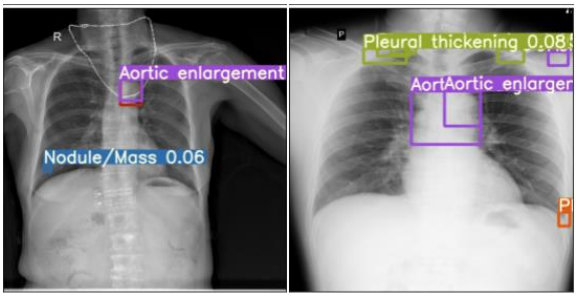

Fig 34: Representation of X-ray and heart mask

It consists of the following classes: "Aortic enlargement", "Atelectasis", "Calcification", "Cardiomegaly", "Consolidation", "ILD", "Infiltration", "Lung Opacity", "Nodule/Mass", "Other lesion", "Pleural effusion", "Pleural thickening", "Pneumothorax", "Pulmonary fibrosis"

2.3.2 Data Processing

This situation can be seen as a multi-label classification consisting of 15 classes. The first step is to create a target vector of all the classes for each instance in the data set. This is done in a manner similar to one hot encoding where we have an array of length 15 consisting of 1s indicating the presence of a disease and 0s indicating the absence.

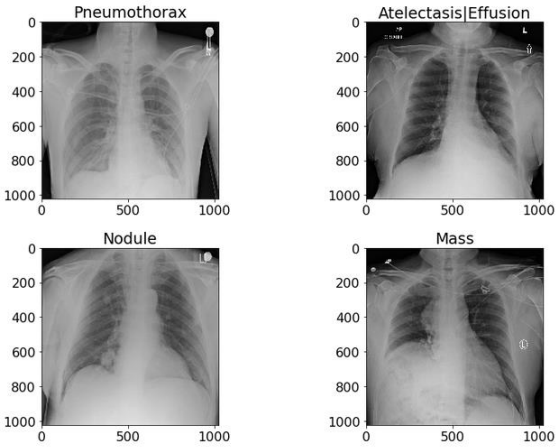

Fig 35: Sample images of classes

This is a multi-label classification task where we have a limited amount of data due to class imbalance. The raw image size is too big to fit the batch in memory and hence need to be resized. A data class is created mapping each image with its respective 14 masks. Each image is resized to 224x224, gray scaled and converted to tensors. It is then split in an 80-20% ratio for training and testing. Due to the lack of data, I opted to forgo the validation set. This data is then loaded using PyTorch Data Loaders and is displayed below.

2.3.3 Model

I first download the Swin transformer model using PyTorch from the main GitHub repository. This gives the underlying structure to the model we will use. In addition, I add additional layers to the output of this mode to modify it for this problem. This includes additional Linear layers and a Dropout to reduce overfitting.

```
HUB_URL = "SharanSMenon/swin-transformer-hub:main"
MODEL_NAME = "swin_tiny_patch4_window7_224"
# check hubconf for more models.
model = torch.hub.load(HUB_URL, MODEL_NAME, pretrained=True)
```

Fig 36: Loading Swin Transformer

```
print(model)
```

```
SwinTransformer(
  (patch_embed): PatchEmbed(
    (proj): Conv2d(3, 96, kernel_size=(4, 4), stride=(4, 4))
    (norm): LayerNorm((96,), eps=1e-05, elementwise_affine=True)
  )
  (pos_drop): Dropout(p=0.0, inplace=False)
  (layers): ModuleList(
    (0): BasicLayer(
      dim=96, input_resolution=(56, 56), depth=2
      (blocks): ModuleList(
        (0): SwinTransformerBlock(
          dim=96, input_resolution=(56, 56), num_heads=3, window_size=7, shift_size=0, mlp_ratio=4.0
          (norm1): LayerNorm((96,), eps=1e-05, elementwise_affine=True)
          (attn): WindowAttention(
            dim=96, window_size=(7, 7), num_heads=3
            (qkv): Linear(in_features=96, out_features=288, bias=True)
            (attn_drop): Dropout(p=0.0, inplace=False)
            (proj): Linear(in_features=96, out_features=96, bias=True)
            (proj_drop): Dropout(p=0.0, inplace=False)
            (softmax): Softmax(dim=-1)
          )
          (drop_path): Identity()
          (norm2): LayerNorm((96,), eps=1e-05, elementwise_affine=True)
          (mlp): Mlp(
            (fc1): Linear(in_features=96, out_features=384, bias=True)
            (act): GELU(approximate=none)
            (fc2): Linear(in_features=384, out_features=96, bias=True)
            (drop): Dropout(p=0.0, inplace=False)
          )
```

Fig 37: Swin Transformer Architecture

```
from torch import nn, optim

for param in model.parameters(): #freeze model
    param.requires_grad = False

n_inputs = model.head.in_features
model.head = nn.Sequential(
    nn.Linear(n_inputs, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, len(classes))
)
model = model.to(device)
print(model.head)
```

```
Sequential(
  (0): Linear(in_features=768, out_features=512, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.3, inplace=False)
  (3): Linear(in_features=512, out_features=14, bias=True)
)
```

Fig 38: Modifying Swin Architecture

In the selection for loss function, I use Label Smoothing Cross Entropy to work with multi label classification and the Adam optimizer. The learning rate is also reduced when it plateaus using the 'Step learning Rate' function. The model is run for 7-12 epochs.

```
Epoch 00012: val_loss did not improve from 0.23968
Epoch 13/35
431/431 [==============================] - 408s 946ms/step - loss: 0.1837
- auc: 0.9026 - val_loss: 0.2426 - val_auc: 0.8239
Restoring model weights from the end of the best epoch.
```

Fig 10: Model Training

2.3.4 Results

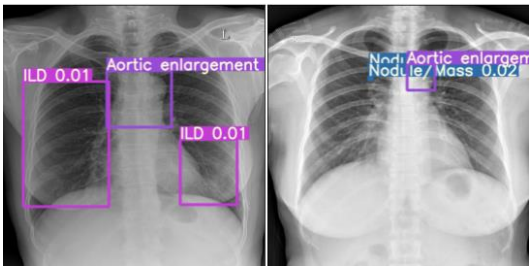

Fig 39: Model prediction on localization

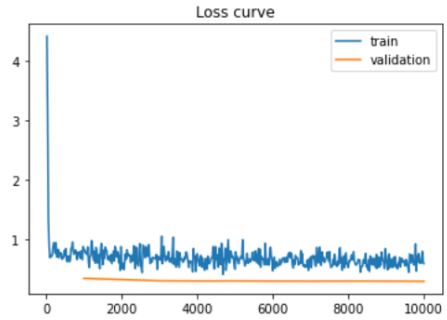

Fig 40: Model training loss curve

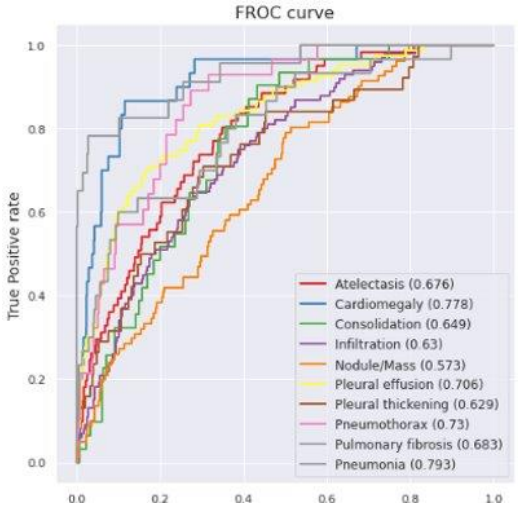

Fig 41: FROC curve for localization model

3. Project 3 (Pulmonary Embolisms)

3.1. PE Presence Classification (Classification)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can go through 3D CT scan slices and identify the location of pulmonary embolism.

3.1.1 Dataset Exploration

The dataset consists of 12,195 instances of data including image data and patient data like their ID, chronic issues, left and right-side pulmonary embolisms (PE) and indeterminate cases.

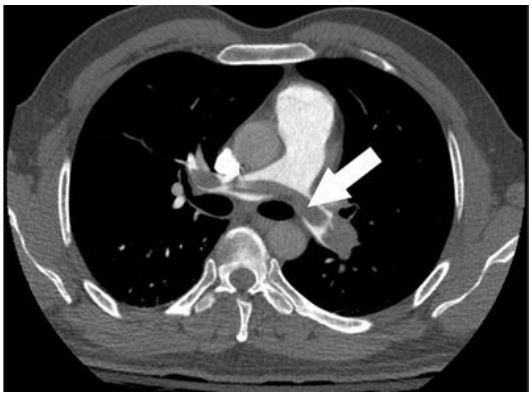

Fig 42: CT Scan image of a central PE

The dataset consists of the following classes: 'Negative Exam', 'Indeterminate', 'Right side PE', 'Left side PE', 'Central PE', 'RV/LV>=1', 'RV/LV<1', 'Chronic PE', 'Acute and Chronic PE', 'QA Motion', 'QA Contrast', 'Flow Contrast' and 'Filling Defect'

3.1.2 Data Processing

To process this data, we need to define a custom python class to identify the patient data and the multi class data. The images must be read from their DICOM filetype to individual slices. Gray scaling and conversion to Tensors follow before passing to the model. These are split into patient wise portions to maintain consistency.

3.1.3 Model

For this task, I found most appropriate to use a Resnet 50 based model pretrained on the ImageNet weights. I use the Adam Optimizer with lr=0.001 and a learning rate scheduler that reduces it on plateauing. The Binary Cross entropy function is used for loss and run for 70 epochs.

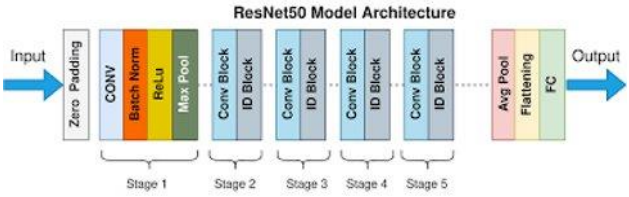

Fig 43: Resnet 50 architecture used

3.1.4 Results

The model achieves and accuracy of 92% and a loss of 0.26. The AUC score is 0.93 indicating good performance. I plot the accuracy over epochs as a check.

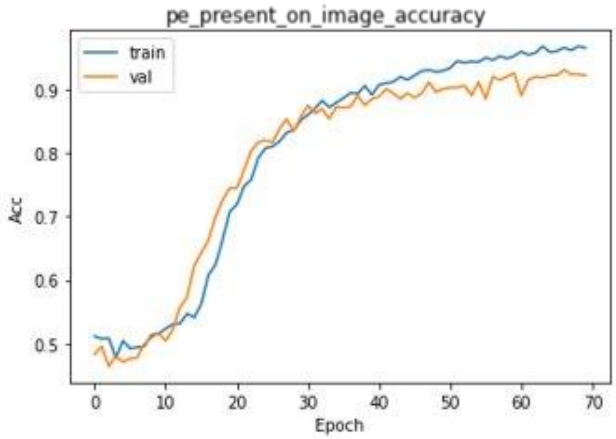

Fig 44: Training and validation accuracy over epochs

I plot some sample predictions of the model to check its efficacy.

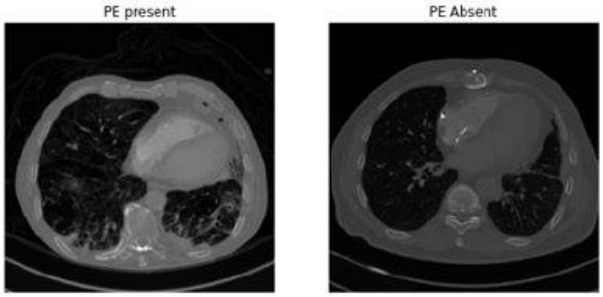

Fig 45: Model prediction of both classes

3.2. PE Segmentation (Segmentation)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can go through 3D CT scan slices and segment the location of pulmonary embolism.

3.2.1 Dataset Exploration

The FUMPE dataset (Ferdowsi University of Mashhad's PE) consists of over 8,000 slices of CT scans along with the respective masks. CT scans are stored as DICOM files which need to be processed using the 'pydicom' package. The masks are stored as .mat files.

3.2.2 Data Processing

To process this data, we need to define a custom python class to identify the patient data and the multi class data. The images must be read from their DICOM filetype to individual slices. Gray scaling and conversion to Tensors follow before passing to the model. These are split into patient wise portions to maintain consistency.

	patient_id	pixels	image	mask
57	PAT001	[0, 1]	/content/FUMPE/CT_scans/PAT001/D0058.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
58	PAT001	[0, 1]	/content/FUMPE/CT_scans/PAT001/D0059.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
59	PAT001	[0, 1]	/content/FUMPE/CT_scans/PAT001/D0060.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
60	PAT001	[0, 1]	/content/FUMPE/CT_scans/PAT001/D0061.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
61	PAT001	[0, 1]	/content/FUMPE/CT_scans/PAT001/D0062.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
...	...	...	...	...
8518	PAT035	[0, 1]	/content/FUMPE/CT_scans/PAT035/D0178.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
8519	PAT035	[0, 1]	/content/FUMPE/CT_scans/PAT035/D0179.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
8520	PAT035	[0, 1]	/content/FUMPE/CT_scans/PAT035/D0180.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
8521	PAT035	[0, 1]	/content/FUMPE/CT_scans/PAT035/D0181.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
8522	PAT035	[0, 1]	/content/FUMPE/CT_scans/PAT035/D0182.dcm	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...

2304 rows × 4 columns

Fig 46: FUMPE dataset unpacking

3.2.3 Model

For this task, I found most effective to use a UNet architecture with a Resnet 50 backbone. It is pretrained on the ImageNet weights. I use the Adam Optimizer with lr=0.001 and a learning rate scheduler that reduces it on plateauing. The Binary Cross entropy function is used for loss and run for 15 epochs.

3.2.4 Results

After training the model, I used the Dice Score as a metric to gauge its performance. The model achieved a Dice score of 0.77 and a loss of 0.352. While the model performs well, there are still instances where the segmentation masks are inaccurate. I plot a few sample predictions of the model below:

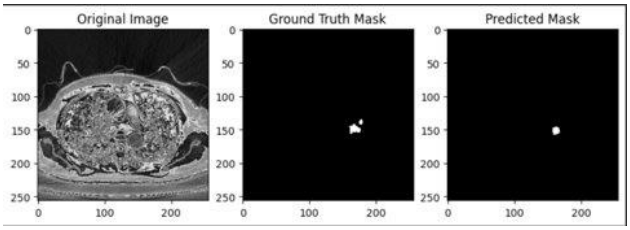

Fig 47.1: PE segmentation mask prediction

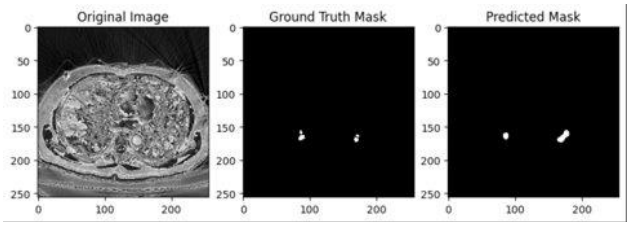

Fig 47.2: PE segmentation mask prediction

3.3. PE Localization (Localization)

The goal of this exercise is to create a State-of-the-art (SoTA) model that can go through 3D CT scan slices and identify the location of pulmonary embolism.

3.3.1 Dataset

The CAD PE dataset consists of 41,256 instances of CT scan data including ground truth box coordinates for localization.

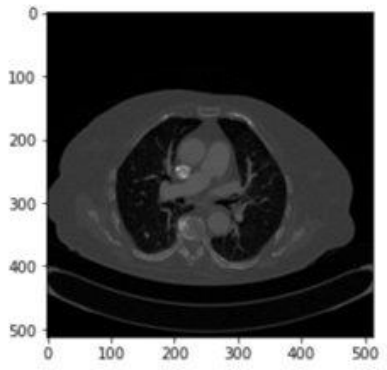

Fig 48: CAD PE dataset

3.3.2 Data Processing

To process this data, we need to define a custom python class to identify the patient data and the multi class data. The images must be read from their DICOM filetype to individual slices. Gray scaling and conversion to Tensors follow before passing to the model. These are split into patient wise portions to maintain consistency. I use 2000 slices from a single patient for training and 1000 slices from different patients for testing.

3.3.3 Model

For this task, I used the Faster RCNN model for generating coordinates for the bounding boxes. It uses the ResNet50 backbone from scratch. Further experiments can be done using the pretrained weights as well. As for the parameters, it uses a Stochastic Gradient Descent optimizer with a learning rate of 0.0001 and a momentum of 0.9. The model is run for 30 epochs.

3.3.4 Results

After training the model, I use the IoU score as metric to gauge its performance.

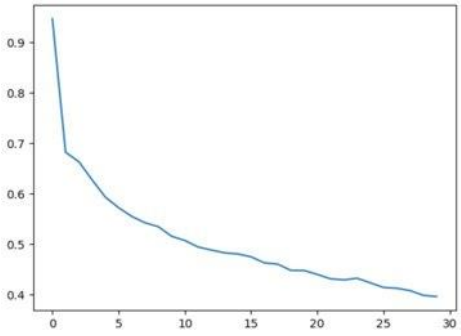

Fig 49: Training loss of PE model over epochs

The standard model faces issues of sensitivity where multiple bounding boxes are drawn around regions of interest (ROI). To solve this, I apply non-maximum suppression to get more realistic results.

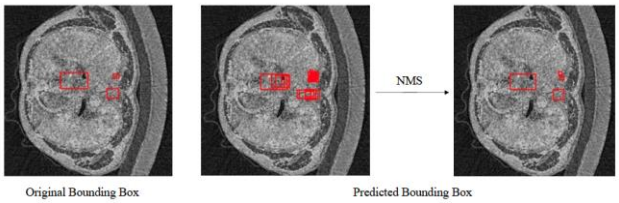

Fig 50: Non-maximum suppression of predictions

Even with these measures, the model achieves an IoU score of just 0.37. To figure out the performance issues, I plot the false positive rates against the sensitivity.

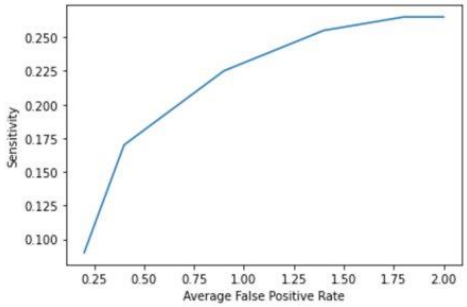

Fig 51: False positive rates vs Sensitivity

It is easier to visualize this by plotting a few examples of the model's predictions to get a better understanding.

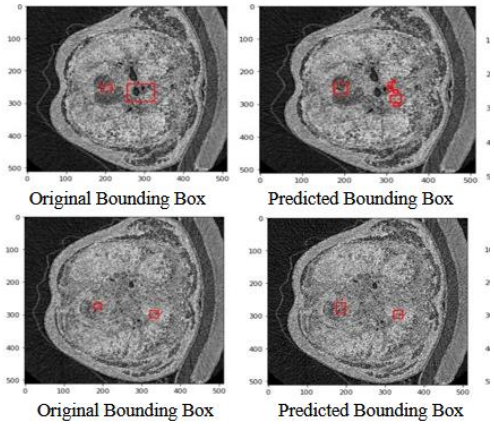

Fig 52: Ground truths vs model predictions

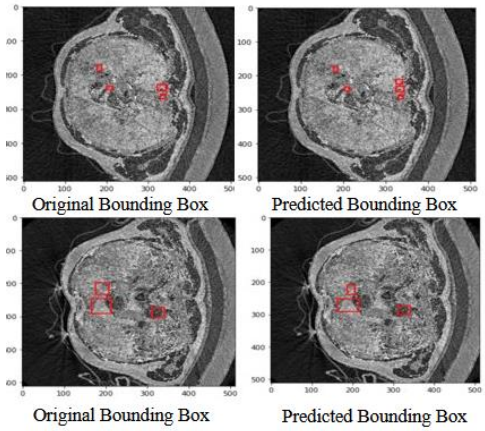

Fig 53: Ground truths vs model predictions