Report

March 21, 2021

1 Introduction

The goal of this project is to create a model that will assist doctors with identification of patients infected with pneumonia symptoms either due to covid or non-covid disease.

The choice is between double-binary or 3-class classifier. We have chosen to do double-binary classifiers where the first classifier (henceforth, called `Normal Classifier`) will attempt to classify between `normal` vs `infected (non-covid + covid)` cases and the second classifier (henceforth, called `Covid Classifier`) will attempt to classify between `non-covid` vs `covid` cases.

1.1 Project Structure

The following diagram shows the project structure one should have when attempting to reproduce or run the project:

```
root
|--dataset          # original dataset structure
|    |--test
|    |    |--infected
|    |        |--covid
|    |        |--non-covid
|    |    |--normal
|    |--train
|    |--val
|
|--models           # contains saved models to load for testing and reproducing result
|    |--binaryModelCovidBest
|    |--binaryModelCovidBestSensitivity
|    |--binaryModelCovidSecondBestSensitivity
|    |--binaryModelNormalBest
|    |--binaryModelNormalBestSensitivity
|
|--dataset.py       # contains custom dataset and dataloader functions
|--model.py         # contains all model architecture that we tested;
|                   # final best result uses resnet18 (the Net class)
|
|--test.py          # contains loading & testing code to check for metrics
|                   # like accuracy and sensitivity (recall)
|
```

```
|--train.py         # contains saving, preprocessing & training code
|                   # including loss function's weight adjustment
|--report.ipynb
|--report.pdf
```

2 Dataset & Dataloader

A custom dataset and dataloader has been written in `dataset.py`. The module contains 3 classes: - `ImageDataset` which inherits from `torch.utils.data.Dataset` - `BinaryClassDataset` which inherits from `ImageDataset` - `TrinaryClassDataset` which inherits from `ImageDataset`

The same class is used 3 times to load the `train`, `test` and `validation` sets by passing the appropriate arguments. The following code shows how to load the `train` and `validation` sets for the first classifier that attempts to separate between `normal` and `infected`.

Since the `infected` folder contains 2 dataset `covid` and `non-covid`, we use concatenation to **concatenate the dataset**. As a consequence, one of the `normal` dataset have to be **set to 0, so they're not double counted.**

However, note that in the actual training, we have decided to **oversample the minority class**, hence purposefully double counting some images as it increases the model performance by a slight amount as it helps remedy the imbalance in the dataset (discussed later).

[2]:
```
from dataset import BinaryClassDataset, TrinaryClassDataset
from torch.utils.data import DataLoader, ConcatDataset

trainingBatchSize = 4
img_size = (150, 150)
class_dict = {0: 'normal', 1: 'infected'}

# load TRAIN dataset
groups = ['train']
dataset_numbers = {'train_normal': 0, # 0 so it is not double counted when␣
 ↪concatenated
                   'train_infected': 2530,
                   }

dataset_paths = {'train_normal': './dataset/train/normal/',
                 'train_infected': './dataset/train/infected/non-covid',
                 }

trainset1 = BinaryClassDataset('train', img_size, class_dict, groups,␣
 ↪dataset_numbers, dataset_paths)

dataset_numbers = {'train_normal': 1341,
                   'train_infected': 1345,
                   }
```

```
dataset_paths = {'train_normal': './dataset/train/normal/',
                 'train_infected': './dataset/train/infected/covid',
                 }

trainset2 = BinaryClassDataset('train', img_size, class_dict, groups,␣
 ↪dataset_numbers, dataset_paths)

trainsets = ConcatDataset([trainset1, trainset2])
trainloader = DataLoader(trainsets, batch_size=trainingBatchSize, shuffle=True)
```

[3]:
```
# load VALIDATION dataset
val_groups = ['val']
val_numbers = {'val_normal': 0, # 0 so it is not double counted when␣
 ↪concatenated
               'val_infected': 8,
               }

valset_paths = {'val_normal': './dataset/test/normal',
                'val_infected': './dataset/test/infected/non-covid',
                }

valset1 = BinaryClassDataset('val', img_size, class_dict, val_groups,␣
 ↪val_numbers, valset_paths)

val_numbers = {'val_normal': 8,
               'val_infected': 8,
               }

valset_paths = {'val_normal': './dataset/val/normal',
                'val_infected': './dataset/val/infected/covid',
                }

valset2 = BinaryClassDataset('val', img_size, class_dict, val_groups,␣
 ↪val_numbers, valset_paths)

valsets = ConcatDataset([valset1, valset2])
validationloader = DataLoader(valsets, batch_size=trainingBatchSize,␣
 ↪shuffle=True)
```

Checking that the dataset and dataloader works as intended:

[4]:
```
trainset1.describe()
trainset2.describe()
valset1.describe()
valset2.describe()
```

```
                It contains a total of 2530 images of size (150, 150).
                Images have been split in 1 groups: ['train'] sets.
                The images are stored in the following locations, each
containing the following images:

 - train_normal, in folder ./dataset/train/normal/: 0 images.
 - train_infected, in folder ./dataset/train/infected/non-covid: 2530 images.

                It contains a total of 2686 images of size (150, 150).
                Images have been split in 1 groups: ['train'] sets.
                The images are stored in the following locations, each
containing the following images:

 - train_normal, in folder ./dataset/train/normal/: 1341 images.
 - train_infected, in folder ./dataset/train/infected/covid: 1345 images.

                It contains a total of 8 images of size (150, 150).
                Images have been split in 1 groups: ['val'] sets.
                The images are stored in the following locations, each
containing the following images:

 - val_normal, in folder ./dataset/test/normal: 0 images.
 - val_infected, in folder ./dataset/test/infected/non-covid: 8 images.

                It contains a total of 16 images of size (150, 150).
                Images have been split in 1 groups: ['val'] sets.
                The images are stored in the following locations, each
containing the following images:

 - val_normal, in folder ./dataset/val/normal: 8 images.
 - val_infected, in folder ./dataset/val/infected/covid: 8 images.
```

[5]:

```python
import matplotlib.pyplot as plt

axes = []
def show_tensor_imgs(tensor_imgs, labels):
    '''quick and dirty function to display tensor image'''
    n = len(tensor_imgs)
    fig = plt.figure()
    for i in range(n):
        axes.append(fig.add_subplot(2, 3, i+1)) # add subplot
        subplot_title = (str(i) + ': ' + labels[i])    # name subplot by index
        axes[-1].set_title(subplot_title)
        plt.imshow(tensor_imgs[i])
    fig.tight_layout()

labels = [
```

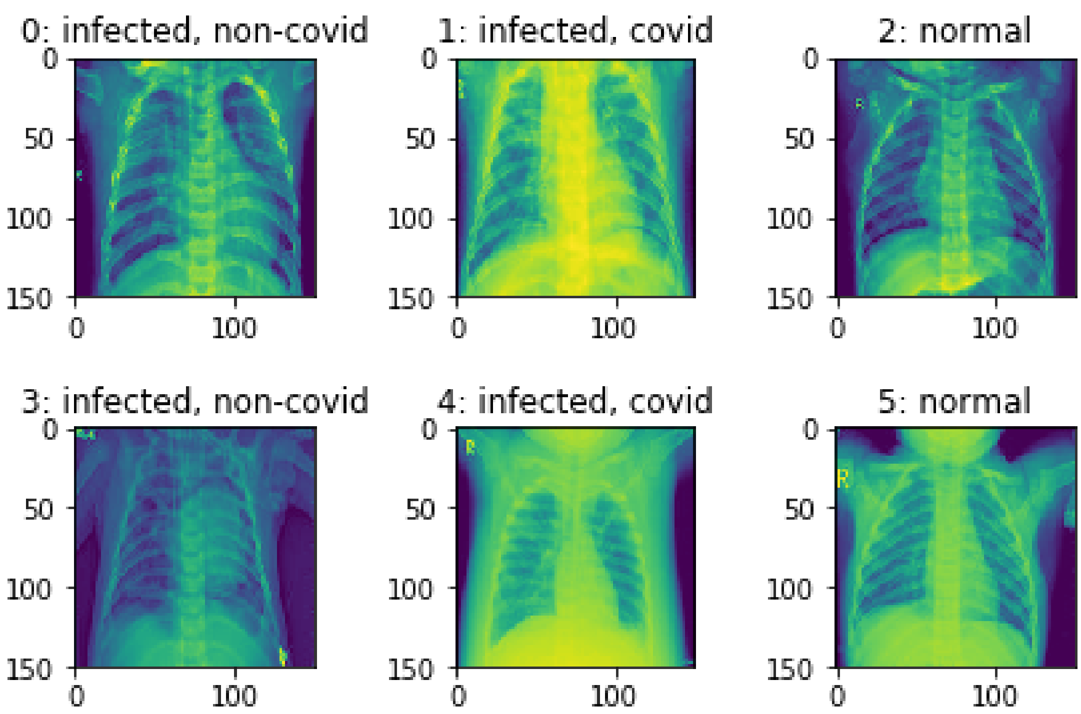
```
    'infected, non-covid',
    'infected, covid',
    'normal',

    'infected, non-covid',
    'infected, covid',
    'normal'
]

imgs = [
    trainset1.open_img('train', 'infected', 1), # infected, non-covid
    trainset2.open_img('train', 'infected', 1), # infected, covid
    trainset2.open_img('train', 'normal', 1),   # normal

    valset1.open_img('val', 'infected', 1), # infected, non-covid
    valset2.open_img('val', 'infected', 1), # infected, covid
    valset2.open_img('val', 'normal', 1)    # normal
]

show_tensor_imgs(imgs, labels)
plt.show()
```



3 Data Exploration

3.1 Dataset Imbalance

A quick exploration of the dataset shows that there is a clear imbalance:

- 1341 images for the train dataset, normal class,
- 2530 images for the train dataset, infected and non-covid class,
- 1345 images for the train dataset, infected and covid class,
- 234 images for the test dataset, normal class,
- 242 images for the test dataset, infected and non-covid class,
- 138 images for the test dataset, infected and covid class,
- 8 images for the val dataset, normal class,
- 8 images for the val dataset, infected and non-covid class,
- 8 images for the val dataset, infected and covid class.

Plotting the distribtion of cases for `train` dataset and `validation` dataset, we get the following:

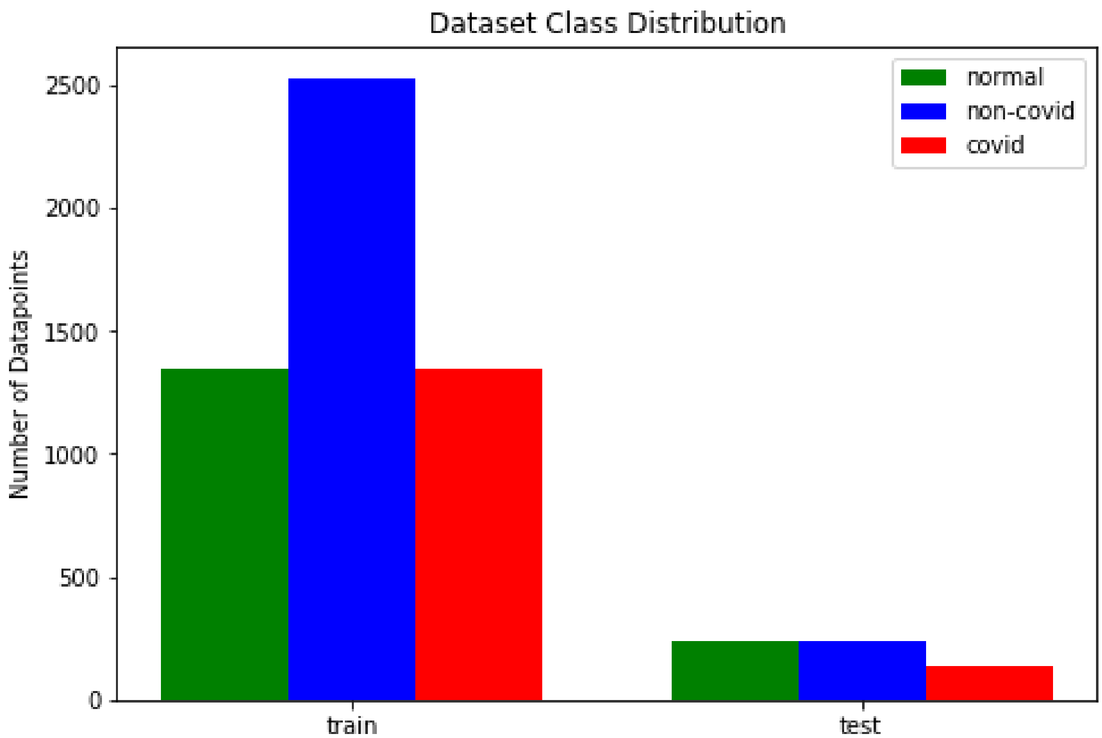
[6]:
```python
import numpy as np

data = [
    [1341, 234],
    [2530, 242],
    [1345, 138]
]

X = np.arange(2)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(X + 0.00, data[0], color = 'g', width = 0.25)
ax.bar(X + 0.25, data[1], color = 'b', width = 0.25)
ax.bar(X + 0.50, data[2], color = 'r', width = 0.25)

ax.set_xticks(X + 0.25)
ax.set_xticklabels(['train', 'test'])
ax.set_ylabel('Number of Datapoints')
ax.legend(labels=['normal', 'non-covid', 'covid'])
ax.set_title('Dataset Class Distribution')
```

[6]:
```
Text(0.5, 1.0, 'Dataset Class Distribution')
```



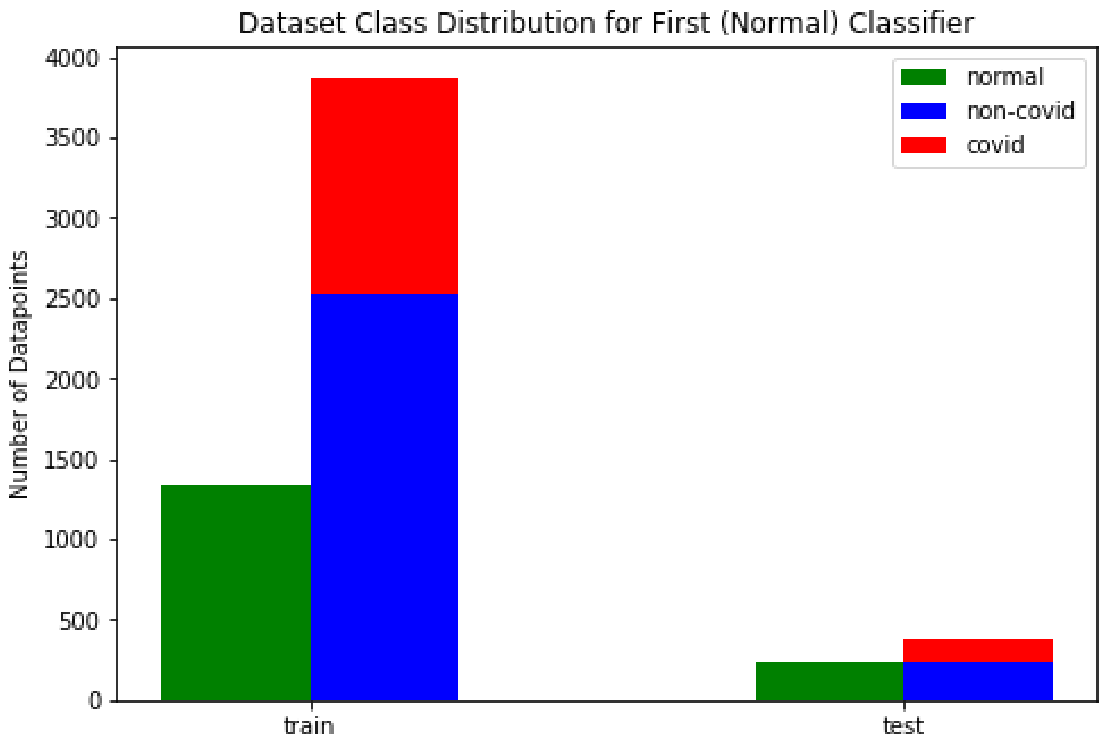
As can be seen there are twice as many `non-covid` cases as there are `normal` or `covid` cases for the `train` dataset. This is especially problematic for the second classifier that attempts to separate `covid` cases from `non-covid` as `non-covid` cases are the majority (the class that we're not interested in).

Whereas in the case of the first classifier that attempts to seperate `normal` cases from `infected` cases, this will be less of a problem as the `infected` class `(covid + non-covid)` (the class we're interested in) is the overwhelming majority in the `train` dataset.

[13]:
```
X = np.arange(2)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(X + 0.00, data[0], color = 'g', width = 0.25)
ax.bar(X + 0.25, data[1], color = 'b', width = 0.25)
ax.bar(X + 0.25, data[2], color = 'r', width = 0.25, bottom=data[1])

ax.set_xticks(X + 0.125)
ax.set_xticklabels(['train', 'test'])
ax.set_ylabel('Number of Datapoints')
ax.legend(labels=['normal', 'non-covid', 'covid'])
ax.set_title('Dataset Class Distribution for First (Normal) Classifier')
```

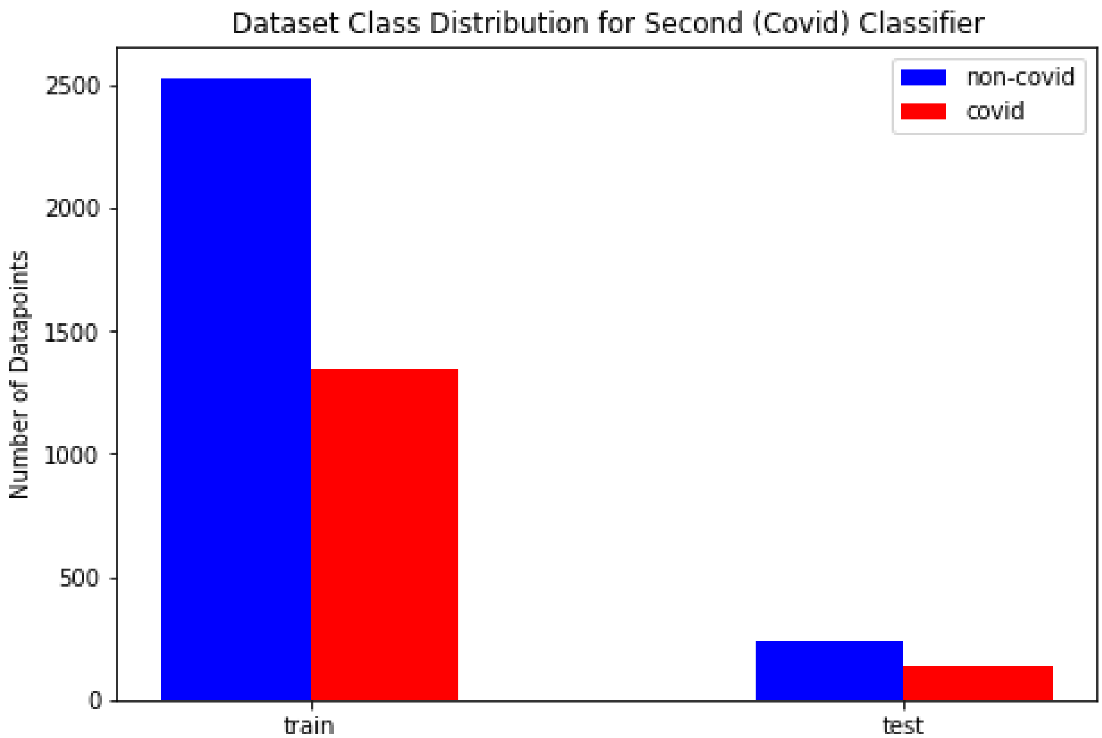
[13]:
```
Text(0.5, 1.0, 'Dataset Class Distribution for First (Normal) Classifier')
```



[15]:
```
X = np.arange(2)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(X, data[1], color = 'b', width = 0.25)
ax.bar(X + 0.25, data[2], color = 'r', width = 0.25)

ax.set_xticks(X + 0.125)
ax.set_xticklabels(['train', 'test'])
ax.set_ylabel('Number of Datapoints')
ax.legend(labels=['non-covid', 'covid'])
ax.set_title('Dataset Class Distribution for Second (Covid) Classifier')
```

[15]: Text(0.5, 1.0, 'Dataset Class Distribution for Second (Covid) Classifier')



As a result of this **imbalance in the dataset** for both the first and second classifier, the **trained classifier can become 'naive'** because it may end up just guessing the majority class while still leading to decent accuracies on test set. This is especially problematic for the `covid` classifier where `covid` cases (the one we're interested in) are the minority, and less of a prominent problem in the `normal` classifier as the `infected` class (the one we're interested in) is the majority.

To remedy this, we have decided to **optimize for a different metric** called `sensitivity` or `recall` instead of `accuracy`. In healthcare context, where it is far more important to detect the rare minority class, and far more punishing to classify false negative than false positive, metric like `sensitivity` may make more sense than `accuracy`.

There are also other techniques that we have used to remedy this such as **oversampling the minority class** and **adjusting the weights of the components of the cross entropy loss function** (explained later).

3.2 Preprocessing

There are 2 key preprocessing that needs to be done. **Normalization and data augmentation.**

Normalization is important to centralize the values of the pixels to a certain range. This helps to ensure that the gradient values do not become too small or too large to some feature values, which means one common learning rate can then be used to update the weights across the network.

[17]:
```
# getting normalization value for classifier 1 & 2
trainset_len = 2530 + 1341 + 1345
```

```
train_data = DataLoader(trainsets, batch_size=trainset_len, shuffle=True)
data = next(iter(train_data))
mean = data[0].mean()
std = data[0].std()

mean, std
```

[17]: (tensor(0.4824), tensor(0.2363))

Another preprocessing that we have done is to add random rotation (max 45 degrees) and random horizontal flip to the image data before it is passed into the network to train.

These data augmentations make sense because the distinguishing feature between `normal` and `infected` cases are the white pneumonia patterns in the x-ray images. **These patterns are rotation and translation invariant.** Furthermore, it also do not matter whether the pneumonia patterns are found on the right or left lung.

Thus, by adding these random transformations during training, we hope to help the model learn these characteristics.

These data augmentations and normalization are done at training time. During test time, only normalization is done on the input image.

4 Classifier Architecture:

From researching the topic, our group was initially unable to form a definitive conclusion since most answers seemed to (vaguely) point that the best approach would depend on the nature of the problem, the amount of data available, and computation power.

As recommended by Prof Mari, we decided to build out both architectures with a relatively simple model as an experiment. Even though the multiclass classifier achieved a higher overall accuracy, we then realised that it would be far easier to diagnose and tweak two binary classifiers separately than in a single model, and hence went with the cascaded structure instead.

Our rationale for this choice is that our overall goal would be to build a model with high sensitivity (rather than accuracy) in order to pick out infected cases from the data and ensure that these patients can seek medical treatment - favouring Type 1 over Type 2 errors. Additionally, since Covid-19 is highly contagious, there is an impetus to try detect as many of these from the infected cases as well. Hence, the cascaded binary classifier structure naturally aligns with this by allowing us to finetune for sensitivity at each stage.

Below is a brief summary of what we believe to be the tradeoffs of each model.

4.1 Comparison Summary

Multiclass Pros

- Simple to use since we only have to deal with a single model
- Seemed to be able to obtain higher overall accuracy than the cascaded structure on this particular dataset (85% versus 90%*75%=~70%)

Multiclass Cons

- Hard to diagnose, and thus difficult to finetune the relationship between the three classes

Cascaded Binary Pros

- Easy to input weights for different classifications in the step-wise process to finetune sensitivity
- Converges faster on a per-model basis
- More accurate in theory due to the potential of modeling pair-wise relationships between classes

Cascaded Binary Cons

- Irritating to tweak and evaluate, in addition to needing extra steps to build the dataloaders for each one
- Overall time to train is longer due to the presence of two models

Logically speaking, this may be why doctors in practice do not use x-ray scans to classify the source of lung damage, but only to confirm the presence of it. Hence, this exercise of actually determining what caused the lung damage is a fallacy in itself, and the best way to account for this is to use a separate classifier.

5 Choice of CNN Architecture

Rather than conceive a completely new architecture from our limited knowledge, our group adopted an approach where we trained and evaluated a few of the available predefined models in torchvision, then selected the best contender to build from scratch so that we could tweak it.

Given the constraints of 1) a relatively small training dataset and 2) limited computation power, we decided to keep the number of parameters on the lower side to follow the golden rule of machine learning. Hence, from the illustration below plotting top-1 accuracy against the number of operations, we selected the ResNet, DenseNet, MobileNet, and Inception architectures. Specifically, we chose ResNet-18, DenseNet-121, MobileNetV2 and Inception-v3.

After testing the models, we found that ResNet-18 was the clear winner for our base model, consistently achieving the highest accuracy on the validation set with the lowest loss on the training set, while also taking the shortest time to train. The figures below show the best models achieved within 10 epochs for both binary classifiers #1 and #2.

	ResNet-18	DenseNet121	Inception-v3	MobileNetV2
Train Avg. Loss #1	0.2689	0.2829	0.4727	0.4377
Val Avg. Loss #1	0.0288	0.0534	0.1573	0.1125
Val Accuracy #1	92%	88%	71%	83%
Training Time #1	6min 32s	21min 58s	16min 25s	8min 49s
Train Avg. Loss #2	0.5802	0.5992	0.7481	0.6383
Val Avg. Loss #2	0.1159	0.1134	0.1631	0.1508
Val Accuracy #2	75%	75%	56%	62%
Training Time #2	4min 49s	16min	12min	6min 32s

5.0.1 Observations

- DenseNet also performed well, but required significantly more time to train than ResNet - more than 3x
- Inception showed subpar performance (due to the removal of the aux channel), but still required significantly more time to train than ResNet.
- MobileNet (our favoured contender) unfortunately obtained similar results to Inception, and somehow also took slightly longer to train
- All the architectures struggled with the second task of separating covid and non-covid cases.

[4]:
```
import time
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms

from train import train, validate

resnet = models.resnet18(pretrained=False)
num_ftrs = resnet.fc.in_features
resnet.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3,␣
 ↪3), bias=False)
resnet.fc = nn.Linear(num_ftrs, 2)
# print(resnet)

densenet = models.densenet121(pretrained=False)
num_ftrs = densenet.classifier.in_features
densenet.features.conv0 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2),␣
 ↪padding=(3, 3), bias=False)
densenet.classifier = nn.Linear(num_ftrs, 2)
# print(densenet)

inception = models.inception_v3(pretrained=False, aux_logits=False) #disable␣
 ↪auxiliary channel to accept 150x150 images
inception.Conv2d_1a_3x3.conv=nn.Conv2d(1, 32, kernel_size=(3, 3), stride=(2,␣
 ↪2), bias=False)
num_ftrs = inception.fc.in_features
inception.fc = nn.Linear(num_ftrs,2)
# print(inception)

mobilenet = models.mobilenet_v2(pretrained=False)
mobilenet.features[0][0]=nn.Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2),␣
 ↪padding=(1, 1), bias=False)
num_ftrs = mobilenet.classifier[1].in_features
mobilenet.classifier[1] = nn.Linear(num_ftrs, 2)
# print(mobilenet)
```

[5]:
```python
from train import train_binary_normal_clf, train_binary_covid_clf

train_binary_normal_clf(10, 4, savePath=None, model=resnet, weight=None,␣
 ↪quiet=True)
train_binary_covid_clf(10, 4, savePath=None, model=resnet, weight=None,␣
 ↪quiet=True)
```

```
Train Epoch: 1
Train set: Average loss: 0.5032
Validation set: Average loss: 0.1801, Accuracy: 15/24 (62%)

Found New Minima at epoch 1 loss: 0.18013657381137213

Train Epoch: 2
Train set: Average loss: 0.3512
Validation set: Average loss: 0.1321, Accuracy: 19/24 (79%)

Found New Minima at epoch 2 loss: 0.13212250793973604

Train Epoch: 3
Train set: Average loss: 0.3304
Validation set: Average loss: 0.0604, Accuracy: 22/24 (92%)

Found New Minima at epoch 3 loss: 0.06036650544653336

Train Epoch: 4
Train set: Average loss: 0.2968
Validation set: Average loss: 0.1091, Accuracy: 20/24 (83%)

Train Epoch: 5
Train set: Average loss: 0.3056
Validation set: Average loss: 0.0572, Accuracy: 23/24 (96%)

Found New Minima at epoch 5 loss: 0.05722993488113085

Train Epoch: 6
Train set: Average loss: 0.3002
Validation set: Average loss: 0.0506, Accuracy: 22/24 (92%)

Found New Minima at epoch 6 loss: 0.05057169760887822

Train Epoch: 7
Train set: Average loss: 0.2689
Validation set: Average loss: 0.0288, Accuracy: 22/24 (92%)

Found New Minima at epoch 7 loss: 0.028785567575444777
```

```
Train Epoch: 8
Train set: Average loss: 0.2606
Validation set: Average loss: 0.0490, Accuracy: 22/24 (92%)

Train Epoch: 9
Train set: Average loss: 0.2455
Validation set: Average loss: 0.0670, Accuracy: 22/24 (92%)

Train Epoch: 10
Train set: Average loss: 0.2421
Validation set: Average loss: 0.0510, Accuracy: 23/24 (96%)

Time Elapsed: 0:06:32.241182

Train Epoch: 1
Train set: Average loss: 1.1077
Validation set: Average loss: 0.1547, Accuracy: 12/16 (75%)

Found New Minima at epoch 1 loss: 0.15469163842499256

Train Epoch: 2
Train set: Average loss: 0.6431
Validation set: Average loss: 0.1623, Accuracy: 11/16 (69%)

Train Epoch: 3
Train set: Average loss: 0.6280
Validation set: Average loss: 0.1389, Accuracy: 12/16 (75%)

Found New Minima at epoch 3 loss: 0.13887844793498516

Train Epoch: 4
Train set: Average loss: 0.5990
Validation set: Average loss: 0.1309, Accuracy: 13/16 (81%)

Found New Minima at epoch 4 loss: 0.13087763264775276

Train Epoch: 5
Train set: Average loss: 0.5909
Validation set: Average loss: 0.1481, Accuracy: 12/16 (75%)

Train Epoch: 6
Train set: Average loss: 0.5835
Validation set: Average loss: 0.1207, Accuracy: 13/16 (81%)

Found New Minima at epoch 6 loss: 0.1207497101277113

Train Epoch: 7
Train set: Average loss: 0.5792
```

```
Validation set: Average loss: 0.1421, Accuracy: 13/16 (81%)

Train Epoch: 8
Train set: Average loss: 0.5802
Validation set: Average loss: 0.1159, Accuracy: 12/16 (75%)

Found New Minima at epoch 8 loss: 0.115862637758255

Train Epoch: 9
Train set: Average loss: 0.5792
Validation set: Average loss: 0.1536, Accuracy: 11/16 (69%)

Train Epoch: 10
Train set: Average loss: 0.5776
Validation set: Average loss: 0.1228, Accuracy: 12/16 (75%)

Time Elapsed: 0:04:49.957193
```

[6]:
```
train_binary_normal_clf(10, 4, savePath=None, model=densenet, weight=None,␣
 ↪quiet=True)
train_binary_covid_clf(10, 4, savePath=None, model=densenet, weight=None,␣
 ↪quiet=True)
```

```
Train Epoch: 1
Train set: Average loss: 0.4614
Validation set: Average loss: 0.1324, Accuracy: 16/24 (67%)

Found New Minima at epoch 1 loss: 0.13236750600238642

Train Epoch: 2
Train set: Average loss: 0.3546
Validation set: Average loss: 0.1055, Accuracy: 19/24 (79%)

Found New Minima at epoch 2 loss: 0.10549515672028065

Train Epoch: 3
Train set: Average loss: 0.3266
Validation set: Average loss: 0.1249, Accuracy: 19/24 (79%)

Train Epoch: 4
Train set: Average loss: 0.3046
Validation set: Average loss: 0.0852, Accuracy: 20/24 (83%)

Found New Minima at epoch 4 loss: 0.08515533133565138

Train Epoch: 5
Train set: Average loss: 0.2984
```

```
Validation set: Average loss: 0.0933, Accuracy: 20/24 (83%)

Train Epoch: 6
Train set: Average loss: 0.2980
Validation set: Average loss: 0.0687, Accuracy: 21/24 (88%)

Found New Minima at epoch 6 loss: 0.06870392366545275

Train Epoch: 7
Train set: Average loss: 0.2829
Validation set: Average loss: 0.0534, Accuracy: 21/24 (88%)

Found New Minima at epoch 7 loss: 0.053382359289874635

Train Epoch: 8
Train set: Average loss: 0.2707
Validation set: Average loss: 0.0651, Accuracy: 20/24 (83%)

Train Epoch: 9
Train set: Average loss: 0.2736
Validation set: Average loss: 0.0701, Accuracy: 20/24 (83%)

Train Epoch: 10
Train set: Average loss: 0.2550
Validation set: Average loss: 0.0638, Accuracy: 22/24 (92%)

Time Elapsed: 0:21:58.655482

Train Epoch: 1
Train set: Average loss: 1.5462
Validation set: Average loss: 0.2615, Accuracy: 7/16 (44%)

Found New Minima at epoch 1 loss: 0.2614750377833843

Train Epoch: 2
Train set: Average loss: 0.7773
Validation set: Average loss: 0.1341, Accuracy: 12/16 (75%)

Found New Minima at epoch 2 loss: 0.13412632420659065

Train Epoch: 3
Train set: Average loss: 0.6346
Validation set: Average loss: 0.1205, Accuracy: 13/16 (81%)

Found New Minima at epoch 3 loss: 0.12049849331378937

Train Epoch: 4
Train set: Average loss: 0.6065
```

```
Validation set: Average loss: 0.1334, Accuracy: 12/16 (75%)

Train Epoch: 5
Train set: Average loss: 0.6046
Validation set: Average loss: 0.1284, Accuracy: 12/16 (75%)

Train Epoch: 6
Train set: Average loss: 0.5949
Validation set: Average loss: 0.1461, Accuracy: 11/16 (69%)

Train Epoch: 7
Train set: Average loss: 0.5992
Validation set: Average loss: 0.1134, Accuracy: 12/16 (75%)

Found New Minima at epoch 7 loss: 0.1134311705827713

Train Epoch: 8
Train set: Average loss: 0.5836
Validation set: Average loss: 0.1228, Accuracy: 11/16 (69%)

Train Epoch: 9
Train set: Average loss: 0.5878
Validation set: Average loss: 0.1403, Accuracy: 11/16 (69%)

Train Epoch: 10
Train set: Average loss: 0.5765
Validation set: Average loss: 0.1553, Accuracy: 12/16 (75%)

Time Elapsed: 0:16:19.505228
```

[7]:
```
train_binary_normal_clf(10, 4, savePath=None, model=inception, weight=None,␣
 ↪quiet=True)
train_binary_covid_clf(10, 4, savePath=None, model=inception, weight=None,␣
 ↪quiet=True)
```

```
Train Epoch: 1
Train set: Average loss: 0.5331
Validation set: Average loss: 0.1753, Accuracy: 14/24 (58%)

Found New Minima at epoch 1 loss: 0.17527922677497068

Train Epoch: 2
Train set: Average loss: 0.4877
Validation set: Average loss: 0.1918, Accuracy: 16/24 (67%)

Train Epoch: 3
Train set: Average loss: 0.5236
```

```
Validation set: Average loss: 0.2788, Accuracy: 14/24 (58%)

Train Epoch: 4
Train set: Average loss: 0.5581
Validation set: Average loss: 0.2099, Accuracy: 16/24 (67%)

Train Epoch: 5
Train set: Average loss: 0.5609
Validation set: Average loss: 0.2309, Accuracy: 16/24 (67%)

Train Epoch: 6
Train set: Average loss: 0.5106
Validation set: Average loss: 0.2347, Accuracy: 15/24 (62%)

Train Epoch: 7
Train set: Average loss: 0.4727
Validation set: Average loss: 0.1573, Accuracy: 17/24 (71%)

Found New Minima at epoch 7 loss: 0.15726305293113305

Train Epoch: 8
Train set: Average loss: 0.4305
Validation set: Average loss: 0.2191, Accuracy: 16/24 (67%)

Train Epoch: 9
Train set: Average loss: 0.4347
Validation set: Average loss: 0.3563, Accuracy: 16/24 (67%)

Train Epoch: 10
Train set: Average loss: 0.4249
Validation set: Average loss: 0.2504, Accuracy: 16/24 (67%)

Time Elapsed: 0:16:25.828651

Train Epoch: 1
Train set: Average loss: 2.5412
Validation set: Average loss: 0.5180, Accuracy: 6/16 (38%)

Found New Minima at epoch 1 loss: 0.5179687812924385

Train Epoch: 2
Train set: Average loss: 1.0828
Validation set: Average loss: 0.4637, Accuracy: 9/16 (56%)

Found New Minima at epoch 2 loss: 0.4637055266648531

Train Epoch: 3
Train set: Average loss: 0.7481
```

```
Validation set: Average loss: 0.1631, Accuracy: 9/16 (56%)

Found New Minima at epoch 3 loss: 0.16309987008571625

Train Epoch: 4
Train set: Average loss: 0.7087
Validation set: Average loss: 0.1765, Accuracy: 9/16 (56%)

Train Epoch: 5
Train set: Average loss: 0.7174
Validation set: Average loss: 0.1935, Accuracy: 9/16 (56%)

Train Epoch: 6
Train set: Average loss: 0.7033
Validation set: Average loss: 0.1978, Accuracy: 9/16 (56%)

Train Epoch: 7
Train set: Average loss: 0.6855
Validation set: Average loss: 0.1901, Accuracy: 9/16 (56%)

Train Epoch: 8
Train set: Average loss: 0.7025
Validation set: Average loss: 0.1853, Accuracy: 10/16 (62%)

Train Epoch: 9
Train set: Average loss: 0.6906
Validation set: Average loss: 0.1689, Accuracy: 9/16 (56%)

Train Epoch: 10
Train set: Average loss: 0.6951
Validation set: Average loss: 0.1936, Accuracy: 9/16 (56%)

Time Elapsed: 0:12:12.456683
```

[8]:
```
train_binary_normal_clf(10, 4, savePath=None, model=mobilenet, weight=None,
    quiet=True)
train_binary_covid_clf(10, 4, savePath=None, model=mobilenet, weight=None,
    quiet=True)
```

```
Train Epoch: 1
Train set: Average loss: 0.5432
Validation set: Average loss: 0.1180, Accuracy: 16/24 (67%)

Found New Minima at epoch 1 loss: 0.11798713852961858

Train Epoch: 2
Train set: Average loss: 0.4377
```

```
Validation set: Average loss: 0.1125, Accuracy: 20/24 (83%)

Found New Minima at epoch 2 loss: 0.11252926041682561

Train Epoch: 3
Train set: Average loss: 0.4176
Validation set: Average loss: 0.1160, Accuracy: 17/24 (71%)

Train Epoch: 4
Train set: Average loss: 0.4309
Validation set: Average loss: 0.1445, Accuracy: 18/24 (75%)

Train Epoch: 5
Train set: Average loss: 0.4226
Validation set: Average loss: 0.1505, Accuracy: 17/24 (71%)

Train Epoch: 6
Train set: Average loss: 0.4203
Validation set: Average loss: 0.1665, Accuracy: 18/24 (75%)

Train Epoch: 7
Train set: Average loss: 0.4395
Validation set: Average loss: 0.1420, Accuracy: 18/24 (75%)

Train Epoch: 8
Train set: Average loss: 0.4257
Validation set: Average loss: 0.1222, Accuracy: 17/24 (71%)

Train Epoch: 9
Train set: Average loss: 0.4033
Validation set: Average loss: 0.1789, Accuracy: 17/24 (71%)

Train Epoch: 10
Train set: Average loss: 0.4364
Validation set: Average loss: 0.1826, Accuracy: 15/24 (62%)

Time Elapsed: 0:08:49.410031

Train Epoch: 1
Train set: Average loss: 1.3180
Validation set: Average loss: 0.1964, Accuracy: 8/16 (50%)

Found New Minima at epoch 1 loss: 0.19638646766543388

Train Epoch: 2
Train set: Average loss: 0.6878
Validation set: Average loss: 0.1789, Accuracy: 6/16 (38%)
```

```
Found New Minima at epoch 2 loss: 0.17894272133708

Train Epoch: 3
Train set: Average loss: 0.6449
Validation set: Average loss: 0.1835, Accuracy: 8/16 (50%)

Train Epoch: 4
Train set: Average loss: 0.6416
Validation set: Average loss: 0.1837, Accuracy: 9/16 (56%)

Train Epoch: 5
Train set: Average loss: 0.6416
Validation set: Average loss: 0.1848, Accuracy: 8/16 (50%)

Train Epoch: 6
Train set: Average loss: 0.6404
Validation set: Average loss: 0.1621, Accuracy: 10/16 (62%)

Found New Minima at epoch 6 loss: 0.1621499713510275

Train Epoch: 7
Train set: Average loss: 0.6383
Validation set: Average loss: 0.1508, Accuracy: 10/16 (62%)

Found New Minima at epoch 7 loss: 0.1507614701986313

Train Epoch: 8
Train set: Average loss: 0.6356
Validation set: Average loss: 0.1668, Accuracy: 8/16 (50%)

Train Epoch: 9
Train set: Average loss: 0.6370
Validation set: Average loss: 0.1798, Accuracy: 8/16 (50%)

Train Epoch: 10
Train set: Average loss: 0.6397
Validation set: Average loss: 0.1697, Accuracy: 9/16 (56%)

Time Elapsed: 0:06:32.558262
```

6 Final Modified ResNet:

The final model we used is a modified version of ResNet-18 as it is the highest performing of all that we tested. Additionally, we did not go with deeper layered resnet as it started to show overfitting after longer training, and we believed ResNet-18 has the sufficient layers to allow the model to generalise better.

We also experimented with adding dropout layers and sees a slight increase in performance.

[2]:
```python
from model import ResNet
model=ResNet()
print(model)
```

```
ResNet(
  (layers): Sequential(
    (0): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (4): ResBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (5): ResBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (6): ResBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
```

```
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (7): ResBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (8): ResBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (9): ResBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (10): ResBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (11): ResBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (12): AdaptiveAvgPool2d(output_size=(1, 1))
    (13): Flatten(start_dim=1, end_dim=-1)
  )
  (fc2): Sequential(
    (0): Linear(in_features=512, out_features=256, bias=True)
    (1): Linear(in_features=256, out_features=128, bias=True)
    (2): Dropout(p=0.25, inplace=False)
    (3): Linear(in_features=128, out_features=64, bias=True)
    (4): Dropout(p=0.25, inplace=False)
    (5): Linear(in_features=64, out_features=2, bias=True)
  )
)
```

6.1 Loss Function and Weight Adjustment

We used the Cross Entropy Loss criterion implemented in pytorch for convenience, which combines LogSoftmax and NLLLoss in one single class.

In order to finetune the model to optimize for `sensitivity` of our model at each stage, we input a weight tensor which would **penalise false negative more** (misclassifying `infected` and `covid` as `normal` and `non-covid` for stage 1 and 2 respectively). While perhaps slightly 'hacky', it was successful in increasing the sensitivity. Through some trial and error, we arrived at weights `[1.0, 1.1]` for first classifier (`normal classifier`) and `[1.0, 1.15]` for second classifier (`covid classifier`) which maximised sensitivity without sacrificing too much accuracy.

Note that, the original model without weight adjustment has performed fairly well in terms of `sensitivity` probably thanks to oversampling and the architecture. However, to further optimize for `sensitivity` we find that adjusting the weights in the loss function works best even though it comes at the cost of some `accuracy`.

[1]:
```
!python train.py
```

```
Train Epoch: 1
Train Epoch: 1 [0/6557 (0%)]    Loss: 10.913421
Train Epoch: 1 [400/6557 (6%)]  Loss: 10.566383
Train Epoch: 1 [800/6557 (12%)] Loss: 9.716506
Train Epoch: 1 [1200/6557 (18%)]        Loss: 7.353319
Train Epoch: 1 [1600/6557 (24%)]        Loss: 5.709531
Train Epoch: 1 [2000/6557 (30%)]        Loss: 6.725890
Train Epoch: 1 [2400/6557 (37%)]        Loss: 5.508781
Train Epoch: 1 [2800/6557 (43%)]        Loss: 5.377548
Train Epoch: 1 [3200/6557 (49%)]        Loss: 2.343995
Train Epoch: 1 [3600/6557 (55%)]        Loss: 11.576360
Train Epoch: 1 [4000/6557 (61%)]        Loss: 5.545029
Train Epoch: 1 [4400/6557 (67%)]        Loss: 8.961969
Train Epoch: 1 [4800/6557 (73%)]        Loss: 2.716870
Train Epoch: 1 [5200/6557 (79%)]        Loss: 12.883558
Train Epoch: 1 [5600/6557 (85%)]        Loss: 1.533314
Train Epoch: 1 [6000/6557 (91%)]        Loss: 14.366716
Train Epoch: 1 [6400/6557 (98%)]        Loss: 2.992687
Train set: Average loss: 0.7491
Validation set: Average loss: 0.0703, Accuracy: 22/32 (69%)

Found New Minima at epoch 1 loss: 0.07026839628815651

Train Epoch: 2
Train Epoch: 2 [0/6557 (0%)]    Loss: 3.005091
Train Epoch: 2 [400/6557 (6%)]  Loss: 2.912483
Train Epoch: 2 [800/6557 (12%)] Loss: 1.539295
Train Epoch: 2 [1200/6557 (18%)]        Loss: 4.182378
Train Epoch: 2 [1600/6557 (24%)]        Loss: 0.677988
Train Epoch: 2 [2000/6557 (30%)]        Loss: 3.190234
Train Epoch: 2 [2400/6557 (37%)]        Loss: 9.510954
Train Epoch: 2 [2800/6557 (43%)]        Loss: 0.685837
Train Epoch: 2 [3200/6557 (49%)]        Loss: 11.204675
Train Epoch: 2 [3600/6557 (55%)]        Loss: 2.251018
Train Epoch: 2 [4000/6557 (61%)]        Loss: 0.202888
Train Epoch: 2 [4400/6557 (67%)]        Loss: 0.428268
Train Epoch: 2 [4800/6557 (73%)]        Loss: 0.902640
Train Epoch: 2 [5200/6557 (79%)]        Loss: 0.779482
Train Epoch: 2 [5600/6557 (85%)]        Loss: 3.077314
Train Epoch: 2 [6000/6557 (91%)]        Loss: 0.108082
Train Epoch: 2 [6400/6557 (98%)]        Loss: 2.606936
Train set: Average loss: 0.4654
Validation set: Average loss: 0.0911, Accuracy: 22/32 (69%)
```

```
Train Epoch: 3
Train Epoch: 3 [0/6557 (0%)]    Loss: 13.441779
Train Epoch: 3 [400/6557 (6%)]  Loss: 1.215724
Train Epoch: 3 [800/6557 (12%)] Loss: 0.349729
Train Epoch: 3 [1200/6557 (18%)]        Loss: 1.844385
Train Epoch: 3 [1600/6557 (24%)]        Loss: 1.585422
Train Epoch: 3 [2000/6557 (30%)]        Loss: 0.451049
Train Epoch: 3 [2400/6557 (37%)]        Loss: 19.699875
Train Epoch: 3 [2800/6557 (43%)]        Loss: 0.638687
Train Epoch: 3 [3200/6557 (49%)]        Loss: 6.737168
Train Epoch: 3 [3600/6557 (55%)]        Loss: 0.209818
Train Epoch: 3 [4000/6557 (61%)]        Loss: 0.373560
Train Epoch: 3 [4400/6557 (67%)]        Loss: 7.902338
Train Epoch: 3 [4800/6557 (73%)]        Loss: 4.522370
Train Epoch: 3 [5200/6557 (79%)]        Loss: 5.018671
Train Epoch: 3 [5600/6557 (85%)]        Loss: 0.932905
Train Epoch: 3 [6000/6557 (91%)]        Loss: 2.721285
Train Epoch: 3 [6400/6557 (98%)]        Loss: 0.225533
Train set: Average loss: 0.4198
Validation set: Average loss: 0.1518, Accuracy: 18/32 (56%)

Train Epoch: 4
Train Epoch: 4 [0/6557 (0%)]    Loss: 1.228575
Train Epoch: 4 [400/6557 (6%)]  Loss: 14.199380
Train Epoch: 4 [800/6557 (12%)] Loss: 0.539935
Train Epoch: 4 [1200/6557 (18%)]        Loss: 0.614304
Train Epoch: 4 [1600/6557 (24%)]        Loss: 1.950186
Train Epoch: 4 [2000/6557 (30%)]        Loss: 0.466521
Train Epoch: 4 [2400/6557 (37%)]        Loss: 0.190448
Train Epoch: 4 [2800/6557 (43%)]        Loss: 1.665514
Train Epoch: 4 [3200/6557 (49%)]        Loss: 0.042150
Train Epoch: 4 [3600/6557 (55%)]        Loss: 0.823390
Train Epoch: 4 [4000/6557 (61%)]        Loss: 0.200822
Train Epoch: 4 [4400/6557 (67%)]        Loss: 4.819230
Train Epoch: 4 [4800/6557 (73%)]        Loss: 0.034330
Train Epoch: 4 [5200/6557 (79%)]        Loss: 0.832343
Train Epoch: 4 [5600/6557 (85%)]        Loss: 0.078937
Train Epoch: 4 [6000/6557 (91%)]        Loss: 0.469817
Train Epoch: 4 [6400/6557 (98%)]        Loss: 0.101998
Train set: Average loss: 0.3835
Validation set: Average loss: 0.0712, Accuracy: 22/32 (69%)

Train Epoch: 5
Train Epoch: 5 [0/6557 (0%)]    Loss: 2.178116
Train Epoch: 5 [400/6557 (6%)]  Loss: 2.158122
Train Epoch: 5 [800/6557 (12%)] Loss: 13.791509
Train Epoch: 5 [1200/6557 (18%)]        Loss: 20.483440
```

```
Train Epoch: 5 [1600/6557 (24%)]        Loss: 1.479434
Train Epoch: 5 [2000/6557 (30%)]        Loss: 16.101559
Train Epoch: 5 [2400/6557 (37%)]        Loss: 0.722625
Train Epoch: 5 [2800/6557 (43%)]        Loss: 0.773676
Train Epoch: 5 [3200/6557 (49%)]        Loss: 0.064216
Train Epoch: 5 [3600/6557 (55%)]        Loss: 0.036284
Train Epoch: 5 [4000/6557 (61%)]        Loss: 14.094223
Train Epoch: 5 [4400/6557 (67%)]        Loss: 2.541386
Train Epoch: 5 [4800/6557 (73%)]        Loss: 0.490635
Train Epoch: 5 [5200/6557 (79%)]        Loss: 8.170601
Train Epoch: 5 [5600/6557 (85%)]        Loss: 0.495575
Train Epoch: 5 [6000/6557 (91%)]        Loss: 0.143888
Train Epoch: 5 [6400/6557 (98%)]        Loss: 1.913144
Train set: Average loss: 0.4028
Validation set: Average loss: 0.0753, Accuracy: 22/32 (69%)

Train Epoch: 6
Train Epoch: 6 [0/6557 (0%)]    Loss: 0.096746
Train Epoch: 6 [400/6557 (6%)]  Loss: 2.192136
Train Epoch: 6 [800/6557 (12%)] Loss: 0.013609
Train Epoch: 6 [1200/6557 (18%)]        Loss: 0.779619
Train Epoch: 6 [1600/6557 (24%)]        Loss: 0.232856
Train Epoch: 6 [2000/6557 (30%)]        Loss: 9.384152
Train Epoch: 6 [2400/6557 (37%)]        Loss: 0.680886
Train Epoch: 6 [2800/6557 (43%)]        Loss: 0.154583
Train Epoch: 6 [3200/6557 (49%)]        Loss: 0.111751
Train Epoch: 6 [3600/6557 (55%)]        Loss: 0.184725
Train Epoch: 6 [4000/6557 (61%)]        Loss: 3.666948
Train Epoch: 6 [4400/6557 (67%)]        Loss: 33.122601
Train Epoch: 6 [4800/6557 (73%)]        Loss: 1.627973
Train Epoch: 6 [5200/6557 (79%)]        Loss: 0.478219
Train Epoch: 6 [5600/6557 (85%)]        Loss: 1.578889
Train Epoch: 6 [6000/6557 (91%)]        Loss: 0.055772
Train Epoch: 6 [6400/6557 (98%)]        Loss: 0.147845
Train set: Average loss: 0.4345
Validation set: Average loss: 0.0647, Accuracy: 24/32 (75%)

Found New Minima at epoch 6 loss: 0.06470148742664605

Train Epoch: 7
Train Epoch: 7 [0/6557 (0%)]    Loss: 0.130107
Train Epoch: 7 [400/6557 (6%)]  Loss: 1.140107
Train Epoch: 7 [800/6557 (12%)] Loss: 0.024054
Train Epoch: 7 [1200/6557 (18%)]        Loss: 5.551694
Train Epoch: 7 [1600/6557 (24%)]        Loss: 0.012472
Train Epoch: 7 [2000/6557 (30%)]        Loss: 1.173953
Train Epoch: 7 [2400/6557 (37%)]        Loss: 0.027068
Train Epoch: 7 [2800/6557 (43%)]        Loss: 0.506161
```

```
Train Epoch: 7 [3200/6557 (49%)]        Loss: 0.744899
Train Epoch: 7 [3600/6557 (55%)]        Loss: 0.014669
Train Epoch: 7 [4000/6557 (61%)]        Loss: 0.026483
Train Epoch: 7 [4400/6557 (67%)]        Loss: 0.503039
Train Epoch: 7 [4800/6557 (73%)]        Loss: 2.804198
Train Epoch: 7 [5200/6557 (79%)]        Loss: 0.032361
Train Epoch: 7 [5600/6557 (85%)]        Loss: 0.004565
Train Epoch: 7 [6000/6557 (91%)]        Loss: 4.641452
Train Epoch: 7 [6400/6557 (98%)]        Loss: 0.016408
Train set: Average loss: 0.3790
Validation set: Average loss: 0.0478, Accuracy: 26/32 (81%)

Found New Minima at epoch 7 loss: 0.04779296857304871

Train Epoch: 8
Train Epoch: 8 [0/6557 (0%)]    Loss: 8.142362
Train Epoch: 8 [400/6557 (6%)]  Loss: 1.451730
Train Epoch: 8 [800/6557 (12%)] Loss: 0.116898
Train Epoch: 8 [1200/6557 (18%)]        Loss: 0.237416
Train Epoch: 8 [1600/6557 (24%)]        Loss: 4.703391
Train Epoch: 8 [2000/6557 (30%)]        Loss: 1.309311
Train Epoch: 8 [2400/6557 (37%)]        Loss: 4.592045
Train Epoch: 8 [2800/6557 (43%)]        Loss: 0.012344
Train Epoch: 8 [3200/6557 (49%)]        Loss: 0.261057
Train Epoch: 8 [3600/6557 (55%)]        Loss: 0.197889
Train Epoch: 8 [4000/6557 (61%)]        Loss: 16.453554
Train Epoch: 8 [4400/6557 (67%)]        Loss: 0.848722
Train Epoch: 8 [4800/6557 (73%)]        Loss: 4.529657
Train Epoch: 8 [5200/6557 (79%)]        Loss: 2.578305
Train Epoch: 8 [5600/6557 (85%)]        Loss: 2.578824
Train Epoch: 8 [6000/6557 (91%)]        Loss: 0.076440
Train Epoch: 8 [6400/6557 (98%)]        Loss: 5.054475
Train set: Average loss: 0.3437
Validation set: Average loss: 0.1250, Accuracy: 19/32 (59%)

Train Epoch: 9
Train Epoch: 9 [0/6557 (0%)]    Loss: 0.518210
Train Epoch: 9 [400/6557 (6%)]  Loss: 1.559361
Train Epoch: 9 [800/6557 (12%)] Loss: 0.107641
Train Epoch: 9 [1200/6557 (18%)]        Loss: 0.019909
Train Epoch: 9 [1600/6557 (24%)]        Loss: 0.044614
Train Epoch: 9 [2000/6557 (30%)]        Loss: 0.931150
Train Epoch: 9 [2400/6557 (37%)]        Loss: 0.959457
Train Epoch: 9 [2800/6557 (43%)]        Loss: 0.177521
Train Epoch: 9 [3200/6557 (49%)]        Loss: 3.516927
Train Epoch: 9 [3600/6557 (55%)]        Loss: 0.019684
Train Epoch: 9 [4000/6557 (61%)]        Loss: 0.435936
Train Epoch: 9 [4400/6557 (67%)]        Loss: 4.036559
```

```
Train Epoch: 9 [4800/6557 (73%)]        Loss: 4.777757
Train Epoch: 9 [5200/6557 (79%)]        Loss: 0.063135
Train Epoch: 9 [5600/6557 (85%)]        Loss: 0.758455
Train Epoch: 9 [6000/6557 (91%)]        Loss: 0.897656
Train Epoch: 9 [6400/6557 (98%)]        Loss: 0.121586
Train set: Average loss: 0.3408
Validation set: Average loss: 0.0669, Accuracy: 25/32 (78%)

Train Epoch: 10
Train Epoch: 10 [0/6557 (0%)]   Loss: 0.693939
Train Epoch: 10 [400/6557 (6%)] Loss: 0.053657
Train Epoch: 10 [800/6557 (12%)]        Loss: 0.000213
Train Epoch: 10 [1200/6557 (18%)]       Loss: 0.013121
Train Epoch: 10 [1600/6557 (24%)]       Loss: 0.135257
Train Epoch: 10 [2000/6557 (30%)]       Loss: 0.009869
Train Epoch: 10 [2400/6557 (37%)]       Loss: 0.012400
Train Epoch: 10 [2800/6557 (43%)]       Loss: 1.011444
Train Epoch: 10 [3200/6557 (49%)]       Loss: 0.037029
Train Epoch: 10 [3600/6557 (55%)]       Loss: 0.165183
Train Epoch: 10 [4000/6557 (61%)]       Loss: 6.063089
Train Epoch: 10 [4400/6557 (67%)]       Loss: 0.579866
Train Epoch: 10 [4800/6557 (73%)]       Loss: 12.427422
Train Epoch: 10 [5200/6557 (79%)]       Loss: 0.002017
Train Epoch: 10 [5600/6557 (85%)]       Loss: 0.012929
Train Epoch: 10 [6000/6557 (91%)]       Loss: 0.016079
Train Epoch: 10 [6400/6557 (98%)]       Loss: 0.859191
Train set: Average loss: 0.3366
Validation set: Average loss: 0.0801, Accuracy: 21/32 (66%)

Train Epoch: 11
Train Epoch: 11 [0/6557 (0%)]   Loss: 2.634500
Train Epoch: 11 [400/6557 (6%)] Loss: 7.851204
Train Epoch: 11 [800/6557 (12%)]        Loss: 2.202120
Train Epoch: 11 [1200/6557 (18%)]       Loss: 0.001797
Train Epoch: 11 [1600/6557 (24%)]       Loss: 0.047693
Train Epoch: 11 [2000/6557 (30%)]       Loss: 0.015314
Train Epoch: 11 [2400/6557 (37%)]       Loss: 0.892511
Train Epoch: 11 [2800/6557 (43%)]       Loss: 1.782406
Train Epoch: 11 [3200/6557 (49%)]       Loss: 1.621803
Train Epoch: 11 [3600/6557 (55%)]       Loss: 0.003357
Train Epoch: 11 [4000/6557 (61%)]       Loss: 4.193816
Train Epoch: 11 [4400/6557 (67%)]       Loss: 0.026110
Train Epoch: 11 [4800/6557 (73%)]       Loss: 0.137113
Train Epoch: 11 [5200/6557 (79%)]       Loss: 0.451956
Train Epoch: 11 [5600/6557 (85%)]       Loss: 1.001984
Train Epoch: 11 [6000/6557 (91%)]       Loss: 0.000489
Train Epoch: 11 [6400/6557 (98%)]       Loss: 0.004192
Train set: Average loss: 0.3462
```

```
Validation set: Average loss: 0.0808, Accuracy: 21/32 (66%)

Train Epoch: 12
Train Epoch: 12 [0/6557 (0%)]   Loss: 0.906866
Train Epoch: 12 [400/6557 (6%)] Loss: 0.019737
Train Epoch: 12 [800/6557 (12%)]        Loss: 0.021294
Train Epoch: 12 [1200/6557 (18%)]       Loss: 1.895911
Train Epoch: 12 [1600/6557 (24%)]       Loss: 0.000335
Train Epoch: 12 [2000/6557 (30%)]       Loss: 0.025165
Train Epoch: 12 [2400/6557 (37%)]       Loss: 0.000065
Train Epoch: 12 [2800/6557 (43%)]       Loss: 7.604950
Train Epoch: 12 [3200/6557 (49%)]       Loss: 0.851316
Train Epoch: 12 [3600/6557 (55%)]       Loss: 0.436386
Train Epoch: 12 [4000/6557 (61%)]       Loss: 2.536125
Train Epoch: 12 [4400/6557 (67%)]       Loss: 1.590642
Train Epoch: 12 [4800/6557 (73%)]       Loss: 13.502479
Train Epoch: 12 [5200/6557 (79%)]       Loss: 0.902547
Train Epoch: 12 [5600/6557 (85%)]       Loss: 0.001424
Train Epoch: 12 [6000/6557 (91%)]       Loss: 1.107133
Train Epoch: 12 [6400/6557 (98%)]       Loss: 0.128182
Train set: Average loss: 0.3297
Validation set: Average loss: 0.0720, Accuracy: 21/32 (66%)

Time Elapsed: 0:07:06.264763

Train Epoch: 1
Train Epoch: 1 [0/5220 (0%)]    Loss: 6.008618
Train Epoch: 1 [400/5220 (8%)]  Loss: 5.864100
Train Epoch: 1 [800/5220 (15%)] Loss: 5.835737
Train Epoch: 1 [1200/5220 (23%)]        Loss: 6.355176
Train Epoch: 1 [1600/5220 (31%)]        Loss: 5.325252
Train Epoch: 1 [2000/5220 (38%)]        Loss: 5.525688
Train Epoch: 1 [2400/5220 (46%)]        Loss: 6.100036
Train Epoch: 1 [2800/5220 (54%)]        Loss: 6.141569
Train Epoch: 1 [3200/5220 (61%)]        Loss: 5.870410
Train Epoch: 1 [3600/5220 (69%)]        Loss: 5.323463
Train Epoch: 1 [4000/5220 (77%)]        Loss: 6.165069
Train Epoch: 1 [4400/5220 (84%)]        Loss: 5.468444
Train Epoch: 1 [4800/5220 (92%)]        Loss: 5.374788
Train Epoch: 1 [5200/5220 (100%)]       Loss: 5.079737
Train set: Average loss: 0.7119
Validation set: Average loss: 0.0799, Accuracy: 10/16 (62%)

Found New Minima at epoch 1 loss: 0.07992818206548691

Train Epoch: 2
Train Epoch: 2 [0/5220 (0%)]    Loss: 5.588284
Train Epoch: 2 [400/5220 (8%)]  Loss: 4.916242
```

```
Train Epoch: 2 [800/5220 (15%)] Loss: 5.580169
Train Epoch: 2 [1200/5220 (23%)]        Loss: 4.798948
Train Epoch: 2 [1600/5220 (31%)]        Loss: 4.344351
Train Epoch: 2 [2000/5220 (38%)]        Loss: 5.820178
Train Epoch: 2 [2400/5220 (46%)]        Loss: 4.766841
Train Epoch: 2 [2800/5220 (54%)]        Loss: 4.587791
Train Epoch: 2 [3200/5220 (61%)]        Loss: 4.509837
Train Epoch: 2 [3600/5220 (69%)]        Loss: 4.437029
Train Epoch: 2 [4000/5220 (77%)]        Loss: 4.966983
Train Epoch: 2 [4400/5220 (84%)]        Loss: 4.645201
Train Epoch: 2 [4800/5220 (92%)]        Loss: 4.754342
Train Epoch: 2 [5200/5220 (100%)]       Loss: 4.023651
Train set: Average loss: 0.6774
Validation set: Average loss: 0.0826, Accuracy: 9/16 (56%)

Train Epoch: 3
Train Epoch: 3 [0/5220 (0%)]    Loss: 7.577713
Train Epoch: 3 [400/5220 (8%)]  Loss: 4.507255
Train Epoch: 3 [800/5220 (15%)] Loss: 5.466890
Train Epoch: 3 [1200/5220 (23%)]        Loss: 6.120241
Train Epoch: 3 [1600/5220 (31%)]        Loss: 3.093391
Train Epoch: 3 [2000/5220 (38%)]        Loss: 7.855576
Train Epoch: 3 [2400/5220 (46%)]        Loss: 4.207525
Train Epoch: 3 [2800/5220 (54%)]        Loss: 7.261671
Train Epoch: 3 [3200/5220 (61%)]        Loss: 5.306428
Train Epoch: 3 [3600/5220 (69%)]        Loss: 4.740294
Train Epoch: 3 [4000/5220 (77%)]        Loss: 5.249031
Train Epoch: 3 [4400/5220 (84%)]        Loss: 8.380112
Train Epoch: 3 [4800/5220 (92%)]        Loss: 5.299091
Train Epoch: 3 [5200/5220 (100%)]       Loss: 4.277110
Train set: Average loss: 0.6506
Validation set: Average loss: 0.0966, Accuracy: 10/16 (62%)

Train Epoch: 4
Train Epoch: 4 [0/5220 (0%)]    Loss: 4.876611
Train Epoch: 4 [400/5220 (8%)]  Loss: 5.561759
Train Epoch: 4 [800/5220 (15%)] Loss: 3.845500
Train Epoch: 4 [1200/5220 (23%)]        Loss: 3.645385
Train Epoch: 4 [1600/5220 (31%)]        Loss: 3.945632
Train Epoch: 4 [2000/5220 (38%)]        Loss: 6.144691
Train Epoch: 4 [2400/5220 (46%)]        Loss: 4.311610
Train Epoch: 4 [2800/5220 (54%)]        Loss: 5.458856
Train Epoch: 4 [3200/5220 (61%)]        Loss: 8.269901
Train Epoch: 4 [3600/5220 (69%)]        Loss: 7.280322
Train Epoch: 4 [4000/5220 (77%)]        Loss: 5.489301
Train Epoch: 4 [4400/5220 (84%)]        Loss: 4.741888
Train Epoch: 4 [4800/5220 (92%)]        Loss: 4.970676
Train Epoch: 4 [5200/5220 (100%)]       Loss: 4.612357
```

```
Train set: Average loss: 0.6374
Validation set: Average loss: 0.0895, Accuracy: 11/16 (69%)

Train Epoch: 5
Train Epoch: 5 [0/5220 (0%)]    Loss: 3.753958
Train Epoch: 5 [400/5220 (8%)]  Loss: 5.924087
Train Epoch: 5 [800/5220 (15%)] Loss: 3.212911
Train Epoch: 5 [1200/5220 (23%)]        Loss: 3.814188
Train Epoch: 5 [1600/5220 (31%)]        Loss: 8.233854
Train Epoch: 5 [2000/5220 (38%)]        Loss: 3.261796
Train Epoch: 5 [2400/5220 (46%)]        Loss: 4.396448
Train Epoch: 5 [2800/5220 (54%)]        Loss: 4.207638
Train Epoch: 5 [3200/5220 (61%)]        Loss: 4.072484
Train Epoch: 5 [3600/5220 (69%)]        Loss: 4.696505
Train Epoch: 5 [4000/5220 (77%)]        Loss: 4.751857
Train Epoch: 5 [4400/5220 (84%)]        Loss: 4.693773
Train Epoch: 5 [4800/5220 (92%)]        Loss: 4.335231
Train Epoch: 5 [5200/5220 (100%)]       Loss: 4.018120
Train set: Average loss: 0.6229
Validation set: Average loss: 0.0759, Accuracy: 10/16 (62%)

Found New Minima at epoch 5 loss: 0.07594491168856621

Train Epoch: 6
Train Epoch: 6 [0/5220 (0%)]    Loss: 6.420358
Train Epoch: 6 [400/5220 (8%)]  Loss: 4.004513
Train Epoch: 6 [800/5220 (15%)] Loss: 7.143463
Train Epoch: 6 [1200/5220 (23%)]        Loss: 6.096256
Train Epoch: 6 [1600/5220 (31%)]        Loss: 4.621164
Train Epoch: 6 [2000/5220 (38%)]        Loss: 5.971611
Train Epoch: 6 [2400/5220 (46%)]        Loss: 3.550537
Train Epoch: 6 [2800/5220 (54%)]        Loss: 1.759524
Train Epoch: 6 [3200/5220 (61%)]        Loss: 8.084679
Train Epoch: 6 [3600/5220 (69%)]        Loss: 5.892495
Train Epoch: 6 [4000/5220 (77%)]        Loss: 2.525831
Train Epoch: 6 [4400/5220 (84%)]        Loss: 2.787439
Train Epoch: 6 [4800/5220 (92%)]        Loss: 4.270261
Train Epoch: 6 [5200/5220 (100%)]       Loss: 3.448928
Train set: Average loss: 0.6066
Validation set: Average loss: 0.0942, Accuracy: 11/16 (69%)

Train Epoch: 7
Train Epoch: 7 [0/5220 (0%)]    Loss: 4.488837
Train Epoch: 7 [400/5220 (8%)]  Loss: 7.157794
Train Epoch: 7 [800/5220 (15%)] Loss: 11.756270
Train Epoch: 7 [1200/5220 (23%)]        Loss: 1.958828
Train Epoch: 7 [1600/5220 (31%)]        Loss: 3.750852
Train Epoch: 7 [2000/5220 (38%)]        Loss: 3.955179
```

```
Train Epoch: 7 [2400/5220 (46%)]        Loss: 5.912403
Train Epoch: 7 [2800/5220 (54%)]        Loss: 2.541599
Train Epoch: 7 [3200/5220 (61%)]        Loss: 7.514783
Train Epoch: 7 [3600/5220 (69%)]        Loss: 5.934039
Train Epoch: 7 [4000/5220 (77%)]        Loss: 4.173022
Train Epoch: 7 [4400/5220 (84%)]        Loss: 7.196945
Train Epoch: 7 [4800/5220 (92%)]        Loss: 4.802866
Train Epoch: 7 [5200/5220 (100%)]       Loss: 4.385753
Train set: Average loss: 0.6046
Validation set: Average loss: 0.1127, Accuracy: 10/16 (62%)

Train Epoch: 8
Train Epoch: 8 [0/5220 (0%)]    Loss: 5.984366
Train Epoch: 8 [400/5220 (8%)]  Loss: 2.932850
Train Epoch: 8 [800/5220 (15%)] Loss: 3.726764
Train Epoch: 8 [1200/5220 (23%)]        Loss: 3.329394
Train Epoch: 8 [1600/5220 (31%)]        Loss: 3.516551
Train Epoch: 8 [2000/5220 (38%)]        Loss: 5.843719
Train Epoch: 8 [2400/5220 (46%)]        Loss: 5.269373
Train Epoch: 8 [2800/5220 (54%)]        Loss: 7.495790
Train Epoch: 8 [3200/5220 (61%)]        Loss: 4.008193
Train Epoch: 8 [3600/5220 (69%)]        Loss: 6.361718
Train Epoch: 8 [4000/5220 (77%)]        Loss: 4.181150
Train Epoch: 8 [4400/5220 (84%)]        Loss: 5.002705
Train Epoch: 8 [4800/5220 (92%)]        Loss: 4.166010
Train Epoch: 8 [5200/5220 (100%)]       Loss: 6.192740
Train set: Average loss: 0.5981
Validation set: Average loss: 0.0882, Accuracy: 11/16 (69%)

Train Epoch: 9
Train Epoch: 9 [0/5220 (0%)]    Loss: 5.594940
Train Epoch: 9 [400/5220 (8%)]  Loss: 4.554133
Train Epoch: 9 [800/5220 (15%)] Loss: 5.209601
Train Epoch: 9 [1200/5220 (23%)]        Loss: 4.527403
Train Epoch: 9 [1600/5220 (31%)]        Loss: 2.897386
Train Epoch: 9 [2000/5220 (38%)]        Loss: 5.984015
Train Epoch: 9 [2400/5220 (46%)]        Loss: 4.999108
Train Epoch: 9 [2800/5220 (54%)]        Loss: 3.919450
Train Epoch: 9 [3200/5220 (61%)]        Loss: 5.205777
Train Epoch: 9 [3600/5220 (69%)]        Loss: 3.175725
Train Epoch: 9 [4000/5220 (77%)]        Loss: 1.639878
Train Epoch: 9 [4400/5220 (84%)]        Loss: 4.218152
Train Epoch: 9 [4800/5220 (92%)]        Loss: 6.607555
Train Epoch: 9 [5200/5220 (100%)]       Loss: 6.303160
Train set: Average loss: 0.5796
Validation set: Average loss: 0.1020, Accuracy: 10/16 (62%)

Train Epoch: 10
```

```
Train Epoch: 10 [0/5220 (0%)]   Loss: 6.282562
Train Epoch: 10 [400/5220 (8%)] Loss: 3.032850
Train Epoch: 10 [800/5220 (15%)]        Loss: 5.607584
Train Epoch: 10 [1200/5220 (23%)]       Loss: 2.275177
Train Epoch: 10 [1600/5220 (31%)]       Loss: 2.817718
Train Epoch: 10 [2000/5220 (38%)]       Loss: 3.813203
Train Epoch: 10 [2400/5220 (46%)]       Loss: 5.436763
Train Epoch: 10 [2800/5220 (54%)]       Loss: 5.010543
Train Epoch: 10 [3200/5220 (61%)]       Loss: 3.991203
Train Epoch: 10 [3600/5220 (69%)]       Loss: 2.730826
Train Epoch: 10 [4000/5220 (77%)]       Loss: 3.858616
Train Epoch: 10 [4400/5220 (84%)]       Loss: 5.786072
Train Epoch: 10 [4800/5220 (92%)]       Loss: 4.971880
Train Epoch: 10 [5200/5220 (100%)]      Loss: 4.344845
Train set: Average loss: 0.5853
Validation set: Average loss: 0.1159, Accuracy: 10/16 (62%)

Train Epoch: 11
Train Epoch: 11 [0/5220 (0%)]   Loss: 7.629343
Train Epoch: 11 [400/5220 (8%)] Loss: 4.922375
Train Epoch: 11 [800/5220 (15%)]        Loss: 7.401924
Train Epoch: 11 [1200/5220 (23%)]       Loss: 3.160268
Train Epoch: 11 [1600/5220 (31%)]       Loss: 6.030275
Train Epoch: 11 [2000/5220 (38%)]       Loss: 6.154574
Train Epoch: 11 [2400/5220 (46%)]       Loss: 2.861617
Train Epoch: 11 [2800/5220 (54%)]       Loss: 4.526337
Train Epoch: 11 [3200/5220 (61%)]       Loss: 3.403824
Train Epoch: 11 [3600/5220 (69%)]       Loss: 5.099963
Train Epoch: 11 [4000/5220 (77%)]       Loss: 6.503198
Train Epoch: 11 [4400/5220 (84%)]       Loss: 4.688989
Train Epoch: 11 [4800/5220 (92%)]       Loss: 2.978997
Train Epoch: 11 [5200/5220 (100%)]      Loss: 4.243023
Train set: Average loss: 0.5794
Validation set: Average loss: 0.1511, Accuracy: 8/16 (50%)

Train Epoch: 12
Train Epoch: 12 [0/5220 (0%)]   Loss: 2.712164
Train Epoch: 12 [400/5220 (8%)] Loss: 6.251606
Train Epoch: 12 [800/5220 (15%)]        Loss: 5.120265
Train Epoch: 12 [1200/5220 (23%)]       Loss: 4.581521
Train Epoch: 12 [1600/5220 (31%)]       Loss: 5.692718
Train Epoch: 12 [2000/5220 (38%)]       Loss: 4.033245
Train Epoch: 12 [2400/5220 (46%)]       Loss: 3.172640
Train Epoch: 12 [2800/5220 (54%)]       Loss: 2.859644
Train Epoch: 12 [3200/5220 (61%)]       Loss: 4.076803
Train Epoch: 12 [3600/5220 (69%)]       Loss: 4.513265
Train Epoch: 12 [4000/5220 (77%)]       Loss: 5.294413
Train Epoch: 12 [4400/5220 (84%)]       Loss: 6.108240
```

```
Train Epoch: 12 [4800/5220 (92%)]       Loss: 1.948889
Train Epoch: 12 [5200/5220 (100%)]      Loss: 2.811039
Train set: Average loss: 0.5655
Validation set: Average loss: 0.1266, Accuracy: 9/16 (56%)

Time Elapsed: 0:05:38.494729
```

7 Results

The results will be delivered by specifying args to the test file. For brevity, we will compare results in the following manner:

1. Independent Binary Accuracy
2. Independent Binary Sensitivity
3. Independent Binary Confusion Matrix
4. Piped Binary Accuracy
5. Piped Binary Sensitivity
6. Comparison of Validation Images

2 classifiers will be used throughout. One of it is referred to as the `normal classifier`, which outputs 0 for `normal`, 1 for `infected` (both covid and non-covid). The other one is referred to as the `covid classifier`, which outputs 0 for `non-covid`, 1 for `covid`. Since we are interested in seeing how accurate and sensitive they can get, we have 2 different set of models for both of these metrics. The model being used will be stated clearly in the following sections.

7.1 Independent Metrics

For the following metrics, both binary classifiers are treated as independent as we are interested in them individually. The following notations are used:

- `TP`: True Positive
- `FP`: False Positive
- `TN`: True Negative
- `FN`: False Negative

TAKE NOTE The **first** set of returned result will treat **normal as negative, infected as positive**. The **second** set of returned result will treat **non-covid as negative, covid as positive**

7.1.1 1. Independent Binary Accuracy

We are interested in their individual accuracy in terms of predicting labels that are the same as the ground truth. The metric that we will be focusing on will be **"Testing Accuracy"**. This is obtained by taking (TP + TN) / len(testloader). Both the accuracy model and the sensitivity model will be tested.

7.1.2 2. Independent Binary Sensitivity

The metric that we will be focusing on will be the **"Testing Sensitivity"**. This metric reflects how well the model is prioritising misclassifying cases as infected/covid ones as a preventive measure.

This is obtained by taking safe_division(TP, (TP+FN)). The safe_division method returns 0 if there is division by zero from TP+FN, otherwise returns the regular division. Both the accuracy model and the sensitivity model will be tested.

7.1.3 3. Independent Binary Confustion Matrix

This metric puts together the confusion matrix for both classifiers. We will be looking at **Confusion Matrix**. Both the accuracy model and the sensitivity model will be tested

[2]:
```
# best accuracy model
!python test.py --independent 1 --validation 0 --print 1 --output_var 2␣
↪--normalclf binaryModelNormalBest --covidclf binaryModelCovidBest
```

```
Starting: Test set on Normal Independent classifier
Total=614, TP=369, FP=79, FN=11, TN=155
Testing Accuracy: 0.853
Testing Sensitivity: 0.971
Testing Specificity: 0.662
Testing PPV: 0.824
Testing NPV: 0.934
Testing F1 Score: 0.891
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 155  |   FP = 79   |
Ground P |   FN = 11  |   TP = 369   |


Starting Test set on Covid Independent classifier
Total=380, TP=126, FP=36, FN=12, TN=206
Testing Accuracy: 0.874
Testing Sensitivity: 0.913
Testing Specificity: 0.851
Testing PPV: 0.778
Testing NPV: 0.945
Testing F1 Score: 0.840
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 206  |   FP = 36   |
Ground P |   FN = 12  |   TP = 126   |
```

[3]:
```
# best sensitivity model
!python test.py --independent 1 --validation 0 --print 1 --output_var 2␣
↪--normalclf binaryModelNormalBestSensitivity --covidclf␣
↪binaryModelCovidBestSensitivity
```

```
Starting: Test set on Normal Independent classifier
Total=614, TP=377, FP=116, FN=3, TN=118
Testing Accuracy: 0.806
Testing Sensitivity: 0.992
Testing Specificity: 0.504
Testing PPV: 0.765
Testing NPV: 0.975
Testing F1 Score: 0.864
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 118  |   FP = 116   |
Ground P |   FN = 3  |   TP = 377   |


Starting Test set on Covid Independent classifier
Total=380, TP=129, FP=61, FN=9, TN=181
Testing Accuracy: 0.816
Testing Sensitivity: 0.935
Testing Specificity: 0.748
Testing PPV: 0.679
Testing NPV: 0.953
Testing F1 Score: 0.787
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 181  |   FP = 61   |
Ground P |   FN = 9  |   TP = 129   |
```

As can be seen, the `binaryModelNormalBest` and `binaryModelCovidBest` model scores better in terms of `accuracy` at the cost of some amount of false negatives.

Whereas if we were to optimize for `sensitivity` by penalizing FN through weight adjustment in the Cross Entropy Loss function, we can improve the `sensitivity` at the cost of `accuracy` as shown in the `binaryModelNormalBestSensitivity` and `binaryModelCovidBestSensitivity` model.

7.2 Piped Binary Metrics

As we ultimately chose to use a piped double binary model for this project (second diagram in the given project handout), the printed metric values may look different from its independently considered counterparts in the above section. Once again, TP FP TN FN will be used.

TAKE NOTE The **first** set of returned result will treated **normal as negative, infected as positive**. The **second** set of returned result will treat **non-covid/normal as negative, covid as positive**

7.2.1 4. Piped Binary Accuracy

The metric that we will be focusing on will be **"Testing Accuracy"**. This is obtained by taking (TP + TN) / len(testloader). Both the accuracy model and the sensitivity model will be tested.

7.2.2 5. Piped Binary Sensitivity

The metric that we will be focusing on will be **"Testing Sensitivity"**. This metric reflects how well the model is prioritising misclassifying cases as infected/covid ones as a preventive measure. This is obtained by taking safe_division(TP, (TP+FN)). The safe_division method returns 0 if there is division by zero from TP+FN, otherwise returns the regular division. Both the accuracy model and the sensitivity model will be tested.

[4]:
```
# best accuracy
!python test.py --independent 0 --validation 0 --print 1 --output_var 2␣
↪--normalclf binaryModelNormalBest --covidclf binaryModelCovidBest
```

```
Starting: Test set on Normal Piped classifier
Total=614, TP=369, FP=79, FN=11, TN=155
Testing Accuracy: 0.853
Testing Sensitivity: 0.971
Testing Specificity: 0.662
Testing PPV: 0.824
Testing NPV: 0.934
Testing F1 Score: 0.891
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 155   |   FP = 79    |
Ground P |   FN = 11   |   TP = 369    |


Starting: Test set on Covid Piped classifier
Total=448, TP=129, FP=308, FN=2, TN=9
Testing Accuracy: 0.308
Testing Sensitivity: 0.985
Testing Specificity: 0.028
Testing PPV: 0.295
Testing NPV: 0.818
Testing F1 Score: 0.454
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 9  |   FP = 308    |
Ground P |   FN = 2  |   TP = 129    |
```

[5]:

```
# best sensitivity
!python test.py --independent 0 --validation 0 --print 1 --output_var 2␣
↪--normalclf binaryModelNormalBestSensitivity --covidclf␣
↪binaryModelCovidBestSensitivity
```

```
Starting: Test set on Normal Piped classifier
Total=614, TP=377, FP=116, FN=3, TN=118
Testing Accuracy: 0.806
Testing Sensitivity: 0.992
Testing Specificity: 0.504
Testing PPV: 0.765
Testing NPV: 0.975
Testing F1 Score: 0.864
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 118  |   FP = 116   |
Ground P |   FN = 3  |   TP = 377   |


Starting: Test set on Covid Piped classifier
Total=493, TP=138, FP=355, FN=0, TN=0
Testing Accuracy: 0.280
Testing Sensitivity: 1.000
Testing Specificity: 0.000
Testing PPV: 0.280
Testing NPV: 0.000
Testing F1 Score: 0.437
Confusion Matrix
          Predicted N | Predicted P
         ---------------------------
Ground N |   TN = 0  |   FP = 355   |
Ground P |   FN = 0  |   TP = 138   |
```

We believe that this piped metric are **NOT as indicative** of the classifiers whole performance especially for the second classifier. For example, in the case of `binaryModelNormalBestSensitivity`, it will pass in too many False Positives into the second covid classifier, as these are supposed to be classified as `normal`, the second classifier will have no choice but to be penalized because it only has the option to classify it as either `covid` or `non-covid`, both of which are wrong.

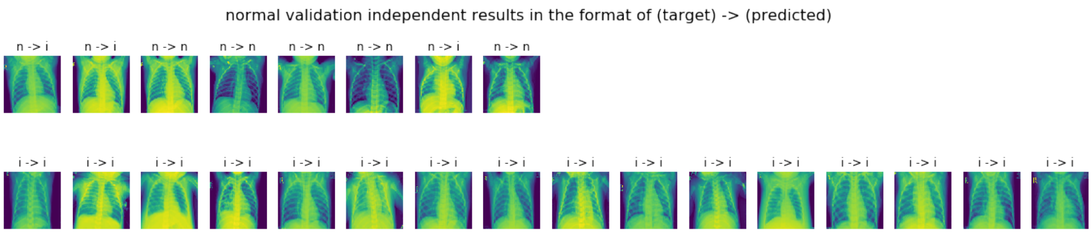
Thus, we recommend looking at the **independent** metrics to evaluate the performance.

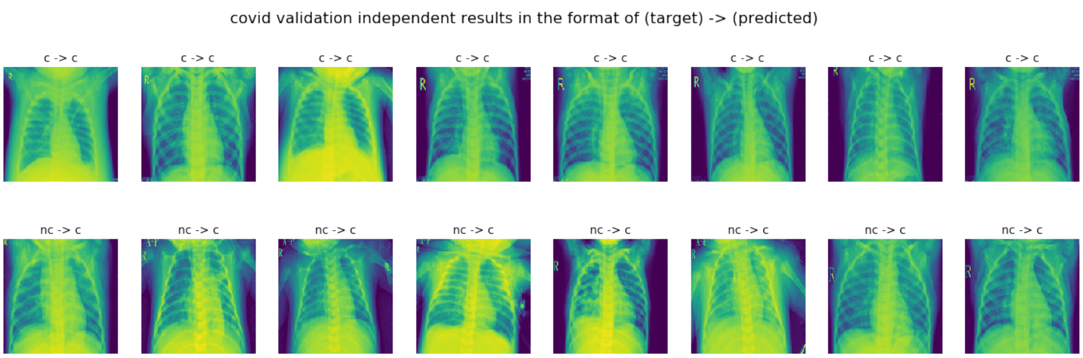
7.2.3 6. Comparison of Validation Images

6.1 On Best Accuracy Model The following shows the plots of the best accuracy model on the validation plot. The first one shows the indepenent classifiers, while the second shows the piped classifiers.

[2]:
```
# independent best accuracy
%run test.py --independent 1 --validation 1 --print 0 --output_var 2 --normalclf binaryModelNormalBest --covidclf binaryModelCovidBest
```

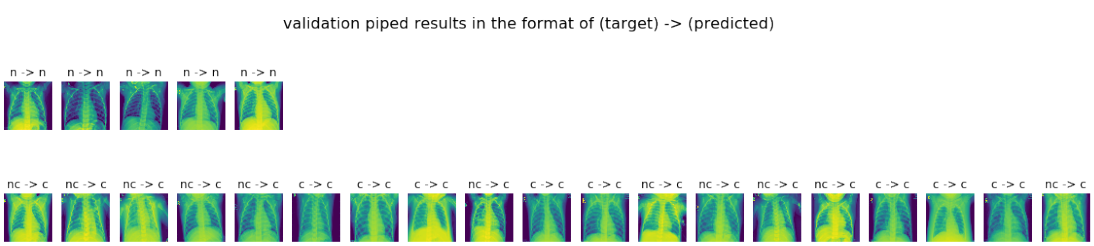
```
Starting: Validation set on Normal Independent classifier
Starting: Validation set on Covid Independent classifier
```





[3]:
```
# piped best accuracy
%run test.py --independent 0 --validation 1 --print 0 --output_var 2 --normalclf binaryModelNormalBest --covidclf binaryModelCovidBest
```

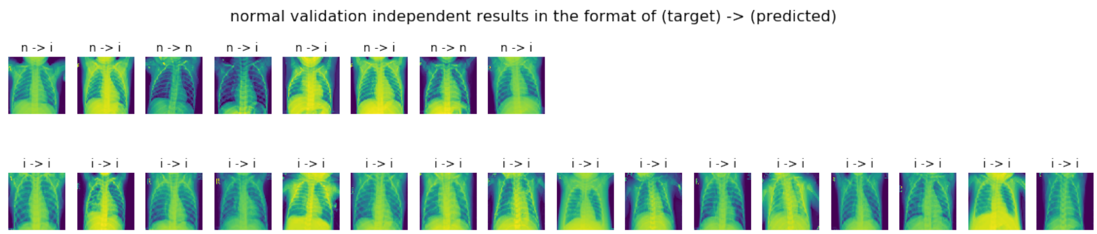
```
Starting: Validation set on Normal Piped classifier
Starting: Validation set on Covid Piped classifier
```

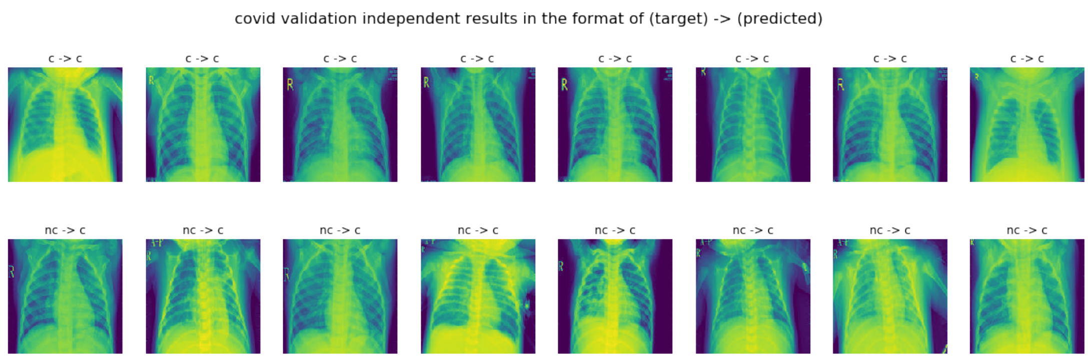


6.2 On Best Sensitivity Model The following shows the plots of the best accuracy model on the validation plot. The first one shows the indepenent classifiers, while the second shows the piped classifiers.

[4]:
```
# independent best sensitivity
%run test.py --independent 1 --validation 1 --print 0 --output_var 2 --normalclf binaryModelNormalBestSensitivity --covidclf binaryModelCovidBestSensitivity
```

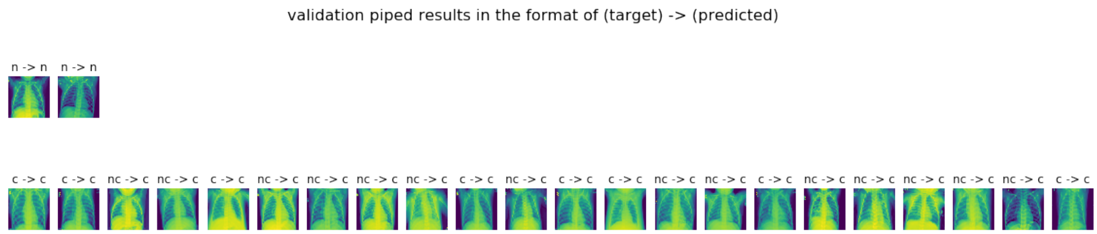
```
Starting: Validation set on Normal Independent classifier
Starting: Validation set on Covid Independent classifier
```





[6]:
```
# piped best sensitivity
%run test.py --independent 0 --validation 1 --print 0 --output_var 2 --normalclf binaryModelNormalBestSensitivity --covidclf binaryModelCovidBestSensitivity
```

```
Starting: Validation set on Normal Piped classifier
Starting: Validation set on Covid Piped classifier
```



7.3 Final Question

Q: Would it be better to have a model with high overall accuracy or low true negatives/false positives rates on certain classes. Discuss.

A: While generally the rule of thumb is to try and aim to have the model report high accuracy, in this case where real life stakes are high since COVID19 has a high infection rate, it is more preferable to do the latter to reduce the risks of outbreaks from one false positive case. This would mean that more samples may be falsely classified as infected/covid. Although more preventive, in the real world this may instead pose a challenge to the capacities of the healthcare sector. Rather than starting from high accuracy and trying to make the model more sensitive in identifying infected/covid cases, it is easier to push the model towards higher accuracy after having been tuned to have high sensitivity, which is the approach that we have taken.