Pneumonia Detection Device - **FDA Submission** - Hamza Mughal

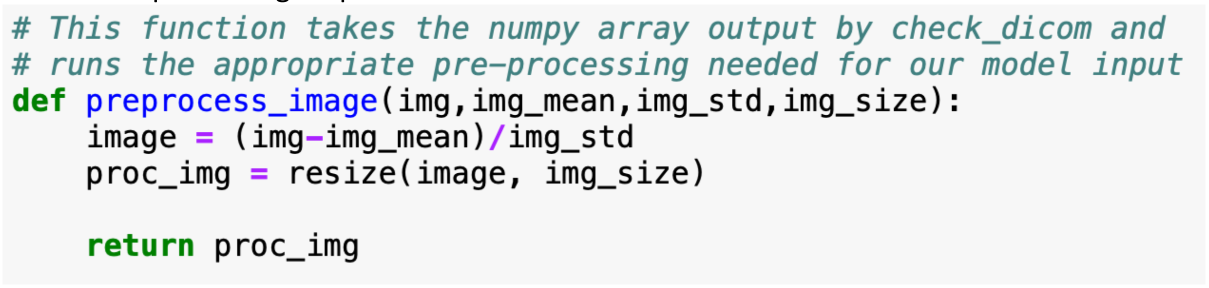
Algorithm Description

General Description

- Intended Use
  - To be used by Emergency Room physician to diagnose patient with pneumonia before x-rays are formally reported by a radiologist.
- Indications for Use
  - To be used on adults. Chest Xray from AP or PA view.
- Device Limitations
  - Not to be used on children.
- Clinical Impact of Performance
  - Speedier diagnosis of pneumonia. Reduced time pressures on radiologist to report CXRs.

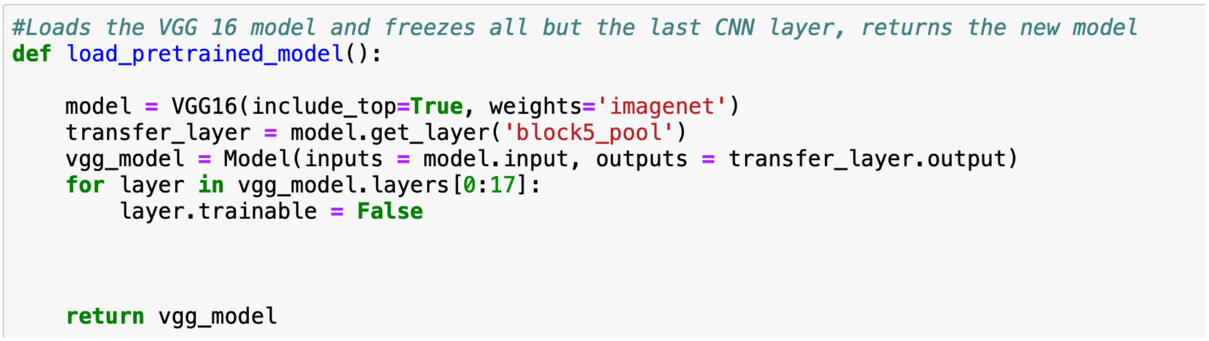
Algorithm Design and Function

- Algorithm Flowchart
  - EDA -> Data Cleaning -> Creating test and training data -> Model-building -> Training -> Performance statistics -> Predicting pneumonia using created models and weights.
- DICOM Checking Steps
  - Line 1 – 11 – File -> Inference.ipynb
- Pre-processing Steps



```
# This function takes the numpy array output by check_dicom and
# runs the appropriate pre-processing needed for our model input
def preprocess_image(img,img_mean,img_std,img_size):
    image = (img-img_mean)/img_std
    proc_img = resize(image, img_size)

    return proc_img
```

- CNN Architecture



```
#Loads the VGG 16 model and freezes all but the last CNN layer, returns the new model
def load_pretrained_model():

    model = VGG16(include_top=True, weights='imagenet')
    transfer_layer = model.get_layer('block5_pool')
    vgg_model = Model(inputs = model.input, outputs = transfer_layer.output)
    for layer in vgg_model.layers[0:17]:
        layer.trainable = False



    return vgg_model
```

```
def build_my_model(vgg_model):

    my_model = Sequential()
    my_model.add(vgg_model)
    my_model.add(Flatten())
    my_model.add(Dropout(0.5))
    my_model.add(Dense(1024, activation = 'relu'))
    my_model.add(Dropout(0.5))
    my_model.add(Dense(512, activation = 'relu'))
    my_model.add(Dropout(0.5))
    my_model.add(Dense(256, activation = 'relu'))
    my_model.add(Dense(1, activation = 'sigmoid'))



    optimizer = Adam(lr = .0001)
    loss = 'binary_crossentropy'
    metrics = ['binary_accuracy']

    my_model.compile(optimizer = optimizer, loss = loss, metrics = metrics)




    return my_model
```

<u>Algorithm Training</u>

- Parameters:
    - Types of augmentation used during training

```
def my_image_augmentation():

    my_idg = ImageDataGenerator(rescale = 1. / 255.0,
                                horizontal_flip = True,
                                vertical_flip = False,
                                height_shift_range = 0.1,
                                width_shift_range = 0.1,
                                rotation_range = 20,
                                shear_range = 0.1,
                                zoom_range = 0.1)


    return my_idg
```

    - Batch size
    - Optimizer learning rate

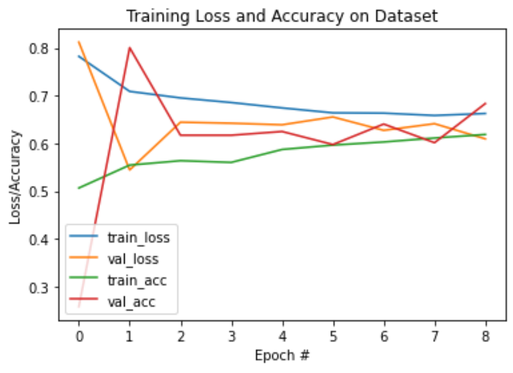
```
def make_train_gen(vargs):

    ## Create the actual generators using the output of my_image_augmentation for
    ## your training data
    ## This generator uses a batch size of 32
    idg = my_image_augmentation()
    train_gen = idg.flow_from_dataframe(dataframe=vargs,
                                        directory=None,
                                        x_col = 'path',
                                        y_col = 'Pneumonia Class',
                                        class_mode = 'binary',
                                        target_size = (224,224),
                                        batch_size = 32)
```

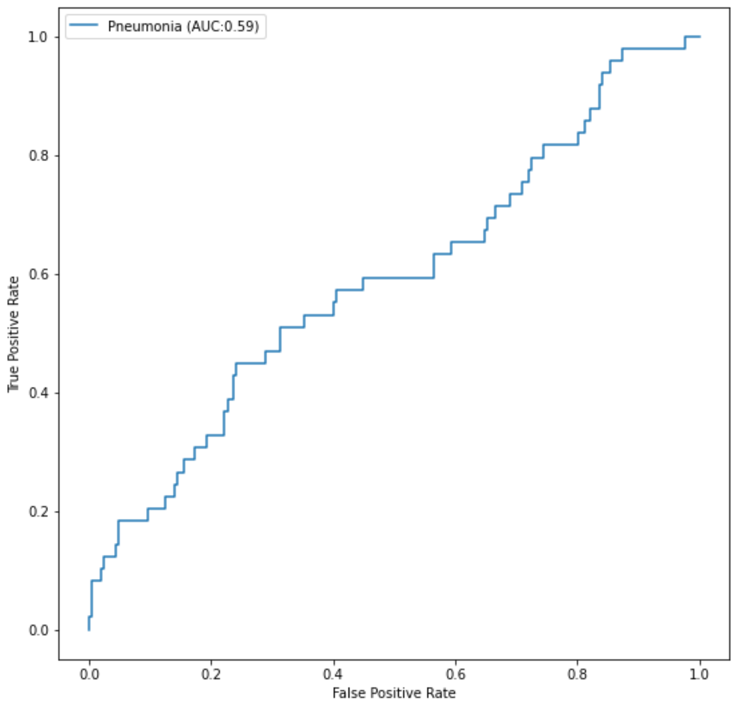
    - Layers of pre-existing architecture that were frozen
    - Layers of pre-existing architecture that were fine-tuned
    - Layers added to pre-existing architecture

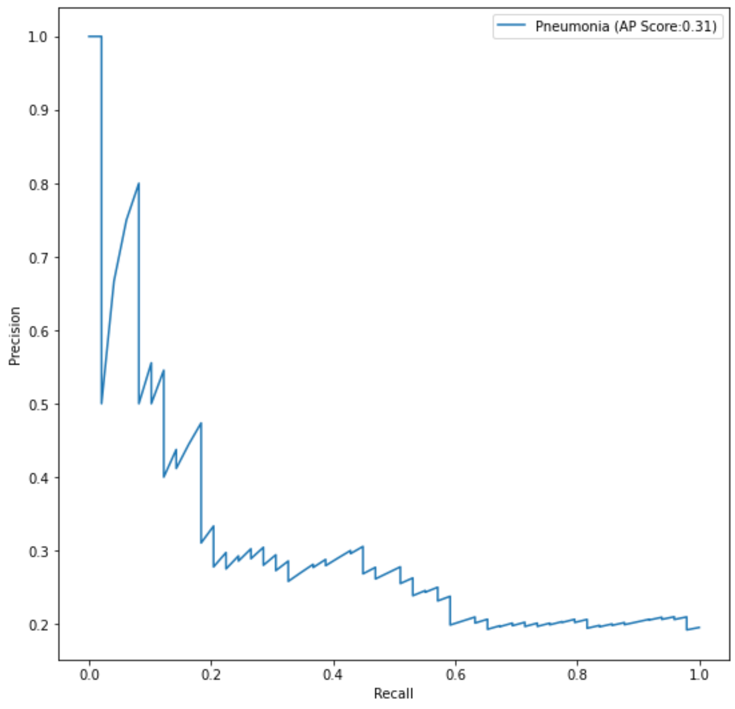
```
#Loads the VGG 16 model and freezes all but the last CNN layer, returns the new model
def load_pretrained_model():

    model = VGG16(include_top=True, weights='imagenet')
    transfer_layer = model.get_layer('block5_pool')
    vgg_model = Model(inputs = model.input, outputs = transfer_layer.output)
    for layer in vgg_model.layers[0:17]:
        layer.trainable = False



    return vgg_model
```

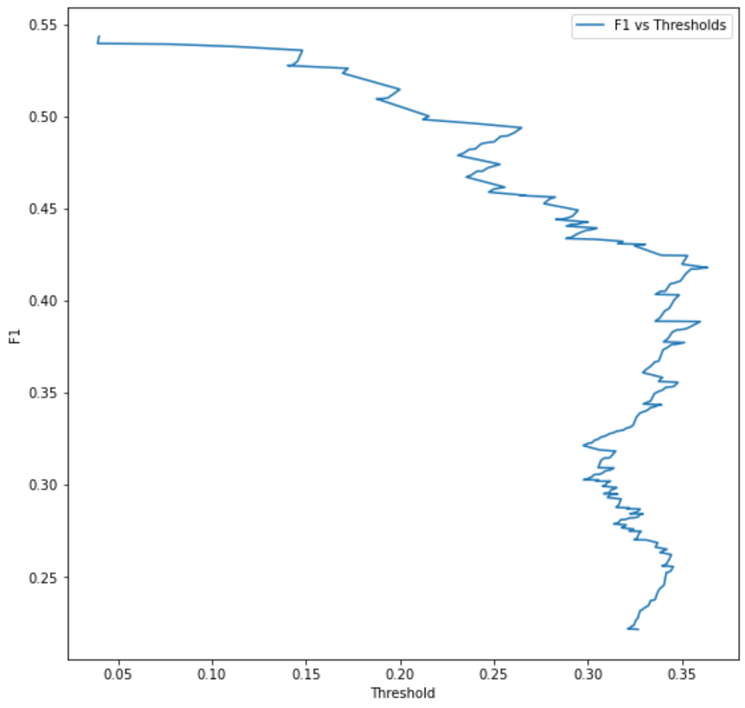
```
#This model has 4 Dense layers and uses a dropout of 0.5 and is added on top of the
#VGG 16 model with all but its last CNN layer frozen


def build_my_model(vgg_model):

    my_model = Sequential()
    my_model.add(vgg_model)
    my_model.add(Flatten())
    my_model.add(Dropout(0.5))
    my_model.add(Dense(1024, activation = 'relu'))
    my_model.add(Dropout(0.5))
    my_model.add(Dense(512, activation = 'relu'))
    my_model.add(Dropout(0.5))
    my_model.add(Dense(256, activation = 'relu'))
    my_model.add(Dense(1, activation = 'sigmoid'))



    optimizer = Adam(lr = .0001)
    loss = 'binary_crossentropy'
    metrics = ['binary_accuracy']

    my_model.compile(optimizer = optimizer, loss = loss, metrics = metrics)




    return my_model
```

- Insert algorithm training performance visualization




Pneumonia (AUC:0.59)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Pneumonia (AP Score:0.31)
Precision
Recall
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0

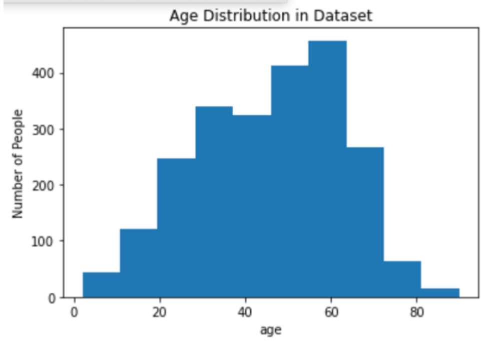


- Final Threshold and Explanation

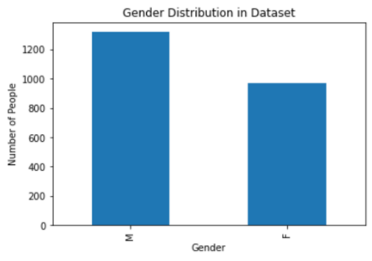
```
# Threshold of 0.35 will result in F1 score of 0.32
threshold_select = 0.35
index = (np.abs(threshold_2 - threshold_select)).argmin()
precision = prec[index]
rec = recall[index]
```

Databases (For the below, include visualizations as they are useful and relevant)

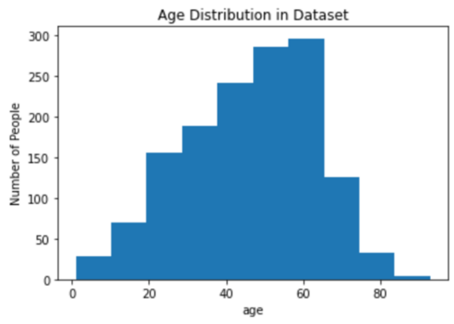
- Description of Training Dataset



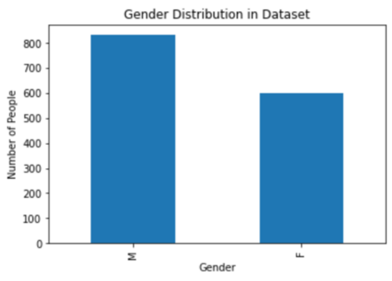
```
M    1320
F     970
Name: Patient Gender, dtype: int64
```



- Description of Validation Dataset



```
M    832
F    598
Name: Patient Gender, dtype: int64
```



Ground Truth

- Labels from the NIH data sample – using label defined by radiologist.

FDA Validation Plan

- Patient Population Description for FDA Validation Dataset
  - Adults 18 and over who presents with symptoms of Pneumonia
- Ground Truth Acquisition Methodology
  - Labels captured using NLP from experienced radiologist.