FDA Submission

Mark Abrenio

DIGITAL Pneumonia Detector

Algorithm Description

1. General Information

Intended Use Statement:

This software can be utilized to prescan XRAY images, to help give a clinical physician some early indication that the patient might be suffering from Pneumonia.

Indications for Use:

This software is not to be utilized as a way of definitively determining an XRAY image does or does not contain Pneumonia. It is simply to be utilized as a diagnostic assistant to the clinician.

Device Limitations:

The precision and recall of the model is not high enough. The accuracy of our model on our simple training data was 0.814, and our loss was 0.41. On our validation data we received the following scores:

```
Precision / recall / f1-score from VALIDATION data
---------------
              precision    recall  f1-score   support

         0.0       0.77      0.78      0.77      3000
```

1.0	0.77	0.77	0.77	3000

We need to increase our precision and recall to at least 90 percent, especially on the Pneumonia Negative labels. It is critical that we reduce our false negatives as much as possible. Having false negatives could give a clinician a false sense of security that the patient does not actually have Pneumonia. Whereas, a false positive would at least prod the clinician to carefully analyze the XRAY image to manually exclude a Pneumonia diagnoses.

This is not possible with this data set because it is not very clean and there are not enough high quality Pneumonia XRAYs and clear XRAYs.

Clinical Impact of Performance:

This algorithm was never designed to replace a physician. Simply to give them a hint of what to look for with a specific patient. Just as a high bp reading does not immediately mean a clinician diagnoses hypertension, a positive Pneumonia notification from this algorithm would not mean a clinician would automatically diagnose pneumonia. This is simply a diagnostic aid. However, an accuracy rate of over 90 percent would greatly assist a clinician in accurately diagnosing Pneumonia.

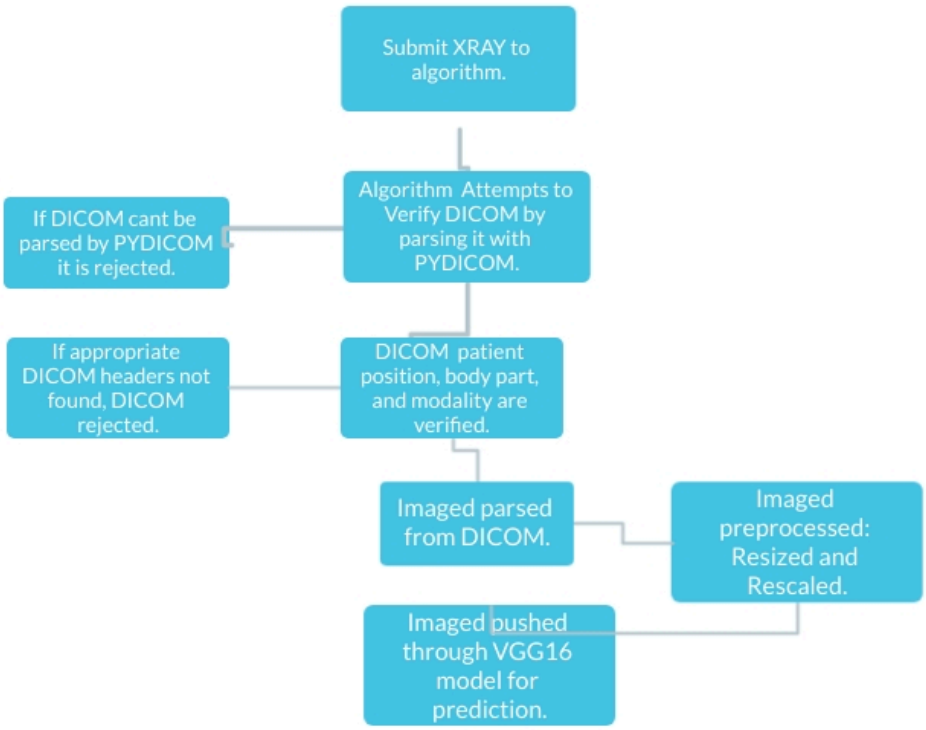
2. Algorithm Design and Function

<< Insert Algorithm Flowchart >>

DICOM Checking Steps:

To verify our DICOMS we utilize a function named check_dicom(). This function carries out all the steps of verifying the DICOM including reading and parsing it using the Python pydicom library. When the app calls this function, the whole thing is wrapped in an exception handler so if anything goes wrong, the DICOM is rejected. For example, if

an exception is triggered when trying to read the DICOM using the pydicom library, the DICOM will not be processed. The function also checks three header values including the patient position (Must be AP or PA), the body part examined (Must be chest), and modality (Must be an XRAY image).



Preprocessing Steps:

We use the TensorFlow ImageDataGenerator class to create an easy to use iterable data loader for our XRAY images. Some effective augmentations we use are samplewise_std_normalization = True, rotation_range = 15, zoom_range = 0.25,brightness_range=[0.35, 1.1], and samplewise_center = True.

CNN Architecture:

This model uses the VGG 16 architecture.

3. Algorithm Training

Parameters:
* Types of augmentation used during training
* Batch size
* Optimizer learning rate
* Layers of pre-existing architecture that were frozen
* Layers of pre-existing architecture that were fine-tuned
* Layers added to pre-existing architecture

We primarily use the function below to load the pertained VGG16 model for training. The only thing we really do to the model is change the input layer to accept 224 x 224 images, and then set the entire model except for the output layer to non trainable so we don’t have to train the entire model again. It takes too long to train the entire model and we are limited in GPU speed.

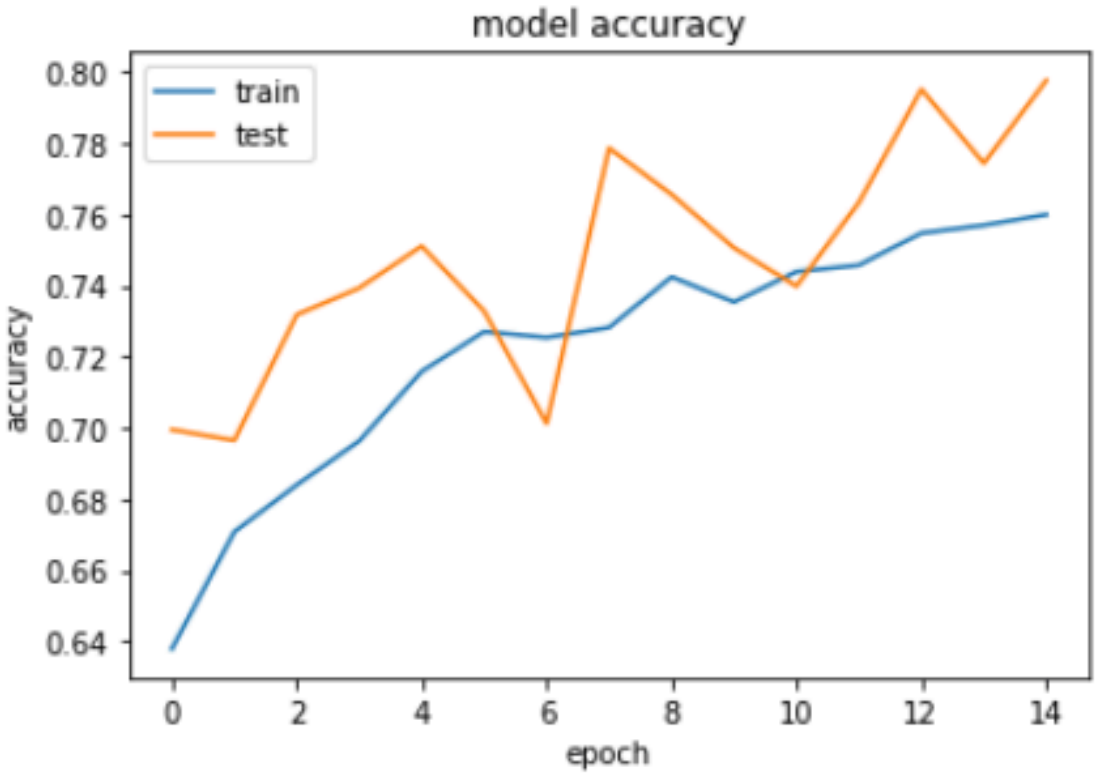
```
def load_pretrained_model(number_output, load_old_model=0, file_name_json=0, file_name_

    # model = VGG16(include_top=True, weights='imagenet')
    # transfer_layer = model.get_layer(lay_of_interest)
    # vgg_model = Model(inputs = model.input, outputs = transfer_layer.output)

    # Todo
    from keras.optimizers import Adam
    number_output = 1 #**need to whittle it down to only look for pneumonia!

    print(device_lib.list_local_devices()) #show if we have GPU ready or not!

    if load_old_model:
        try:
            print("Loading Saved Model")
            model = load_model1(file_name_json, file_name_weights)
            print("SUCCESS: Returning saved model!")
            from datetime import datetime
            from keras.callbacks import ModelCheckpoint, LearningRateScheduler
            from keras.callbacks import ReduceLROnPlateau
            lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                                           cooldown=0,
                                           patience=5,
                                           min_lr=0.5e-6)

            return model, lr_reducer
        except Exception as e:
            print("ERROR: Could Not Load Saved Model. TERMINATING")
            print(e)
            sys.exit(1)

    IMAGE_SIZE = [224, 224] #**use a little bigger images, to get more xray data!
    vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

    # don't train existing weights GPU TOO SLOW
    for layer in vgg.layers:
        layer.trainable = False

    x = Flatten()(vgg.output)
    prediction = Dense(number_output, activation='sigmoid')(x)
    model = Model(inputs=vgg.input, outputs=prediction)

    model.compile(optimizer = Adam(lr=0.0007), loss='binary_crossentropy', metrics=['bi
    model.summary()

    from datetime import datetime
    from keras.callbacks import ModelCheckpoint, LearningRateScheduler
    from keras.callbacks import ReduceLROnPlateau
    lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                                   cooldown=0,
                                   patience=5,
                                   min_lr=0.5e-6)
```

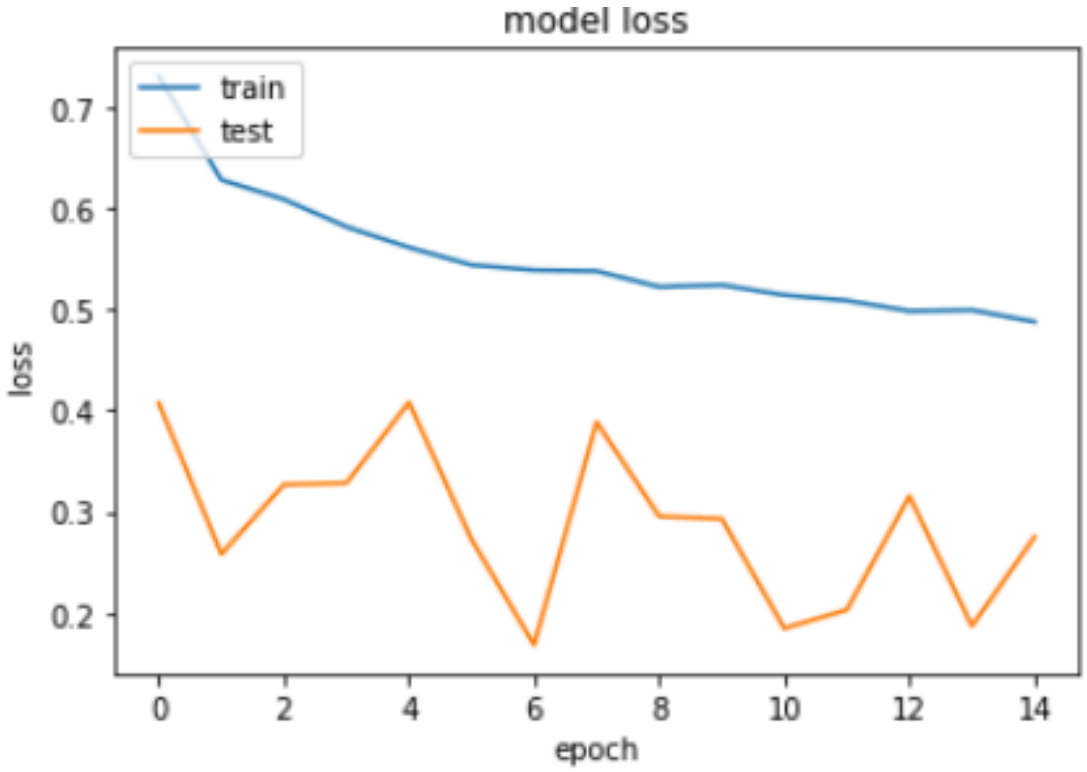
The code below displays the augmentations we used for the XRAY images. The varying brightness range and slight in and out zoom seem to help the model learn the XRAYs a lot better. With these simple augmentation the VGG16 model learns the XRAYs pretty well. But it could learn them better if the data was a little cleaner. Some of the

Pneumonia XRAYs don't look very clear, and I think it confuses the model some.

```
def make_train_gen(class_column_list, patched_df): #**lets provide the dataframe here

    class_column_list = ["Infiltration", "Pneumonia"] #*just narrow it down to notice
                                                      #*Pneumonia images!

    batch_size = 64

    datagen = ImageDataGenerator(samplewise_std_normalization = True,
                                 rotation_range = 15,
                                 zoom_range = 0.25,
                                 brightness_range=[0.35, 1.1],
                                 samplewise_center = True
                                )

    train_generator=datagen.flow_from_dataframe(
    dataframe=patched_df,
    directory="None",
    x_col="path",
    y_col="Pneumonia_or_Infiltration",
    subset="training",
    batch_size=batch_size,
    seed=42,
    shuffle=True,
    class_mode="binary",
    target_size=(224,224)) #**use bigger images to get more xray data! :)

    print("Train Loader Ready")

    print("Number of Batches of Train_Generator: " + str(len(train_generator)))

    return train_generator#, valid_generator
```

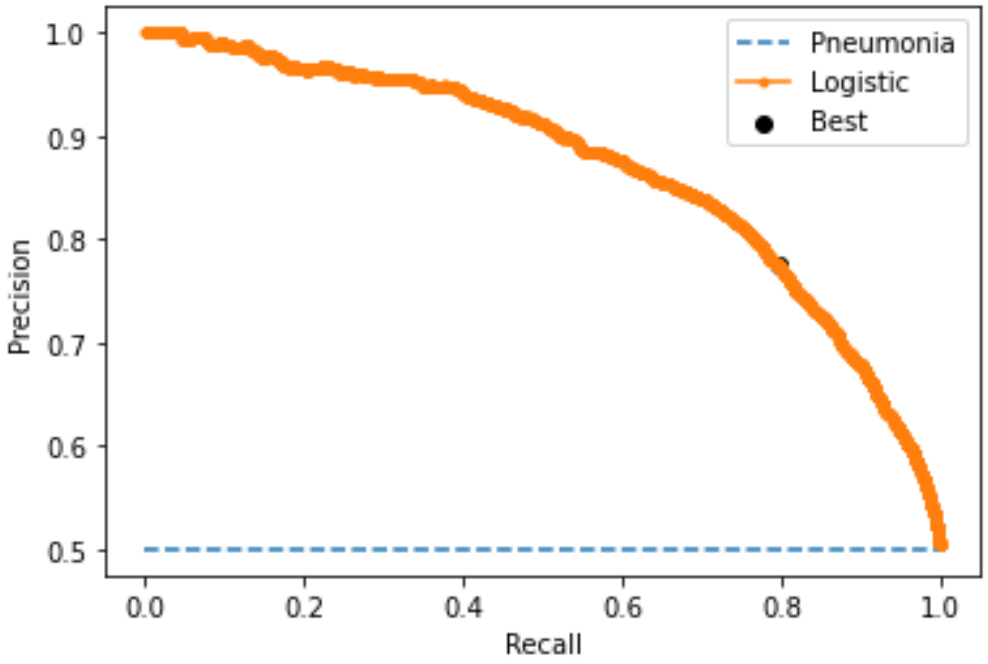
<< Insert algorithm training performance visualization >>


model accuracy
train
test
accuracy
epoch
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0.66
0.64
0
2
4
6
8
10
12
14



<< Insert P-R curve >>

```
Best Threshold=0.392611, F-Score=0.786
0.39261103
```



We can see from the illustration below that our model for the XRAYs is not multi class, but rather based on binary cross entropy — either Pneumonia or not Pneumonia. Also we use an ADAM optimizer with a learning rate of 0.0007.

```
    model.compile(optimizer = Adam(lr=0.0007), loss='binary_crossentropy', metrics=['binary_accuracy'])
    model.summary()

    from datetime import datetime
    from keras.callbacks import ModelCheckpoint, LearningRateScheduler
    from keras.callbacks import ReduceLROnPlateau
    lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                                   cooldown=0,
                                   patience=5,
                                   min_lr=0.5e-6)

    print("SUCCESS: Returning built model!")

    return model, lr_reducer
```

Final Threshold and Explanation:

I utilized information from this source to find the optimal threshold: https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/
Specific information from this site is illustrated below:

There are many ways we could locate the threshold with the optimal balance between false

positive and true positive rates.

Firstly, the true positive rate is called the Sensitivity. The inverse of the false-positive rate is called the Specificity.

- Sensitivity = TruePositive / (TruePositive + FalseNegative)
- Specificity = TrueNegative / (FalsePositive + TrueNegative)

Where:

- Sensitivity = True Positive Rate
- Specificity = 1 – False Positive Rate

The Geometric Mean or G-Mean is a metric for imbalanced classification that, if optimized, will seek a balance between the sensitivity and the specificity.

- G-Mean = sqrt(Sensitivity * Specificity)

One approach would be to test the model with each threshold returned from the call roc_auc_score() and select the threshold with the largest G-Mean value.

Given that we have already calculated the Sensitivity (TPR) and the complement to the Specificity when we calculated the ROC Curve, we can calculate the G-Mean for each threshold directly.

```
...
# calculate the g-mean for each threshold
gmeans = sqrt(tpr * (1-fpr))
```

Once calculated, we can locate the index for the largest G-mean score and use that index to determine which threshold value to use.

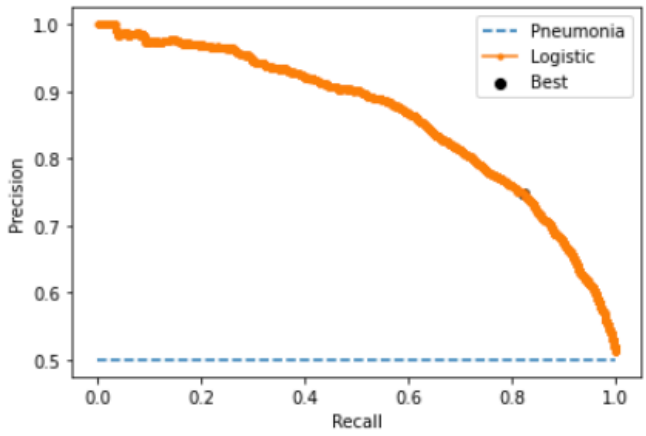
```
...
# locate the index of the largest g-mean
ix = argmax(gmeans)
print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix], gmeans[ix]))
```

We can also re-draw the ROC Curve and highlight this point.

This blog resulted in me utilizing the following code in the project, to find a good threshold to use for the model. The optimal threshold decided on was 0.435366.

```
precision, recall, thresholds = metrics.precision_recall_curve(valY_val, pred_Y_val)
# convert to f score
fscore = (2 * precision * recall) / (precision + recall)
# locate the index of the largest f score
ix = argmax(fscore)
print('Best Threshold=%f, F-Score=%.3f' % (thresholds[ix], fscore[ix]))
# plot the roc curve for the model
no_skill = len(valY_val[valY_val==1]) / len(valY_val)
pyplot.plot([0,1], [no_skill,no_skill], linestyle='--', label='Pneumonia')
pyplot.plot(recall, precision, marker='.', label='Logistic')
pyplot.scatter(recall[ix], precision[ix], marker='o', color='black', label='Best')
# axis labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()
# show the plot
pyplot.show()

best_thresh = thresholds[ix]

print("Best Threshold. Threshold we shall use: " + str(best_thresh))
```

```
Best Threshold=0.435366, F-Score=0.784
```




```
Best Threshold. Threshold we shall use: 0.43536562
```

4. Databases

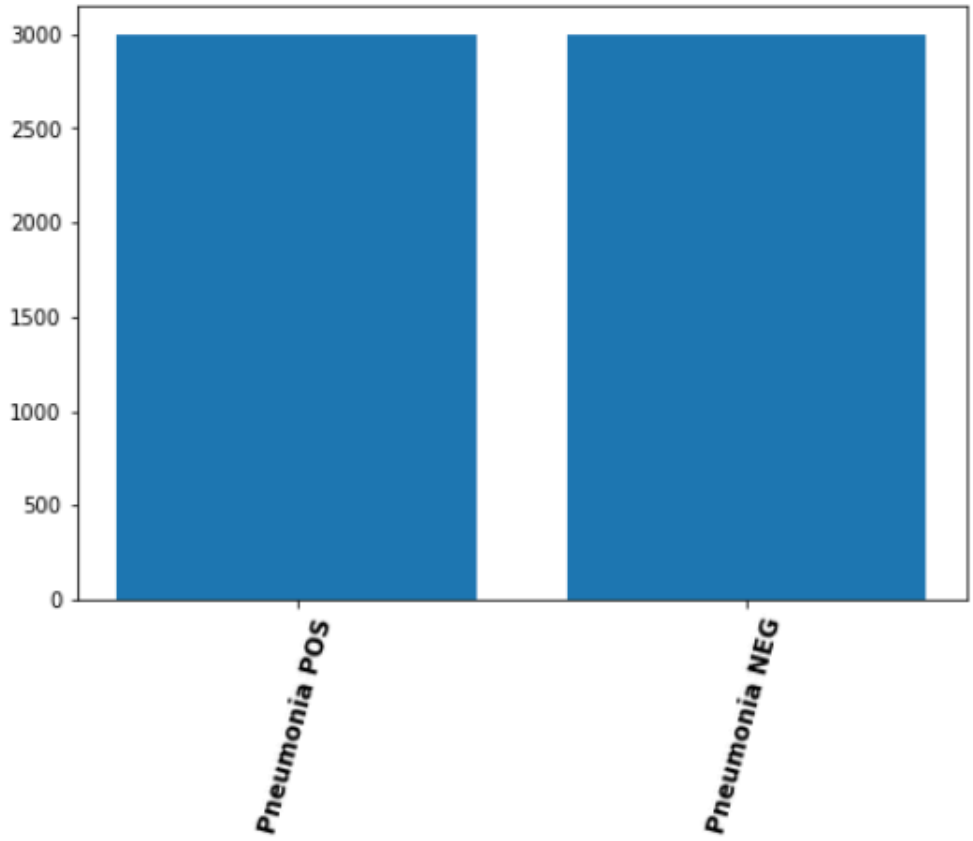
(For the below, include visualizations as they are useful and relevant)

Description of Training Dataset:

For the dataset I grouped infiltration and pneumonia together, since the XRAY images look similar in nature. Both tend to show a highlighted blob within the lung, and there were not very many Pneumonia XRAY images to begin with. The training set is

balanced and plotted below. There are more images available to use since we group Pneumonia and Infiltration together, but the GPU we have for this project is too slow. So I am only using 6 thousand total images for training.

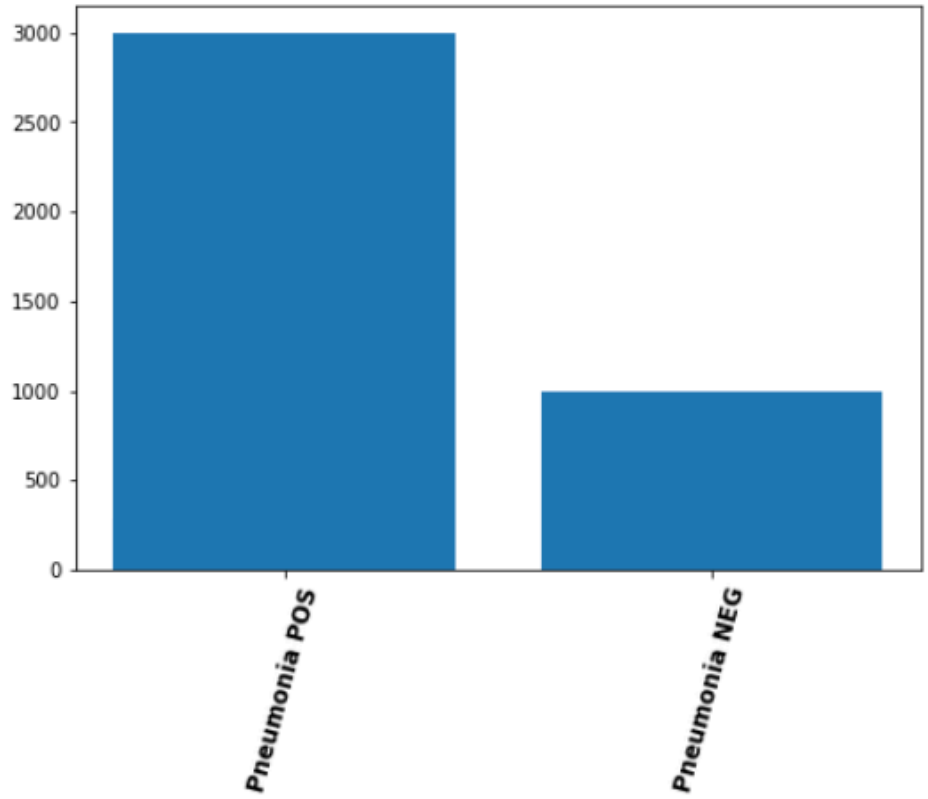
```
PLOT TRAINING DATA. POSITIVE AND NEGATIVE PNEUMONIA IMAGES (BALANCED)
Positive Images: 3000
Negative Images: 3000
```



Description of Validation Dataset:

I used the same methodology for the validation set. I counted all images that were either Pneumonia or Infiltration positive, as Pneumonia positive XRAY images. Per the Udacity Reviewers recommendation I am validating with a data set that contains 3000 Pneumonia positive images, and 1000 Pneumonia negative images.

```
PLOT VALIDATION DATA. POSITIVE AND NEGATIVE PNEUMONIA IMAGES (BALANCED)
Positive Images: 3000
Negative Images: 1000
```



5. Ground Truth

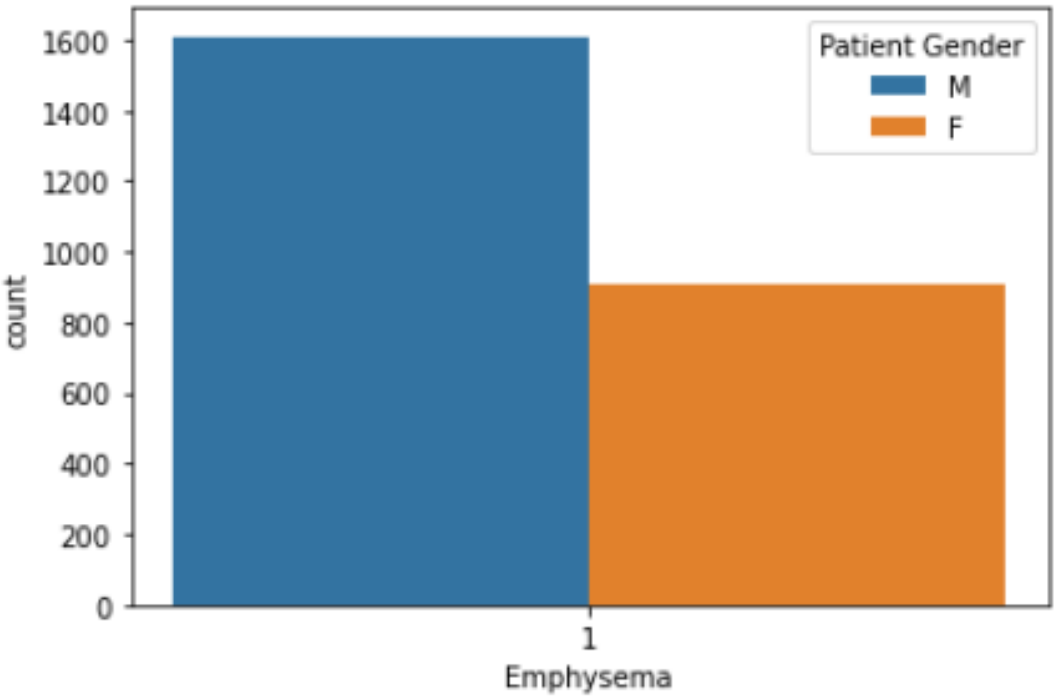
The ground truth I used was simply based on the truth labels provided in the NIH dataset. Although , I did notice that some of the images appeared to be false positives and some of the images were very low quality. If the dataset was cleaned up I think the model would be much more effective at generalizing on Pneumonia XRAY images.

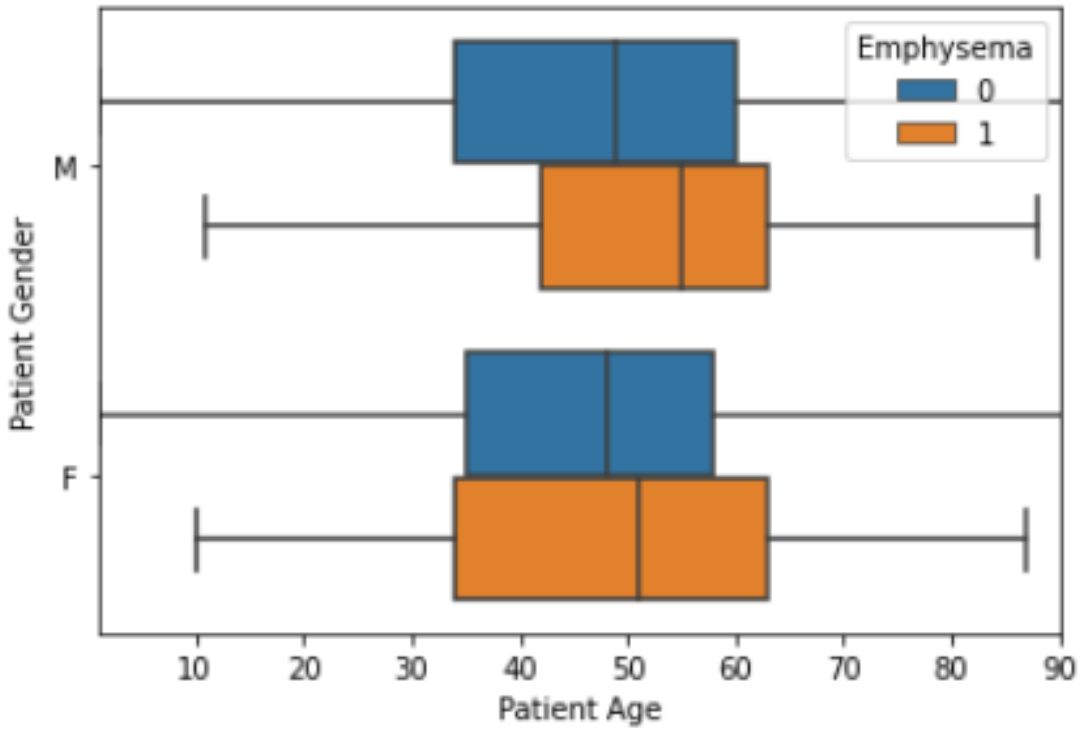
6. FDA Validation Plan

Patient Population Description for FDA Validation Dataset:

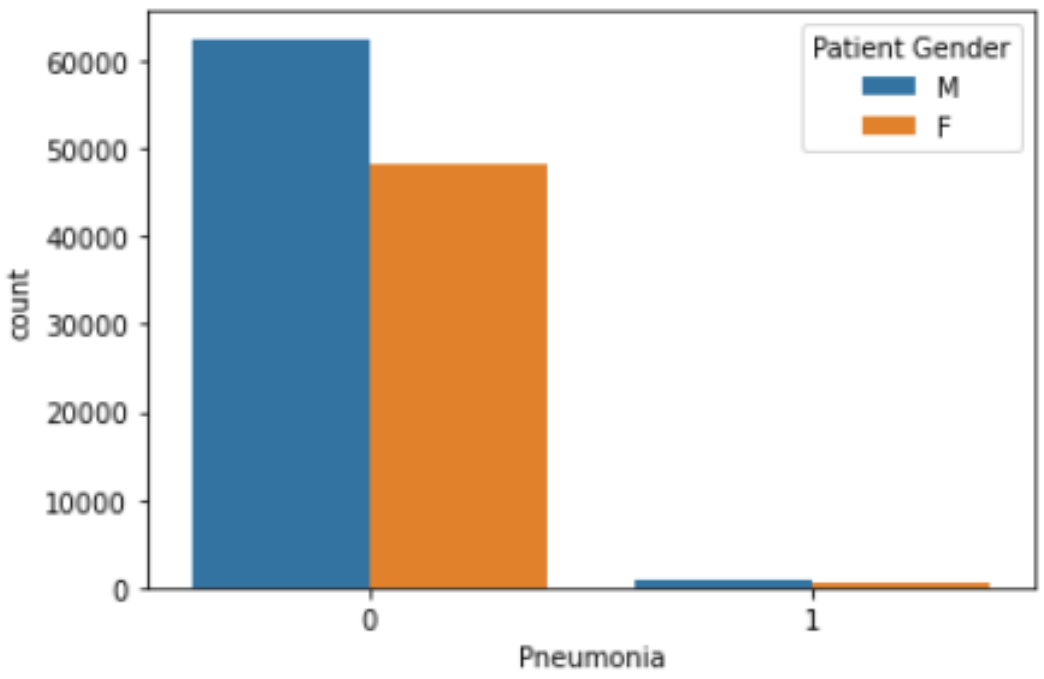
Ideally, this model needs to be trained with thousands of known pneumonia images, and thousands of known clear images. The NIH dataset is not really clean. It has some very low quality XRAYs and some apparent false positives. Ideally, this algorithm would be continually trained alongside an active clinicians ongoing manually diagnoses of many patients, so it gets better the longer it works beside an actual physician.

We also need more data because, some of our data does not seem to reflect real life. For example, if you look at the gender and age plots for emphysema, the ages tend to trend a little lower. In real life most people who get emphysema are a bit older.





We can also see another problem in our dataset is simply, most of the images are not marked Pneumonia.



Fix

Ground Truth Acquisition Methodology:

Ideally, this algorithm would be continually trained alongside an active clinicians ongoing manually diagnoses of many patients, so it gets better the longer it works beside an actual physician.

Algorithm Performance Standard:

This algorithm was never designed to replace a physician. Simply to give them a hint of what to look for with a specific patient. Just as a high bp reading does not immediately mean a clinician diagnoses hypertension, a positive Pneumonia notification from this algorithm would not mean a clinician would automatically diagnose pneumonia. This is simply a diagnostic aid. However, an accuracy rate of over 90 percent would greatly assist a clinician in accurately diagnosing Pneumonia.

Reviewer Concerns

The previous reviewer mentioned that I needed to address an ideal F1 score for the model, ground truth acquisition method, and describe an ideal dataset. To complete this I read the research paper entitled CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. To make sure I correctly understood the F1 scoring algorithm I did a few calculations. The documented formula for F1 is pretty simple. It is:

F1 = True Positive / True Positive + .5(False Positive + True Positive)

So lets assume we have 100 images. If all of them are predicted correctly by the model F1 score is 1. If 50 were predicted correct and 50 incorrectly predicted, assuming 50 true positives and 50 false negatives, the F1 score would be 0.666667. If 80 were predicted correctly, assuming 80 true positives and 20 false negatives, the F1 score would be 0,888889.

Given my new understanding of F1 score it was surprising to me to see that CheXNet only had an F1 score of predicting Pneumonia at 0.435. And an average F1 score of highly trained radiologist trying to perform manual predictions of the same XRAY images had an F1 score of 0.387. This indicates to me that a lot of Pneumonia images were misdiagnosed! It does make sense though because, its just very difficult to diagnose a Pneumonia just from an image, given no prior clinical information from the patient.

For example, maybe the XRAY image looks pretty clear — however the patient is complaining of shortness of breath and has a fever, and the lung sounds very congested during breathing. The patient presenting in this way would certainly cause the clinician to be suspicious of a Pneumonia, even if the imaging said otherwise.

This is why I do not consider this code to be a product that would be utilized on its own, without the guidance of a trained physician, in a clinical environment. I would assume

that the model would need to be trained to where it had at least an F1 score of 0.387, and then it could be used as a diagnostic assistant to the clinician. In addition to the F1 score of 0.387, we also want the false negatives rate to be as low as possible. The threshold should be tuned to minimize false negatives.

I believe the best use of this software would be to have it working side by side with a clinician. It could be utilized to pre screen images. And if it says it thinks an image has pneumonia in it, that would trigger the clinician to take a much closer look at the image if it initially looks clear to them. Negative predictions from this algorithm, should trigger no change in steps for the clinician. But, a positive detection from this system, along with a negative prediction from the clinician — should trigger an additional review of the XRAY to make sure pneumonia is not missed — as this could be fatal to a patient in a clinical setting.

XRAY images should be acquired in a clinical setting along side actual physicians. And if ultimately the clinician has a different prediction of the image than this system, the new image should be fed back into the system with the new label, and the system should be retrained with this new knowledge. In this sense, the system will begin to, over time, be trained on the “ideal” dataset — which would be XRAYs from the hospital or clinic’s typical demographic. Of course, most clinics tend to treat a similar demographic and age range depending on their location. For example, is the clinic in a rural setting with older people? Is the clinic located inside a University campus, and most of its patients under 30?

Overtime, as this knowledge on specific demographics and age ranges is acquired by individual clinic systems, it could continually be shared across health systems. Thereby granting remote health systems heavy access to systems trained with other demographic data.