I. Introduction

a. *Problem Statement*

How to reduce 70% of radiologists/clinicians' workload by building a deep learning model to differentiate between normal and abnormal frontal chest radiographs with a higher accuracy score (at least 80% accurate) than previous models on Kaggle by the end of 2021?

b. *Context*

Chest X-ray exams are one of the most frequent and cost-effective medical imaging examinations available. As one of the most ubiquitous diagnostic imaging tests in medical practice, chest radiography requires timely reporting of potential findings and diagnosis of diseases in the images. Automated, fast, and reliable detection of diseases based on chest radiography is a critical step in radiology workflow. Therefore, we will be developing a transfer learning model differentiating between normal and abnormal frontal chest radiographs, in order to help alert radiologists and clinicians of potential abnormal findings as a means of work list triaging and reporting prioritization.

c. *Criteria for success*

Successfully building a model that achieves at least 80% of accuracy score (using AUC metric).

d. *Scope of solution space*

Patients in the U.S.

e. *Constraints*

- Large dataset, which takes a long time to train the model.
- The labels are expected to be > 90% accurate as the authors used Natural Language Processing to text-mine disease classifications from the associated radiological reports. In other words, some labels might not interpret the image well.

f. *Stakeholders*

Radiologists/clinicians.

g. *Data source(s) and References*

https://www.kaggle.com/nickuzmenkov/nih-chest-xrays-tfrecords

II. Datasets

[NIH Chest X-rays TFRecords] - This is a non-official TFRecord version of NIH Chest X-rays dataset. The initial DataFrame has been preprocessed to format more suitable for CNN training purposes (see preprocessed_data.csv file). Only images (downscaled to 600x600 and encoded as 1-channel jpegs) and corresponding diagnosis (15 categories) were left with all additional patient information excluded (e.g. age, sex, etc.). All 112,120 samples were kept (no filtration, grouping, or removing were performed).

This dataset includes:

- ***Preprocessed DataFrame:*** preprocessed_data.csv
- ***TFRecords:*** 256 serialized .tfrec files under the /data directory with name format f'{number}-{number_or_samples}'

```
# count of TFRecords
tfr_list = os.listdir('data/')
print('TFRecord file count: {}'.format(len(tfr_list)))
```

```
TFRecord file count: 256
```

```
# read in csv
df = pd.read_csv('preprocessed_data.csv').drop(columns='Unnamed: 0')
print(df.shape)
df.head()
```

```
(112120, 15)
```

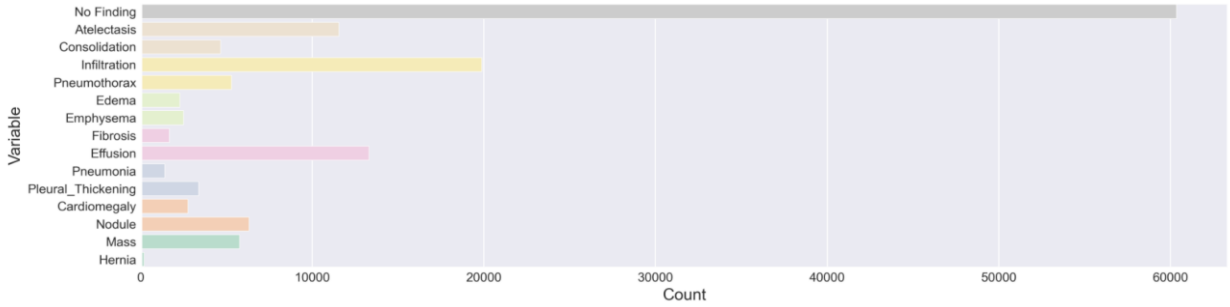
	No Finding	Atelectasis	Consolidation	Infiltration	Pneumothorax	Edema	Emphysema	Fibrosis	Effusion	Pneumonia	Pleural_Thickening	Cardiomegaly	I
0	False	False	False	False	False	False	False	False	False	False	False	True	
1	False	False	False	False	False	False	True	False	False	False	False	True	
2	False	False	False	False	False	False	False	False	True	False	False	True	
3	True	False	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	False	False	

III. Data Cleaning/Wrangling

Since this dataset primarily contains the images, and the dataframe has been preprocessed to format more suitable for CNN/transfer learning purpose, there is nothing much we can do to wrangle the data. In other words, the dataset is ready for us to move on to the Exploratory Data Analysis and the modeling phases.

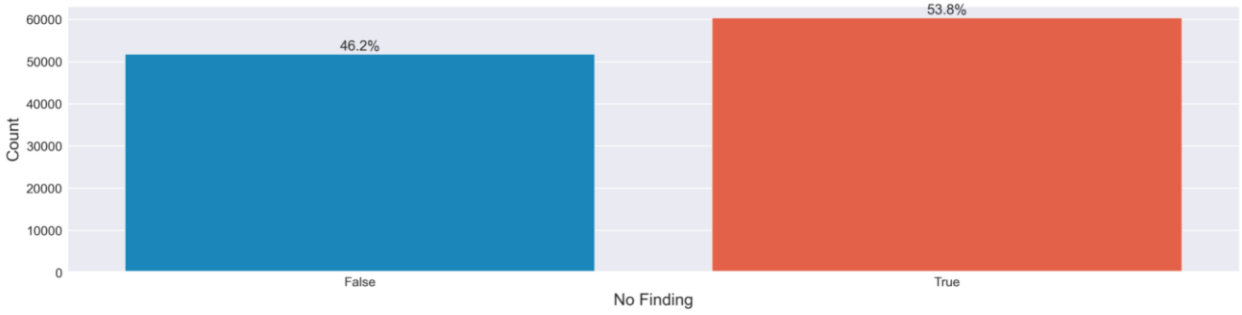
IV. Exploratory Data Analysis

a) Preprocessed DataFrame Features:



- *No Finding: 60361*
- *Atelectasis: 11559*
- *Consolidation: 4667*
- *Infiltration: 19894*
- *Pneumothorax: 5302*
- *Edema: 2303*
- *Emphysema: 2516*
- *Fibrosis: 1686*
- *Effusion: 13317*
- *Pneumonia: 1431*
- *Pleural_Thickening: 3385*
- *Cardiomegaly: 2776*
- *Nodule: 6331*
- *Mass: 5782*
- *Hernia: 227*

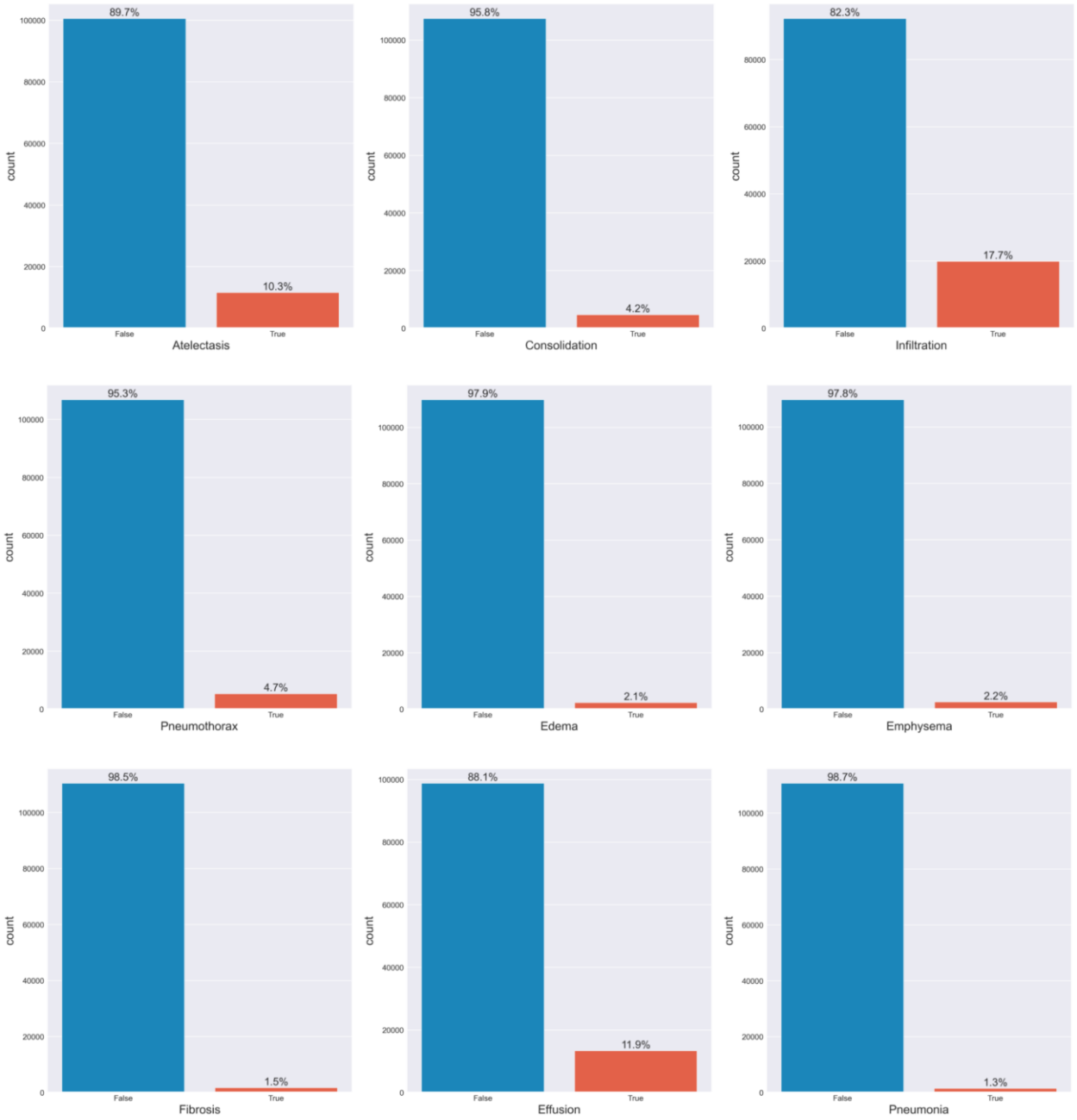
We can see that this is an imbalanced dataset, in which we have 60,361 observations with no finding and 51,759 observations with findings (other diseases).

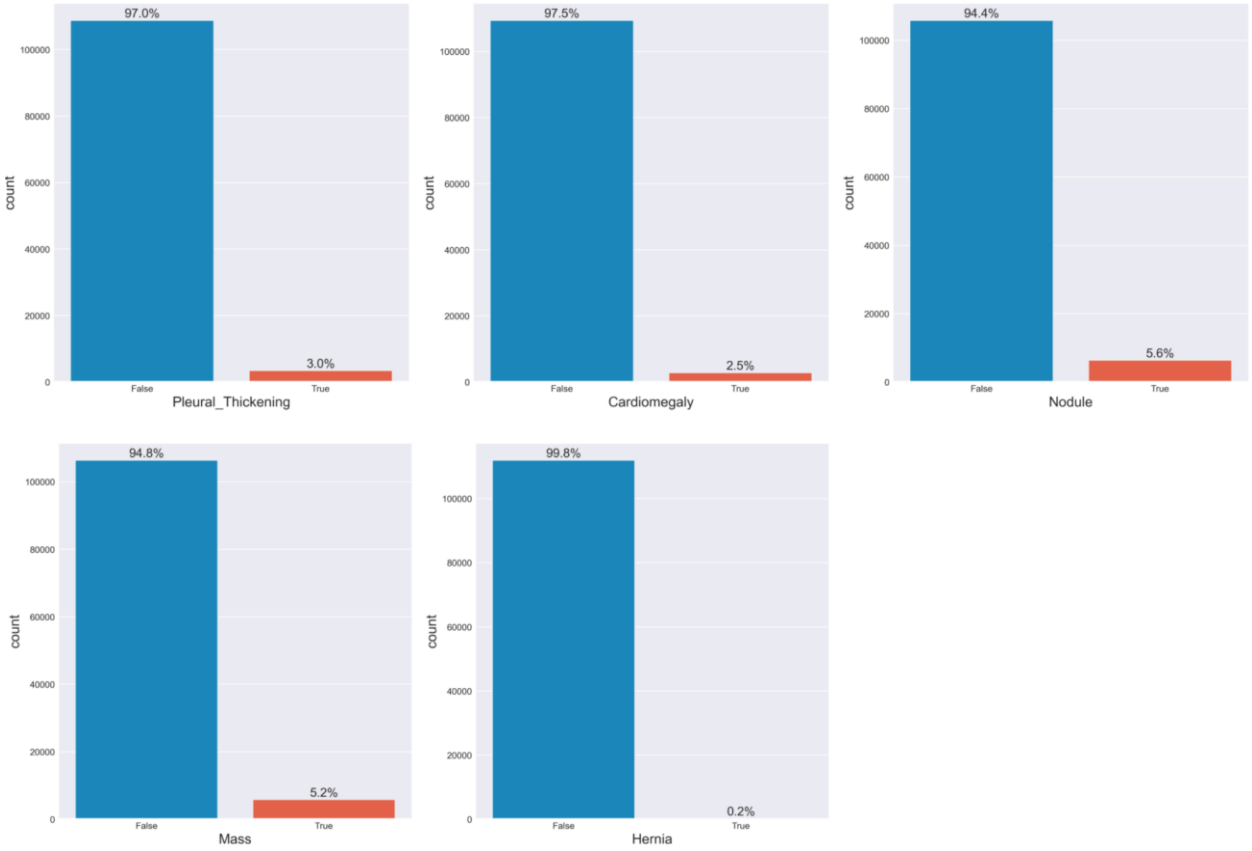


Let’s convert the number to percentage. 53.8% of observations are no finding and 46.2% of observations are other findings, which can be Atelectasis, Consolidation, Infiltration, Pneumothorax, Edema, Emphysema, Fibrosis, Effusion, Pneumonia, Pleural_Thickening, Cardiomegaly, Nodule, Mass, Hernia.

Let's look at each finding below.

NIH CHEST X-RAYS CLASSIFICATION



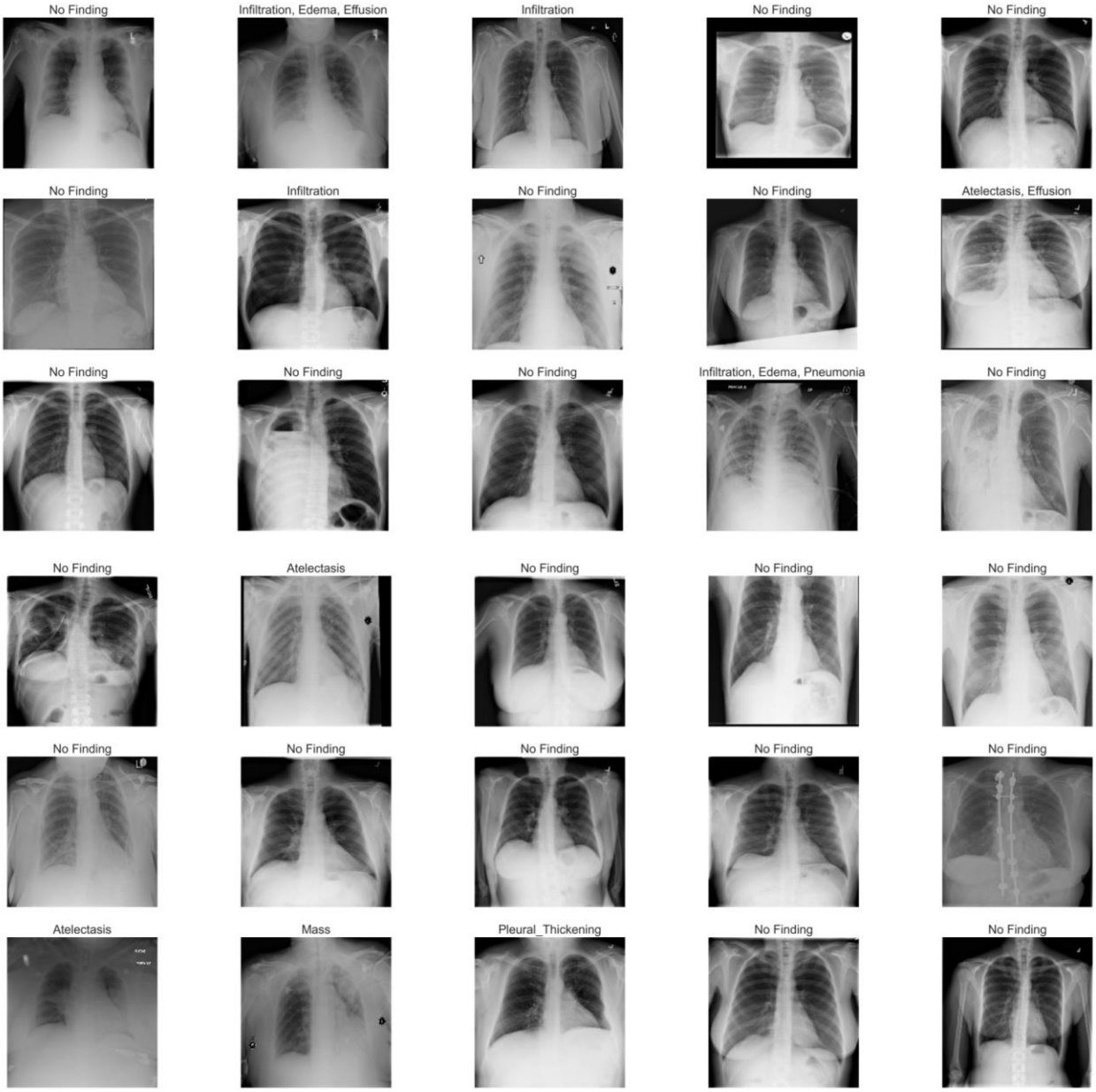


- *Infiltration: 17.7%*
- *Effusion: 11.9%*
- *Atelectasis: 10.3%*
- *Nodule: 5.6%*
- *Mass: 5.2%*
- *Pneumothorax: 4.7%*
- *Consolidation: 4.2%*
- *Pleural_Thickening: 3.0%*
- *Cardiomegaly: 2.5%*
- *Emphysema: 2.2%*
- *Edama: 2.1%*
- *Fibrosis: 1.5%*
- *Pneumonia: 1.3%*
- *Hernia: 0.2%*

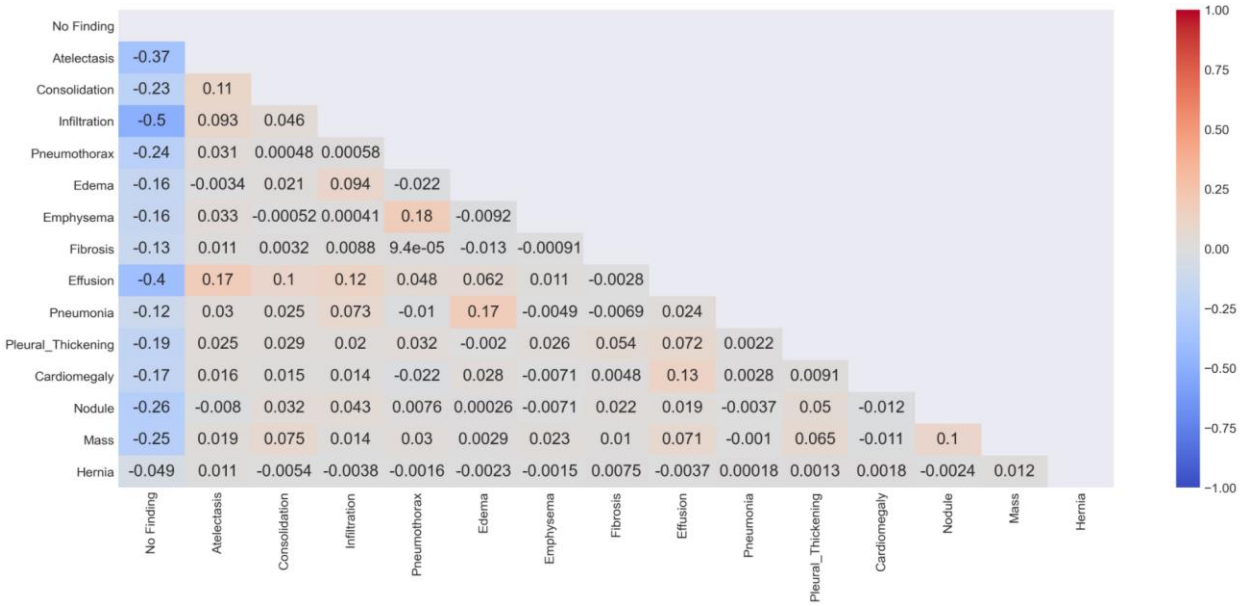
To summarize, Infiltration has the most observations in the dataset for the findings as there are 17.7% of them, whereas Hernia has the least observations in the dataset since there are only 0.2% of them.

b) TFRecords (Chest X-Rays)

Now, we will check out all the images saved in TFRecords. We defined three functions help us load the data in a random order. Please see more details in the modeling section below. Let's visualize 30 random samples.



c) Correlation



As shown in the heatmap, Emphysema and Pneumothorax have the strongest correlation in the dataset with a value of 0.18. The following two variables are Effusion and Atelectasis as their correlation coefficient is 0.17. Similarly, Pneumonia and Edema have 0.17 as their correlation coefficient. Thus, we should keep our eyes on the relationship between:

- *Emphysema and Pneumothorax*
- *Effusion and Atelectasis*
- *Pneumonia and Edema*

Based on their correlation coefficient, we can assume that a patient who has one disease is more likely to have other as well. For instance, if a person who has Emphysema, he/she may also have Pneumothorax.

V. Modeling

Before defining a model, we will be importing the TFRecord Files. It is good practice to divide the training set data into two. The smaller dataset will be the validation set. Having a validation set is useful to prevent overfitting as the finetuning of the model will be done by calculating metrics on the validation set and not the training set.

```
# grabbing TFRecords
tfr_list = os.listdir('data/')
tfrlist = ['data/' + x for x in tfr_list]
FILENAMES = tf.io.gfile.glob(tfrlist)

# train_test_split
ALL_INDEX = list(range(len(FILENAMES)))
TRAIN_VALID_INDEX = random.sample(ALL_INDEX, int(len(ALL_INDEX) * 0.8))
'''
random.sample's params:
    sequence: can be a list, tuple, string, or set
    k: an integer value, it specify the length of a sample
'''

TRAIN_INDEX = random.sample(TRAIN_VALID_INDEX, int(len(TRAIN_VALID_INDEX) * 0.9))
VALID_INDEX = list(set(TRAIN_VALID_INDEX) - set(TRAIN_INDEX))
TEST_INDEX = list(set(ALL_INDEX) - set(TRAIN_VALID_INDEX))

# Getting lists for Train, Validation and Test
TRAINING_FILENAMES, VALID_FILENAMES, TEST_FILENAMES = [FILENAMES[index] for index in TRAIN_INDEX], \
                                                      [FILENAMES[index] for index in VALID_INDEX], \
                                                      [FILENAMES[index] for index in TEST_INDEX]

# Check
print('Train TFRecord Files: {}'.format(len(TRAINING_FILENAMES)))
print('Validation TFRecord Files: {}'.format(len(VALID_FILENAMES)))
print('Test TFRecord Files: {}'.format(len(TEST_FILENAMES)))
```

```
Train TFRecord Files: 183
Validation TFRecord Files: 21
Test TFRecord Files: 52
```

We set up the parameters to pass them in some of the functions below.

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
BATCH_SIZE = 70
IMAGE_SIZE = [1024, 1024]
IMAGE_RESIZE = [256, 256]

# feature space for parsing
feature_description = {}
features=list(df.columns)[1:]
for elem in features:
    feature_description[elem] = tf.io.FixedLenFeature([], tf.int64)

feature_description['image'] = tf.io.FixedLenFeature([], tf.string)
feature_description
```

```
{'Atelectasis': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Consolidation': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Infiltration': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Pneumothorax': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Edema': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Emphysema': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Fibrosis': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Effusion': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Pneumonia': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Pleural_Thickening': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Cardiomegaly': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Nodule': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Mass': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'Hernia': FixedLenFeature(shape=[], dtype=tf.int64, default_value=None),
 'image': FixedLenFeature(shape=[], dtype=tf.string, default_value=None)}
```

We then define the functions to help us load in our data.

```
def decode_image(image):
    '''
    The images have to be converted to tensors so that it will be a valid input in our model.
    As images utilize an RBG scale, we specify 3 channels.
    It is also best practice to normalize data before it is is fed into the model.
    For our image data, we will scale it down so that the value of each pixel will range
    from [0, 1] instead of [0. 255].
    We also reshape our data so that all of the images will be the same shape.
    Although the TFRecord files have already been reshaped for us, it is best practice
    to reshape the input so that we know exactly what's going in to our model.
    '''
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.reshape(image, [*IMAGE_SIZE, 3])
    return image

def read_tfrecord(example):   # Reading the file
    '''
    This function uses the above defined feature description to decode the image and its label.
    With the loop, it extracts the feature's labels as a one-hot encoded list.
    No Findings would be a zero vector.
    '''
    example = tf.io.parse_single_example(example, feature_description)
    image = tf.io.decode_jpeg(example['image'], channels=3)
    image = tf.image.resize(image, IMAGE_RESIZE)
    image = tf.cast(image, tf.float32) / 255.0
    label = []
    for val in features: label.append(example[val])
    return image, label

def load_dataset(filenames, labeled=True, ordered=False):
    '''
    The TFRecordDataset weaves together the individiual TFRecords, essentially treating them as one dataset.
    Our dataset is not ordered in any meaningful way, so the order can be ignored when loading our dataset.
    By ignoring the order and reading files as soon as they come in, it will take a shorter time to load the data.
    '''
    ignore_order = tf.data.Options()
    if not ordered:
        ignore_order.experimental_deterministic = False # disable order, increase speed
    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads=AUTOTUNE) # automatically interleaves reads from mul
    dataset = dataset.with_options(ignore_order) # uses data as soon as it streams in, rather than in its original order
    dataset = dataset.map(partial(read_tfrecord), num_parallel_calls=AUTOTUNE)
    # returns a dataset of (image, label) pairs if labeled=True or (image, id) pairs if labeled=False
    return dataset
```

We define the following three functions to get our three different datasets.

```
def get_training_dataset():
    dataset = load_dataset(TRAINING_FILENAMES, labeled=True)
#     dataset = dataset.map(num_parallel_calls=AUTOTUNE)
    dataset = dataset.repeat()
    dataset = dataset.shuffle(2048)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset

def get_validation_dataset(ordered=False):
    dataset = load_dataset(VALID_FILENAMES, labeled=True, ordered=ordered)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.cache()
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset

def get_test_dataset(ordered=False):
    dataset = load_dataset(TEST_FILENAMES, labeled=False, ordered=ordered)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(AUTOTUNE)
    return dataset
```

The following cell returns the number of images we have in each dataset. Notice that the training dataset has the most images. This is normal, as having more training points will allow the model to learn more.

```
def count_data_items(filenames):
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1)) for filename in filenames]
    return np.sum(n)

NUM_TRAINING_IMAGES = count_data_items(TRAINING_FILENAMES)
NUM_VALIDATION_IMAGES = count_data_items(VALID_FILENAMES)
NUM_TEST_IMAGES = count_data_items(TEST_FILENAMES)
STEPS_PER_EPOCH = NUM_TRAINING_IMAGES // BATCH_SIZE
print(
    'Dataset: {} training images, {} validation images, {} unlabeled test images'.format(
        NUM_TRAINING_IMAGES, NUM_VALIDATION_IMAGES, NUM_TEST_IMAGES
    )
)
```

```
Dataset: 80150 training images, 9198 validation images, 22772 unlabeled test images
```

Now that our data has been loaded, we will now start building our model. Let us first define The Learning Rate.

```
def exponential_decay(lr0, s):
    '''
    This function allows for the model to change the learning rate as it runs each epoch.
    Having a learning rate that is too high will prevent our model from converging.
    However, having a learning rate that is too small will cause our model to run for far too long.
    With this function, the model will know how to change its learning rate after each epoch and
    update the learning rate itself to allow for increased efficiency while still allowing the model to converge.
    '''
    def exponential_decay_fn(epoch):
        return lr0 * 0.1 **(epoch / s)
    return exponential_decay_fn

exponential_decay_fn = exponential_decay(0.01, 20)
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(exponential_decay_fn)
```

As mentioned in the beginning, we will build our base model by using transfer learning. Transfer learning is a great way to reap the benefits of a well-trained model without having the train the model ourselves. For this notebook, we want to import the VGG16 model. As this model has already been trained, we do not want to change the weights in this model and must set its trainable attribute to False.

```
# modeling
def make_model(output_bias = None, metrics = None):
    if output_bias is not None:
        output_bias = tf.keras.initializers.Constant(output_bias)

    base_model = tf.keras.applications.Xception(input_shape=(IMAGE_RESIZE[0], IMAGE_RESIZE[1], 3),
                                                include_top=False,
                                                weights='imagenet')

    base_model.trainable = False

    # define a for loop for layers
    for layer in base_model.layers[-5:]:
        layer.trainable=True

    model = tf.keras.Sequential([
        base_model,
        tf.keras.layers.GlobalAveragePooling2D(),
        tf.keras.layers.Dense(8, activation='relu'),
        tf.keras.layers.Dense(14, activation='sigmoid',
                              bias_initializer=output_bias)
    ])

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=metrics)

    return model
```

When we compile our model, we do not want our metric to be accuracy. If we run the model, with an accuracy metric, it will give us false confidence in our model.

```
model = make_model(metrics=tf.keras.metrics.AUC(name='auc'))
```

We can use callbacks to stop training when there are no improvements in our validation set predictions, and this stops overfitting.

```
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("xray_model.h5",
                                                   save_best_only=True)

early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=10,
                                                     restore_best_weights=True)
```

Let's fit our model.

```
history = model.fit(
    train_dataset, epochs=100,
    steps_per_epoch=STEPS_PER_EPOCH,
    validation_data=valid_dataset,
    validation_steps=VALID_STEPS,
    callbacks=[checkpoint_cb, early_stopping_cb, lr_scheduler]
)
```

```
Epoch 1/100
1145/1145 [==============================] - 8194s 7s/step - loss: 0.1789 - auc: 0.7783 - val_loss: 0.1704 - val_auc: 0.8102

D:\Users\tvo10\AppData\Roaming\Python\Python38\site-packages\tensorflow\python\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers require a config and must override get_config. When loading, the custom mask layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '

Epoch 2/100
1145/1145 [==============================] - 7941s 7s/step - loss: 0.1708 - auc: 0.8049 - val_loss: 0.1692 - val_auc: 0.8145
Epoch 3/100
1145/1145 [==============================] - 7930s 7s/step - loss: 0.1689 - auc: 0.8126 - val_loss: 0.1684 - val_auc: 0.8169
Epoch 4/100
1145/1145 [==============================] - 7920s 7s/step - loss: 0.1675 - auc: 0.8173 - val_loss: 0.1693 - val_auc: 0.8136
Epoch 5/100
1145/1145 [==============================] - 7936s 7s/step - loss: 0.1665 - auc: 0.8211 - val_loss: 0.1684 - val_auc: 0.8183
Epoch 6/100
1145/1145 [==============================] - 7951s 7s/step - loss: 0.1657 - auc: 0.8236 - val_loss: 0.1694 - val_auc: 0.8136
Epoch 7/100
1145/1145 [==============================] - 7932s 7s/step - loss: 0.1655 - auc: 0.8250 - val_loss: 0.1676 - val_auc: 0.8209
Epoch 8/100
1145/1145 [==============================] - 7912s 7s/step - loss: 0.1647 - auc: 0.8276 - val_loss: 0.1688 - val_auc: 0.8163
Epoch 9/100
1145/1145 [==============================] - 7876s 7s/step - loss: 0.1642 - auc: 0.8290 - val_loss: 0.1677 - val_auc: 0.8215
Epoch 10/100
1145/1145 [==============================] - 7854s 7s/step - loss: 0.1635 - auc: 0.8310 - val_loss: 0.1671 - val_auc: 0.8220

Epoch 11/100
1145/1145 [==============================] - 7844s 7s/step - loss: 0.1634 - auc: 0.8318 - val_loss: 0.1680 - val_auc: 0.8214
Epoch 12/100
1145/1145 [==============================] - 7816s 7s/step - loss: 0.1629 - auc: 0.8335 - val_loss: 0.1676 - val_auc: 0.8213
Epoch 13/100
1145/1145 [==============================] - 7797s 7s/step - loss: 0.1626 - auc: 0.8344 - val_loss: 0.1682 - val_auc: 0.8186
Epoch 14/100
1145/1145 [==============================] - 7775s 7s/step - loss: 0.1623 - auc: 0.8349 - val_loss: 0.1678 - val_auc: 0.8196
Epoch 15/100
1145/1145 [==============================] - 7765s 7s/step - loss: 0.1622 - auc: 0.8360 - val_loss: 0.1685 - val_auc: 0.8194
Epoch 16/100
1145/1145 [==============================] - 7768s 7s/step - loss: 0.1620 - auc: 0.8365 - val_loss: 0.1676 - val_auc: 0.8211
Epoch 17/100
1145/1145 [==============================] - 7756s 7s/step - loss: 0.1617 - auc: 0.8372 - val_loss: 0.1681 - val_auc: 0.8204
Epoch 18/100
1145/1145 [==============================] - 7723s 7s/step - loss: 0.1615 - auc: 0.8376 - val_loss: 0.1676 - val_auc: 0.8212
Epoch 19/100
1145/1145 [==============================] - 7721s 7s/step - loss: 0.1613 - auc: 0.8385 - val_loss: 0.1675 - val_auc: 0.8213
Epoch 20/100
1145/1145 [==============================] - 7752s 7s/step - loss: 0.1613 - auc: 0.8388 - val_loss: 0.1674 - val_auc: 0.8213
```

Even though I set 100 as the value for Epoch, it stopped at 20 since the model's accuracy score will not be improved anymore. As shown in the image, we got 0.8388 as the highest score. Now we will evaluate our model using our test dataset.

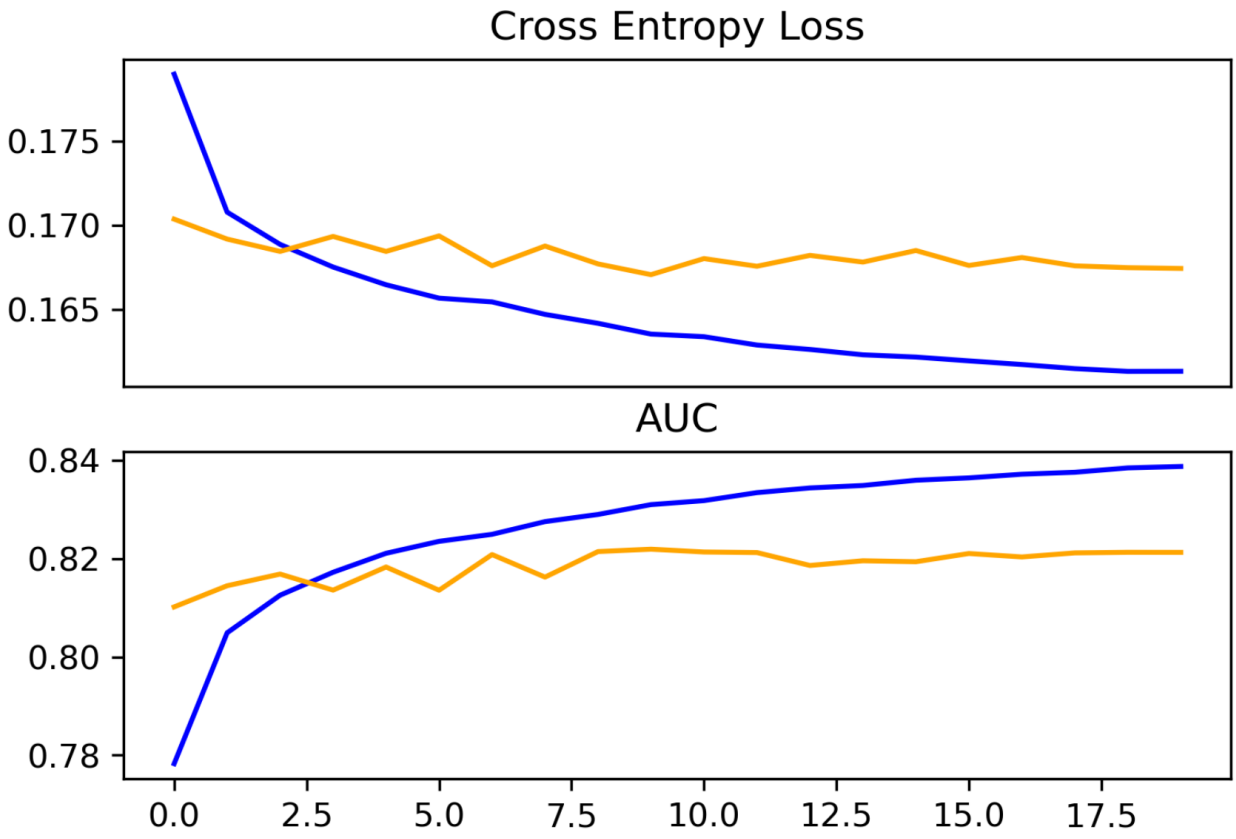
```
test_dataset = get_test_dataset()

_, test_auc = model.evaluate(test_dataset, verbose=0)

print('Test auc:', test_auc)
```

```
Test auc: 0.8237475752830505
```

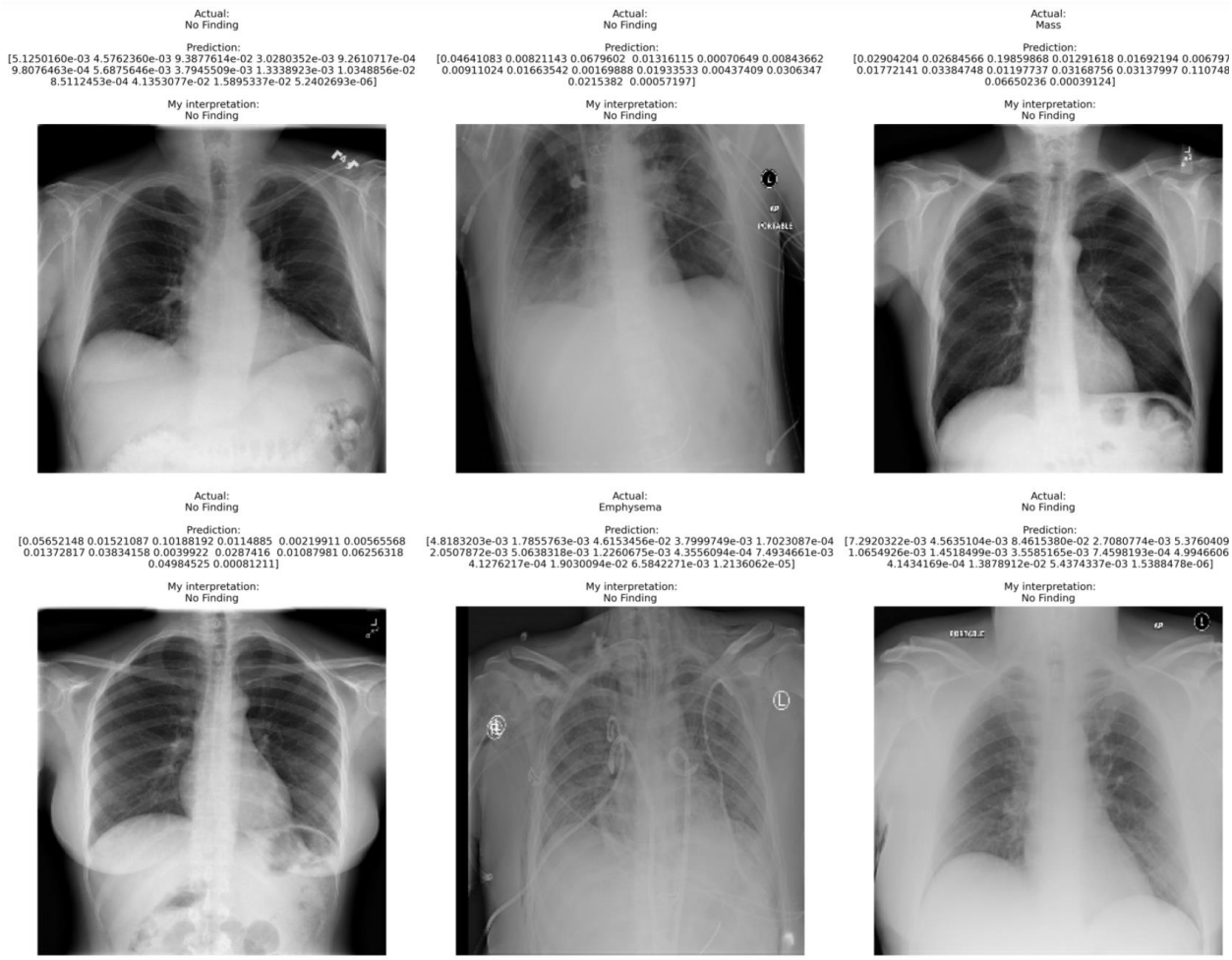
Based on our test dataset, we got 0.82 for the accuracy score, which is already accomplished our objective. Even though the model still needs to be improved, we decided to temporarily accept 0.82 accuracy score as an achievement for this notebook as this score is higher than several models on Kaggle.

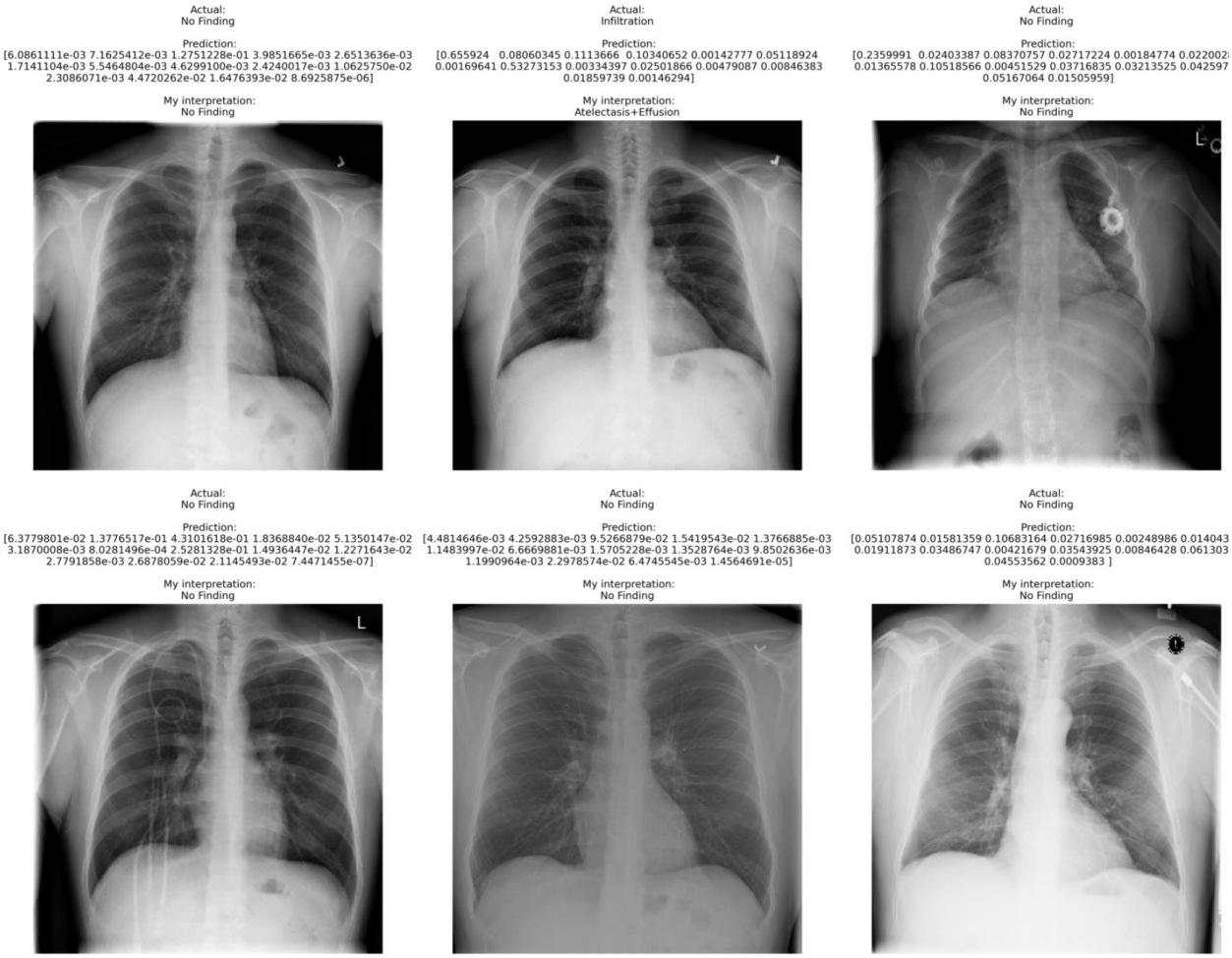


As the cross-entropy loss decreases, AUC increases, which is a common sense.

NIH CHEST X-RAYS CLASSIFICATION

Using our model, let us visualize some of our predictions, whether the chest x-ray images are normal or abnormal using our testing dataset.





VI. Conclusion and Feature Work

The model predicts most of the images as "No Finding" since the number of "No Finding" observations is larger than the number of "Findings" observations that belong to 14 classes. Hence, we have two methods to improve our model in the feature work:

1. Merging all 14 disease classes to one class as "Findings", so we will only have 2 classes "No Finding" and "Findings". This will help us boost our model by balancing the class weight.
2. Finding a way to make sure that we have a balanced dataset, so all 15 classes in the dataset will have approximately the same number of observations. This can also boost our model performance.