IDS 560 – Analytics Strategy & Practice

IYKA Enterprises, Inc.

Project Technical Discussion

Team Members

Charu Yadav [653762591]

Deepak Singhal [672190946]

Medhavi Pokhriyal [650144850]

Reza Amini [672783344]

Sunny Patel [676645654]

Technical Documentation

Introduction

Artificial intelligence (AI) is the simulation of human processes by computers, and its simulation includes learning, reasoning, and self-correction. As the number of tasks grows by the day, we require AI to automate routine processes. In terms of the healthcare industry, AI can be used to improve the lives of patients, physicians, and healthcare workers by performing tasks that doctors frequently perform, but at a much lower cost and in a much shorter time. Furthermore, using artificial intelligence to diagnose and treat patient conditions improves and accelerates decision-making. The best thing about using AI in healthcare is that it improves a wide range of areas such as disease detection, optimal treatments, and so on, potentially saving many lives.

AI in healthcare refers to a collection of technologies that allow machines/robots to detect, comprehend, act, and learn to perform administrative and clinical healthcare tasks. By addressing some of the industry's most pressing issues, AI has the potential to transform healthcare. AI, for example, can lead to better patient outcomes as well as increased productivity and efficiency in care delivery. It can also improve healthcare practitioners' daily lives by allowing them to use the time saved by AI to treat other patients. When the high volume of images a radiologist is tasked with reviewing is combined with the fact that some medical centers may not have a radiologist on-call around the clock, pneumothorax diagnosis can be delayed for many hours. Looking at these scenarios, AI can play a critical role in relieving many workloads where labor is not required. As a result, staff morale and retention will improve.

As we all know, chest X-rays are one of the most common medical imaging examinations. However, when compared to other imaging examinations, such as a CT scan, the diagnosis of a chest X-ray can be extremely difficult. Thus, the goal of this project is to automate the diagnostic process in order to achieve clinically relevant computer-aided detection and diagnosis (CAD) with chest X-rays. In terms of business, hospitals rely heavily on doctors for diagnostics via various imaging examinations. As a result, this CAD project is geared toward hospitals that want to increase efficiency by automating diagnosis procedures and making better use of their doctors.

Dataset

One significant challenge in creating large X-ray image datasets is the lack of resources for labeling so many images. This NIH Chest X-ray Dataset is a publicly available dataset that was extracted by using Natural Language Processing to text-mine disease classifications from associated radiological reports. The dataset contains 112,120 X-ray images with disease labels from 30,805 distinct patients. The label’s accuracy in the dataset is more than 90% and hence, is suitable for simulating real-world analysis.

Data Preprocessing

Once the data is downloaded, we start with the first step in the data analysis lifecycle which is Exploratory Data Analysis. In the EDA, we start by loading the data & transforming it into a more readable format for further analysis. This is a crucial step, as the path of the images is mapped corresponding to each patient’s row.

In [2]:

```
all_xray_df = pd.read_csv('../input/Data_Entry_2017.csv')
all_image_paths = {os.path.basename(x): x for x in
                   glob(os.path.join('..', 'input', 'images*', '*', '*.png'))}
print('Scans found:', len(all_image_paths), ', Total Headers', all_xray_df.shape[0])
all_xray_df['path'] = all_xray_df['Image Index'].map(all_image_paths.get)
all_xray_df['Patient Age'] = all_xray_df['Patient Age'].map(lambda x: int(x[:-1]))
all_xray_df.sample(3)
```

```
Scans found: 112120 , Total Headers 112120
```

Out[2]:

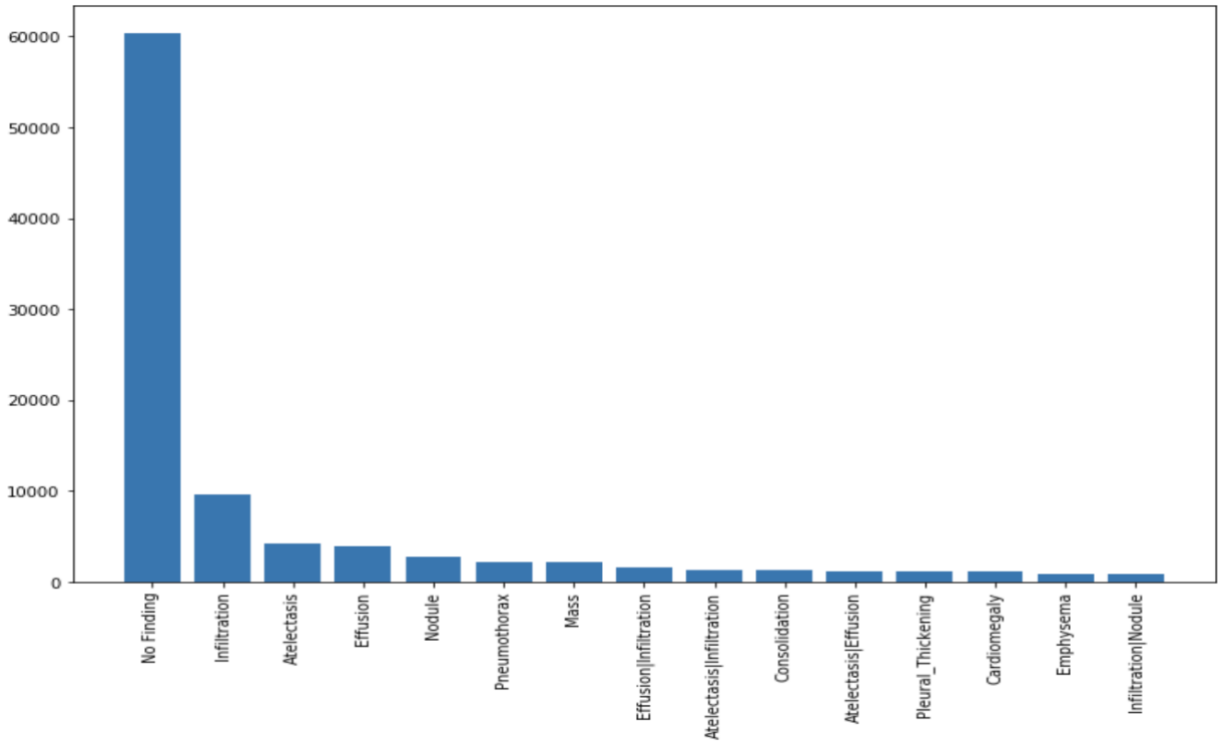
	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position	OriginalImageWidth	OriginalImageHeight
57219	00014203_013.png	No Finding	13	14203	30	M	PA	2946	2953
56096	00013996_002.png	No Finding	2	13996	60	M	PA	2500	2048
47004	00011984_004.png	No Finding	4	11984	25	F	AP	2500	2048

Once the data is successfully loaded and mapped, the preprocessing begins. Here, we take the disease labels and apply descriptive analytics to gain information on the data. Visualizing the data seems to be the most viable option in this step & hence, we generate a plot that describes the number of patients with different disease labels.

In [3]:

```
label_counts = all_xray_df['Finding Labels'].value_counts()[:15]
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
_ = ax1.set_xticklabels(label_counts.index, rotation = 90)
```

Plot generate before cleaning categories



In [4]:

```
all_xray_df['Finding Labels'] = all_xray_df['Finding Labels'].map(lambda x: x.r
eplace('No Finding', ''))
from itertools import chain
all_labels = np.unique(list(chain(*all_xray_df['Finding Labels'].map(lambda x:
x.split('|')).tolist())))
all_labels = [x for x in all_labels if len(x)>0]
print('All Labels ({}): {}'.format(len(all_labels), all_labels))
for c_label in all_labels:
    if len(c_label)>1: # leave out empty labels
        all_xray_df[c_label] = all_xray_df['Finding Labels'].map(lambda findin
g: 1.0 if c_label in finding else 0)
all_xray_df.sample(3)
```

```
All Labels (14): ['Atelectasis', 'Cardiomegaly', 'Consolidation', 'Edema',
'Effusion', 'Emphysema', 'Fibrosis', 'Hernia', 'Infiltration', 'Mass', 'Nodu
le', 'Pleural_Thickening', 'Pneumonia', 'Pneumothorax']
```

Out[4]:

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position	OriginalImageWid
7700	00002016_002.png		2	2016	66	F	AP	2500
86114	00021231_002.png	Atelectasis	2	21231	46	F	AP	3056
46568	00011896_012.png		12	11896	43	F	PA	2580

Preprocessing aims to clean categories, since, there are many categories that have less impact and so we proceed by pruning, by taking only a few examples.

In [5]:

```
# keep at least 1000 cases
MIN_CASES = 1000
all_labels = [c_label for c_label in all_labels if all_xray_df[c_label].sum()>M
IN_CASES]
print('Clean Labels ({})'.format(len(all_labels)),
      [(c_label,int(all_xray_df[c_label].sum())) for c_label in all_labels])
```

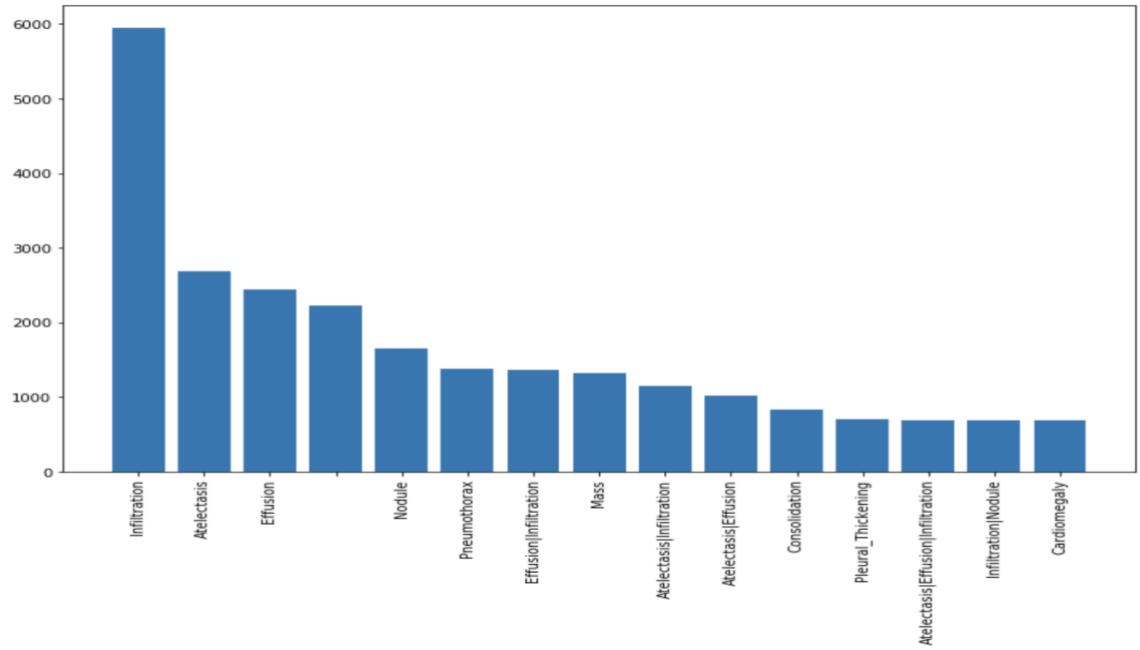
```
Clean Labels (13) [('Atelectasis', 11535), ('Cardiomegaly', 2772), ('Consoli
dation', 4667), ('Edema', 2303), ('Effusion', 13307), ('Emphysema', 2516),
('Fibrosis', 1686), ('Infiltration', 19870), ('Mass', 5746), ('Nodule', 632
3), ('Pleural_Thickening', 3385), ('Pneumonia', 1353), ('Pneumothorax', 529
8)]
```

During preprocessing, we discovered that there was resampling needed due to the bias of the dataset.

In [6]:

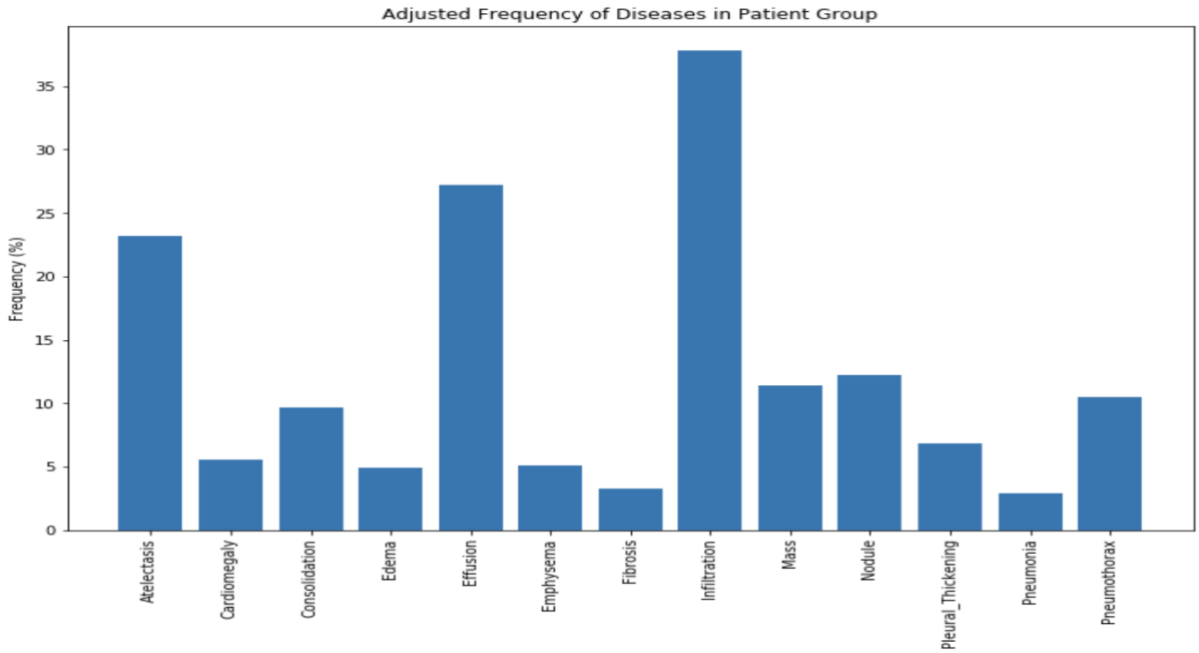
```
# since the dataset is very unbiased, we can resample it to be a more reasonable
collection
# weight is 0.1 + number of findings
sample_weights = all_xray_df['Finding Labels'].map(lambda x: len(x.split('|'))
if len(x)>0 else 0).values + 4e-2
sample_weights /= sample_weights.sum()
all_xray_df = all_xray_df.sample(40000, weights=sample_weights)

label_counts = all_xray_df['Finding Labels'].value_counts()[:15]
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
_ = ax1.set_xticklabels(label_counts.index, rotation = 90)
```



Hence, we adjust the frequency of the diseases by changing the weights that were initially assigned.

In [7]:
```
label_counts = 100*np.mean(all_xray_df[all_labels].values,0)
fig, ax1 = plt.subplots(1,1,figsize = (12, 8))
ax1.bar(np.arange(len(label_counts))+0.5, label_counts)
ax1.set_xticks(np.arange(len(label_counts))+0.5)
ax1.set_xticklabels(all_labels, rotation = 90)
ax1.set_title('Adjusted Frequency of Diseases in Patient Group')
_ = ax1.set_ylabel('Frequency (%)')
```



Prepare Training Data

In this step, we divide the data into training and validation sets and construct a single vector with Binary (0/1) outputs for disease status, i.e. 0 depicts the absence of a disease, and 1 depicts the presence of a disease.

In [8]:
```
all_xray_df['disease_vec'] = all_xray_df.apply(lambda x: [x[all_labels].value
s], 1).map(lambda x: x[0])
```

In [9]:
```
from sklearn.model_selection import train_test_split
train_df, valid_df = train_test_split(all_xray_df,
                                      test_size = 0.25,
                                      random_state = 2018,
                                      stratify = all_xray_df['Finding Labels'].map
(lambda x: x[:4]))
print('train', train_df.shape[0], 'validation', valid_df.shape[0])
```

```
train 30000 validation 10000
```

Data Generators

Since, the X-Ray Image dataset is too large to be processed in one go, hence, we use data generators. Data Generators use batch gradient descent, to train a model with smaller chunks of data. Moreover, almost all the real-world datasets pertaining to deep learning cannot fit into memory in a single pass, therefore, this is frequently the only viable solution. In this section, we create data generators for loading and randomly transforming images by using a deep-learning library called Keras.

In [10]:
```
from keras.preprocessing.image import ImageDataGenerator
IMG_SIZE = (128, 128)
core_idg = ImageDataGenerator(samplewise_center=True,
                              samplewise_std_normalization=True,
                              horizontal_flip = True,
                              vertical_flip = False,
                              height_shift_range= 0.05,
                              width_shift_range=0.1,
                              rotation_range=5,
                              shear_range = 0.1,
                              fill_mode = 'reflect',
                              zoom_range=0.15)
```

```
/opt/conda/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: C
onversion of the second argument of issubdtype from `float` to `np.floating`
is deprecated. In future, it will be treated as `np.float64 == np.dtype(floa
t).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [11]:

```
def flow_from_dataframe(img_data_gen, in_df, path_col, y_col, **dflow_args):
    base_dir = os.path.dirname(in_df[path_col].values[0])
    print('## Ignore next message from keras, values are replaced anyways')
    df_gen = img_data_gen.flow_from_directory(base_dir,
                                     class_mode = 'sparse',
                                    **dflow_args)
    df_gen.filenames = in_df[path_col].values
    df_gen.classes = np.stack(in_df[y_col].values)
    df_gen.samples = in_df.shape[0]
    df_gen.n = in_df.shape[0]
    df_gen._set_index_array()
    df_gen.directory = '' # since we have the full path
    print('Reinserting dataframe: {} images'.format(in_df.shape[0]))
    return df_gen
```

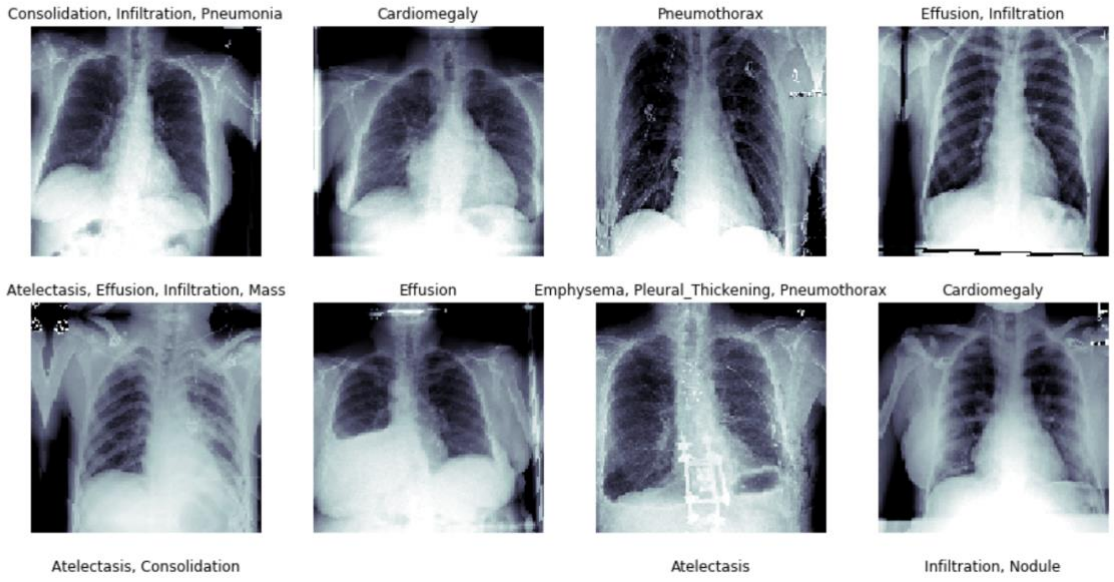
In [12]:

```
train_gen = flow_from_dataframe(core_idg, train_df,
                             path_col = 'path',
                            y_col = 'disease_vec',
                            target_size = IMG_SIZE,
                             color_mode = 'grayscale',
                            batch_size = 32)

valid_gen = flow_from_dataframe(core_idg, valid_df,
                             path_col = 'path',
                            y_col = 'disease_vec',
                            target_size = IMG_SIZE,
                             color_mode = 'grayscale',
                            batch_size = 256) # we can use much larger batches f
or evaluation
# used a fixed dataset for evaluating the algorithm
test_X, test_Y = next(flow_from_dataframe(core_idg,
                               valid_df,
                             path_col = 'path',
                            y_col = 'disease_vec',
                            target_size = IMG_SIZE,
                             color_mode = 'grayscale',
                            batch_size = 1024)) # one big batch
```

In [13]:

```
t_x, t_y = next(train_gen)
fig, m_axs = plt.subplots(4, 4, figsize = (16, 16))
for (c_x, c_y, c_ax) in zip(t_x, t_y, m_axs.flatten()):
    c_ax.imshow(c_x[:,:,0], cmap = 'bone', vmin = -1.5, vmax = 1.5)
    c_ax.set_title(', '.join([n_class for n_class, n_score in zip(all_labels, c
_y)
                              if n_score>0.5]))
    c_ax.axis('off')
```



Deep Learning Algorithm - CNN

Convolutional neural networks (CNN) are a type of artificial neural network architecture that has been widely applied to a wide range of pattern recognition problems such as image classification, image recognition, medical diagnostics, and so on. A CNN is made up of layers that filter (convolve) inputs to produce useful information. These convolutional layers have learned parameters (kernel) that allow these filters to be automatically adjusted to extract the most useful information for the task at hand without feature selection. CNN works better with images. Normal neural networks do not perform well in image classification tasks.

Detecting abnormal components in medical imaging is a significant logistical problem for physicians, hence, the more accurate the detection results lead to a more efficient process of diagnosis and treatment. The process begins with the development of a simple preprocessing pipeline using digital image processing techniques, followed by the development of a pipeline that can apply neural network architectures, and finally, the development of a pipeline that can use neural network visualization techniques to understand what types of features our model weights the most heavily.

To apply DL Algorithm, we start by creating a simple model using MobileNet as the base layer, followed by, adding a gap layer (such as Dense, Flatten), dropout & fully connected layer.

In [14]:

```
from keras.applications.mobilenet import MobileNet
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten
from keras.models import Sequential
base_mobilenet_model = MobileNet(input_shape =  t_x.shape[1:],
                                 include_top = False, weights = None)
multi_disease_model = Sequential()
multi_disease_model.add(base_mobilenet_model)
multi_disease_model.add(GlobalAveragePooling2D())
multi_disease_model.add(Dropout(0.5))
multi_disease_model.add(Dense(512))
multi_disease_model.add(Dropout(0.5))
multi_disease_model.add(Dense(len(all_labels), activation = 'sigmoid'))
multi_disease_model.compile(optimizer = 'adam', loss = 'binary_crossentropy',
                            metrics = ['binary_accuracy', 'mae'])
multi_disease_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenet_1.00_128 (Model)   (None, 4, 4, 1024)        3228288
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 13)                6669
=================================================================
Total params: 3,759,757
Trainable params: 3,737,869
Non-trainable params: 21,888
_________________________________________________________________
```

ModelCheckpoint callback function that is used in conjunction with model.fit() training to save a model or weights (in a checkpoint file) at some interval so that the model or weights can be loaded later to continue training from the saved state.

The ModelCheckpoint callback function provides a few options-

a. Whether to keep only the model with the "best performance" so far or to save the model at the end of each epoch regardless of performance.
b. The definition of "best" is which quantity to track and whether it should be maximized or minimized.

c. The frequency at which it should save. The callback currently supports saving at the end of each epoch or after a set number of training batches.
d. Whether only the weights are saved or the entire model.

EarlyStopping function's goal is to minimize loss. The metric to be monitored would be loss, and the mode would be minimum.' model.fit() function training loop will check at the end of each epoch to see if the loss is still decreasing.

In [15]:
```
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStoppi
ng, ReduceLROnPlateau
weight_path="{}_weights.best.hdf5".format('xray_class')

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min', save_weights_only
= True)

early = EarlyStopping(monitor="val_loss",
                      mode="min",
                      patience=3)
callbacks_list = [checkpoint, early]
```

Model Fitting

.fit_generator() uses a data augmentation method to artificially create a new dataset for training from the existing training dataset to improve the performance of deep learning neural networks with the amount of data available. It is a form of regularization which makes a model generalize better than before.

In [16]:
```
multi_disease_model.fit_generator(train_gen,
                                  steps_per_epoch=100,
                                  validation_data = (test_X, test_Y),
                                  epochs = 1,
                                  callbacks = callbacks_list)
```

```
Epoch 1/1
 99/100 [============================>.] - ETA: 4s - loss: 0.4498 - binary_a
ccuracy: 0.8516 - mean_absolute_error: 0.2011
Epoch 00001: val_loss improved from inf to 0.79686, saving model to xray_cla
ss_weights.best.hdf5
100/100 [==============================] - 484s 5s/step - loss: 0.4496 - bin
ary_accuracy: 0.8516 - mean_absolute_error: 0.2012 - val_loss: 0.7969 - val_
binary_accuracy: 0.8631 - val_mean_absolute_error: 0.1542
```

Out[16]:
```
<keras.callbacks.History at 0x7ff2b0ed0cc0>
```

Checking Output

Examining output to check the number of positive patients present in each disease category.

In [17]:
```
for c_label, s_count in zip(all_labels, 100*np.mean(test_Y,0)):
    print('%s: %2.2f%%' % (c_label, s_count))
```

```
Atelectasis: 24.41%
Cardiomegaly: 5.18%
Consolidation: 10.55%
Edema: 4.39%
Effusion: 26.27%
Emphysema: 4.69%
Fibrosis: 2.64%
Infiltration: 37.21%
Mass: 11.91%
Nodule: 12.40%
Pleural_Thickening: 5.96%
Pneumonia: 2.73%
Pneumothorax: 11.04%
```

In [18]:
```
pred_Y = multi_disease_model.predict(test_X, batch_size = 32, verbose = True)
```

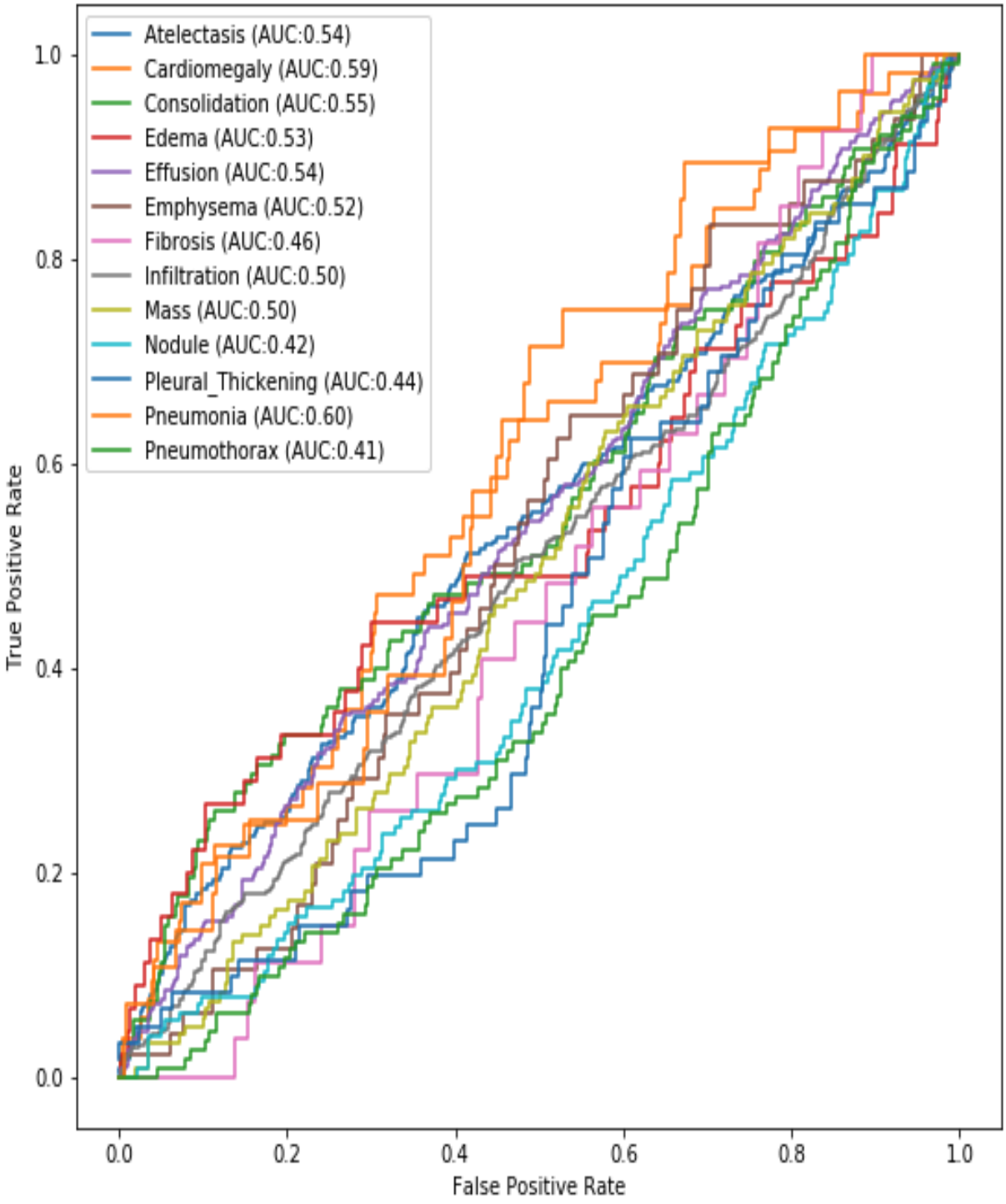
```
1024/1024 [==============================] - 45s 44ms/step
```

Evaluation Metric - ROC & AUC Curves

The receiver operating characteristic (ROC) curve is a useful tool in binary classification, and the Area Under the ROC Curve (AUC) is a popular metric for reporting binary classifier performance.

In [19]:
```
from sklearn.metrics import roc_curve, auc
fig, c_ax = plt.subplots(1,1, figsize = (9, 9))
for (idx, c_label) in enumerate(all_labels):
    fpr, tpr, thresholds = roc_curve(test_Y[:,idx].astype(int), pred_Y[:,idx])
    c_ax.plot(fpr, tpr, label = '%s (AUC:%0.2f)'  % (c_label, auc(fpr, tpr)))
c_ax.legend()
c_ax.set_xlabel('False Positive Rate')
c_ax.set_ylabel('True Positive Rate')
fig.savefig('barely_trained_net.png')
```

Plot for barely trained data



Continued Training

We are now conducting a much longer training process to see how the results improve.

In [20]:
```
multi_disease_model.fit_generator(train_gen,
                                  steps_per_epoch = 100,
                                  validation_data =  (test_X, test_Y),
                                  epochs = 5,
                                  callbacks = callbacks_list)
```

```
Epoch 1/5
 99/100 [============================>.] - ETA: 4s - loss: 0.3723 - binary_accuracy: 0.8664
- mean_absolute_error: 0.1972
Epoch 00001: val_loss improved from 0.79686 to 0.34932, saving model to xray_class_weights.
best.hdf5
100/100 [==============================] - 483s 5s/step - loss: 0.3721 - binary_accuracy:
0.8666 - mean_absolute_error: 0.1972 - val_loss: 0.3493 - val_binary_accuracy: 0.8738 - val
_mean_absolute_error: 0.2016
Epoch 2/5
 99/100 [============================>.] - ETA: 4s - loss: 0.3494 - binary_accuracy: 0.8718
- mean_absolute_error: 0.1990
Epoch 00002: val_loss did not improve
100/100 [==============================] - 490s 5s/step - loss: 0.3495 - binary_accuracy:
0.8718 - mean_absolute_error: 0.1991 - val_loss: 0.3515 - val_binary_accuracy: 0.8616 - val
_mean_absolute_error: 0.2050
Epoch 3/5
 99/100 [============================>.] - ETA: 4s - loss: 0.3417 - binary_accuracy: 0.8727
- mean_absolute_error: 0.1985
Epoch 00003: val_loss did not improve
100/100 [==============================] - 464s 5s/step - loss: 0.3417 - binary_accuracy:
0.8727 - mean_absolute_error: 0.1985 - val_loss: 0.3554 - val_binary_accuracy: 0.8582 - val
_mean_absolute_error: 0.1865
Epoch 4/5
 99/100 [============================>.] - ETA: 3s - loss: 0.3389 - binary_accuracy: 0.8732
- mean_absolute_error: 0.1969
```

In [21]:
```
# load the best weights
multi_disease_model.load_weights(weight_path)
```

In [22]:
```
pred_Y = multi_disease_model.predict(test_X, batch_size = 32, verbose = True)
```

```
1024/1024 [==============================] - 34s 34ms/step
```

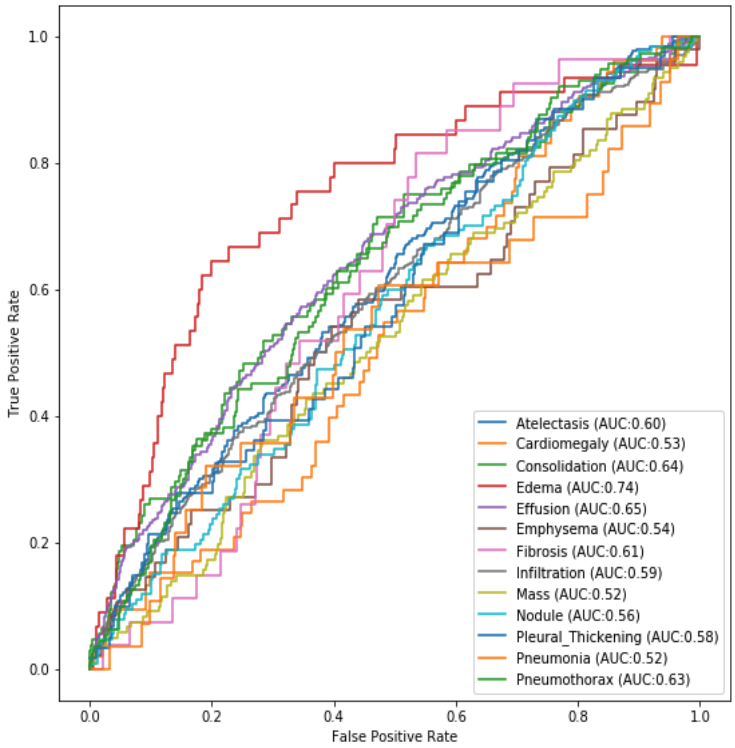
In [23]:
```
# look at how often the algorithm predicts certain diagnoses
for c_label, p_count, t_count in zip(all_labels,
                                     100*np.mean(pred_Y,0),
                                     100*np.mean(test_Y,0)):
    print('%s: Dx: %2.2f%%, PDx: %2.2f%%' % (c_label, t_count, p_count))
```

```
Atelectasis: Dx: 24.41%, PDx: 25.93%
Cardiomegaly: Dx: 5.18%, PDx: 3.92%
Consolidation: Dx: 10.55%, PDx: 13.94%
Edema: Dx: 4.39%, PDx: 7.24%
Effusion: Dx: 26.27%, PDx: 26.18%
Emphysema: Dx: 4.69%, PDx: 2.29%
Fibrosis: Dx: 2.64%, PDx: 3.05%
Infiltration: Dx: 37.21%, PDx: 49.67%
Mass: Dx: 11.91%, PDx: 9.15%
Nodule: Dx: 12.40%, PDx: 9.72%
Pleural_Thickening: Dx: 5.96%, PDx: 5.32%
Pneumonia: Dx: 2.73%, PDx: 4.52%
Pneumothorax: Dx: 11.04%, PDx: 6.85%
```

Plot for Trained Data

In [24]:

```
from sklearn.metrics import roc_curve, auc
fig, c_ax = plt.subplots(1,1, figsize = (9, 9))
for (idx, c_label) in enumerate(all_labels):
    fpr, tpr, thresholds = roc_curve(test_Y[:,idx].astype(int), pred_Y[:,idx])
    c_ax.plot(fpr, tpr, label = '%s (AUC:%0.2f)'  % (c_label, auc(fpr, tpr)))
c_ax.legend()
c_ax.set_xlabel('False Positive Rate')
c_ax.set_ylabel('True Positive Rate')
fig.savefig('trained_net.png')
```

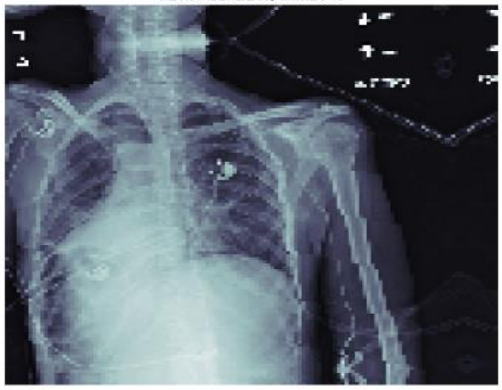


Results - Showing Images & Associated Predictions

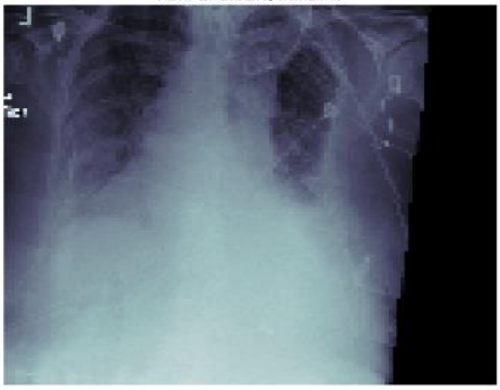
In [25]:

```
sickest_idx = np.argsort(np.sum(test_Y, 1)<1)
fig, m_axs = plt.subplots(4, 2, figsize = (16, 32))
for (idx, c_ax) in zip(sickest_idx, m_axs.flatten()):
    c_ax.imshow(test_X[idx, :,:,0], cmap = 'bone')
    stat_str = [n_class[:6] for n_class, n_score in zip(all_labels,
                                                        test_Y[idx])
                            if n_score>0.5]
    pred_str = ['%s:%2.0f%%' % (n_class[:4], p_score*100)  for n_class, n_score, p_score in zi
p(all_labels,
                                                        test_Y[idx], pred_Y[idx])
                            if (n_score>0.5) or (p_score>0.5)]
    c_ax.set_title('Dx: '+', '.join(stat_str)+'\nPDx: '+', '.join(pred_str))
    c_ax.axis('off')
fig.savefig('trained_img_predictions.png')
```

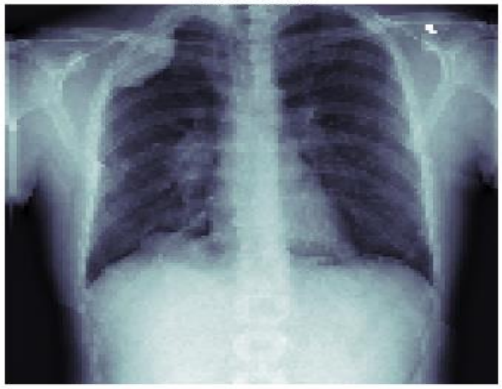
Dx: Atelec
PDx: Atel:21%, Infi:57%



Dx: Effusi
PDx: Effu:32%, Infi:57%



Dx: Mass
PDx: Mass:10%

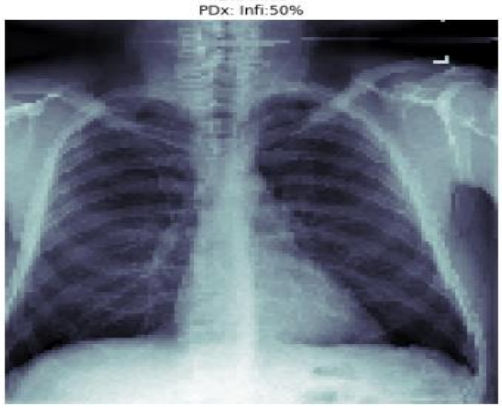


Dx: Infilt
PDx: Infi:54%

Dx: Mass
PDx: Mass:10%

Dx: Atelec
PDx: Atel:30%, Infi:56%

Dx: Infilt
PDx: Infi:50%



Dx: Infilt, Mass
PDx: Infi:33%, Mass: 9%