# PNEUMONIA DETECTION FROM THE CHEST X-RAY IMAGE

Harsh Kevadia

# Table of Contents

## Domain Background

Coronavirus disease is widely spread around the world in 2020. Coronavirus detection kit is limited as more people are getting infected than producing the Covid-19 diagnosis kit. The kit cost is much higher than developing countries like India, Nepal, Pakistan, and many African countries cannot afford it. These countries are still fighting with the diseases that spread by bugs and mosquitos like Malaria, Dengue, and many more. Covid-19 symptoms are variable, but majorly it includes fever and cough. Fever is most common among many diseases like Malaria, Covid-19, Dengue and others. To predict Covid-19 without using a Coronavirus detection kit is to check the amount of the cough present in the lungs. If a certain amount of cough increase in the lungs, it calls Pneumonia. So, Pneumonia detection might be helpful for the pre-stage of identifying Covid-19 patients. Pneumonia can be detected through the X-ray image of the chest or lungs.

## Problem Statement

As per the CDC, Coronavirus infected rate is near to 5. That means one infected person can transfer the Covid-19 virus to the other 5 persons. As an early detection, physicians are recommending doing Pneumonia test through chest X-ray. As a result of that, radiologists supposed to look more X-ray images and take a decision whether a patient has Pneumonia or not. A radiologist is also human, and human makes a mistake. Sometime this mistake has to pay off another human precious life. I can reduce mistake by providing assistant on Pneumonia detection to the radiologist. Artificial Intelligence (AI) has the potential to diagnose disease through image classification. As a part of this project, I am going to train the Machine Learning model which will detect the Pneumonia through chest X-ray image and providing a web application to the radiologist, who can upload the chest X-ray image and our Machine Learning algorithm diagnose Pneumonia disease.

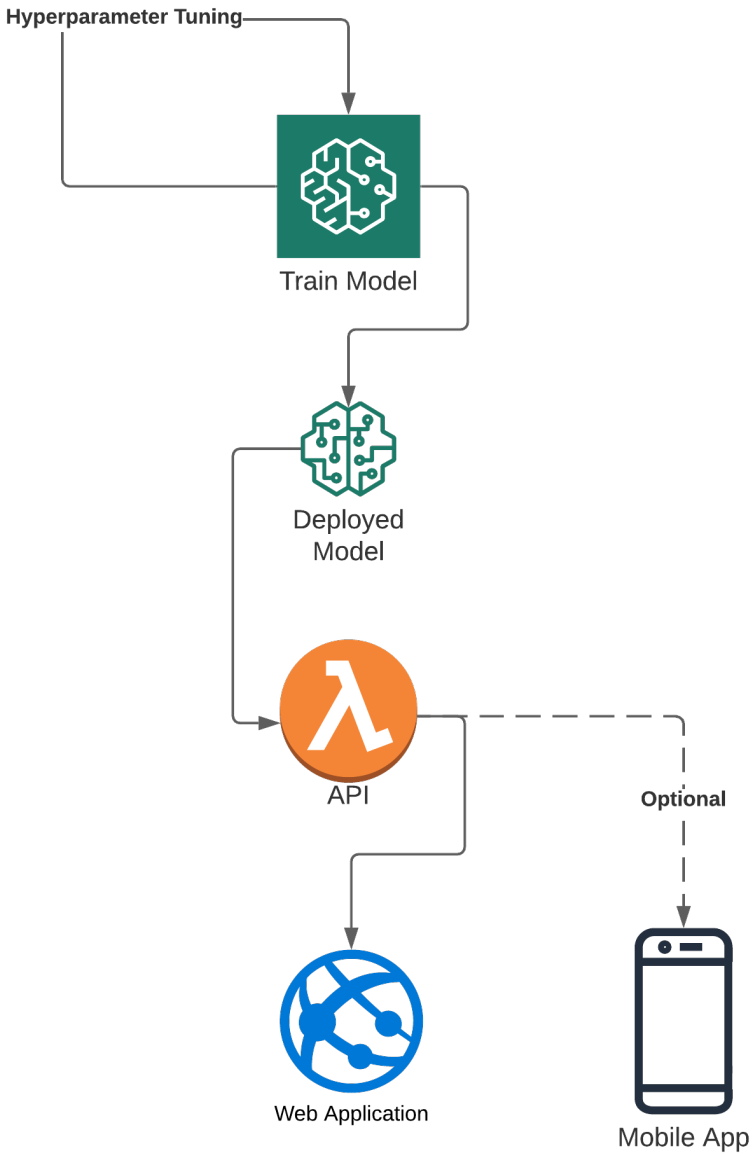
## Datasets and Inputs

I am going to use a chest X-ray dataset from the Kaggle. This dataset has total of 5856 chest X-ray images. Among them, 4,273 images are marked as lungs are infected by Pneumonia and 1,583 images marked as healthy or normal lungs. I am using pixels of the image as an input feature of our machine learning model and I am going to use binary image classification to distinguish between Pneumonia infected lungs vs. normal healthy lungs.

## Solution Statement

As a part of the project, I am going to build the model which will classify or predict that given chest X-ray image has Pneumonia or not. This prediction will have greater than 75% accuracy on test data, so it will assist radiologists accurately.

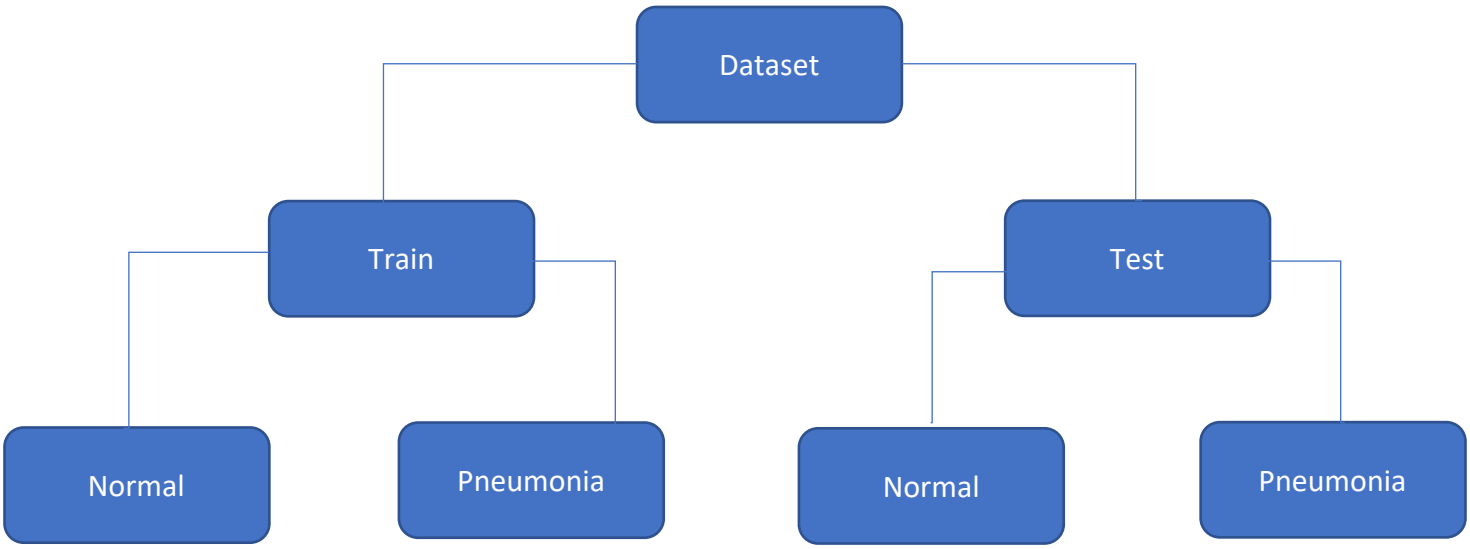
## Project Design/Architecture



## Data Exploration

The dataset compressed file size is 1.5 GB. The dataset is divided into 2 subsets: train and test respectively. Each subset is divided into 2 parts: a normal person's chest x-ray images and pneumonia person's chest x-ray images. Images are in **jpeg** format.

### File Structure



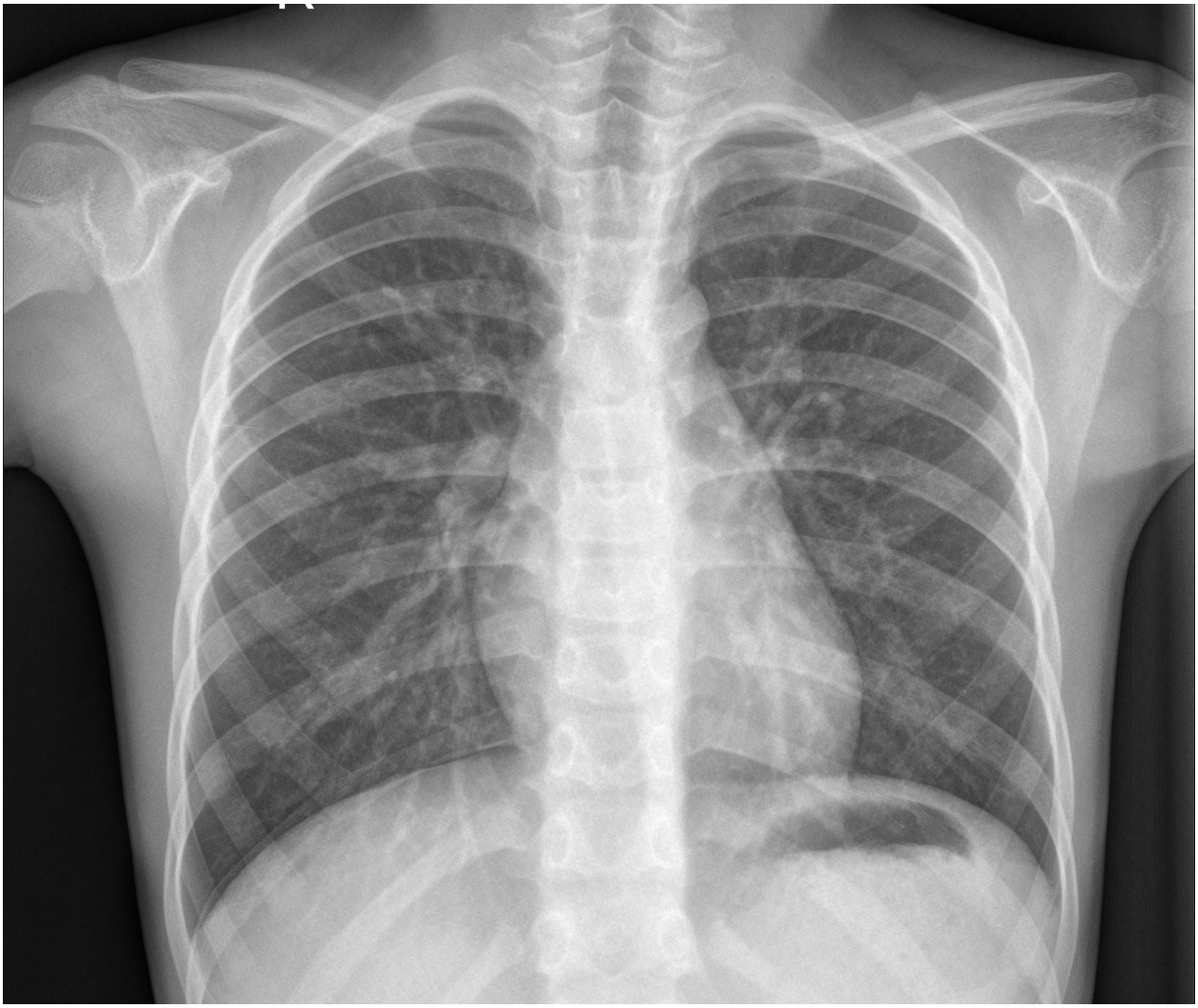

The data **train** dataset contains **5232** total jpeg formatted chest x-ray images and the **test** dataset contains **624** total jpeg formatted chest x-ray images. Following is the screenshot of the counting number of images in each dataset.

**Size of Training, Test and Validation Dataset**

```
print("Number of Images in Training Data: ")
!wc -l chest_xray/train.lst

print("Number of Images in Validation Data: ")
!wc -l chest_xray/validation.lst

print("Number of Images in Test Data: ")
!wc -l chest_xray/test.lst
```

```
Number of Images in Training Data:
5112 chest_xray/train.lst
Number of Images in Validation Data:
120 chest_xray/validation.lst
Number of Images in Test Data:
624 chest_xray/test.lst
```

Normal Person’s Chest X-Ray:



### Pneumonia Infected Person's Chest X-ray:


As you can see from both the images, Pneumonia infected person has more white color on chest x-ray than a normal person x-ray. So, in that data, I am only going to consider a portion of white color over the black color in the image and I can convert color image to black and white image for more accurate results and fewer data points to consider.

## Algorithms and Techniques

The Pneumonia detection is based on image, so I need to build the model such a way that it will classify the image into two categories: Pneumonia, and Normal person's chest x-ray.

As we are using SageMaker as our machine learning pipeline, I found that SageMaker has an *image-classifier* algorithm, it uses a convolutional neural network (ResNet). This is a supervised machine learning algorithm that support multi-label classification. It takes as images and it's belonging class to train the machine learning model. This machine-learning algorithm converts

image to fixed height, width image size. After resizing the image, it uses each pixel as a data point or feature for the image class.

## Benchmark

I am going to split the train dataset into 2 parts: train, and validation dataset. The validation dataset is an important part of training the image classification algorithm. I am separating 60 images from the normal person's chest x-ray images and 60 images from pneumonia infected person's chest x-ray images. As a result, I am getting 120 images as a part of the validation dataset. Validation datasets play a very critical role to train the model properly, it prevents from underfitting and also overfitting of the machine learning model. The machine learning algorithm try to minimize the RMSE and increase validation accuracy. So, I am planning to set epoch to 25 and it will pick up the configuration which will give us maximum validation accuracy.

```
[09/22/2020 03:32:50 INFO 140020406769472] Epoch[18] Train-accuracy=0.993189
[09/22/2020 03:32:50 INFO 140020406769472] Epoch[18] Time cost=68.976
[09/22/2020 03:32:53 INFO 140020406769472] Epoch[18] Validation-accuracy=0.992188
[09/22/2020 03:33:30 INFO 140020406769472] Epoch[19] Batch [20]#011Speed: 69.367 samples/sec#011accuracy=0.995908
[09/22/2020 03:34:02 INFO 140020406769472] Epoch[19] Train-accuracy=0.996595
[09/22/2020 03:34:02 INFO 140020406769472] Epoch[19] Time cost=69.008
[09/22/2020 03:34:04 INFO 140020406769472] Epoch[19] Validation-accuracy=1.000000
[09/22/2020 03:34:04 INFO 140020406769472] Storing the best model with validation accuracy: 1.000000
[09/22/2020 03:34:04 INFO 140020406769472] Saved checkpoint to "/opt/ml/model/image-classification-0020.params"
[09/22/2020 03:34:41 INFO 140020406769472] Epoch[20] Batch [20]#011Speed: 69.690 samples/sec#011accuracy=0.996280
[09/22/2020 03:35:14 INFO 140020406769472] Epoch[20] Train-accuracy=0.995393
[09/22/2020 03:35:14 INFO 140020406769472] Epoch[20] Time cost=69.027
[09/22/2020 03:35:16 INFO 140020406769472] Epoch[20] Validation-accuracy=0.960938
[09/22/2020 03:35:53 INFO 140020406769472] Epoch[21] Batch [20]#011Speed: 70.208 samples/sec#011accuracy=0.993304
[09/22/2020 03:36:25 INFO 140020406769472] Epoch[21] Train-accuracy=0.993590
[09/22/2020 03:36:25 INFO 140020406769472] Epoch[21] Time cost=68.987
[09/22/2020 03:36:27 INFO 140020406769472] Epoch[21] Validation-accuracy=0.984375
[09/22/2020 03:37:04 INFO 140020406769472] Epoch[22] Batch [20]#011Speed: 69.944 samples/sec#011accuracy=0.991443
[09/22/2020 03:37:37 INFO 140020406769472] Epoch[22] Train-accuracy=0.992989
[09/22/2020 03:37:37 INFO 140020406769472] Epoch[22] Time cost=69.132
[09/22/2020 03:37:39 INFO 140020406769472] Epoch[22] Validation-accuracy=0.968750
[09/22/2020 03:38:16 INFO 140020406769472] Epoch[23] Batch [20]#011Speed: 69.544 samples/sec#011accuracy=0.990699
[09/22/2020 03:38:49 INFO 140020406769472] Epoch[23] Train-accuracy=0.992989
[09/22/2020 03:38:49 INFO 140020406769472] Epoch[23] Time cost=69.032
[09/22/2020 03:38:51 INFO 140020406769472] Epoch[23] Validation-accuracy=0.984375
[09/22/2020 03:39:28 INFO 140020406769472] Epoch[24] Batch [20]#011Speed: 70.240 samples/sec#011accuracy=0.992932
[09/22/2020 03:40:00 INFO 140020406769472] Epoch[24] Train-accuracy=0.993590
[09/22/2020 03:40:00 INFO 140020406769472] Epoch[24] Time cost=68.842
[09/22/2020 03:40:02 INFO 140020406769472] Epoch[24] Validation-accuracy=0.992188

2020-09-22 03:40:28 Uploading - Uploading generated training model
2020-09-22 03:40:28 Completed - Training job completed
Training seconds: 1944
Billable seconds: 1944
CPU times: user 4.59 s, sys: 225 ms, total: 4.81 s
Wall time: 35min 40s
```

The above screenshot is the part logs of the machine learning model training. As you can see that on epoch 19, I am getting maximum accuracy of 100% so the algorithm saved checkpoint to use configuration as the best model.

## Data Preprocessing

SageMaker Image Classification Algorithm uses Apache MXNet RecordIO format to read the training image and identify the class type of the image. To build a *RecordIO* format file I need to create an *LST* format file to set the class type of the particular image. The following is the LST format file structure.

### LST File Structure

The LST file is like metadata of the training images. CSV file is comma-separated file, as like the LST file is tab-separated file. LST file contains 3 columns, that means two tabs per row. The first column is an index of the file. The Index should be unique to the entire LST file. The second column is for the image class index and the last or 3rd column is the path of the image.

### Example

Following is the part of the training dataset LST file.

```
1730    1.000000        PNEUMONIA/person1238_bacteria_3194.jpeg
2062    1.000000        PNEUMONIA/person1375_virus_2366.jpeg
1688    1.000000        PNEUMONIA/person1214_virus_2059.jpeg
3375    1.000000        PNEUMONIA/person347_bacteria_1599.jpeg
2841    1.000000        PNEUMONIA/person1888_bacteria_4775.jpeg
750     0.000000        NORMAL/NORMAL2-IM-0637-0001.jpeg
3095    1.000000        PNEUMONIA/person281_bacteria_1326.jpeg
3137    1.000000        PNEUMONIA/person294_bacteria_1386.jpeg
336     0.000000        NORMAL/IM-0533-0001-0001.jpeg
1273    0.000000        NORMAL/NORMAL2-IM-1396-0001.jpeg
4507    1.000000        PNEUMONIA/person656_virus_1238.jpeg
4215    1.000000        PNEUMONIA/person563_bacteria_2337.jpeg
2426    1.000000        PNEUMONIA/person1527_bacteria_3990.jpeg
949     0.000000        NORMAL/NORMAL2-IM-0947-0001.jpeg
892     0.000000        NORMAL/NORMAL2-IM-0873-0001.jpeg
4893    1.000000        PNEUMONIA/person881_bacteria_2805.jpeg
2824    1.000000        PNEUMONIA/person1865_bacteria_4737.jpeg
3228    1.000000        PNEUMONIA/person313_bacteria_1458.jpeg
881     0.000000        NORMAL/NORMAL2-IM-0860-0001.jpeg
4705    1.000000        PNEUMONIA/person75_bacteria_367.jpeg
4508    1.000000        PNEUMONIA/person657_bacteria_2549.jpeg
2846    1.000000        PNEUMONIA/person1901_bacteria_4795.jpeg
1328    1.000000        PNEUMONIA/person1023_bacteria_2954.jpeg
4424    1.000000        PNEUMONIA/person621_virus_1195.jpeg
649     0.000000        NORMAL/NORMAL2-IM-0487-0001.jpeg
1445    1.000000        PNEUMONIA/person1089_virus_1808.jpeg
2337    1.000000        PNEUMONIA/person1491_bacteria_3893.jpeg
3509    1.000000        PNEUMONIA/person380_virus_763.jpeg
2980    1.000000        PNEUMONIA/person258_virus_539.jpeg
304     0.000000        NORMAL/IM-0509-0001-0002.jpeg
2660    1.000000        PNEUMONIA/person1651_bacteria_4381.jpeg
2550    1.000000        PNEUMONIA/person1592_bacteria_4178.jpeg
524     0.000000        NORMAL/IM-0703-0001.jpeg
1285    0.000000        NORMAL/NORMAL2-IM-1437-0001.jpeg
3972    1.000000        PNEUMONIA/person501_bacteria_2112.jpeg
2804    1.000000        PNEUMONIA/person1819_bacteria_4677.jpeg
5008    1.000000        PNEUMONIA/person947_bacteria_2872.jpeg
1531    1.000000        PNEUMONIA/person1138_virus_1879.jpeg
```

As you can see the first column is an index that is unique to the entire LST file index, the second column is the index of the class type of the particular image, and 3rd is the relative path of the image from the training dataset.

Let's consider the 1st row of the sample LST file, 1730 is the unique index among LST file, 1.0 is the class type of the image. As you notice, the 1.0 class index type means the specific image is belong to the Pneumonia chest x-ray image and the 0.0 class index type means the specific image belongs to a normal chest x-ray image.

Now, you might have a question is it any way to create an LST file based on our file structure of the training data? The answer is yes, I found the library which will scan our training data folder and create an LST file. You can get this library from Apache GitHub.

```
!wget 'https://raw.githubusercontent.com/apache/incubator-mxnet/master/tools/im2rec.py'
```

```
--2020-09-18 11:32:36--  https://raw.githubusercontent.com/apache/incubator-mxnet/master/tools/im2rec.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 199.232.64.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|199.232.64.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15838 (15K) [text/plain]
Saving to: 'im2rec.py'

im2rec.py           100%[===================>]  15.47K  --.-KB/s    in 0.001s

2020-09-18 11:32:37 (26.6 MB/s) - 'im2rec.py' saved [15838/15838]
```

### *How we use im2rec library?*

We can create the LST file from the following command:

```
%%bash

python im2rec.py --list --recursive chest_xray/train chest_xray/train/
```

```
NORMAL 0
PNEUMONIA 1
```

As you can see, we are giving our training dataset path and the library itself figure out that the train dataset has 2 types of classes based on the file structure of the training dataset. This library also assigns the class index type: 0 for the normal chest X-ray image and 1 for the Pneumonia chest X-ray image.

The same way we are going to create the LST files for the test and validation dataset:

```
%%bash

python im2rec.py --list --recursive chest_xray/validation chest_xray/validation/
```

```
NORMAL 0
PNEUMONIA 1
```

```
%%bash

python im2rec.py --list --recursive chest_xray/test chest_xray/test/
```

```
NORMAL 0
PNEUMONIA 1
```

## Creating of RecordIO

SageMaker image classification algorithm uses Apache MXNet RecordIO file as a building image classification model. The same im2rec library is helping us to create a RecordIO file based on the LST file of the dataset.

After creating the validation set, we have 5112 images as a training dataset. RecordIO file does contain all 5112 images into a single file, so it will easy to send to a training instance from the S3 bucket.

### Steps to convert RecordIO file

Let's convert our training dataset into RecordIO file:

```
%%bash
cd chest_xray/train/
python3 ../../im2rec.py --pack-label ../train.lst .
```

```
Creating .rec file from /home/ec2-user/SageMaker/pneumonia-detection/chest_xray/train.lst in /home/ec2-user/SageMaker/pneumoni
a-detection/chest_xray
multiprocessing not available, fall back to single threaded encoding
time: 0.17641782760620117  count: 0
time: 154.4530484676361  count: 1000
time: 103.63465118408203  count: 2000
time: 90.44755506515503  count: 3000
time: 74.69310426712036  count: 4000
time: 68.39177823066711  count: 5000
```

The same way we have to convert our test and validation dataset into RecordIO file format:

```
%%bash
cd chest_xray/validation/
python3 ../../im2rec.py --pack-label ../validation.lst .
```

```
Creating .rec file from /home/ec2-user/SageMaker/pneumonia-detection/chest_xray/validation.lst in /home/ec2-user/SageMaker/pne
umonia-detection/chest_xray
multiprocessing not available, fall back to single threaded encoding
time: 0.058873891830444336  count: 0
```

```
%%bash
cd chest_xray/test/
python3 ../../im2rec.py --pack-label ../test.lst .
```

```
Creating .rec file from /home/ec2-user/SageMaker/pneumonia-detection/chest_xray/test.lst in /home/ec2-user/SageMaker/pneumonia
-detection/chest_xray
multiprocessing not available, fall back to single threaded encoding
time: 0.041243791580200195  count: 0
```

## Train and Validation Dataset Upload to S3 Bucket

In order to use the SageMaker Image Classification algorithm, I need to upload the training and validation dataset RecordIO file to a different folder in the Amazon S3 bucket.

First, I need to define the S3 bucket and folder for the training and validation dataset upload location and after I need to move the training and validation dataset RecordIO file from the notebook instance to the Amazon S3 bucket.

```
%%time

import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)

session = sagemaker.Session()
default_bucket = session.default_bucket()

prefix = 'pneumonia-detection'

s3train = 's3://{}/{}/train/'.format(default_bucket, prefix)
s3validation = 's3://{}/{}/validation/'.format(default_bucket, prefix)
```

```
arn:aws:iam::929462117807:role/service-role/AmazonSageMaker-ExecutionRole-20200723T233556
CPU times: user 260 ms, sys: 20.3 ms, total: 281 ms
Wall time: 2.62 s
```

```
# upload the image files to train and validation channels
!aws s3 cp chest_xray/train.rec $s3train --quiet
!aws s3 cp chest_xray/validation.rec $s3validation --quiet
```

## Model Train

The model training part requires input data or training data to build the model. In an earlier step, we prepare our input data which is compatible with the SageMaker image classification algorithm.

Let's create the estimator object:

```
s3_output_location = 's3://{}/{}/output'.format(default_bucket, prefix)
estimator = sagemaker.estimator.Estimator(training_image,
                                          role,
                                          train_instance_count=1,
                                          train_instance_type='ml.p2.xlarge',
                                          train_volume_size = 3,
                                          train_max_run = 7200,
                                          input_mode= 'File',
                                          output_path=s3_output_location,
                                          sagemaker_session=session)
```

```
Parameter image_name will be renamed to image_uri in SageMaker Python SDK v2.
```

The *train_volumn_size* is the size of training data in GB.

### Hyperparameter Tuning

Tuning hyperparameter is giving us the best model from the training and validation dataset. As a part of hyperparameter tuning we are setting up the image height and width, number of RXNet layers, number of classes or types of the training image, minimum batch size to train the model, number of an epoch, learning rate, optimizer, number of the training sample, multi-classification or not.

```
estimator.set_hyperparameters(num_layers=18,
                              use_pretrained_model=1,
                              image_shape = "3,224,224",
                              num_classes=2,
                              mini_batch_size=128,
                              resize=256,
                              epochs=25,
                              learning_rate=0.0005,
                              optimizer='adam',
                              num_training_samples=5112,
                              augmentation_type = 'crop_color_transform',
                              precision_dtype='float32',
                              multi_label = 0)
```

The interesting thing is the *augmentation_type* hyperparameter, as you know X-ray images are always in black and white image format. If something you see in color, it's might be X-ray software add it or someone might temper image for the internal purpose. So, we can skip that part, SageMaker provides this functionality by setting up ***crop_color_transformation*** to *augmentation_type* hyperparameter.

## Set the input and Validation Channel for the Model Training

```
train_data = sagemaker.session.s3_input(s3train, distribution='FullyReplicated',
                                        content_type='application/x-recordio',
                                        s3_data_type='S3Prefix')
validation_data = sagemaker.session.s3_input(s3validation, distribution='FullyReplicated',
                                        content_type='application/x-recordio',
                                        s3_data_type='S3Prefix')

data_channels = {'train': train_data,
                 'validation': validation_data}
```

```
's3_input' class will be renamed to 'TrainingInput' in SageMaker Python SDK v2.
's3_input' class will be renamed to 'TrainingInput' in SageMaker Python SDK v2.
```

## Build Model

```
%%time

estimator.fit(inputs=data_channels, logs=True)
```

```
2020-09-22 03:05:15 Starting - Starting the training job...
2020-09-22 03:05:18 Starting - Launching requested ML instances......
2020-09-22 03:06:30 Starting - Preparing the instances for training.........
2020-09-22 03:08:04 Downloading - Downloading input data.........
2020-09-22 03:09:33 Training - Downloading the training image..Docker entrypoint called with argument(s): train
[09/22/2020 03:10:01 INFO 140020406769472] Reading default configuration from /opt/amazon/lib/python2.7/site-packages/image_cl
assification/default-input.json: {u'beta_1': 0.9, u'gamma': 0.9, u'beta_2': 0.999, u'optimizer': u'sgd', u'use_pretrained_mode
l': 0, u'eps': 1e-08, u'epochs': 30, u'lr_scheduler_factor': 0.1, u'num_layers': 152, u'image_shape': u'3,224,224', u'precisio
n_dtype': u'float32', u'mini_batch_size': 32, u'weight_decay': 0.0001, u'learning_rate': 0.1, u'momentum': 0}
[09/22/2020 03:10:01 INFO 140020406769472] Merging with provided configuration from /opt/ml/input/config/hyperparameters.json:
{u'learning_rate': u'0.0005', u'optimizer': u'adam', u'image_shape': u'3,224,224', u'num_layers': u'18', u'epochs': u'25', u'r
esize': u'256', u'augmentation_type': u'crop_color_transform', u'multi_label': u'0', u'precision_dtype': u'float32', u'mini_ba
tch_size': u'128', u'use_pretrained_model': u'1', u'num_classes': u'2', u'num_training_samples': u'5112'}
[09/22/2020 03:10:01 INFO 140020406769472] Final configuration: {u'optimizer': u'adam', u'learning_rate': u'0.0005', u'multi_l
abel': u'0', u'epochs': u'25', u'lr_scheduler_factor': 0.1, u'num_layers': u'18', u'num_classes': u'2', u'precision_dtype':
u'float32', u'mini_batch_size': u'128', u'resize': u'256', u'beta_1': 0.9, u'beta_2': 0.999, u'use_pretrained_model': u'1',
u'eps': 1e-08, u'augmentation_type': u'crop_color_transform', u'weight_decay': 0.0001, u'momentum': 0, u'image_shape': u'3,22
4,224', u'gamma': 0.9, u'num_training_samples': u'5112'}
```

## Test the Model

Calculating accuracy, precision, recall, and other performance metrics of the model is very important in Machine learning. It gives confidence to how well a machine learning model performs. The test dataset is useful to derive all the performance metrics including accuracy, as the model never seen these data.

SageMaker provides two ways to predict the output from the model: Batch Transformation also known as offline prediction and runtime prediction also known as the online prediction. Batch transformation is a one-time job, which takes multiple inputs at the same time and predicts output in some size of the batch and gives the result back of all the input at the same time.

I am going to send all test images to the image classification model for the prediction of pneumonia in the chest x-ray. Machine Learning model classifies all the test image in batch transformation mode and gives the output of all the image and then we can compare with the actual ground truth. To accommodate this, I am going to divide batch transformation into 2 parts. In the first part, I am going to send all the images which actually belong to the normal person's chest X-ray and set the test label to 0. In the 2nd phase, I am going to send all the images, which belong to pneumonia infected person's chest X-ray and set their test labels to 1.

### Batch Transformation

SageMaker batch transformation needs input data into the Amazon S3 bucket. As I am going to divide batch transformation into 2 parts, I am uploading normal person images to a test-normal folder and pneumonia person images to the pneumonia-test folder.

```
%%time

role = get_execution_role()
print(role)

session = sagemaker.Session()
default_bucket = session.default_bucket()

prefix = 'pneumonia-detection'

s3_normal_test_location = 's3://{}/{}/test-normal/'.format(default_bucket, prefix)
s3_normal_out = 's3://{}/{}/batch-normal-out/'.format(default_bucket, prefix)

s3_pneumonia_test_location = 's3://{}/{}/test-pneumonia/'.format(default_bucket, prefix)
s3_pneumonia_out = 's3://{}/{}/batch-pneumonia-out/'.format(default_bucket, prefix)

!aws s3 cp chest_xray/test/NORMAL/ $s3_normal_test_location --recursive --quiet
!aws s3 cp chest_xray/test/PNEUMONIA/ $s3_pneumonia_test_location --recursive --quiet
```

```
arn:aws:iam::929462117807:role/service-role/AmazonSageMaker-ExecutionRole-20200723T233556
CPU times: user 963 ms, sys: 76.1 ms, total: 1.04 s
Wall time: 12.1 s
```

## Define Low-Level Batch Transformation

SageMaker Low-level batch transformation needs a unique name for the transformation job, input data source, model name, maximum concurrent transform, batch strategy, output S3 folder, content type, instance type, instance count and etc.

```
%%time
import time
from time import gmtime, strftime

timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
batch_job_name = "image-classification-model" + timestamp
request = \
{
    "TransformJobName": batch_job_name,
    "ModelName": "image-classification-2020-09-22-03-05-15-236",
    "MaxConcurrentTransforms": 16,
    "MaxPayloadInMB": 6,
    "BatchStrategy": "SingleRecord",
    "TransformOutput": {
        "S3OutputPath": s3_normal_out
    },
    "TransformInput": {
        "DataSource": {
            "S3DataSource": {
                "S3DataType": "S3Prefix",
                "S3Uri": s3_normal_test_location
            }
        },
        "ContentType": "application/x-image",
        "SplitType": "None",
        "CompressionType": "None"
    },
    "TransformResources": {
            "InstanceType": "ml.p2.xlarge",
            "InstanceCount": 1
    }
}

print('Transform job name: {}'.format(batch_job_name))
print('\nInput Data Location: {}'.format(s3_normal_test_location))
```

```
Transform job name: image-classification-model-2020-09-22-23-57-35

Input Data Location: s3://sagemaker-us-east-1-929462117807/pneumonia-detection/test-normal/
CPU times: user 155 µs, sys: 13 µs, total: 168 µs
Wall time: 135 µs
```

After defining the batch transformation job, need to start the prediction job.

```
sagemaker = boto3.client('sagemaker')
sagemaker.create_transform_job(**request)

print("Created Transform job with name: ", batch_job_name)

while(True):
    response = sagemaker.describe_transform_job(TransformJobName=batch_job_name)
    status = response['TransformJobStatus']
    if status == 'Completed':
        print("Transform job ended with status: " + status)
        break
    if status == 'Failed':
        message = response['FailureReason']
        print('Transform failed with the following error: {}'.format(message))
        raise Exception('Transform job failed')
    time.sleep(30)
```

```
Created Transform job with name:  image-classification-model-2020-09-23-00-08-00
Transform job ended with status: Completed
```

After the transformation job completes, it writes output data to a specific location on the Amazon S3 bucket. To work with the output data, I need to copy that data from S3 to notebook instance memory.

```
!mkdir chest_xray/out
```

```
!mkdir chest_xray/out/normal
```

```
!aws s3 cp --recursive $s3_normal_out chest_xray/out/normal/
```

```
download: s3://sagemaker-us-east-1-929462117807/pneumonia-detection/batch-normal-out/IM-0001-0001.jpeg.out to chest_xray/out/n
ormal/IM-0001-0001.jpeg.out
download: s3://sagemaker-us-east-1-929462117807/pneumonia-detection/batch-normal-out/IM-0005-0001.jpeg.out to chest_xray/out/n
ormal/IM-0005-0001.jpeg.out
download: s3://sagemaker-us-east-1-929462117807/pneumonia-detection/batch-normal-out/IM-0010-0001.jpeg.out to chest_xray/out/n
ormal/IM-0010-0001.jpeg.out
download: s3://sagemaker-us-east-1-929462117807/pneumonia-detection/batch-normal-out/IM-0011-0001.jpeg.out to chest_xray/out/n
ormal/IM-0011-0001.jpeg.out
```

As you can see from the output, the transformation job creates .out file to write the prediction for each image individually because mentioned into the batch strategy in low-level transformation job define.

Each output file contains 2 element 1-D array. This array contains the probability of Image class and the image class index is the array index. So, the $0^{th}$ index element of the array has the probability of a normal person's chest X-ray, and the $1^{st}$ index element of the array has probability of the pneumonia infected person's chest X-ray.

Following is one of the batch transformation job output:

```
import numpy as np
import pandas as pd
import os


with open("chest_xray/out/normal/NORMAL2-IM-0366-0001.jpeg.out") as f:
        data = json.load(f)
        print(data['prediction'])
```

```
[0.999991774559021, 8.17106592876371e-06]
```

As you can see, the $0^{th}$ index value of an array is near to 1, which is the probability of a particular image is belonging to a normal person's chest x-ray.

Convert our understanding into the code, I create a function named *accuracy_prediction,* which takes input as an output image array and gives whether the image belongs to pneumonia infected person's chest X-ray or not.

```
def accuracy_prediction(result):
    if((result[1] > result[0]) and (result[1] > 0.50)):
        return 1
    elif((result[0] > result[1]) and (result[0] > 0.50)):
        return 0
    else:
        return -1
```

## Calculating Metrics

The following python function helps us to gives the model performance metrics, which include true positive, true negative, false positive, false negative, precision, recall, and accuracy.

```
def evaluate(batch_out_dir, verbose=True):
    test_labels = []
    pred_labels = []
    normal_expected_dir = batch_out_dir + "normal/"
    pneumonia_expected_dir = batch_out_dir + "pneumonia/"

    # Working on Normal Expected X-Ray Image
    for file in os.listdir(normal_expected_dir):
        with open(os.path.join(normal_expected_dir, file)) as f:
            data = json.load(f)
            pred_labels.append(accuracy_prediction(data['prediction']))
            test_labels.append(0)

    # Working on Pneumonia Expected X-Ray Image
    for file in os.listdir(pneumonia_expected_dir):
        with open(os.path.join(pneumonia_expected_dir, file)) as f:
            data = json.load(f)
            pred_labels.append(accuracy_prediction(data['prediction']))
            test_labels.append(1)

    test_labels = np.array(test_labels)
    test_preds = np.array(pred_labels)

    # calculate true positives, false positives, true negatives, false negatives
    tp = np.logical_and(test_labels, test_preds).sum()
    fp = np.logical_and(1-test_labels, test_preds).sum()
    tn = np.logical_and(1-test_labels, 1-test_preds).sum()
    fn = np.logical_and(test_labels, 1-test_preds).sum()

    # calculate binary classification metrics
    recall = tp / (tp + fn)
    precision = tp / (tp + fp)
    accuracy = (tp + tn) / (tp + fp + tn + fn)

    # printing a table of metrics
    if verbose:
        print(pd.crosstab(test_labels, test_preds, rownames=['actual (row)'], colnames=['prediction (col)']))
        print("\n{:<11} {:.3f}".format('Recall:', recall))
        print("{:<11} {:.3f}".format('Precision:', precision))
        print("{:<11} {:.3f}".format('Accuracy:', accuracy))
        print()

    return {'TP': tp, 'FP': fp, 'FN': fn, 'TN': tn,
            'Precision': precision, 'Recall': recall, 'Accuracy': accuracy}
```

## Results

The following the result, we get from the test data prediction:

```
print('Metrics for Image Classifier.\n')

# get metrics for linear predictor
metrics = evaluate('chest_xray/out/',
                   verbose=True) # verbose means we'll print out the metrics
```

```
Metrics for Image Classifier.

prediction (col)    0    1
actual (row)
0                 232    2
1                 148  242

Recall:     0.621
Precision:  0.992
Accuracy:   0.760
```

As you notice the output, I got 99% precision, with 76% overall accuracy, based on that radiologist confidently tell the result for the normal person's chest X-ray without much investigation. This machine learning model might save a lot of radiologist time.

## Build Web Application

A Radiologist is not a software engineer to run the python notebook and send an image through python notebook. So, now let's build a web application that will easy to use for any non-technical person. The web application has upload image button or drag and drops for the image. The uploaded image goes to the machine learning model and the model predicts the output and the website will show the result.

For the real-prediction, I use SageMaker inference endpoint functionality. This feature provides an endpoint for the AWS tools to upload the image and predict the image class and gives the result back to the client over the network.

### Deploy the Model

```
estimator = sagemaker.estimator.Estimator.attach('image-classification-2020-09-22-03-05-15-236')

classifier = estimator.deploy(initial_instance_count = 1,
                              instance_type = 'ml.t2.medium')
```

```
[09/22/2020 03:36:27 INFO 140020406769472] Epoch[21] Validation-accuracy=0.984375
[09/22/2020 03:37:04 INFO 140020406769472] Epoch[22] Batch [20]#011Speed: 69.944 samples/sec#011a
[09/22/2020 03:37:37 INFO 140020406769472] Epoch[22] Train-accuracy=0.992989
[09/22/2020 03:37:37 INFO 140020406769472] Epoch[22] Time cost=69.132
[09/22/2020 03:37:39 INFO 140020406769472] Epoch[22] Validation-accuracy=0.968750
[09/22/2020 03:38:16 INFO 140020406769472] Epoch[23] Batch [20]#011Speed: 69.544 samples/sec#011a
[09/22/2020 03:38:49 INFO 140020406769472] Epoch[23] Train-accuracy=0.992989
[09/22/2020 03:38:49 INFO 140020406769472] Epoch[23] Time cost=69.032
[09/22/2020 03:38:51 INFO 140020406769472] Epoch[23] Validation-accuracy=0.984375
[09/22/2020 03:39:28 INFO 140020406769472] Epoch[24] Batch [20]#011Speed: 70.240 samples/sec#011a
[09/22/2020 03:40:00 INFO 140020406769472] Epoch[24] Train-accuracy=0.993590
[09/22/2020 03:40:00 INFO 140020406769472] Epoch[24] Time cost=68.842
[09/22/2020 03:40:02 INFO 140020406769472] Epoch[24] Validation-accuracy=0.992188
Training seconds: 1944
Billable seconds: 1944
Parameter image will be renamed to image_uri in SageMaker Python SDK v2.
Using already existing model: image-classification-2020-09-22-03-05-15-236

----------------------!
```

After deploying the model, it's time to test the output and convert results probability into human understand language.

```
pneumonia_sample_image_payload = create_payload_from_image(pneumonia_sample_image)
normal_sample_image_payload = create_payload_from_image(normal_sample_image)
```

**Send Sample Pneumonia Detected Person's Chest X-Ray image and see the prediction**

```
classifier.content_type = 'application/x-image'
pneumonia_image_result = json.loads(classifier.predict(pneumonia_sample_image_payload))
```

```
pneumonia_image_result
```

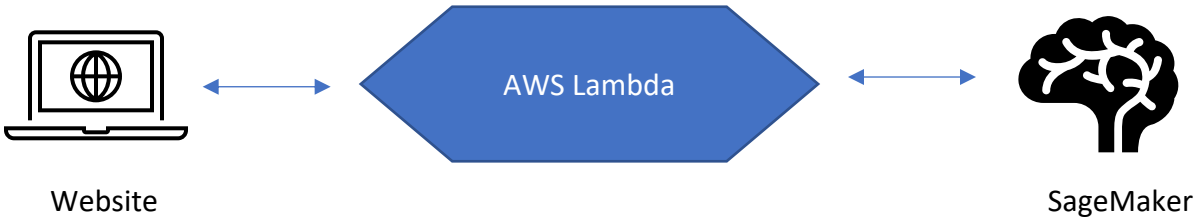
```
[0.00032388418912887573, 0.9996761083602905]
```

```
prediction(pneumonia_image_result)
```

```
'Pneumonia'
```

## Build REST Endpoint

The website sends the image in POST request to the model, but the deployed model is not working outside of the AWS world. AWS Lambda is the bridge between handling website form POST request and converts the request to such a way that SageMaker deployed model can understand and predicts the image class or pneumonia prediction.



## AWS Lambda Code:

Following is the AWS Lambda code, which will act as a bridge between REST Http Call to SageMaker Inference Endpoint.

```
import json
import boto3
from cgi import parse_header, parse_multipart, FieldStorage
from io import BytesIO
import base64


def create_payload_from_image(image):
    payload = bytearray(image)
    return payload

def prediction(result):
    if((result[1] > result[0]) and (result[1] > 0.50)):
        return 'Pneumonia'
    elif((result[0] > result[1]) and (result[0] > 0.50)):
        return 'Normal'
    else:
        return 'Failed to Predict'

def lambda_handler(event, context):
    c_type, c_data = parse_header(event['headers']['Content-Type'])
    assert c_type == 'multipart/form-data'
    c_data['boundary'] = bytes(c_data['boundary'], "utf-8")
    fp = BytesIO(bytes(event['body'], 'utf-8'))
    form_data = parse_multipart(fp, c_data)

    img_data = form_data["file"][0]
    payload = create_payload_from_image(img_data)

    # Call to the run-time inference endpoint
    runtime = boto3.Session().client('sagemaker-runtime')
    response = runtime.invoke_endpoint(EndpointName = 'image-classification-2020-09-22-03-05-15-236',
                                       ContentType = 'application/x-image',
                                       Body = payload)

    prediction_raw_data = json.loads(response['Body'].read().decode('utf-8'))
    pred = prediction(prediction_raw_data)


    return {
        'statusCode' : 200,
        'headers' : { 'Content-Type' : '*', 'Access-Control-Allow-Origin' : '*' },
        'body' : pred
    }
```

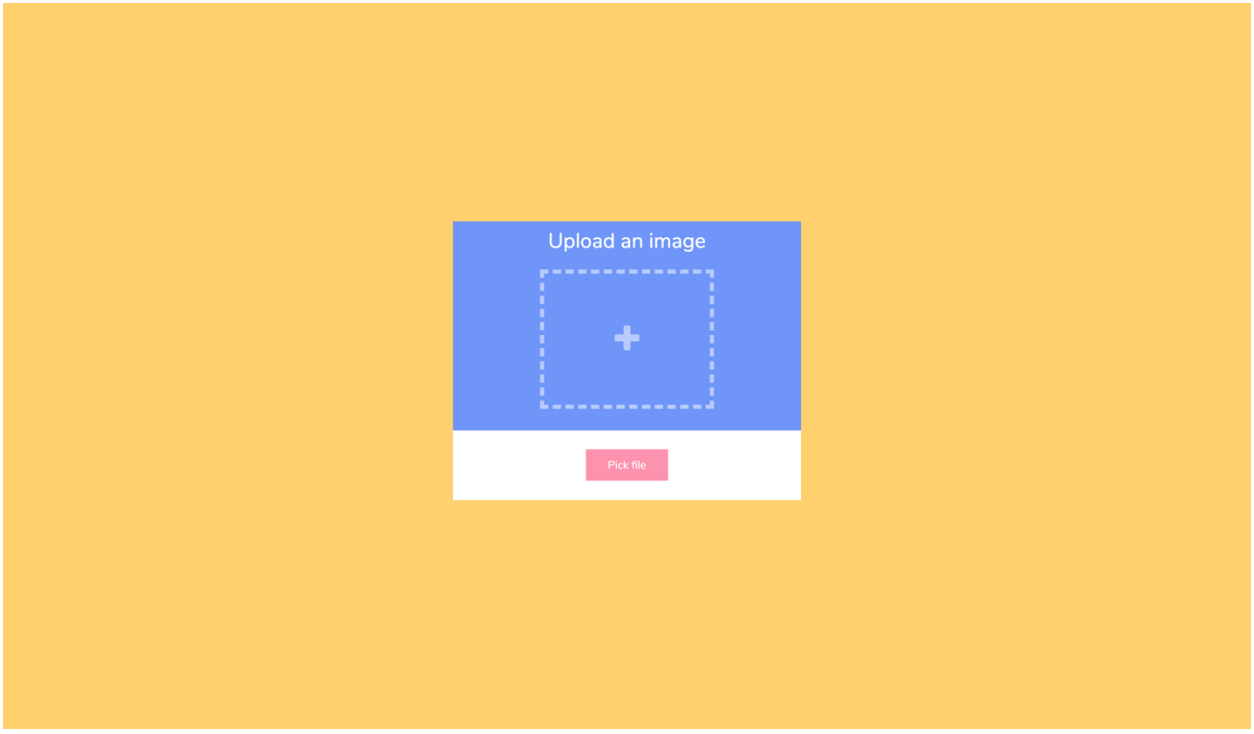
## JavaScript AJAX function

Following is the JavaScript Ajax function, which will call every time a new image is going to upload from the website. This Ajax function sends a POST request to the AWS lambda function endpoint and this endpoint sends the image to the online deployed model, the model predicts the result, and sends back to the Ajax function.

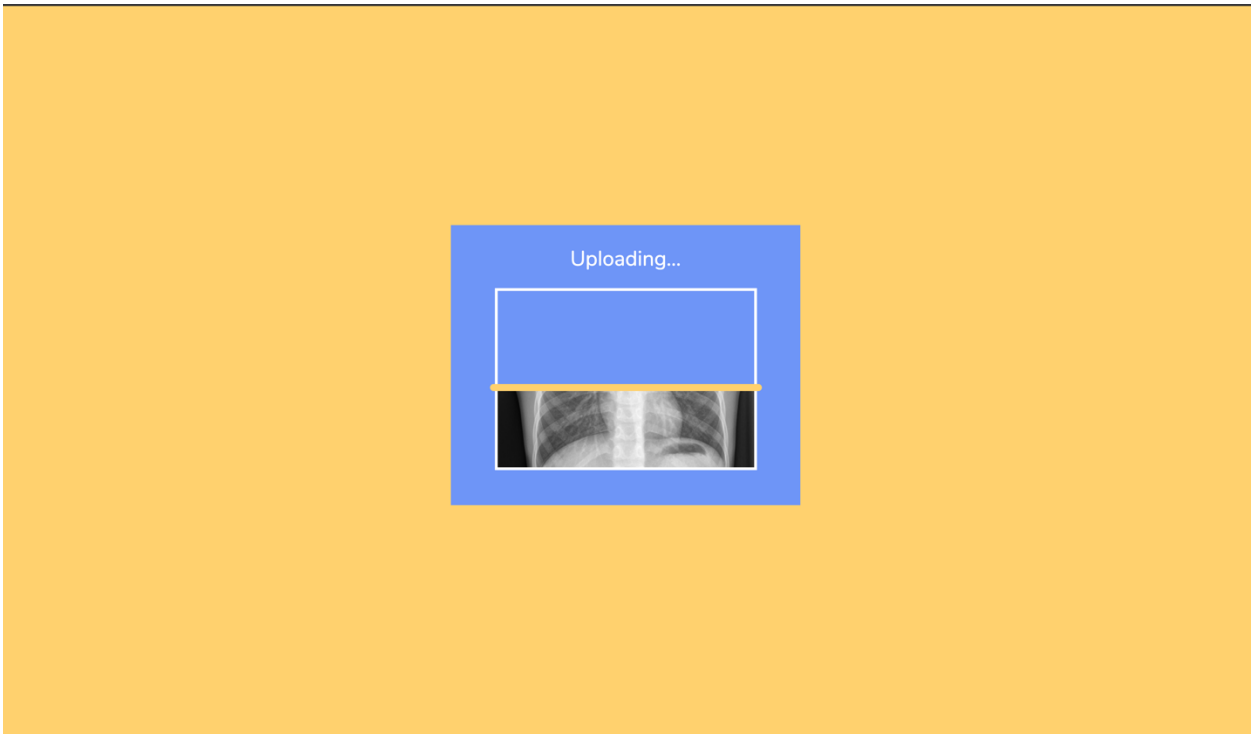
```
function fire_ajax_submit(file) {
        var fd = new FormData();
        fd.append("file", file);

        $.ajax({
            type: "POST",
            enctype: 'multipart/form-data',
            url: "https://u7k24dtheg.execute-api.us-east-1.amazonaws.com/prod",
            data: fd,
            processData: false, //prevent jQuery from automatically transforming the data into a query string
            contentType: false,
            cache: false,
            timeout: 600000,
            success: function (data) {
                if (data === 'Pneumonia') {
                    isNormal = 0;
                }
                else if (data === 'Normal') {
                    isNormal = 1;
                }
                console.log("SUCCESS : ", data);
            },
            error: function (e) {
                console.log("ERROR : ", e);
            }
        });

    }
```

# Demo Screenshots

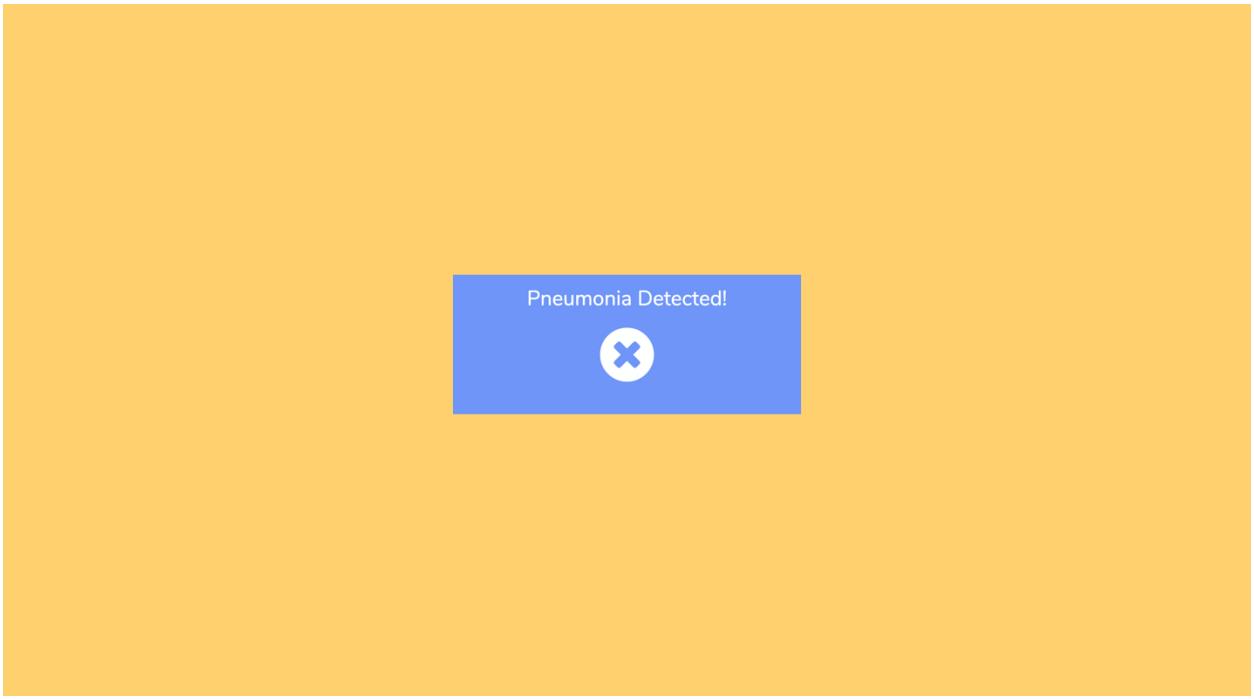
## Home Page



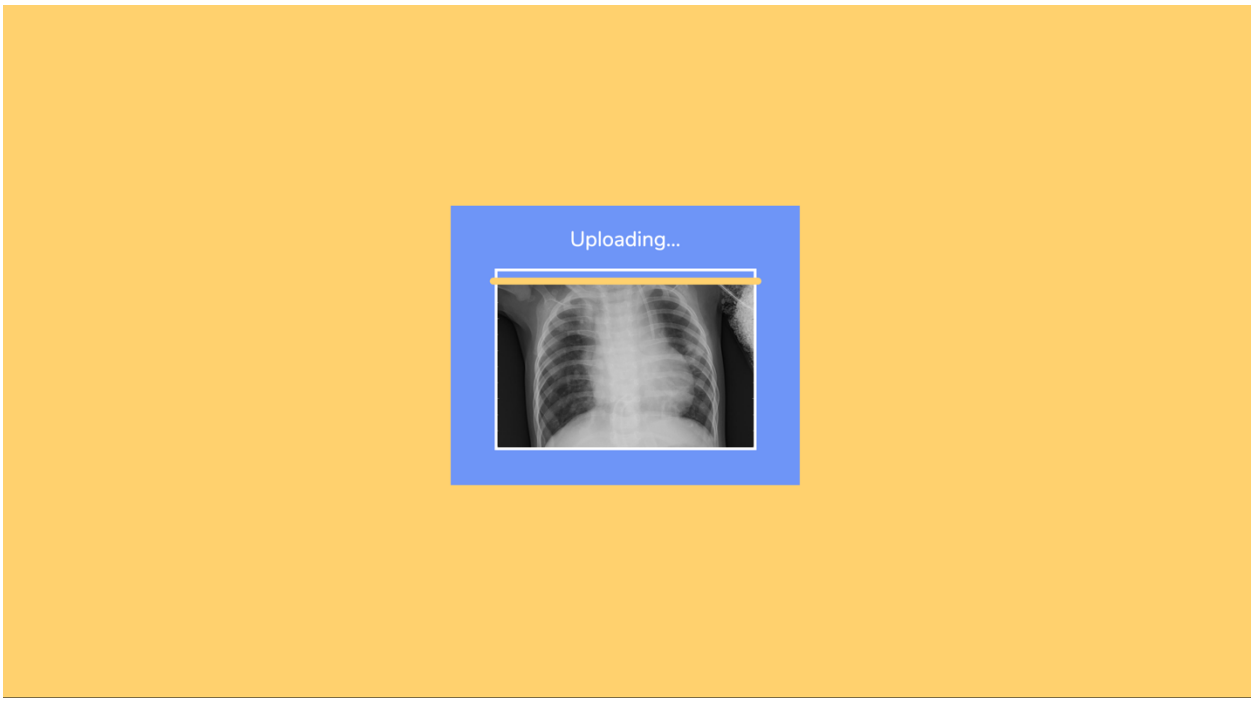

## Uploading



## Prediction Output




## Uploading



## Prediction