Deep Learning - Chest x-ray Examination for Pneumonia

by Mike Yang

Table of Contents

Main objective of analysis

- Business initiative

Pneumonia is a common yet sometimes fatal disease. It affects patient’s lungs and other organs through inflammation and could be vital to young kids and elders without proper and in - time treatment.

The chest x-ray image is the first line diagnostic tool to assist doctors to find out the symptom. In order to provide a more efficient diagnostic process, we want to conduct an analysis to screen chest x-ray images and analyze them through the convolutional neural network (CNN) to filter out potential patients, so that doctors can spend less time at the initial inspection stage and let the model to pinpoint the patients with high risks.

- Expect outcome

The analysis will create a CNN model which is trained by an x-ray image repository that contains both normal and pneumonia chest images. Then the model should be capable of classifying pneumonia images with an acceptable accuracy.

Brief description of the data set and a summary of its attributes

The image repository is downloaded from:

https://www.kaggle.com/jonaspalucibarbosa/chest-x-ray-pneumonia-cnn-transfer-learning/data

There are 5828 images in the image repository. The images are firstly categorized into train and test data sets. Inside the train and test data set images are further divided into “NORMAL” and “PNEUMONIA” subsets. Initially there is no validation data set defined.

Data load, pre - processing and exploration

Firstly let’s try to load the image data.

Access the image files

We use the chest X ray images as the input.

```
test_path = './data/chest_xray/test/'
train_path = './data/chest_xray/train/'
label_normal = 'NORMAL/'
label_pneumonia = 'PNEUMONIA/'
```

Read and explore the image files

```
suffix = '*.jpeg'
normal_train_path = train_path + label_normal + suffix
pneumonia_train_path = train_path + label_pneumonia + suffix
normal_test_path = test_path + label_normal + suffix
pneumonia_test_path = test_path + label_pneumonia + suffix
normal_train_name_list = glob.glob(normal_train_path)
pneumonia_train_name_list = glob.glob(pneumonia_train_path)
normal_test_name_list = glob.glob(normal_test_path)
pneumonia_test_name_list = glob.glob(pneumonia_test_path)
```

The original image data is organized into “train” and “test” sub - folders. Each of them contains “NORMAL” and “PNEUMONIA” sub - folders that contain pre - classified x-ray images of healthy or infected people respectively.

```
# Generate image array

def get_image_array(name_list):
    img_array = []
    for name in name_list:
        img = plt.imread(name)
        img_array.append(img)
    return np.array(img_array, dtype='object')
```

```
normal_train_img = get_image_array(normal_train_name_list)
normal_test_img = get_image_array(normal_test_name_list)
pneumonia_train_img = get_image_array(pneumonia_train_name_list)
pneumonia_test_img = get_image_array(pneumonia_test_name_list)
```

```
print('The normal train image size is {}'.format(normal_train_img.shape[0]))
print('The pneumonia train image size is {}'.format(pneumonia_train_img.shape[0]))
print('The normal test image size is {}'.format(normal_test_img.shape[0]))
print('The pneumonia test image size is {}'.format(pneumonia_test_img.shape[0]))
```

```
The normal train image size is 922
The pneumonia train image size is 2740
The normal test image size is 206
The pneumonia test image size is 390
```

To better prepare the model, I randomly select images from the train data set and move the images to a new folder for validation usage. The validation image list is prepared through train_test_split function with 30% sampling ratio. As a result, the new folder is named “validate” and for sure it also contains “NORMAL” and “PNEUMONIA” sub - folders with associated images.

```
validate_path = './data/chest_xray/validate/'
test_path = './data/chest_xray/test/'
```

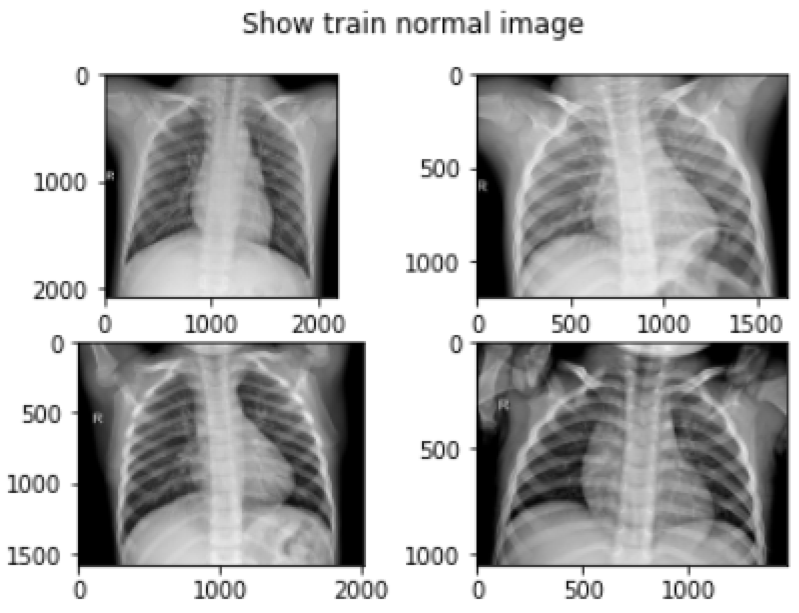
```
# for source in validation_image_set:
#     dest = source.replace('train', 'validate')
#     os.rename(source, dest)

normal_validate_path = validate_path + label_normal + suffix
pneumonia_validate_path = validate_path + label_pneumonia + suffix
normal_validate_name_list = glob.glob(normal_validate_path)
pneumonia_validate_name_list = glob.glob(pneumonia_validate_path)
normal_validate_img = get_image_array(normal_validate_name_list)
pneumonia_validate_img = get_image_array(pneumonia_validate_name_list)
print('The normal validate image size is {}'.format(normal_validate_img.shape[0]))
print('The pneumonia validate image size is {}'.format(pneumonia_validate_img.shape[0]))
```

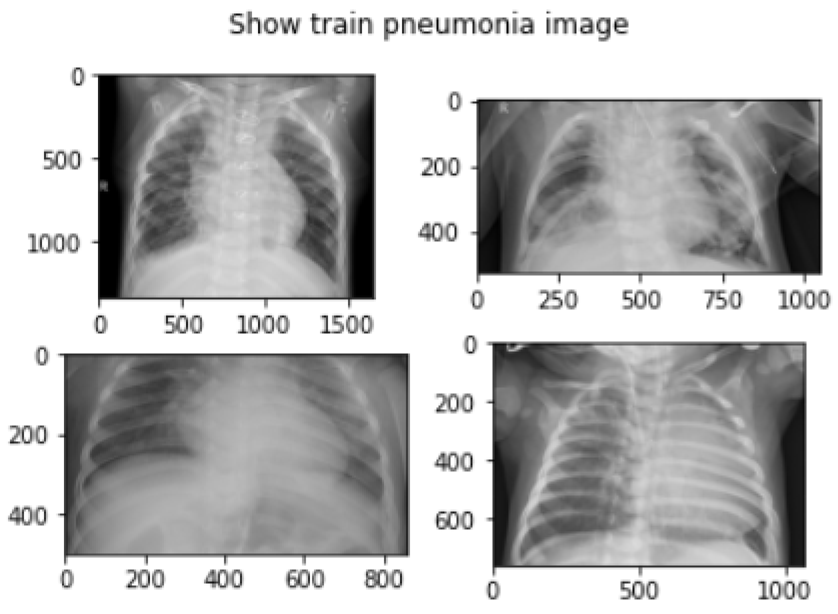
```
The normal validate image size is 427
The pneumonia validate image size is 1143
```

Let’s randomly pick up some training, testing and validation images to explore:

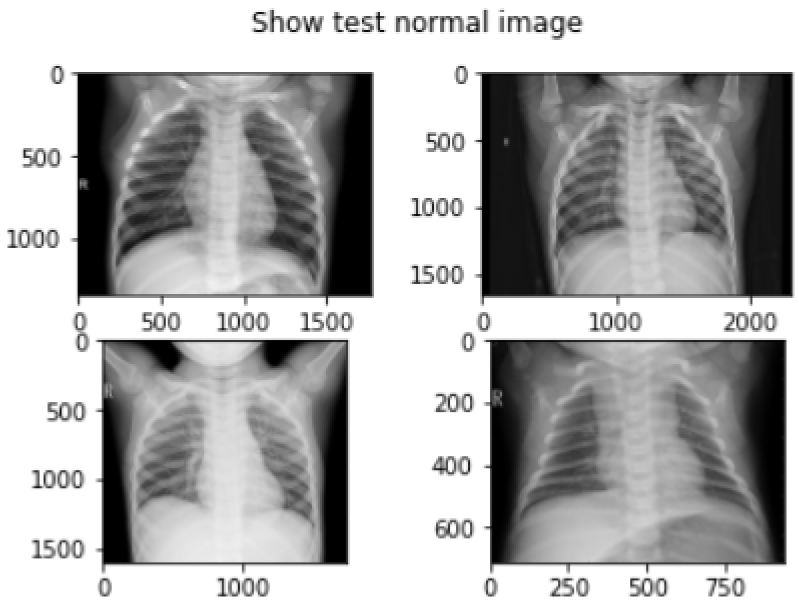
```
def show_sample_image(image_array, title):
    plt.figure()
    plt.suptitle(title)
    for i in range(4):
        plt.subplot(2, 2, i+1)
        plt.imshow(image_array[random.randrange(200)], cmap='gray')

show_sample_image(normal_train_img, 'Show train normal image')
show_sample_image(pneumonia_train_img, 'Show train pneumonia image')

show_sample_image(normal_test_img, 'Show test normal image')
show_sample_image(pneumonia_test_img, 'Show test pneumonia image')

show_sample_image(normal_validate_img, 'Show validation normal image')
show_sample_image(pneumonia_validate_img, 'Show validation pneumonia image')
```

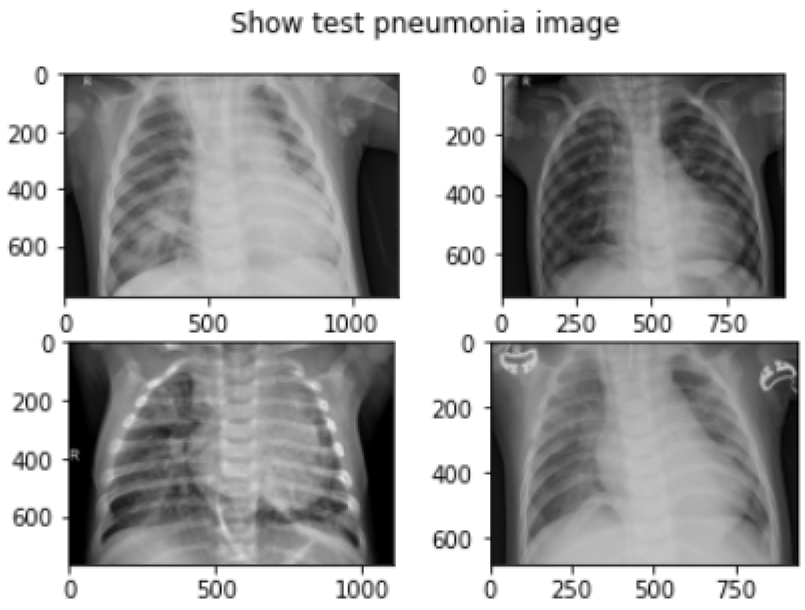
Sample images for training:



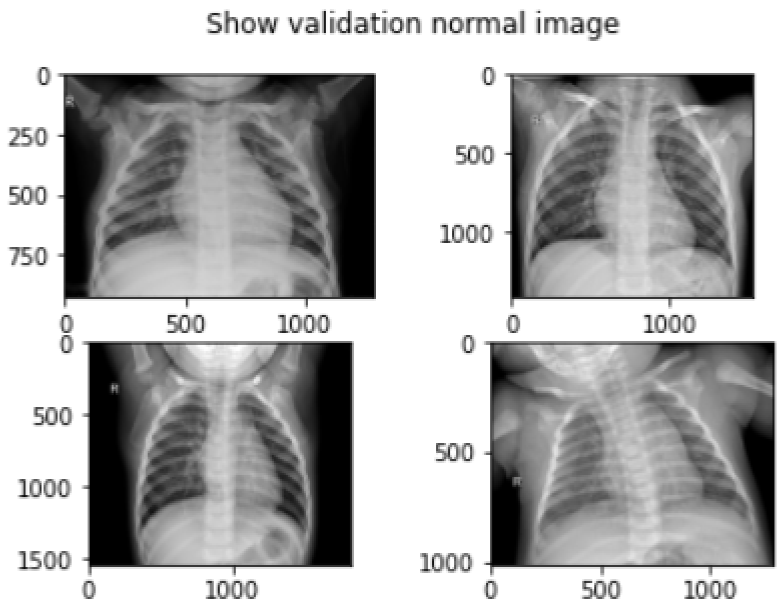


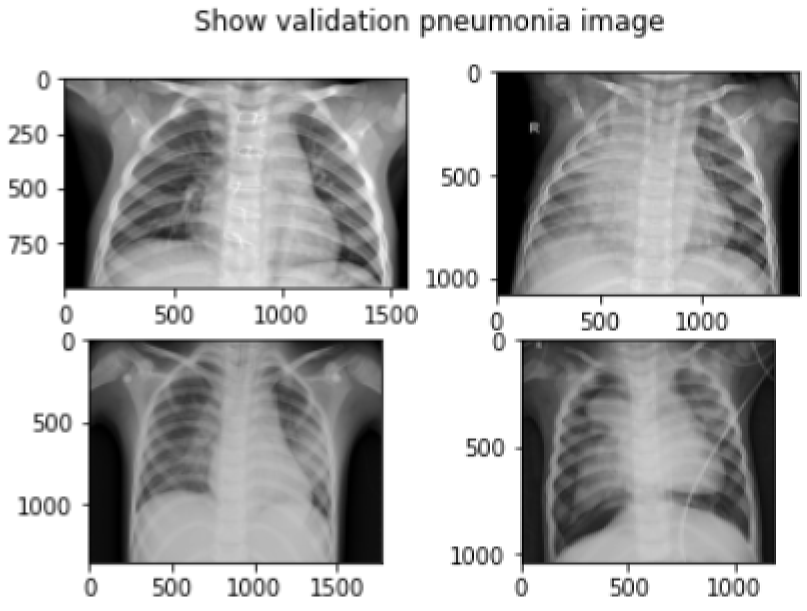
Sample images for testing:





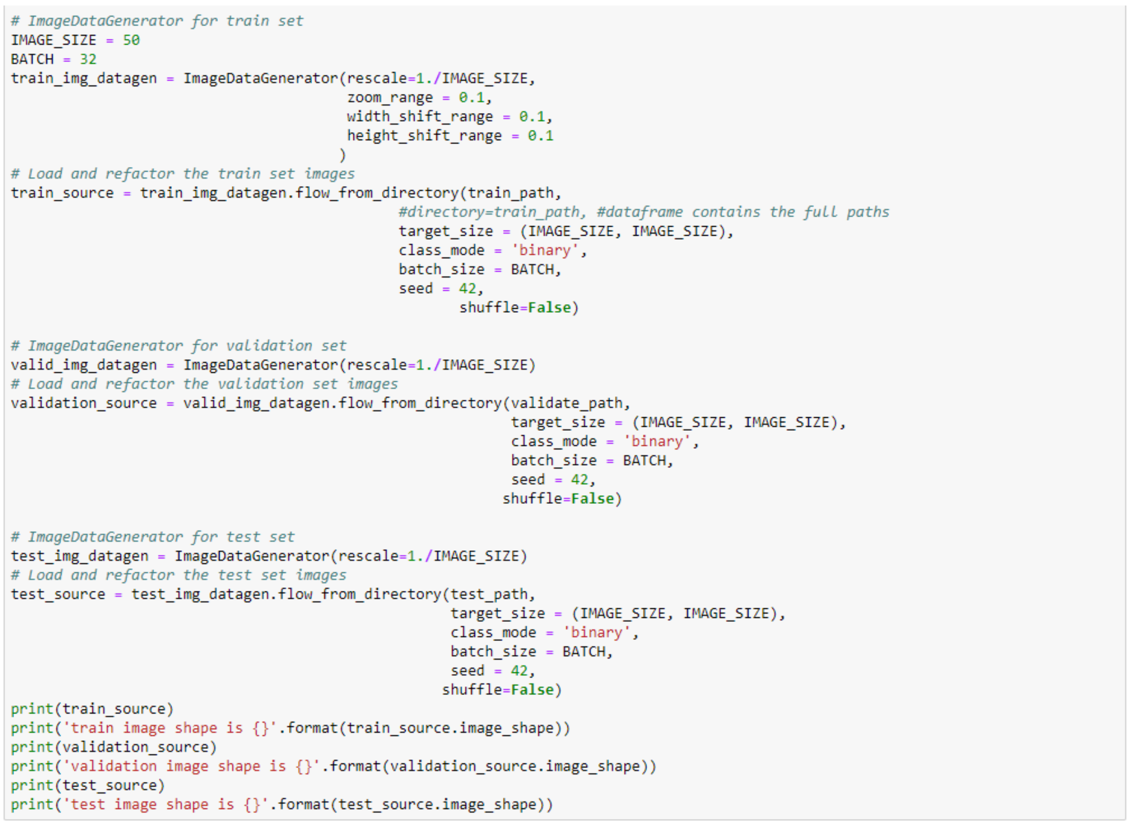
Sample images for validation:





Data screening and preprocessing

From the image exploration in the previous paragraph, it is found that these images are in different sizes. Therefore for better modeling, I am going to resize the images to 50 * 50 through ImageDataGenerator.
In addition to resize the image, to provide more diversity to the train set, the image augmentation is to be introduced with the parameters for image zooming, width and height shifting on the train set:



```
# ImageDataGenerator for train set
IMAGE_SIZE = 50
BATCH = 32
train_img_datagen = ImageDataGenerator(rescale=1./IMAGE_SIZE,
                                       zoom_range = 0.1,
                                       width_shift_range = 0.1,
                                       height_shift_range = 0.1
                                      )
# Load and refactor the train set images
train_source = train_img_datagen.flow_from_directory(train_path,
                                             #directory=train_path, #dataframe contains the full paths
                                             target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                             class_mode = 'binary',
                                             batch_size = BATCH,
                                             seed = 42,
                                                    shuffle=False)

# ImageDataGenerator for validation set
valid_img_datagen = ImageDataGenerator(rescale=1./IMAGE_SIZE)
# Load and refactor the validation set images
validation_source = valid_img_datagen.flow_from_directory(validate_path,
                                                          target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                                          class_mode = 'binary',
                                                          batch_size = BATCH,
                                                          seed = 42,
                                                         shuffle=False)

# ImageDataGenerator for test set
test_img_datagen = ImageDataGenerator(rescale=1./IMAGE_SIZE)
# Load and refactor the test set images
test_source = test_img_datagen.flow_from_directory(test_path,
                                                   target_size = (IMAGE_SIZE, IMAGE_SIZE),
                                                   class_mode = 'binary',
                                                   batch_size = BATCH,
                                                   seed = 42,
                                                  shuffle=False)
print(train_source)
print('train image shape is {}'.format(train_source.image_shape))
print(validation_source)
print('validation image shape is {}'.format(validation_source.image_shape))
print(test_source)
print('test image shape is {}'.format(test_source.image_shape))
```

```
Found 3662 images belonging to 2 classes.
Found 1570 images belonging to 2 classes.
Found 596 images belonging to 2 classes.
<keras.preprocessing.image.DirectoryIterator object at 0x000001D677496390>
train image shape is (50, 50, 3)
<keras.preprocessing.image.DirectoryIterator object at 0x000001D677496F60>
validation image shape is (50, 50, 3)
<keras.preprocessing.image.DirectoryIterator object at 0x000001D6774965C0>
test image shape is (50, 50, 3)
```

Convolutional neural network (CNN) modeling

Here to check again the data set that is going to be analyzed:

- There are 5828 chest x-ray images in total.
- The images are curated into 3 folders, which are “train”, “validate” and “test”. Each of which contains “NORMAL” and “PNEUMONIA” sub - folders as the classification target.
- We split out 30% of train images as the validation set by using the train_test_split function from sklearn.

The image volume of each set is listed as following:

Train	Validate	Test
3662	1570	596

We are going to design a CNN model to classify the chest x-ray images. In addition to regular neural network model components, such as Dense, Activation and Dropout layers, for CNN we also need to introduce some specific layers like Conv2D, Pooling, and Flatten.

The first model

The first model is designed with 3 Conv2D layers, and the design is listed as following:

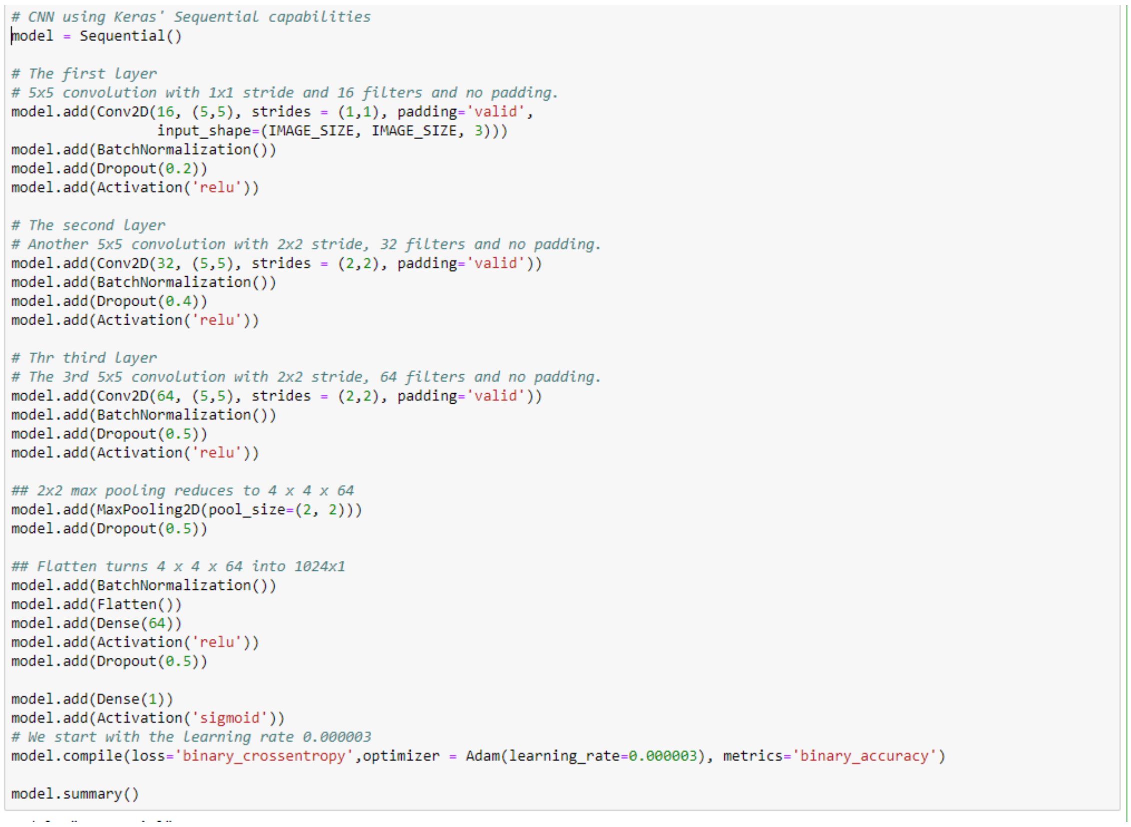
Layer	Filter	Kernal size	Stride	Padding	Dropout
Layer 1	16	5 * 5	1 * 1	valid (no padding)	0.2
Layer 2	32	5 * 5	2 * 2	valid (no padding)	0.4
Layer 3	64	5 * 5	2 * 2	valid (no padding)	0.5

Each Con2D layer is activated by the relu activation layer, and then pooled by MaxPooling2D to reduce the feature numbers so that we can somehow avoid overfitting.

The Dropout layer is set to reduce the internal - reliability and also overfitting during the training process. Here we gradually increase the dropout ratio among the 3 layers (from 0.2 to 0.5).

Then we introduce regular neural network layers to complete the modeling with Flatten and Dense layer come with another Dropout layer. We add the final Dense layer with only 1 unit and activated by sigmoid to get the value between -1 and 1 as the label value.

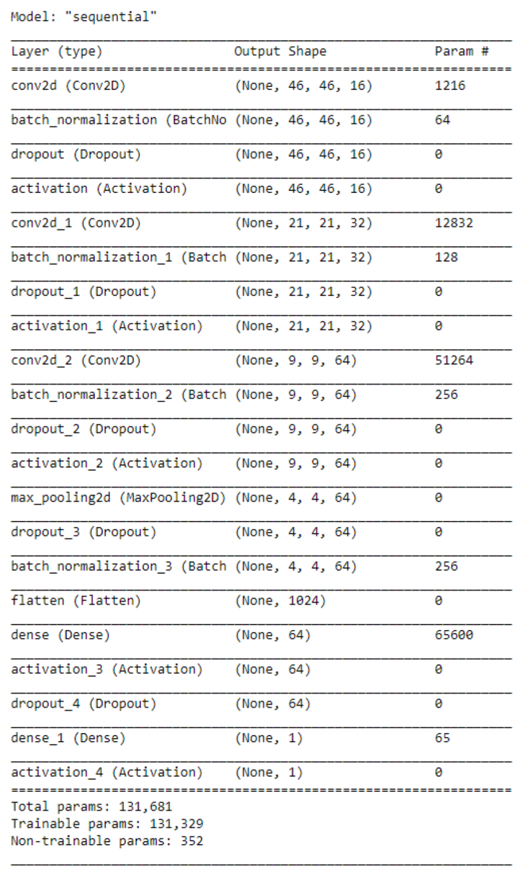
Since we build the model to classify the images, it is going to be optimized by Adam optimizer, compensates the loss by binary_crossentrophy and uses binary_accuracy to evaluate the model effectiveness. Here we set up the learning rate to 0.000003.



```
# CNN using Keras' Sequential capabilities
model = Sequential()

# The first layer
# 5x5 convolution with 1x1 stride and 16 filters and no padding.
model.add(Conv2D(16, (5,5), strides = (1,1), padding='valid',
                 input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Activation('relu'))

# The second layer
# Another 5x5 convolution with 2x2 stride, 32 filters and no padding.
model.add(Conv2D(32, (5,5), strides = (2,2), padding='valid'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Activation('relu'))

# Thr third layer
# The 3rd 5x5 convolution with 2x2 stride, 64 filters and no padding.
model.add(Conv2D(64, (5,5), strides = (2,2), padding='valid'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Activation('relu'))

## 2x2 max pooling reduces to 4 x 4 x 64
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

## Flatten turns 4 x 4 x 64 into 1024x1
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(1))
model.add(Activation('sigmoid'))
# We start with the learning rate 0.000003
model.compile(loss='binary_crossentropy',optimizer = Adam(learning_rate=0.000003), metrics='binary_accuracy')

model.summary()
```

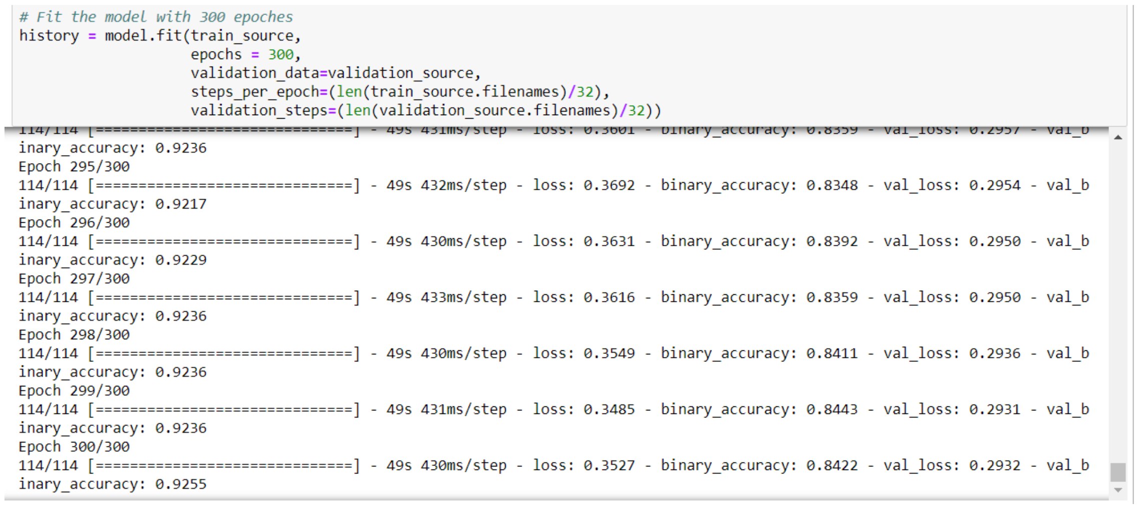
Compile the model

The compile result is as following:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 16)        1216
_________________________________________________________________
batch_normalization (BatchNo (None, 46, 46, 16)        64
_________________________________________________________________
dropout (Dropout)            (None, 46, 46, 16)        0
_________________________________________________________________
activation (Activation)      (None, 46, 46, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 32)        12832
_________________________________________________________________
batch_normalization_1 (Batch (None, 21, 21, 32)        128
_________________________________________________________________
dropout_1 (Dropout)          (None, 21, 21, 32)        0
_________________________________________________________________
activation_1 (Activation)    (None, 21, 21, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 64)          51264
_________________________________________________________________
batch_normalization_2 (Batch (None, 9, 9, 64)          256
_________________________________________________________________
dropout_2 (Dropout)          (None, 9, 9, 64)          0
_________________________________________________________________
activation_2 (Activation)    (None, 9, 9, 64)          0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 4, 4, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 64)          256
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
activation_3 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 131,681
Trainable params: 131,329
Non-trainable params: 352
_________________________________________________________________
```

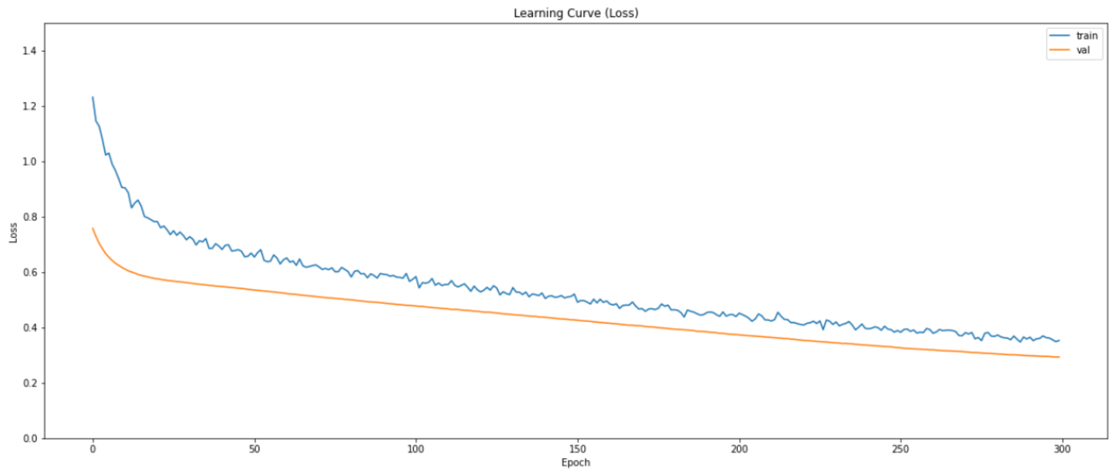
Then the model is fit with the train data set. We run the fit with 300 epochs to see if the train and validation sets could have an organic performance on losses and accuracy, and since the data set is in the form of a generator, we do not assign the batch size parameter according to the official API document.



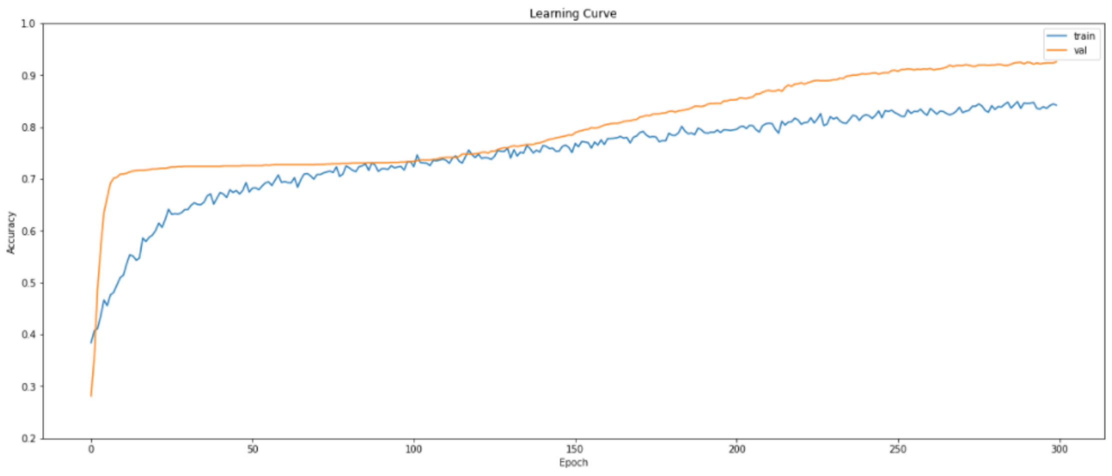
```
# Fit the model with 300 epoches
history = model.fit(train_source,
                    epochs = 300,
                    validation_data=validation_source,
                    steps_per_epoch=(len(train_source.filenames)/32),
                    validation_steps=(len(validation_source.filenames)/32))
```

```
114/114 [==============================] - 49s 431ms/step - loss: 0.3601 - binary_accuracy: 0.8359 - val_loss: 0.2957 - val_b
inary_accuracy: 0.9236
Epoch 295/300
114/114 [==============================] - 49s 432ms/step - loss: 0.3692 - binary_accuracy: 0.8348 - val_loss: 0.2954 - val_b
inary_accuracy: 0.9217
Epoch 296/300
114/114 [==============================] - 49s 430ms/step - loss: 0.3631 - binary_accuracy: 0.8392 - val_loss: 0.2950 - val_b
inary_accuracy: 0.9229
Epoch 297/300
114/114 [==============================] - 49s 433ms/step - loss: 0.3616 - binary_accuracy: 0.8359 - val_loss: 0.2950 - val_b
inary_accuracy: 0.9236
Epoch 298/300
114/114 [==============================] - 49s 430ms/step - loss: 0.3549 - binary_accuracy: 0.8411 - val_loss: 0.2936 - val_b
inary_accuracy: 0.9236
Epoch 299/300
114/114 [==============================] - 49s 431ms/step - loss: 0.3485 - binary_accuracy: 0.8443 - val_loss: 0.2931 - val_b
inary_accuracy: 0.9236
Epoch 300/300
114/114 [==============================] - 49s 430ms/step - loss: 0.3527 - binary_accuracy: 0.8422 - val_loss: 0.2932 - val_b
inary_accuracy: 0.9255
```

Once the model is fit, let's evaluate the losses and accuracy scores of the train set and validation set respectively.
The figure of the train set and validation set is listed below. The loss curve of the train set is looking good and reasonable since the values fluctuate but diminish consistently with the increasing epochs.
The loss curve of the validation set looks nicer with lower losses all the way and a more stable diminishing trend.



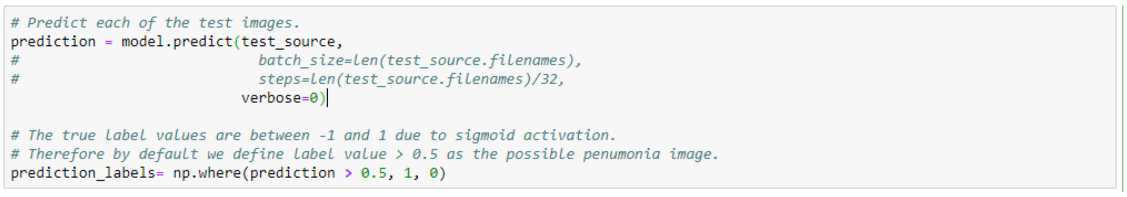
Now we can expect good accuracy scores of the train set and validation set. Similar to the loss curve, the train set has a fluctuate but increasing accuracy scores correspond to accumulated epochs. We also produce a less fluctuating accuracy curve for the validation set with a sharp increased accuracy scores at the beginning, then going smoothly to the middle and going up significantly again to the end.



In general, the first model shows some positive information and gives us some confidence to proceed with the follow - up supervised prediction. We are going to utilize the model to predict the test data set.

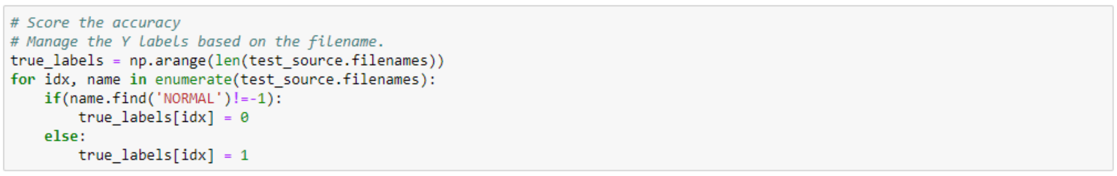
Predict

The final step is to conduct the prediction on the test data set. As mentioned previously we use the test data generator to generate the source iterator as the prediction input. Since we are using the image data generator, we do not assign batch size and steps parameters.
The prediction would generate the classification probabilities to each item. At this moment we define the image as “Pneumonia” if the probability is greater than 0.5, otherwise it is regarded as “Normal”.



```
# Predict each of the test images.
prediction = model.predict(test_source,
#                              batch_size=len(test_source.filenames),
#                              steps=len(test_source.filenames)/32,
                           verbose=0)

# The true label values are between -1 and 1 due to sigmoid activation.
# Therefore by default we define label value > 0.5 as the possible penumonia image.
prediction_labels= np.where(prediction > 0.5, 1, 0)
```

For the actual “Y”, we use a simple loop to iterate the whole test date set and assign value “0” to the image that contains “Normal” in its file name, and in contrast assign value “1” to those contain “Pneumonia” in the file names. As a result we create a Numpy array to keep the “Y true” values.

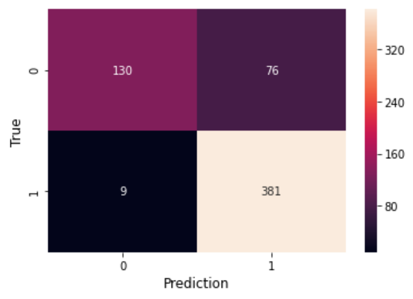


```
# Score the accuracy
# Manage the Y labels based on the filename.
true_labels = np.arange(len(test_source.filenames))
for idx, name in enumerate(test_source.filenames):
    if(name.find('NORMAL')!=-1):
        true_labels[idx] = 0
    else:
        true_labels[idx] = 1
```

Now we have both prediction and true classification labels. We then leverage the heatmap and classification report from sklearn to check the prediction quality. In addition to a 85.7% prediction accuracy, another noticeable factor about the prediction is the model generates few type 2 error cases, which is important for medical diagnosis since the costs of type 2 error could be patients' lives.

```
# Check the accuracy with heatmap
print(prediction_labels.shape)
print(accuracy_score(true_labels, prediction_labels))
confusion_matrix = confusion_matrix(true_labels, prediction_labels)
sns.heatmap(confusion_matrix, annot=True, fmt="d")
plt.xlabel("Prediction", fontsize= 12)
plt.ylabel("True", fontsize= 12)
plt.show()
```

```
(596, 1)
0.8573825503355704
```



```
print('                ====The classification report====')
print(classification_report(true_labels, prediction_labels, labels = [0, 1]))
```

```
                ====The classification report====
              precision    recall  f1-score   support

           0       0.94      0.63      0.75       206
           1       0.83      0.98      0.90       390

    accuracy                           0.86       596
   macro avg       0.88      0.80      0.83       596
weighted avg       0.87      0.86      0.85       596
```

The second model

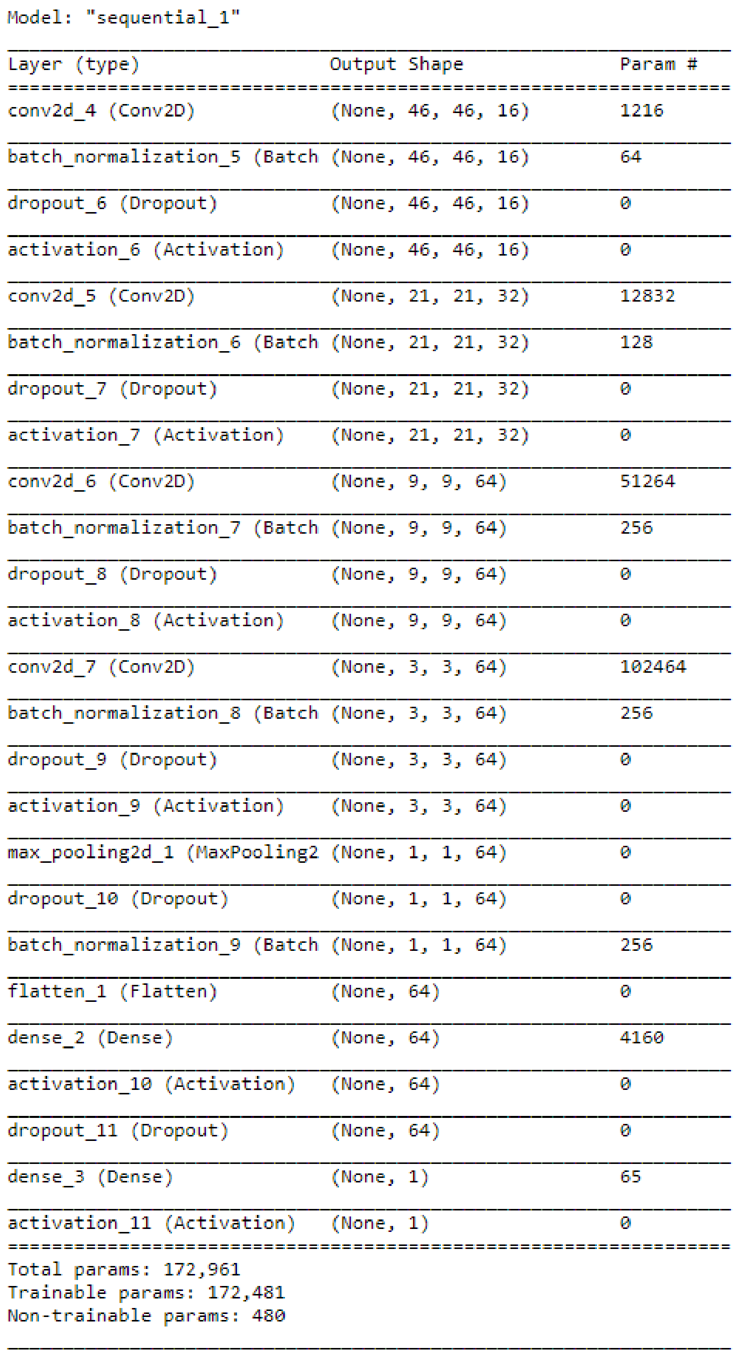
Add one more Conv2D layer

The first model provides 85.7% prediction accuracy with 3 convolution layers. Now let's take a look if there would be a better prediction with an extra convolution layer in the model while keeping other configurations intact.
The fourth CNN layer is with the same parameters as the third one. The overall configuration is as following:

Layer	Filter	Kernal size	Stride	Padding	Drop
Layer 1	16	5 * 5	1 * 1	valid (no padding)	0.2

Layer 2	32	5 * 5	2 * 2	valid (no padding)	0.4
Layer 3	64	5 * 5	2 * 2	valid (no padding)	0.5
Layer 4	64	5 * 5	2 * 2	valid (no padding)	0.5

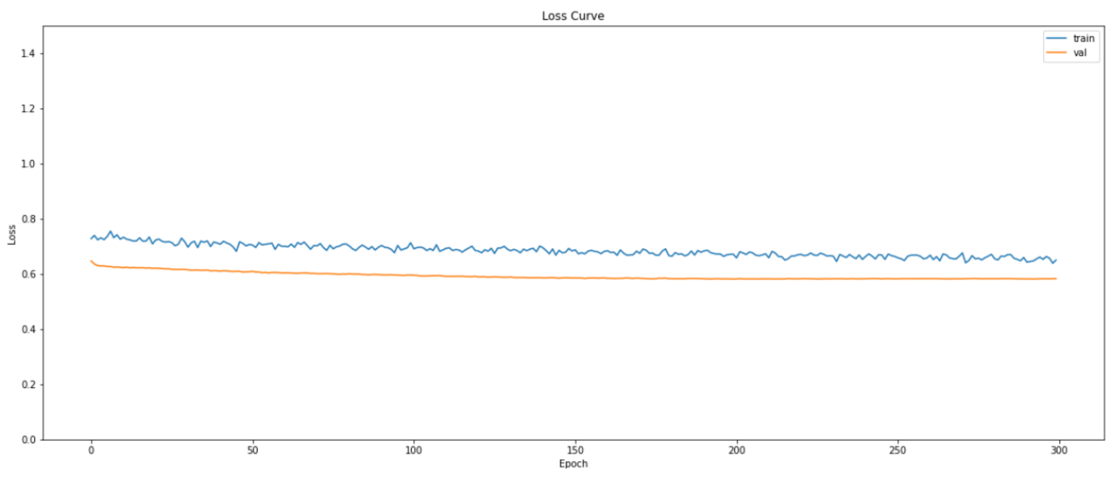
The compile summary is described as below:



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 46, 46, 16)        1216      
_________________________________________________________________
batch_normalization_5 (Batch (None, 46, 46, 16)        64        
_________________________________________________________________
dropout_6 (Dropout)          (None, 46, 46, 16)        0         
_________________________________________________________________
activation_6 (Activation)    (None, 46, 46, 16)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 21, 21, 32)        12832     
_________________________________________________________________
batch_normalization_6 (Batch (None, 21, 21, 32)        128       
_________________________________________________________________
dropout_7 (Dropout)          (None, 21, 21, 32)        0         
_________________________________________________________________
activation_7 (Activation)    (None, 21, 21, 32)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 9, 9, 64)          51264     
_________________________________________________________________
batch_normalization_7 (Batch (None, 9, 9, 64)          256       
_________________________________________________________________
dropout_8 (Dropout)          (None, 9, 9, 64)          0         
_________________________________________________________________
activation_8 (Activation)    (None, 9, 9, 64)          0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 3, 3, 64)          102464    
_________________________________________________________________
batch_normalization_8 (Batch (None, 3, 3, 64)          256       
_________________________________________________________________
dropout_9 (Dropout)          (None, 3, 3, 64)          0         
_________________________________________________________________
activation_9 (Activation)    (None, 3, 3, 64)          0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 1, 1, 64)          0         
_________________________________________________________________
dropout_10 (Dropout)         (None, 1, 1, 64)          0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 1, 1, 64)          256       
_________________________________________________________________
flatten_1 (Flatten)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160      
_________________________________________________________________
activation_10 (Activation)   (None, 64)                0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
_________________________________________________________________
activation_11 (Activation)   (None, 1)                 0         
=================================================================
Total params: 172,961
Trainable params: 172,481
Non-trainable params: 480
_________________________________________________________________
```

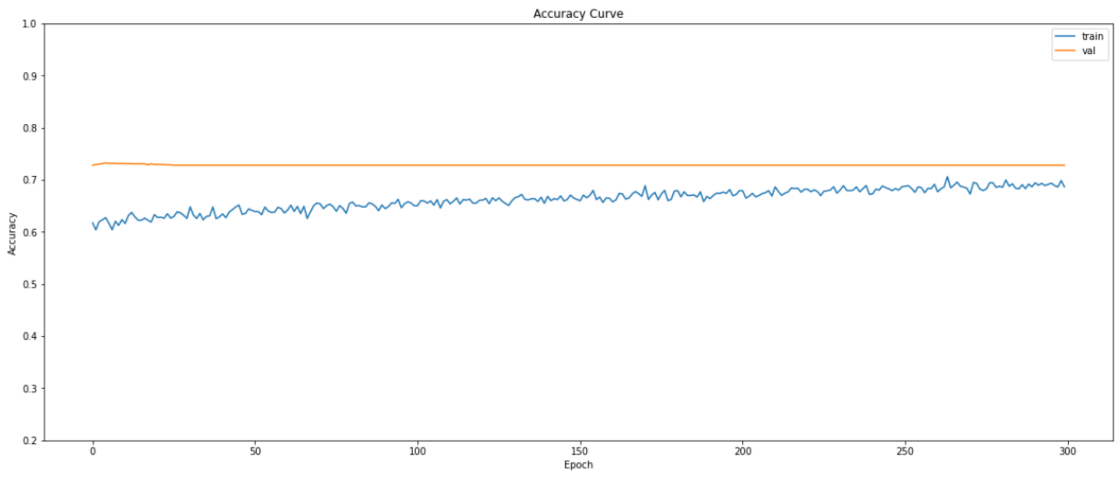
As a result there are nearly 40K more parameters to be trained when comparing with the previous version. Let's check how the losses, accuracy and prediction look like.

The losses:



The loss curves don't look as good as those from the previous model. The curves are flat with no obvious downward trend. In other words, there is no clear diminishing of loss values through the train and validation processes.

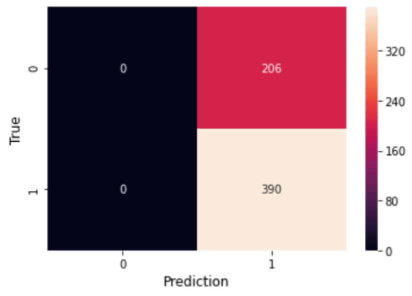
The accuracy:



Again the same phenonium exists in the accuracy figure. There is also no obvious upward trend of accuracy curves. And interestingly but not surprisingly, the prediction result is worse than the one from the previous model. The second model only generates 65% prediction accuracy, and tends to classify all the normal and pneumonia images as "Pneumonia", which is the Type I error.

```
# Check the accuracy with heatmap
print(prediction_labels.shape)
print(accuracy_score(true_labels, prediction_labels))
confusion_matrix = confusion_matrix(true_labels, prediction_labels)
sns.heatmap(confusion_matrix, annot=True, fmt="d")
plt.xlabel("Prediction", fontsize= 12)
plt.ylabel("True", fontsize= 12)
plt.show()
```

```
(596, 1)
0.6543624161073825
```



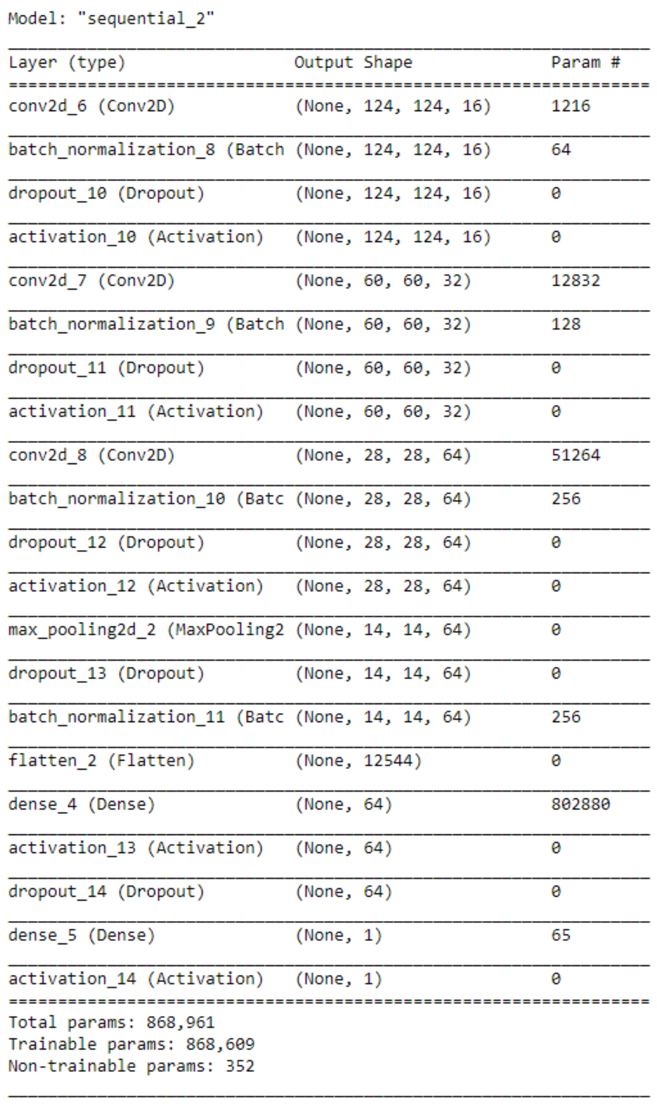
```
# Check the report.
print('                 ====The classification report====')
print(classification_report(true_labels, prediction_labels, labels = [0, 1]))
```

```
                 ====The classification report====
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       206
           1       0.65      1.00      0.79       390

    accuracy                           0.65       596
   macro avg       0.33      0.50      0.40       596
weighted avg       0.43      0.65      0.52       596
```

The third model

Higher image resolution

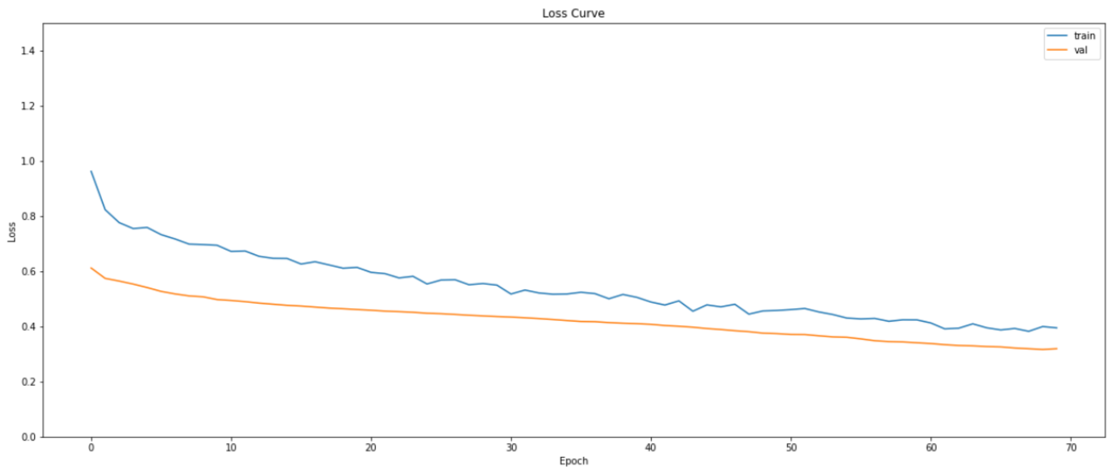
Now let’s re - test the model with 3 convolution layers and higher image resolution. We adjust the image resolution from 50 * 50 to 128 * 128 pixels and as a result, the total parameters have increased to 869K.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 124, 124, 16)      1216      
_________________________________________________________________
batch_normalization_8 (Batch (None, 124, 124, 16)      64        
_________________________________________________________________
dropout_10 (Dropout)         (None, 124, 124, 16)      0         
_________________________________________________________________
activation_10 (Activation)   (None, 124, 124, 16)      0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 60, 60, 32)        12832     
_________________________________________________________________
batch_normalization_9 (Batch (None, 60, 60, 32)        128       
_________________________________________________________________
dropout_11 (Dropout)         (None, 60, 60, 32)        0         
_________________________________________________________________
activation_11 (Activation)   (None, 60, 60, 32)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 64)        51264     
_________________________________________________________________
batch_normalization_10 (Batc (None, 28, 28, 64)        256       
_________________________________________________________________
dropout_12 (Dropout)         (None, 28, 28, 64)        0         
_________________________________________________________________
activation_12 (Activation)   (None, 28, 28, 64)        0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 14, 14, 64)        0         
_________________________________________________________________
batch_normalization_11 (Batc (None, 14, 14, 64)        256       
_________________________________________________________________
flatten_2 (Flatten)          (None, 12544)             0         
_________________________________________________________________
dense_4 (Dense)              (None, 64)                802880    
_________________________________________________________________
activation_13 (Activation)   (None, 64)                0         
_________________________________________________________________
dropout_14 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65        
_________________________________________________________________
activation_14 (Activation)   (None, 1)                 0         
=================================================================
Total params: 868,961
Trainable params: 868,609
Non-trainable params: 352
_________________________________________________________________
```

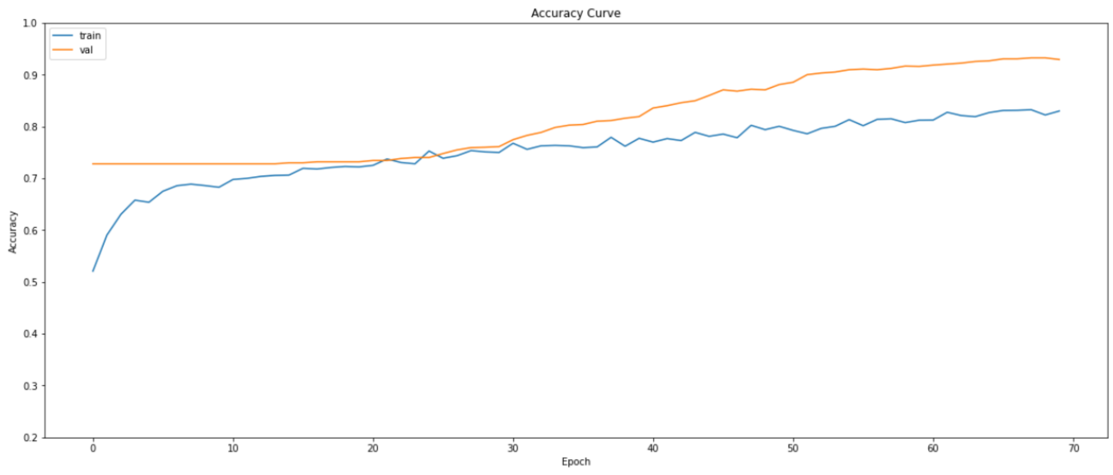
We are not sure if it works out with higher image resolution, therefore we conduct an initial fitting with 70 epochs in order to see the trend.

The losses:



First of all, the loss curves are presenting continuous decreasing loss values. More specifically, comparing with the first model's loss values, the third model generates less losses.

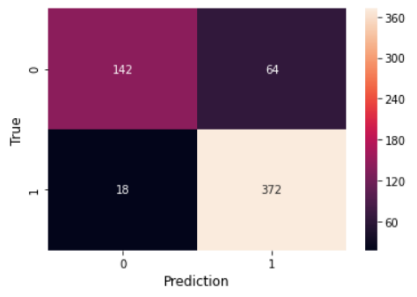
The accuracy:



Similarly, the third model provides better accuracy curves than those from the first model. The 70 - epoch prediction accuracy is around 86.2%.

```
# Check the accuracy with heatmap
print(prediction_labels.shape)
print(accuracy_score(true_labels, prediction_labels))
confusion_matrix = confusion_matrix(true_labels, prediction_labels)
sns.heatmap(confusion_matrix, annot=True, fmt="d")
plt.xlabel("Prediction", fontsize= 12)
plt.ylabel("True", fontsize= 12)
plt.show()
```

```
(596, 1)
0.8624161073825504
```

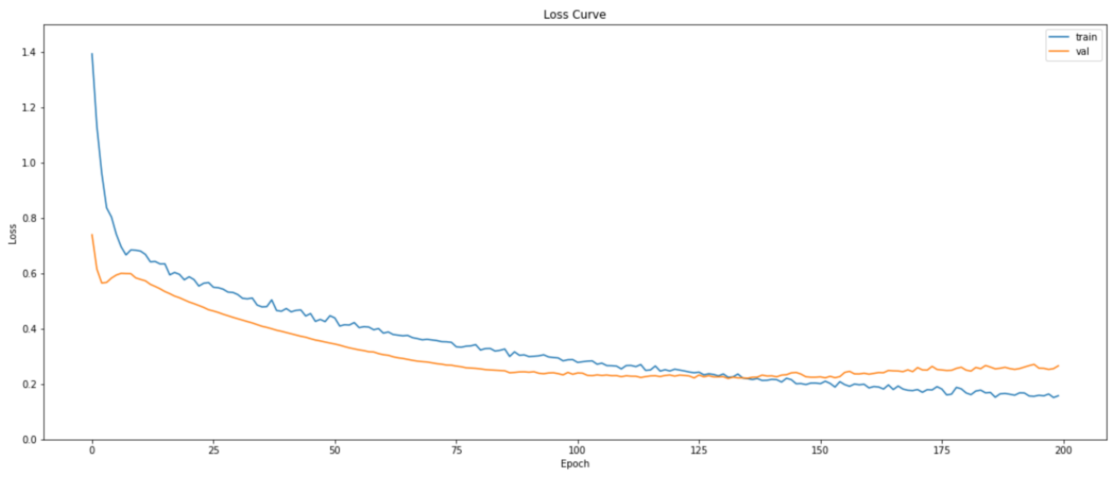


```
# Check the report.
print('                    ====The classification report====')
print(classification_report(true_labels, prediction_labels, labels = [0, 1]))
```

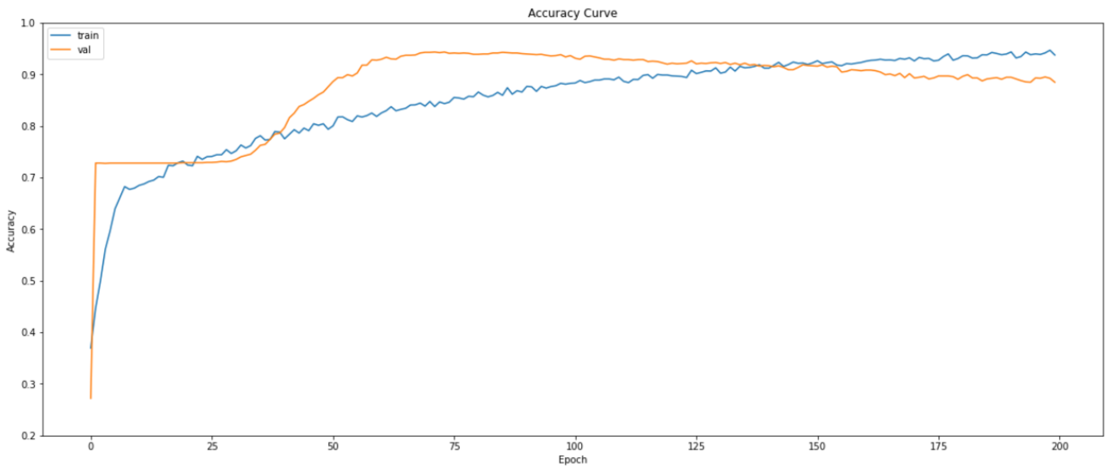
```
                    ====The classification report====
              precision    recall  f1-score   support

           0       0.89      0.69      0.78       206
           1       0.85      0.95      0.90       390

    accuracy                           0.86       596
   macro avg       0.87      0.82      0.84       596
weighted avg       0.87      0.86      0.86       596
```

This observation gives us some confidence to proceed with more epochs. Let’s re - run the model with 200 epochs to check the trends of losses and accuracy.

The losses of 200 epochs:

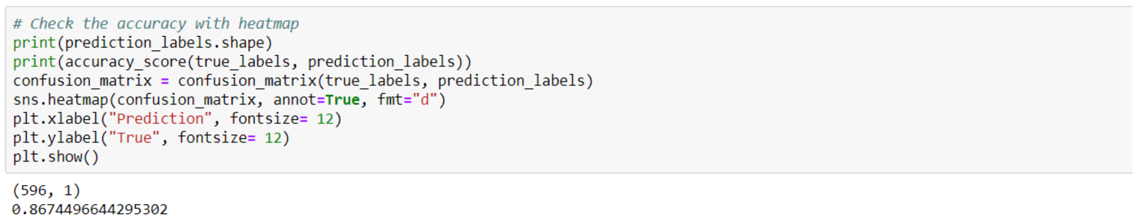


The accuracy of 200 epochs:



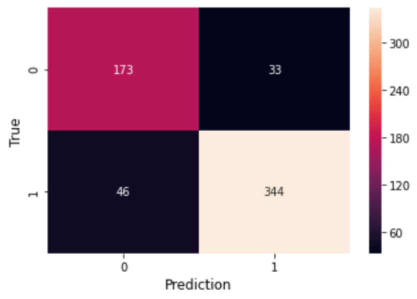
It shows based on this parameter configuration, we could have the best performance around the 100th epoch and after that the accuracy diminishes as well as the loss goes up slightly to the end. It implies that the optimal epoch setting for this model could be 100.

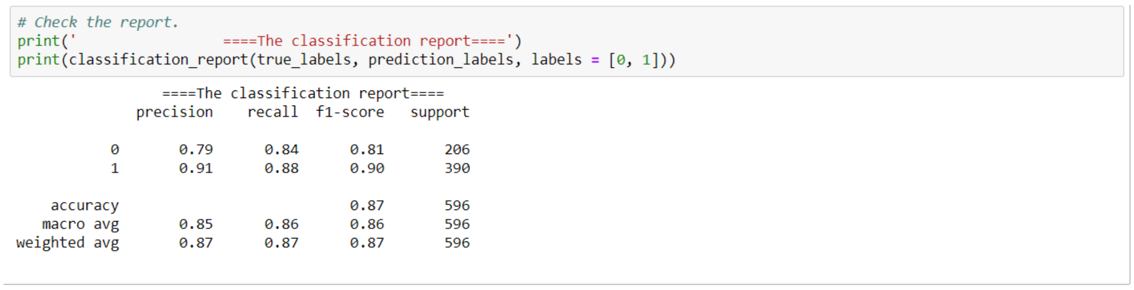
The heatmap and confusion matrix are shown here and with 200 epochs we have a 1% performance improvement when comparing with the first model. The 200 - epoch prediction accuracy is around 86.7%.



```
# Check the accuracy with heatmap
print(prediction_labels.shape)
print(accuracy_score(true_labels, prediction_labels))
confusion_matrix = confusion_matrix(true_labels, prediction_labels)
sns.heatmap(confusion_matrix, annot=True, fmt="d")
plt.xlabel("Prediction", fontsize= 12)
plt.ylabel("True", fontsize= 12)
plt.show()
```

```
(596, 1)
0.8674496644295302
```





```
# Check the report.
print('                 ====The classification report====')
print(classification_report(true_labels, prediction_labels, labels = [0, 1]))
```

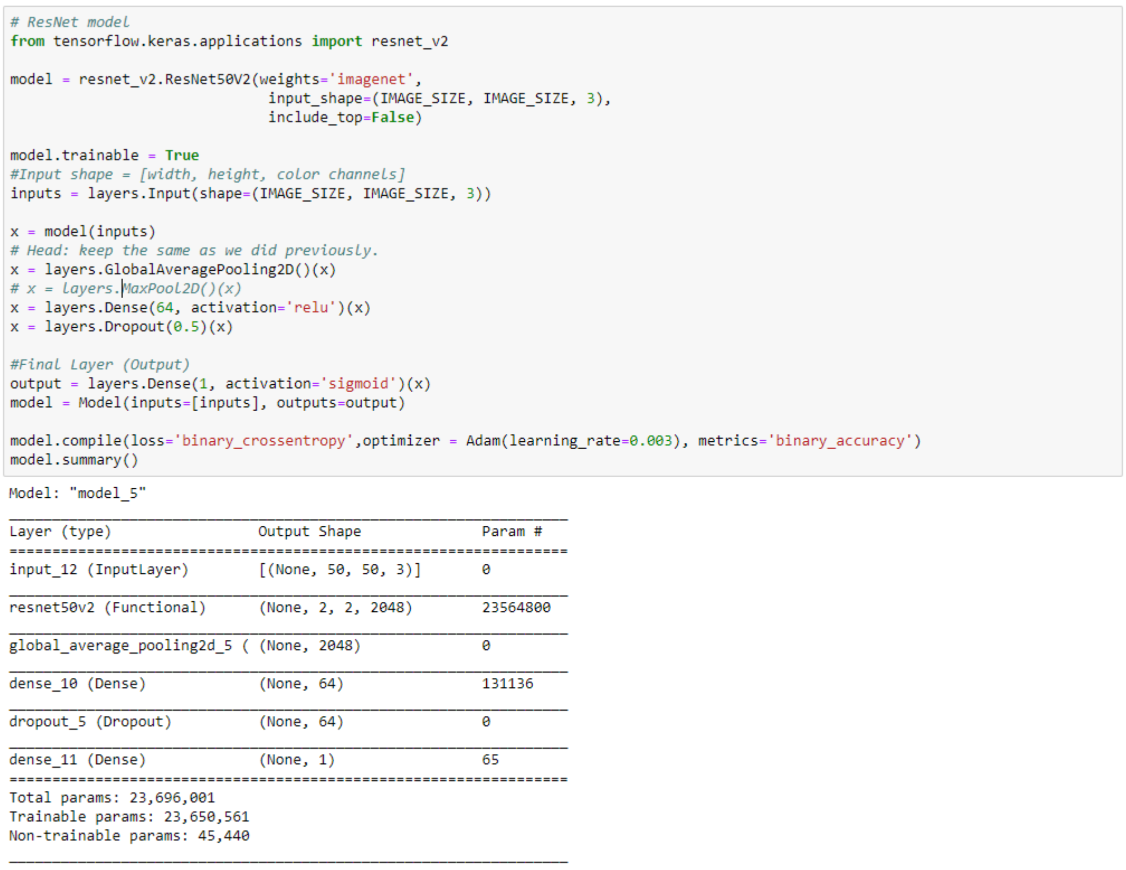
```
                ====The classification report====
              precision    recall  f1-score   support

           0       0.79      0.84      0.81       206
           1       0.91      0.88      0.90       390

    accuracy                           0.87       596
   macro avg       0.85      0.86      0.86       596
weighted avg       0.87      0.87      0.87       596
```

The fourth model

Transfer learning and fine tuning

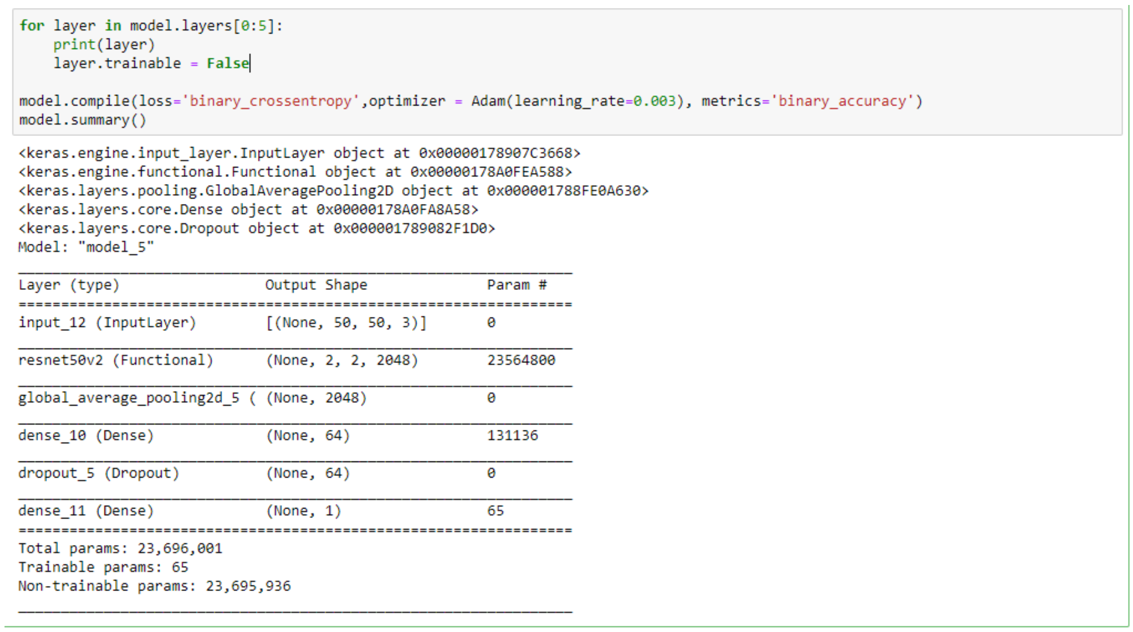
Instead of a purely handmade model, we are going to leverage a pre - trained model provided by ResNet to predict the same data set.

We start from a ResNet50V2 model with a larger learning rate 0.003 and in order to prevent the layers' weights to be modified, we fine - tune the model by freezing all layers except the final Dense and output layer.



```
# ResNet model
from tensorflow.keras.applications import resnet_v2

model = resnet_v2.ResNet50V2(weights='imagenet',
                              input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3),
                              include_top=False)

model.trainable = True
#Input shape = [width, height, color channels]
inputs = layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3))

x = model(inputs)
# Head: keep the same as we did previously.
x = layers.GlobalAveragePooling2D()(x)
# x = layers.MaxPool2D()(x)
x = layers.Dense(64, activation='relu')(x)
x = layers.Dropout(0.5)(x)

#Final Layer (Output)
output = layers.Dense(1, activation='sigmoid')(x)
model = Model(inputs=[inputs], outputs=output)

model.compile(loss='binary_crossentropy',optimizer = Adam(learning_rate=0.003), metrics='binary_accuracy')
model.summary()
```

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_12 (InputLayer)        [(None, 50, 50, 3)]       0
_________________________________________________________________
resnet50v2 (Functional)      (None, 2, 2, 2048)        23564800
_________________________________________________________________
global_average_pooling2d_5 ( (None, 2048)              0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                131136
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65
=================================================================
Total params: 23,696,001
Trainable params: 23,650,561
Non-trainable params: 45,440
_________________________________________________________________
```

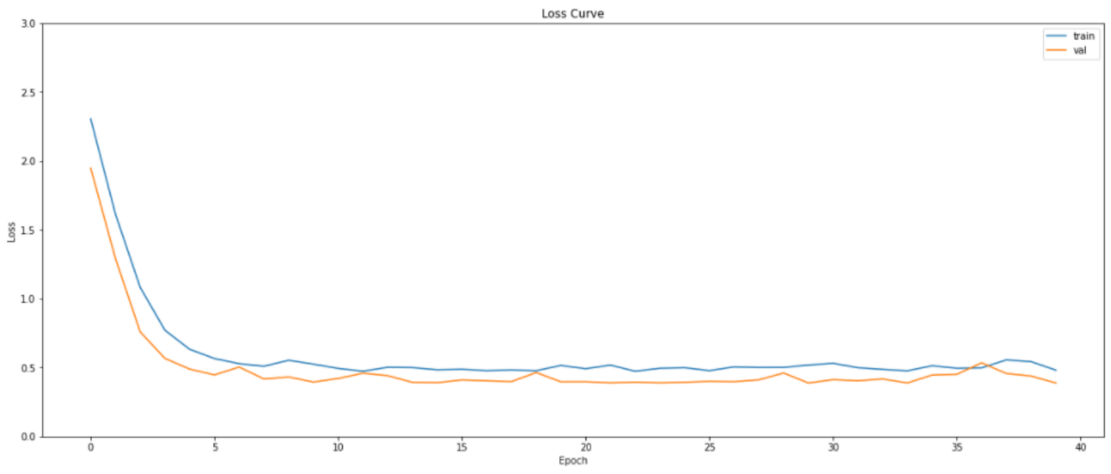


```
for layer in model.layers[0:5]:
    print(layer)
    layer.trainable = False

model.compile(loss='binary_crossentropy',optimizer = Adam(learning_rate=0.003), metrics='binary_accuracy')
model.summary()
```

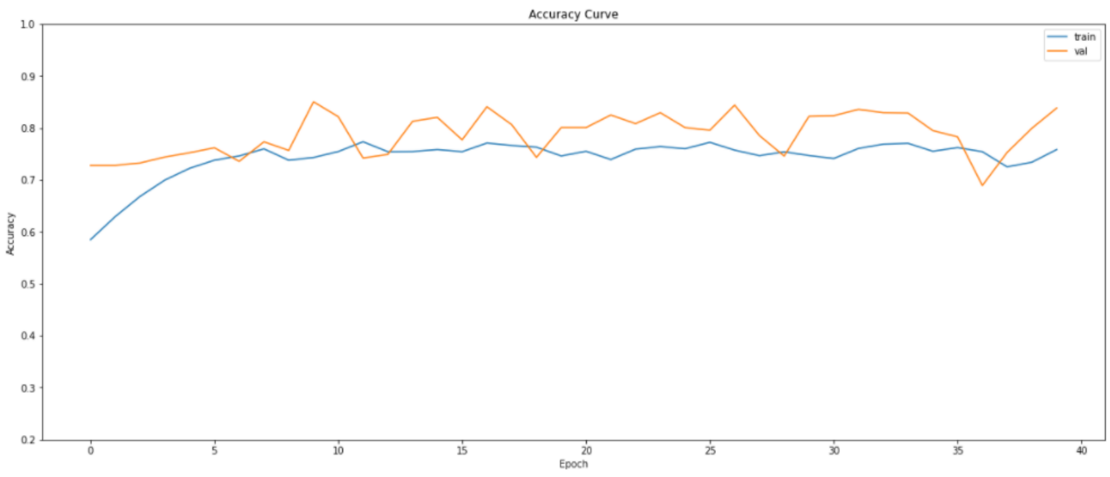
```
<keras.engine.input_layer.InputLayer object at 0x00000178907C3668>
<keras.engine.functional.Functional object at 0x00000178A0FEA588>
<keras.layers.pooling.GlobalAveragePooling2D object at 0x000001788FE0A630>
<keras.layers.core.Dense object at 0x00000178A0FA8A58>
<keras.layers.core.Dropout object at 0x000001789082F1D0>
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_12 (InputLayer)        [(None, 50, 50, 3)]       0         
_________________________________________________________________
resnet50v2 (Functional)      (None, 2, 2, 2048)        23564800  
_________________________________________________________________
global_average_pooling2d_5 ( (None, 2048)              0         
_________________________________________________________________
dense_10 (Dense)             (None, 64)                131136    
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 65        
=================================================================
Total params: 23,696,001
Trainable params: 65
Non-trainable params: 23,695,936
_________________________________________________________________
```

After 40 - epoch fitting, here are loss and accuracy curves:

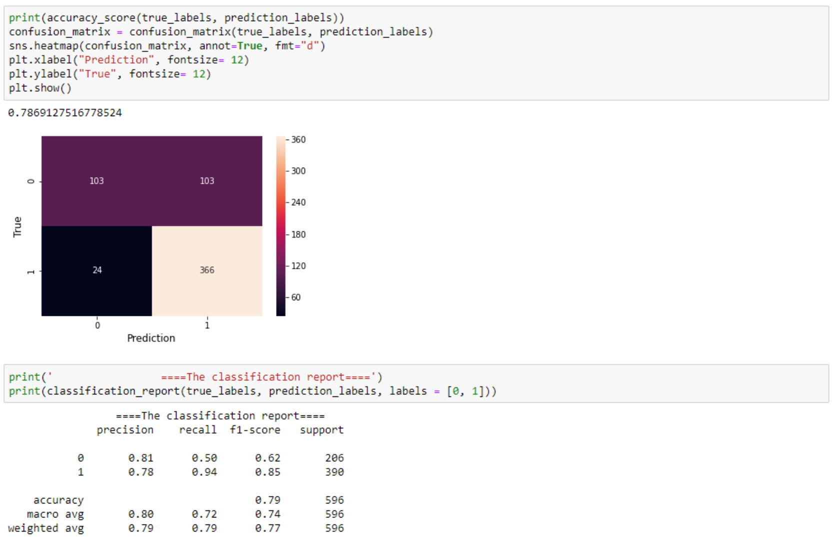
The losses:



The accuracy:



And the prediction figures are listed as follows. The model provides 78.6% prediction accuracy, which is worse than the first and third models.



```
print(accuracy_score(true_labels, prediction_labels))
confusion_matrix = confusion_matrix(true_labels, prediction_labels)
sns.heatmap(confusion_matrix, annot=True, fmt="d")
plt.xlabel("Prediction", fontsize= 12)
plt.ylabel("True", fontsize= 12)
plt.show()
```

```
0.7869127516778524
```

```
print('                 ====The classification report====')
print(classification_report(true_labels, prediction_labels, labels = [0, 1]))
```

```
                 ====The classification report====
              precision    recall  f1-score   support

           0       0.81      0.50      0.62       206
           1       0.78      0.94      0.85       390

    accuracy                           0.79       596
   macro avg       0.80      0.72      0.74       596
weighted avg       0.79      0.79      0.77       596
```

Key findings from the current analysis

1. We are working with a mid - size image dataset which contains 5828 x-ray images in total.
2. The image size is not unified and in general each image is in a different size. Not quite sure if it is a normal case in medical business but we need to resize the images before modeling.
3. When modeling with more convolution layers, it doesn't necessarily generate better results. In fact, as described in the previous section,

keeping others parameters intact and adding one more convolution layer would decrease the prediction accuracy.

4. From the medical and health industry perspective, the proposed model should not only provide acceptable prediction results, but also need to consider minimizing the type II error (False Negative). The purpose is rather to ask patients for a further check but not exclude any potential ones at the beginning.
5. So far we have a better prediction from a self - created model than the pre - trained ResNet50V2 model. However it does not mean that the self - created model performs better than pre - trained one, but rather it means that more time may be needed to adjust and tune the transfer learning model to get better outcomes.

Suggestions for next steps

The follow up steps for the further analysis could include:

1. Accumulate more x-ray “Normal” and “Pneumonia” image data and tune the existing model.
2. Cross - validate the existing models with more image augmentation parameters on the train set. Currently we only introduce image zooming, width and height shifting and it is worth it to have more for further training. In addition, we could also validate the model with more different input image sizes (so far we have tried 50 * 50 and 128 * 128 pixels).
3. Continue to test the model with different configuration and hyperparameter values. We already embed regularization and dropout layers into the model and we can further adjust the values step by step to see if there is any improvement. The learning rate is perhaps the most important parameter among the hyperparameters and should be tuned first. Through previous analysis we realize that the learning of the transfer learning could be very different from the one of pure user - defined CNN model.
4. Continue to try out different parameter combinations with different ResNet models.