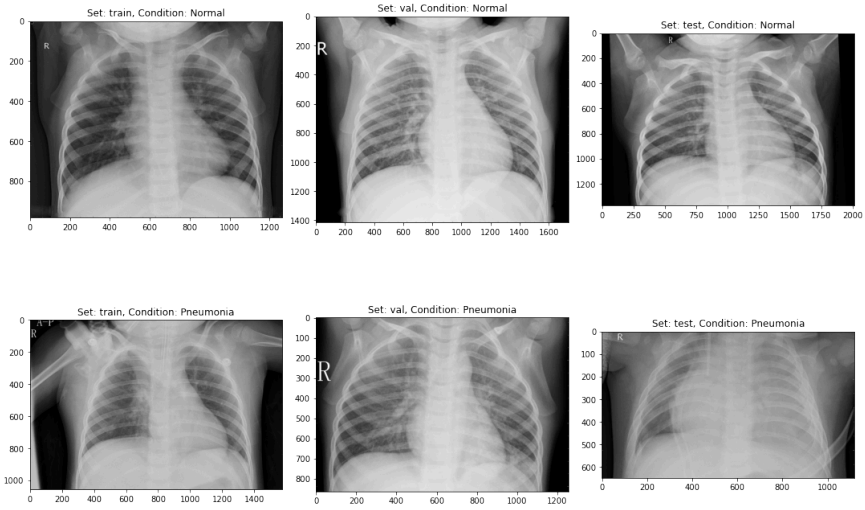
```
import os
import numpy as np
import pandas as pd
import random
import cv2
import matplotlib.pyplot as plt


import keras.backend as K
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, SeparableConv2D, MaxPool2D, LeakyReLU, Activation
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
import tensorflow as tf


seed = 232
np.random.seed(seed)
tf.random.set_seed(seed)
```

+ Code + Text

```
path = './chest_xray/'

fig, ax = plt.subplots(2, 3, figsize=(15, 10))
ax = ax.ravel()
plt.tight_layout()

for i, _set in enumerate(['train', 'val', 'test']):
    set_path = path + _set
    ax[i].imshow(plt.imread(set_path+'/NORMAL/'+os.listdir(set_path+'/NORMAL')[0]), cmap='gray')
    ax[i].set_title('Set: {}, Condition: Normal'.format(_set))

    ax[i+3].imshow(plt.imread(set_path+'/PNEUMONIA/'+os.listdir(set_path+'/PNEUMONIA')[0]), cmap='gray')
    ax[i+3].set_title('Set: {}, Condition: Pneumonia'.format(_set))
```



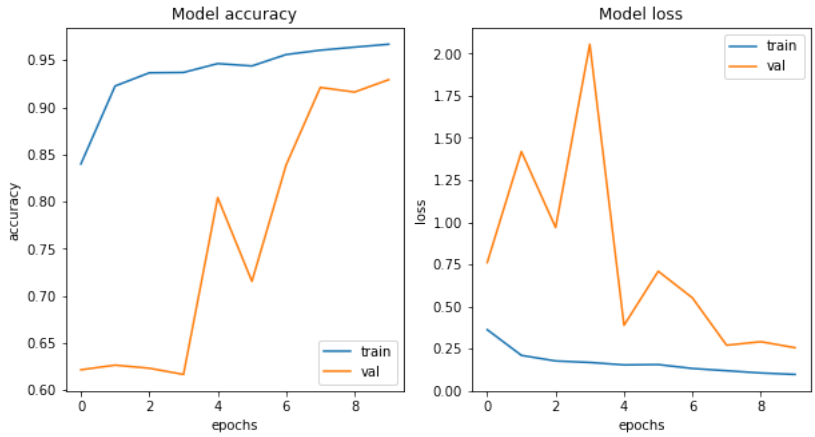
```
def process_data(img_dims, batch_size):

    # Data generation objects
    train_datagen = image_gen = ImageDataGenerator(
                                  rescale = 1./255,
                                  shear_range = 0.2,
                                  zoom_range = 0.2,
                                  horizontal_flip = True,
                                )

    test_val_datagen = ImageDataGenerator(
                                     rescale=1./255
                                 )

    # This is fed to the network in the specified batch sizes and image dimensions
    train_gen = train_datagen.flow_from_directory(
                    directory= path + 'train',
                    target_size=(img_dims, img_dims),
                    batch_size=batch_size,
                    class_mode='binary',
                    shuffle=True)

    test_gen = test_val_datagen.flow_from_directory(
                    directory= path + 'test',
                    target_size=(img_dims, img_dims),
                    batch_size=batch_size,
                    class_mode='binary',
                    shuffle=True)

    val_gen = test_val_datagen.flow_from_directory(
                    directory= path + 'val',
                    target_size=(img_dims, img_dims),
                    batch_size=batch_size,
                    class_mode='binary',
                    shuffle=True)

    test_data = []
    test_labels = []

    for cond in ['/NORMAL/', '/PNEUMONIA/']:
        for img in (os.listdir(path + 'test' + cond)):
            img = plt.imread(path + 'test' + cond + img)
            img = cv2.resize(img, (img_dims, img_dims))
            img = np.dstack([img, img, img])
            img = img.astype('float32') / 255
            if cond=='/NORMAL/':
                label = 0
            elif cond=='/PNEUMONIA/':
                label = 1
            test_data.append(img)
            test_labels.append(label)

    test_data = np.array(test_data)
    test_labels = np.array(test_labels)

    return train_gen, test_gen, test_data, test_labels


img_dims = 150
epochs = 10
batch_size = 32

train_gen, test_gen, test_data, test_labels = process_data(img_dims, batch_size)
```

```
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
```

```
## CNN Architecture

inputs = Input(shape=(img_dims, img_dims, 3))

# First conv block
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(inputs)
x = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Second conv block
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Third conv block
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)

# Fourth conv block
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)

# Fifth conv block
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = SeparableConv2D(filters=256, kernel_size=(3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.2)(x)

# FC layer
x = Flatten()(x)
x = Dense(units=512, activation='relu')(x)
x = Dropout(rate=0.7)(x)
x = Dense(units=128, activation='relu')(x)
x = Dropout(rate=0.5)(x)
x = Dense(units=64, activation='relu')(x)
x = Dropout(rate=0.3)(x)

# Output layer
output = Dense(units=1, activation='sigmoid')(x)

# Creating model and compiling
model = Model(inputs=inputs, outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint(filepath='best_weights.hdf5', save_best_only=True, save_weights_only=True)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=2, verbose=2, mode='max')
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.1, patience=1, mode='min')


hist = model.fit(
           train_gen, steps_per_epoch=train_gen.samples // batch_size,
           epochs=epochs, validation_data=test_gen,
           validation_steps=test_gen.samples // batch_size, callbacks=[checkpoint, lr_reduce])

     Epoch 1/10
     163/163 [==============================] - 122s 750ms/step - loss: 0.3622 - accuracy: 0.8399 - val_loss: 0.7606 - val_accuracy: 0.62
     Epoch 2/10
     163/163 [==============================] - 115s 704ms/step - loss: 0.2096 - accuracy: 0.9225 - val_loss: 1.4191 - val_accuracy: 0.62
     Epoch 3/10
     163/163 [==============================] - 114s 699ms/step - loss: 0.1764 - accuracy: 0.9365 - val_loss: 0.9685 - val_accuracy: 0.62
     Epoch 4/10
     163/163 [==============================] - 114s 698ms/step - loss: 0.1676 - accuracy: 0.9369 - val_loss: 2.0563 - val_accuracy: 0.61
     Epoch 5/10
     163/163 [==============================] - 112s 689ms/step - loss: 0.1529 - accuracy: 0.9463 - val_loss: 0.3881 - val_accuracy: 0.86
     Epoch 6/10
     163/163 [==============================] - ETA: 0s - loss: 0.1549 - accuracy: 0.9438
     Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
     163/163 [==============================] - 112s 688ms/step - loss: 0.1549 - accuracy: 0.9438 - val_loss: 0.7090 - val_accuracy: 0.71
     Epoch 7/10
     163/163 [==============================] - 115s 704ms/step - loss: 0.1317 - accuracy: 0.9559 - val_loss: 0.5510 - val_accuracy: 0.83
     Epoch 8/10
     163/163 [==============================] - ETA: 0s - loss: 0.1184 - accuracy: 0.9605
     Epoch 00008: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
     163/163 [==============================] - 117s 717ms/step - loss: 0.1184 - accuracy: 0.9605 - val_loss: 0.2697 - val_accuracy: 0.92
     Epoch 9/10
     163/163 [==============================] - 116s 712ms/step - loss: 0.1051 - accuracy: 0.9638 - val_loss: 0.2907 - val_accuracy: 0.91
     Epoch 10/10
```

```
163/163 [==============================] - ETA: 0s - loss: 0.0964 - accuracy: 0.9668
Epoch 00010: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
163/163 [==============================] - 114s 702ms/step - loss: 0.0964 - accuracy: 0.9668 - val_loss: 0.2551 - val_accuracy: 0.92
```

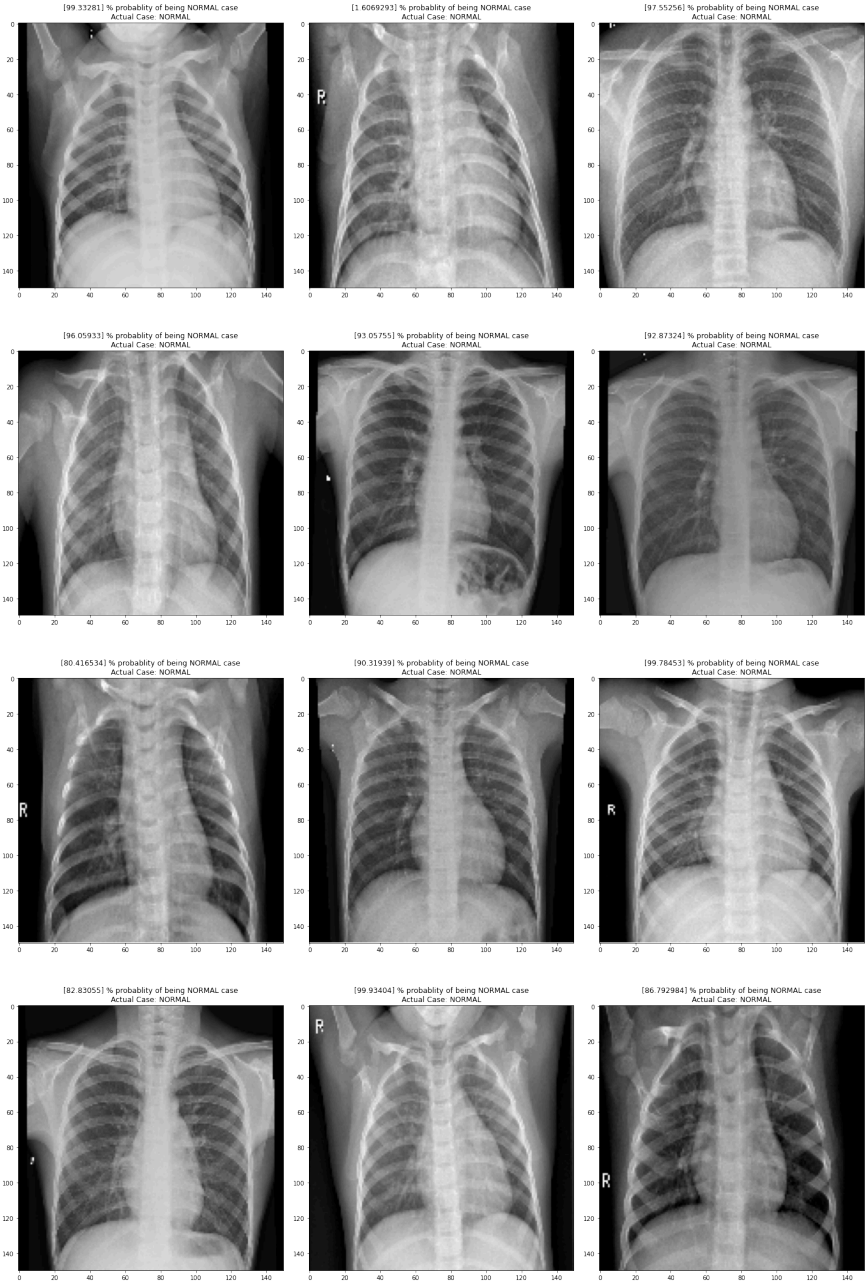
```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax = ax.ravel()

for i, met in enumerate(['accuracy', 'loss']):
    ax[i].plot(hist.history[met])
    ax[i].plot(hist.history['val_' + met])

    ax[i].set_title('Model {}'.format(met))

    ax[i].set_xlabel('epochs')
    ax[i].set_ylabel(met)

    ax[i].legend(['train', 'val'])
```




```
from sklearn.metrics import accuracy_score, confusion_matrix

preds = model.predict(test_data)

acc = accuracy_score(test_labels, np.round(preds))*100
cm = confusion_matrix(test_labels, np.round(preds))
tn, fp, fn, tp = cm.ravel()

print('CONFUSION MATRIX ------------------')
print(cm)

print('\nTEST METRICS ----------------------')
precision = tp/(tp+fp)*100
recall = tp/(tp+fn)*100
print('Accuracy: {}%'.format(acc))
print('Precision: {}%'.format(precision))
print('Recall: {}%'.format(recall))
print('F1-score: {}'.format(2*precision*recall/(precision+recall)))

print('\nTRAIN METRIC ----------------------')
print('Train acc: {}'.format(np.round((hist.history['accuracy'][-1])*100, 2)))
```

```
CONFUSION MATRIX ------------------
[[200  34]
 [  9 381]]

TEST METRICS ----------------------
Accuracy: 93.10897435897436%
Precision: 91.80722891566265%
Recall: 97.69230769230769%
F1-score: 94.6583850931677

TRAIN METRIC ----------------------
Train acc: 96.68
```

```
fig, ax = plt.subplots(4, 3, figsize=(20, 30))
ax = ax.ravel()
plt.tight_layout()

for i in range(12):
    ax[i].imshow(test_data[i], cmap='gray')

    if test_labels[i] == 0:
        ax[i].set_title('{} % probablity of being NORMAL case'.format((1 - preds[i])*100) + '\n' + 'Actual Case: NORMAL')
    else:
        ax[i].set_title('{} % probablity of being PNEUMONIA case'.format(preds[i]*100) + '\n' + 'Actual Case: PNEUMONIA')
```


[99.33281] % probablity of being NORMAL case
Actual Case: NORMAL
[1.6069293] % probablity of being NORMAL case
Actual Case: NORMAL
[97.55256] % probablity of being NORMAL case
Actual Case: NORMAL
[96.05933] % probablity of being NORMAL case
Actual Case: NORMAL
[93.05755] % probablity of being NORMAL case
Actual Case: NORMAL
[92.87324] % probablity of being NORMAL case
Actual Case: NORMAL
[80.416534] % probablity of being NORMAL case
Actual Case: NORMAL
[90.31939] % probablity of being NORMAL case
Actual Case: NORMAL
[99.78453] % probablity of being NORMAL case
Actual Case: NORMAL
[82.83055] % probablity of being NORMAL case
Actual Case: NORMAL
[99.93404] % probablity of being NORMAL case
Actual Case: NORMAL
[86.792984] % probablity of being NORMAL case
Actual Case: NORMAL

```
fig, ax = plt.subplots(4, 3, figsize=(20, 30))
ax = ax.ravel()
plt.tight_layout()

for i in range(12):
    for j in range(234, 246):
        ax[i].imshow(test_data[j])

        if test_labels[j] == 0:
            ax[i].set_title('{} % probablity of being NORMAL case'.format((1 - preds[j])*100) + '\n' + 'Actual Case: NORMAL')
        else:
            ax[i].set_title('{} % probablity of being PNEUMONIA case'.format(preds[j]*100) + '\n' + 'Actual Case: PNEUMONIA')
```