In [1]:

```
import os
import glob
import h5py
import shutil
import imgaug as aug
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mimg
import imgaug.augmenters as iaa
from os import listdir, makedirs, getcwd, remove
from os.path import isfile, join, abspath, exists, isdir, expanduser
from PIL import Image
from pathlib import Path
from skimage.io import imread
from skimage.transform import resize
from keras.models import Sequential, Model
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import ImageDataGenerator,load_img, img_to_array
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Input, Flatten, SeparableConv2D
from keras.layers import GlobalMaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.merge import Concatenate
from keras.models import Model
from keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
import cv2
import tensorflow as tf
from keras import backend as K
color = sns.color_palette()
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [2]:

```
import tensorflow as tf

# Set the seed for hash based operations in python
os.environ['PYTHONHASHSEED'] = '0'

# Set the numpy seed
np.random.seed(111)

# Disable multi-threading in tensorflow ops
session_conf = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)

# Set the random seed in tensorflow at graph level
tf.random.set_seed(111)



# Make the augmentation sequence deterministic
aug.seed(111)
```

In [3]:

```
# Define path to the data directory
data_dir = Path('chest_xray')

# Path to train directory (Fancy pathlib...no more os.path!!)
train_dir = data_dir / 'train'

# Path to test directory
test_dir = data_dir / 'test'
```

In [4]:

```
# Get the path to the normal and pneumonia sub-directories
normal_cases_dir = train_dir / 'NORMAL'
pneumonia_cases_dir = train_dir / 'PNEUMONIA'

# Get the list of all the images
normal_cases = normal_cases_dir.glob('*.*g')
#normal_cases.extend('*.png')
#normal_cases.extend('*.jpg')
pneumonia_cases = pneumonia_cases_dir.glob('*.*g')
#pneumonia_cases = pneumonia_cases_dir.glob('*.jpg')
#pneumonia_cases = pneumonia_cases_dir.glob('*.png')

print(normal_cases)
# An empty list. We will insert the data into this list in (img_path, label) format
train_data = []

# Go through all the normal cases. The label for these cases will be 0
for img in normal_cases:
    train_data.append((img,0))

# Go through all the pneumonia cases. The label for these cases will be 1
for img in pneumonia_cases:
    train_data.append((img, 1))

# Get a pandas dataframe from the data we have in our list
train_data = pd.DataFrame(train_data, columns=['image', 'label'],index=None)

# Shuffle the data
train_data = train_data.sample(frac=1.).reset_index(drop=True)
# How the dataframe looks like?
train_data.head()
```

```
&lt;generator object Path.glob at 0x7f9eb56a9ad0&gt;
```

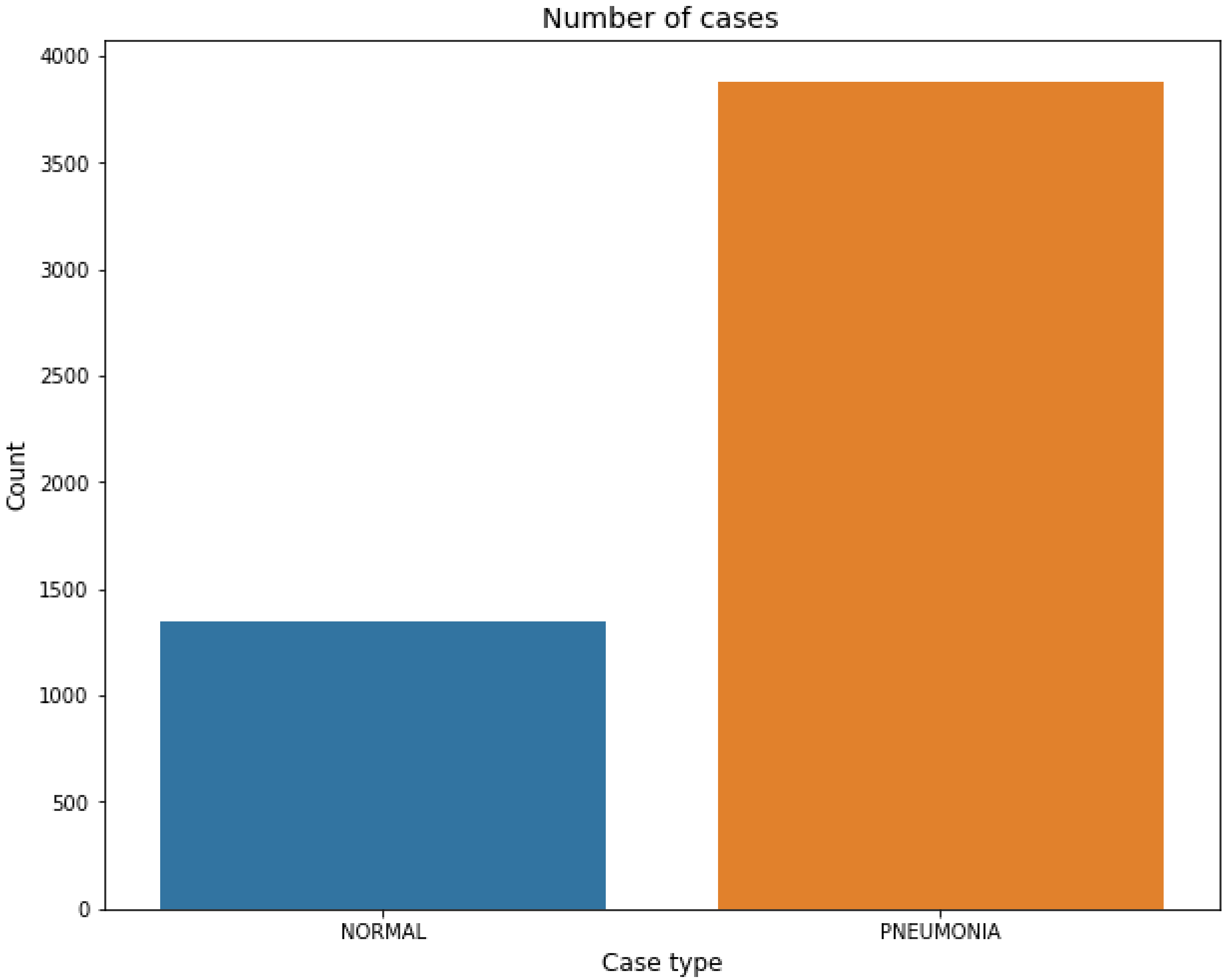
Out[4]:

	image	label
0	chest_xray/train/PNEUMONIA/BACTERIA-3134196-00...	1
1	chest_xray/train/PNEUMONIA/BACTERIA-1083680-00...	1
2	chest_xray/train/PNEUMONIA/BACTERIA-1950119-00...	1
3	chest_xray/train/PNEUMONIA/VIRUS-8028911-0002....	1
4	chest_xray/train/PNEUMONIA/VIRUS-5331068-0002....	1

In [5]:

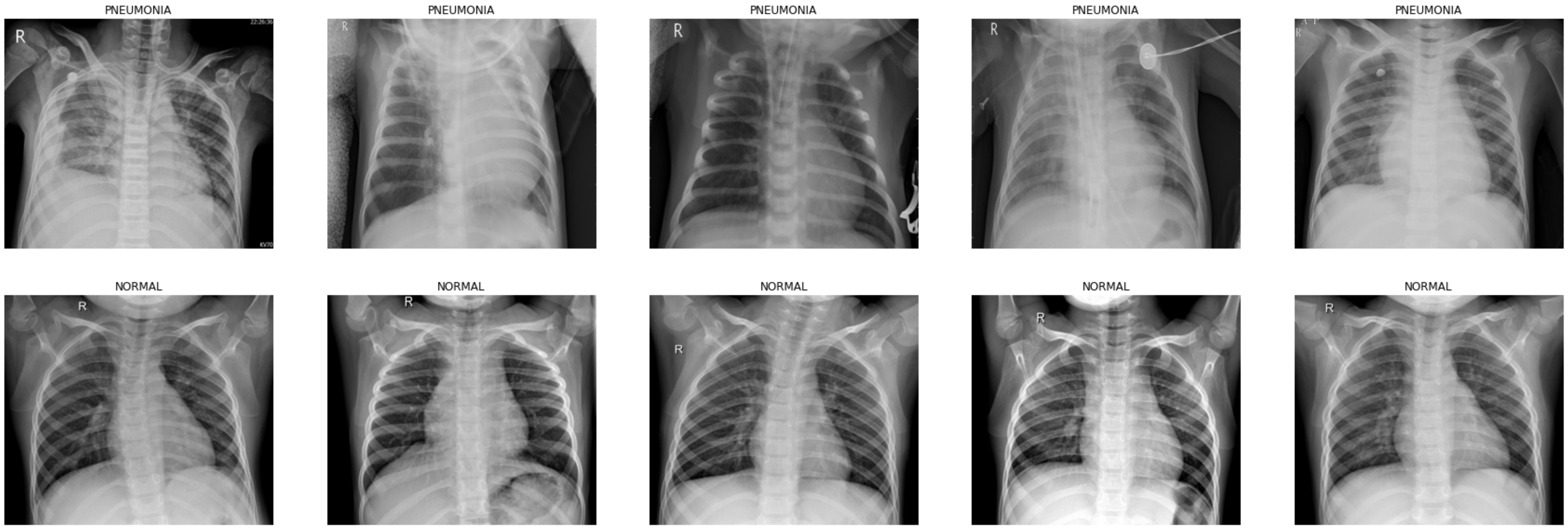
```
# Get the counts for each class
cases_count = train_data['label'].value_counts()
print(cases_count)

# Plot the results
plt.figure(figsize=(10,8))
sns.barplot(x=cases_count.index, y= cases_count.values)
plt.title('Number of cases', fontsize=14)
plt.xlabel('Case type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(range(len(cases_count.index)), ['NORMAL', 'PNEUMONIA'])
plt.show()
```

```
1    3883
0    1349
Name: label, dtype: int64
```


Number of cases
Count
Case type
NORMAL
PNEUMONIA
0
500
1000
1500
2000
2500
3000
3500
4000

In [6]:

```
pneumonia_samples = (train_data[train_data['label']==1]['image'].iloc[:5]).tolist()
normal_samples = (train_data[train_data['label']==0]['image'].iloc[:5]).tolist()

# Concat the data in a single list and del the above two list
samples = pneumonia_samples + normal_samples
del pneumonia_samples, normal_samples

# Plot the data
f, ax = plt.subplots(2,5, figsize=(30,10))
for i in range(10):
    img = imread(samples[i])
    ax[i//5, i%5].imshow(img, cmap='gray')
    if i<5:
        ax[i//5, i%5].set_title("PNEUMONIA")
    else:
        ax[i//5, i%5].set_title("NORMAL")
    ax[i//5, i%5].axis('off')
    ax[i//5, i%5].set_aspect('auto')
plt.show()
```



In [ ]:

```

```

In [7]:

```
# Get the path to the sub-directories
normal_cases_dir = test_dir / 'NORMAL'
pneumonia_cases_dir = test_dir / 'PNEUMONIA'

# Get the list of all the images
normal_cases = normal_cases_dir.glob('*.*g')
pneumonia_cases = pneumonia_cases_dir.glob('*.*g')

# List that are going to contain validation images data and the corresponding labels
valid_data = []
valid_labels = []


# Some images are in grayscale while majority of them contains 3 channels. So, if the image is grayscale, we will convert into a i
mage with 3 channels.
# We will normalize the pixel values and resizing all the images to 224x224

# Normal cases
for img in normal_cases:
    img = cv2.imread(str(img))
    img = cv2.resize(img, (224,224))
    if img.shape[2] ==1:
        img = np.dstack([img, img, img])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)/255.
    label = to_categorical(0, num_classes=2)
    valid_data.append(img)
    valid_labels.append(label)

# Pneumonia cases
for img in pneumonia_cases:
    img = cv2.imread(str(img))
    img = cv2.resize(img, (224,224))
    if img.shape[2] ==1:
        img = np.dstack([img, img, img])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)/255.
    label = to_categorical(1, num_classes=2)
    valid_data.append(img)
```

```
    valid_labels.append(label)

# Convert the list into numpy arrays
valid_data = np.array(valid_data)
valid_labels = np.array(valid_labels)

print("Total number of validation examples: ", valid_data.shape)
print("Total number of labels:", valid_labels.shape)
```

```
Total number of validation examples:  (624, 224, 224, 3)
Total number of labels: (624, 2)
```

In [8]:

```
# # Augmentation sequence
# seq = iaa.OneOf([
#     iaa.Fliplr(), # horizontal flips
#     iaa.Affine(rotate=40), # roatation
#     iaa.Multiply((1.2, 1.5))]) #random brightness


seq = iaa.Sequential([
    iaa.Fliplr(0.5), # horizontal flips
    iaa.Crop(percent=(0, 0.1)), # random crops
    # Small gaussian blur with random sigma between 0 and 0.5.
    # But we only blur about 50% of all images.
    iaa.Sometimes(
        0.5,
        iaa.GaussianBlur(sigma=(0, 0.5))
    ),
    # Strengthen or weaken the contrast in each image.
    iaa.LinearContrast((0.75, 1.5)),
    # Add gaussian noise.
    # For 50% of all images, we sample the noise once per pixel.
    # For the other 50% of all images, we sample the noise per pixel AND
    # channel. This can change the color (not only brightness) of the
    # pixels.
    iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5),
    # Make some images brighter and some darker.
    # In 20% of all cases, we sample the multiplier once per channel,
    # which can end up changing the color of the images.
    iaa.Multiply((0.8, 1.2), per_channel=0.2),
    # Apply affine transformations to each image.
    # Scale/zoom them, translate/move them, rotate them and shear them.
    iaa.Affine(
        scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},
        translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
        rotate=(-25, 25),
        shear=(-8, 8)
    )
], random_order=True) # apply augmenters in random order
```

In [9]:

```
def data_gen(data, batch_size):
    # Get total number of samples in the data
    n = len(data)
    steps = n//batch_size

    # Define two numpy arrays for containing batch data and labels
    batch_data = np.zeros((batch_size, 224, 224, 3), dtype=np.float32)
    batch_labels = np.zeros((batch_size,2), dtype=np.float32)

    # Get a numpy array of all the indices of the input data
    indices = np.arange(n)

    # Initialize a counter
    i =0
    while True:
        np.random.shuffle(indices)
        # Get the next batch
        count = 0
        next_batch = indices[(i*batch_size):(i+1)*batch_size]
        for j, idx in enumerate(next_batch):
            img_name = data.iloc[idx]['image']
            label = data.iloc[idx]['label']

            # one hot encoding
            encoded_label = to_categorical(label, num_classes=2)
            # read the image and resize
            img = cv2.imread(str(img_name))
            img = cv2.resize(img, (224,224))

            # check if it's grayscale
            if img.shape[2]==1:
                img = np.dstack([img, img, img])

            # cv2 reads in BGR mode by default
            orig_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            # normalize the image pixels
            orig_img = img.astype(np.float32)/255.

            batch_data[count] = orig_img
```

```
            batch_labels[count] = encoded_label

            # generating more samples of the undersampled class
            if label==0 and count < batch_size-2:
                aug_img1 = seq.augment_image(img)
                aug_img2 = seq.augment_image(img)
                aug_img1 = cv2.cvtColor(aug_img1, cv2.COLOR_BGR2RGB)
                aug_img2 = cv2.cvtColor(aug_img2, cv2.COLOR_BGR2RGB)
                aug_img1 = aug_img1.astype(np.float32)/255.
                aug_img2 = aug_img2.astype(np.float32)/255.

                batch_data[count+1] = aug_img1
                batch_labels[count+1] = encoded_label
                batch_data[count+2] = aug_img2
                batch_labels[count+2] = encoded_label
                count +=2

            else:
                count+=1

            if count==batch_size-1:
                break

        i+=1
        yield batch_data, batch_labels

        if i>=steps:
            i=0
```

In [10]:

```
def build_model():
    input_img = Input(shape=(224,224,3), name='ImageInput')
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1_1')(input_img)
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1_2')(x)
    x = MaxPooling2D((2,2), name='pool1')(x)

    x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2_1')(x)
    x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2_2')(x)
    x = MaxPooling2D((2,2), name='pool2')(x)

    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_1')(x)
    x = BatchNormalization(name='bn1')(x)
    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_2')(x)
    x = BatchNormalization(name='bn2')(x)
    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_3')(x)
    x = MaxPooling2D((2,2), name='pool3')(x)

    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_1')(x)
    x = BatchNormalization(name='bn3')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_2')(x)
    x = BatchNormalization(name='bn4')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_3')(x)
    x = MaxPooling2D((2,2), name='pool4')(x)

    x = Flatten(name='flatten')(x)
    x = Dense(1024, activation='relu', name='fc1')(x)
    x = Dropout(0.5, name='dropout1')(x)
    x = Dense(512, activation='relu', name='fc2')(x)
    x = Dropout(0.4, name='dropout2')(x)
    x = Dense(2, activation='softmax', name='fc3')(x)

    model = Model(inputs=input_img, outputs=x)
    return model
```

In [11]:

```
def build_modelSOTA():
    input_img = Input(shape=(224,224,3), name='ImageInput')
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1_1')(input_img)
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1_2')(x)
    x = MaxPooling2D((2,2), name='pool1')(x)

    x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2_1')(x)
    x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2_2')(x)
    x = MaxPooling2D((2,2), name='pool2')(x)

    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_1')(x)
    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_2')(x)
    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_3')(x)
    x = MaxPooling2D((2,2), name='pool3')(x)

    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_1')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_2')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_3')(x)
    x = MaxPooling2D((2,2), name='pool4')(x)

    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv5_1')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv5_2')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv5_3')(x)
    x = MaxPooling2D((2,2), name='pool5')(x)

    x = Flatten(name='flatten')(x)
    x = Dense(512, activation='relu', name='fc1')(x)
    x = BatchNormalization(name='bn1')(x)
    x = Dense(512, activation='relu', name='fc2')(x)
    x = BatchNormalization(name='bn2')(x)
    x = Dropout(0.5, name='dropout1')(x)
    x = Dense(2, activation='softmax', name='fc3')(x)

    model = Model(inputs=input_img, outputs=x)
    return model
```

In [12]:

```
model =  build_model()
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
ImageInput (InputLayer)      (None, 224, 224, 3)       0         
_________________________________________________________________
Conv1_1 (Conv2D)             (None, 224, 224, 64)      1792      
_________________________________________________________________
Conv1_2 (Conv2D)             (None, 224, 224, 64)      36928     
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 112, 112, 64)      0         
_________________________________________________________________
Conv2_1 (SeparableConv2D)    (None, 112, 112, 128)     8896      
_________________________________________________________________
Conv2_2 (SeparableConv2D)    (None, 112, 112, 128)     17664     
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 56, 56, 128)       0         
_________________________________________________________________
Conv3_1 (SeparableConv2D)    (None, 56, 56, 256)       34176     
_________________________________________________________________
bn1 (BatchNormalization)     (None, 56, 56, 256)       1024      
_________________________________________________________________
Conv3_2 (SeparableConv2D)    (None, 56, 56, 256)       68096     
_________________________________________________________________
bn2 (BatchNormalization)     (None, 56, 56, 256)       1024      
_________________________________________________________________
Conv3_3 (SeparableConv2D)    (None, 56, 56, 256)       68096     
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 28, 28, 256)       0         
_________________________________________________________________
Conv4_1 (SeparableConv2D)    (None, 28, 28, 512)       133888    
_________________________________________________________________
bn3 (BatchNormalization)     (None, 28, 28, 512)       2048      
_________________________________________________________________
Conv4_2 (SeparableConv2D)    (None, 28, 28, 512)       267264    
_________________________________________________________________
bn4 (BatchNormalization)     (None, 28, 28, 512)       2048      
_________________________________________________________________
Conv4_3 (SeparableConv2D)    (None, 28, 28, 512)       267264    
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 14, 14, 512)       0         
```

```
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
fc1 (Dense)                  (None, 1024)              102761472
_________________________________________________________________
dropout1 (Dropout)           (None, 1024)              0
_________________________________________________________________
fc2 (Dense)                  (None, 512)               524800
_________________________________________________________________
dropout2 (Dropout)           (None, 512)               0
_________________________________________________________________
fc3 (Dense)                  (None, 2)                 1026
=================================================================
Total params: 104,197,506
Trainable params: 104,194,434
Non-trainable params: 3,072
_________________________________________________________________
```

In [13]:

```
model.layers
print(len(model.layers))
```

```
25
```

In [14]:

```
# Open the VGG16 weight file
f = h5py.File('vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5', 'r')

# Select the layers for which you want to set weight.

w,b = f['block1_conv1']['block1_conv1_W_1:0'], f['block1_conv1']['block1_conv1_b_1:0']
model.layers[1].set_weights = [w,b]

w,b = f['block1_conv2']['block1_conv2_W_1:0'], f['block1_conv2']['block1_conv2_b_1:0']
model.layers[2].set_weights = [w,b]

w,b = f['block2_conv1']['block2_conv1_W_1:0'], f['block2_conv1']['block2_conv1_b_1:0']
model.layers[4].set_weights = [w,b]

w,b = f['block2_conv2']['block2_conv2_W_1:0'], f['block2_conv2']['block2_conv2_b_1:0']
model.layers[5].set_weights = [w,b]

f.close()
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
ImageInput (InputLayer)      (None, 224, 224, 3)       0         
_________________________________________________________________
Conv1_1 (Conv2D)             (None, 224, 224, 64)      1792      
_________________________________________________________________
Conv1_2 (Conv2D)             (None, 224, 224, 64)      36928     
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 112, 112, 64)      0         
_________________________________________________________________
Conv2_1 (SeparableConv2D)    (None, 112, 112, 128)     8896      
_________________________________________________________________
Conv2_2 (SeparableConv2D)    (None, 112, 112, 128)     17664     
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 56, 56, 128)       0         
_________________________________________________________________
Conv3_1 (SeparableConv2D)    (None, 56, 56, 256)       34176     
_________________________________________________________________
bn1 (BatchNormalization)     (None, 56, 56, 256)       1024      
_________________________________________________________________
Conv3_2 (SeparableConv2D)    (None, 56, 56, 256)       68096     
_________________________________________________________________
bn2 (BatchNormalization)     (None, 56, 56, 256)       1024      
_________________________________________________________________
Conv3_3 (SeparableConv2D)    (None, 56, 56, 256)       68096     
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 28, 28, 256)       0         
_________________________________________________________________
Conv4_1 (SeparableConv2D)    (None, 28, 28, 512)       133888    
_________________________________________________________________
bn3 (BatchNormalization)     (None, 28, 28, 512)       2048      
_________________________________________________________________
Conv4_2 (SeparableConv2D)    (None, 28, 28, 512)       267264    
_________________________________________________________________
bn4 (BatchNormalization)     (None, 28, 28, 512)       2048      
_________________________________________________________________
Conv4_3 (SeparableConv2D)    (None, 28, 28, 512)       267264    
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 14, 14, 512)       0         
```

```
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
fc1 (Dense)                  (None, 1024)              102761472
_________________________________________________________________
dropout1 (Dropout)           (None, 1024)              0
_________________________________________________________________
fc2 (Dense)                  (None, 512)               524800
_________________________________________________________________
dropout2 (Dropout)           (None, 512)               0
_________________________________________________________________
fc3 (Dense)                  (None, 2)                 1026
=================================================================
Total params: 104,197,506
Trainable params: 104,194,434
Non-trainable params: 3,072
_________________________________________________________________
```

In [15]:

```
opt = RMSprop(lr=0.0001, decay=1e-6)
#opt = RMSprop(lr=1e-4, decay=0.9) # SOTA
#opt = Adam(lr=0.0001, decay=1e-5)
#opt = Adam(lr=0.0001, decay=1e-5)
es = EarlyStopping(patience=15)
chkpt = ModelCheckpoint(filepath='best_modelvgg.hdf5', save_best_only=True, save_weights_only=True)
model.compile(loss='binary_crossentropy', metrics=['accuracy'],optimizer=opt)
```

In [16]:

```
batch_size = 16
nb_epochs = 30

# Get a train data generator
train_data_gen = data_gen(data=train_data, batch_size=batch_size)

# Define the number of training steps
nb_train_steps = train_data.shape[0]//batch_size

print("Number of training and validation steps: {} and {}".format(nb_train_steps, len(valid_data)))
```

```
Number of training and validation steps: 327 and 624
```

In [17]:

```
# # Fit the model
history = model.fit_generator(train_data_gen, epochs=nb_epochs, steps_per_epoch=nb_train_steps,
                              validation_data=(valid_data, valid_labels), callbacks=[chkpt],
                              class_weight={0:1.0, 1:0.4})
```

```
Epoch 1/30
327/327 [==============================] - 110s 338ms/step - loss: 0.2175 - accuracy: 0.7985 - val_loss: 0.9567 - val
_accuracy: 0.3750
Epoch 2/30
327/327 [==============================] - 107s 326ms/step - loss: 0.0870 - accuracy: 0.9455 - val_loss: 0.4487 - val
_accuracy: 0.8702
Epoch 3/30
327/327 [==============================] - 107s 326ms/step - loss: 0.0898 - accuracy: 0.9463 - val_loss: 0.8542 - val
_accuracy: 0.6266
Epoch 4/30
327/327 [==============================] - 107s 326ms/step - loss: 0.0677 - accuracy: 0.9589 - val_loss: 1.3455 - val
_accuracy: 0.4712
Epoch 5/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0655 - accuracy: 0.9602 - val_loss: 0.5250 - val
_accuracy: 0.8237
Epoch 6/30
327/327 [==============================] - 107s 326ms/step - loss: 0.0503 - accuracy: 0.9667 - val_loss: 0.3424 - val
_accuracy: 0.8766
Epoch 7/30
327/327 [==============================] - 107s 326ms/step - loss: 0.0493 - accuracy: 0.9708 - val_loss: 0.6804 - val
_accuracy: 0.8013
Epoch 8/30
327/327 [==============================] - 107s 326ms/step - loss: 0.0404 - accuracy: 0.9776 - val_loss: 0.5727 - val
_accuracy: 0.8638
Epoch 9/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0413 - accuracy: 0.9748 - val_loss: 0.2380 - val
_accuracy: 0.9167
Epoch 10/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0365 - accuracy: 0.9794 - val_loss: 0.2334 - val
_accuracy: 0.9038
Epoch 11/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0326 - accuracy: 0.9820 - val_loss: 0.2114 - val
_accuracy: 0.9103
Epoch 12/30
327/327 [==============================] - 107s 326ms/step - loss: 0.0319 - accuracy: 0.9820 - val_loss: 1.0568 - val
_accuracy: 0.7965
Epoch 13/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0299 - accuracy: 0.9845 - val_loss: 0.5757 - val
_accuracy: 0.8478
Epoch 14/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0274 - accuracy: 0.9843 - val_loss: 1.5206 - val
```

```
_accuracy: 0.7644
Epoch 15/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0303 - accuracy: 0.9849 - val_loss: 0.9085 - val
_accuracy: 0.8397
Epoch 16/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0298 - accuracy: 0.9849 - val_loss: 0.2680 - val
_accuracy: 0.8910
Epoch 17/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0212 - accuracy: 0.9878 - val_loss: 0.6676 - val
_accuracy: 0.8878
Epoch 18/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0247 - accuracy: 0.9876 - val_loss: 0.5556 - val
_accuracy: 0.8846
Epoch 19/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0192 - accuracy: 0.9885 - val_loss: 0.6113 - val
_accuracy: 0.8766
Epoch 20/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0232 - accuracy: 0.9889 - val_loss: 0.3990 - val
_accuracy: 0.9183
Epoch 21/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0180 - accuracy: 0.9916 - val_loss: 0.4197 - val
_accuracy: 0.9215
Epoch 22/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0194 - accuracy: 0.9891 - val_loss: 0.6992 - val
_accuracy: 0.8814
Epoch 23/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0169 - accuracy: 0.9933 - val_loss: 1.2115 - val
_accuracy: 0.7772
Epoch 24/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0180 - accuracy: 0.9925 - val_loss: 0.7636 - val
_accuracy: 0.8910
Epoch 25/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0138 - accuracy: 0.9918 - val_loss: 0.8445 - val
_accuracy: 0.8942
Epoch 26/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0174 - accuracy: 0.9904 - val_loss: 0.3637 - val
_accuracy: 0.8974
Epoch 27/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0224 - accuracy: 0.9912 - val_loss: 1.0082 - val
_accuracy: 0.8381
Epoch 28/30
327/327 [==============================] - 106s 324ms/step - loss: 0.0177 - accuracy: 0.9922 - val_loss: 0.9656 - val
```

```
_accuracy: 0.8686
Epoch 29/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0160 - accuracy: 0.9922 - val_loss: 0.3773 - val
_accuracy: 0.8958
Epoch 30/30
327/327 [==============================] - 106s 325ms/step - loss: 0.0119 - accuracy: 0.9935 - val_loss: 0.7299 - val
_accuracy: 0.8846
```

In [18]:

```
#print(history.history)
```

In [19]:

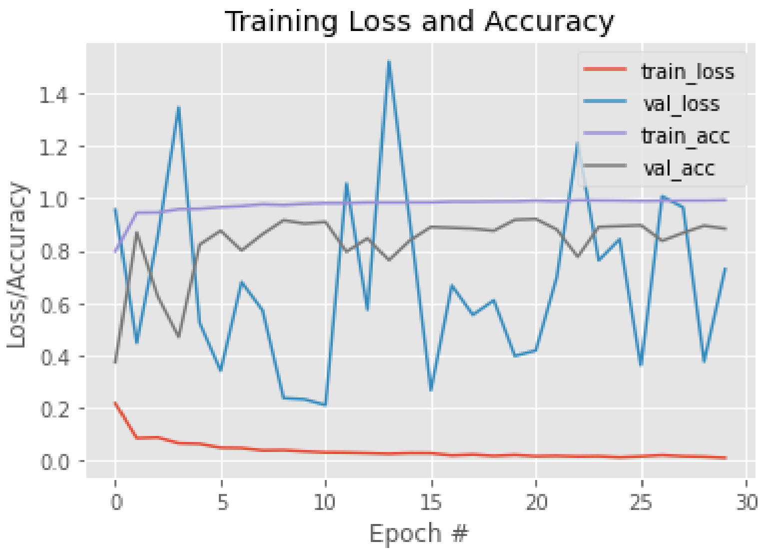
```
#print(history)
```

In [20]:

```
def showGraph(Histroy, epochs):
    # plot the training loss and accuracy
    plt.style.use("ggplot")
    plt.figure()
    plt.plot(np.arange(0, epochs), Histroy.history["loss"], label="train_loss")
    plt.plot(np.arange(0, epochs), Histroy.history["val_loss"], label="val_loss")
    plt.plot(np.arange(0, epochs), Histroy.history["accuracy"], label="train_acc")
    plt.plot(np.arange(0, epochs), Histroy.history["val_accuracy"], label="val_acc")
    plt.title("Training Loss and Accuracy")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss/Accuracy")
    plt.legend()
    plt.show()
```

In [21]:

```
showGraph(history, nb_epochs)
```



In [22]:

```
# # Visualize training history
# from keras.models import Sequential
# from keras.layers import Dense
# import matplotlib.pyplot as plt
# import numpy


# # summarize history for accuracy

# plt.plot(history.history['val_accuracy'])
# plt.plot(history.history['accuracy'])
# plt.title('model accuracy')
# plt.ylabel('accuracy')
# plt.xlabel('epoch')
# plt.legend(['train', 'test'], loc='upper left')
# plt.show()
# # summarize history for loss
# plt.plot(history.history['loss'])
# plt.plot(history.history['val_loss'])
# plt.title('model loss')
# plt.ylabel('loss')
# plt.xlabel('epoch')
# plt.legend(['train', 'test'], loc='upper left')
# plt.show()


# # summarize history for loss
# plt.plot(history.history['loss'])
# plt.plot(history.history['val_loss'])
# plt.plot(history.history['val_accuracy'])
# plt.plot(history.history['accuracy'])
# plt.legend(['loss', 'val_loss','val_accuracy','accuracy' ], loc='upper left')
# plt.show()
```

In [23]:

```
# Load the model weights
model.load_weights("best_modelvgg.hdf5")
```

In [24]:

```
# Preparing test data
normal_cases_dir = test_dir / 'NORMAL'
pneumonia_cases_dir = test_dir / 'PNEUMONIA'

normal_cases = normal_cases_dir.glob('*.*g')
pneumonia_cases = pneumonia_cases_dir.glob('*.*g')

test_data = []
test_labels = []

for img in normal_cases:
    img = cv2.imread(str(img))
    img = cv2.resize(img, (224,224))
    if img.shape[2] ==1:
        img = np.dstack([img, img, img])
    else:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)/255.
    label = to_categorical(0, num_classes=2)
    test_data.append(img)
    test_labels.append(label)

for img in pneumonia_cases:
    img = cv2.imread(str(img))
    img = cv2.resize(img, (224,224))
    if img.shape[2] ==1:
        img = np.dstack([img, img, img])
    else:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)/255.
    label = to_categorical(1, num_classes=2)
    test_data.append(img)
    test_labels.append(label)


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

```
print("Total number of test examples: ", test_data.shape)
print("Total number of labels: ", test_labels.shape)
```

```
Total number of test examples: (624, 224, 224, 3)
Total number of labels: (624, 2)
```

In [25]:

```
# Evaluation on test dataset
test_loss, test_score = model.evaluate(test_data, test_labels, batch_size=16)
print("Loss on test set: ", test_loss)
print("Accuracy on test set: ", test_score)
```

```
624/624 [==============================] - 2s 4ms/step
Loss on test set:  0.2113733634304924
Accuracy on test set:  0.9102563858032227
```

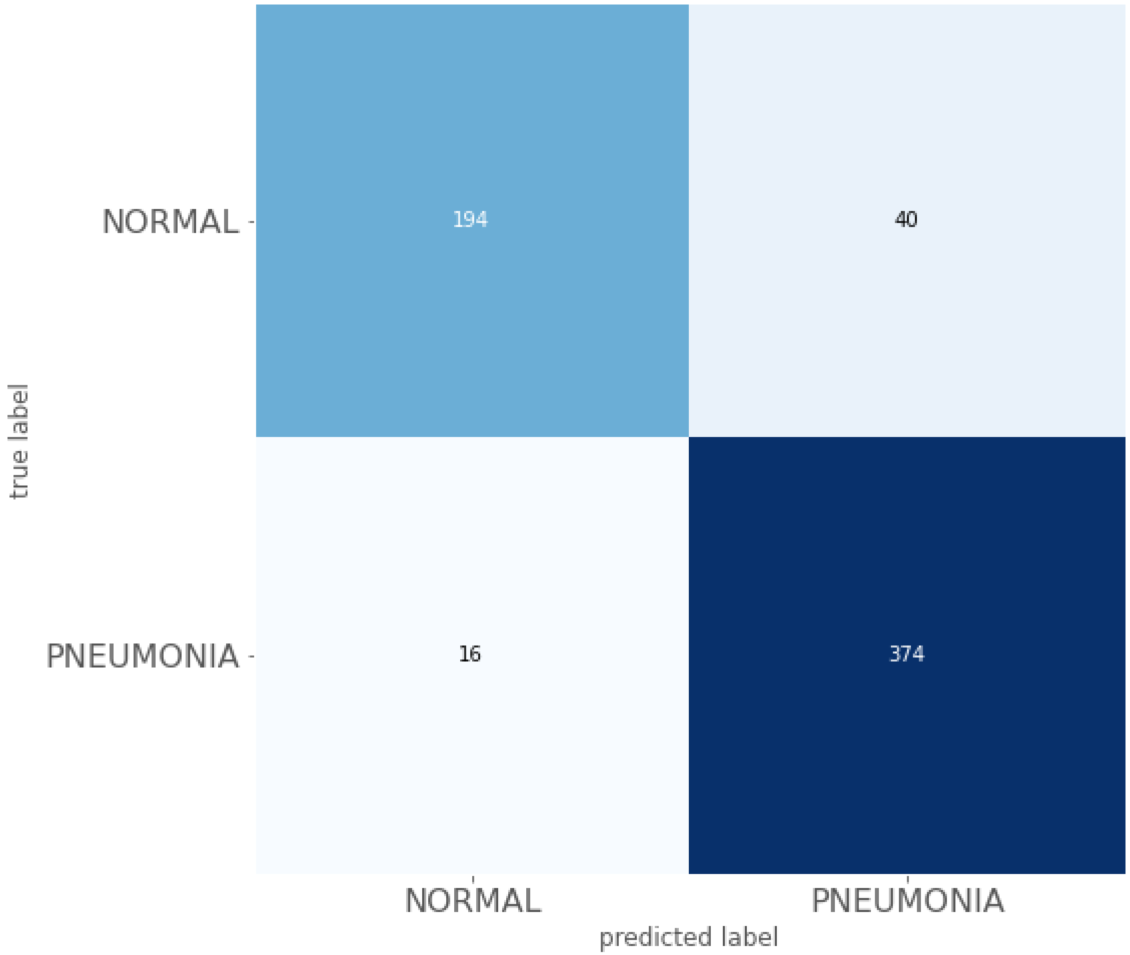
In [26]:

```
# Get the predictions on test set
preds = model.predict(test_data, batch_size=16)
preds = np.squeeze((preds > 0.5).astype('int'))
orig = test_labels.astype('int')
#print(preds)
#print(orig)

# Get predictions
preds = model.predict(test_data, batch_size=16)
preds = np.argmax(preds, axis=-1)

# Original labels
orig = np.argmax(test_labels, axis=-1)

#print(orig)
#print(preds)
```

In [27]:

```
# Get the confusion matrix
cm  = confusion_matrix(orig, preds)
plt.figure()
plot_confusion_matrix
plot_confusion_matrix(cm,figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.xticks(range(2), ['NORMAL', 'PNEUMONIA'], fontsize=16)
plt.yticks(range(2), ['NORMAL', 'PNEUMONIA'], fontsize=16)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [28]:

```
# Calculate Precision and Recall
tn, fp, fn, tp = cm.ravel()

precision = tp/(tp+fp)
recall = tp/(tp+fn)


print("Recall of the model is {:.2f}".format(recall))
print("Precision of the model is {:.2f}".format(precision))
```

```
Recall of the model is 0.96
Precision of the model is 0.90
```

In [29]:

```
model.save("SOTA_V_SOTA_STRUCTURE.h5")
```

Fine tuning

In [30]:

```
def showGraph(Histroy, epochs):
    # plot the training loss and accuracy
    plt.style.use("ggplot")
    plt.figure()
    plt.plot(np.arange(0, epochs), Histroy.history["loss"], label="train_loss")
    plt.plot(np.arange(0, epochs), Histroy.history["val_loss"], label="val_loss")
    plt.plot(np.arange(0, epochs), Histroy.history["accuracy"], label="train_acc")
    plt.plot(np.arange(0, epochs), Histroy.history["val_accuracy"], label="val_acc")
    plt.title("Training Loss and Accuracy")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss/Accuracy")
    plt.legend()
    plt.show()
```

Step 1 & 2: # Freezing all the layers & added a new fully connected layer

In [31]:

```
model = build_model()
# Open the VGG16 weight file
f = h5py.File('vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5', 'r')

# Select the layers for which you want to set weight.

w,b = f['block1_conv1']['block1_conv1_W_1:0'], f['block1_conv1']['block1_conv1_b_1:0']
model.layers[1].set_weights = [w,b]

w,b = f['block1_conv2']['block1_conv2_W_1:0'], f['block1_conv2']['block1_conv2_b_1:0']
model.layers[2].set_weights = [w,b]

w,b = f['block2_conv1']['block2_conv1_W_1:0'], f['block2_conv1']['block2_conv1_b_1:0']
model.layers[4].set_weights = [w,b]

w,b = f['block2_conv2']['block2_conv2_W_1:0'], f['block2_conv2']['block2_conv2_b_1:0']
model.layers[5].set_weights = [w,b]

f.close()
model.summary()
model.trainable = False
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
ImageInput (InputLayer)      (None, 224, 224, 3)       0         
_________________________________________________________________
Conv1_1 (Conv2D)             (None, 224, 224, 64)      1792      
_________________________________________________________________
Conv1_2 (Conv2D)             (None, 224, 224, 64)      36928     
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 112, 112, 64)      0         
_________________________________________________________________
Conv2_1 (SeparableConv2D)    (None, 112, 112, 128)     8896      
_________________________________________________________________
Conv2_2 (SeparableConv2D)    (None, 112, 112, 128)     17664     
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 56, 56, 128)       0         
_________________________________________________________________
Conv3_1 (SeparableConv2D)    (None, 56, 56, 256)       34176     
_________________________________________________________________
bn1 (BatchNormalization)     (None, 56, 56, 256)       1024      
_________________________________________________________________
Conv3_2 (SeparableConv2D)    (None, 56, 56, 256)       68096     
_________________________________________________________________
bn2 (BatchNormalization)     (None, 56, 56, 256)       1024      
_________________________________________________________________
Conv3_3 (SeparableConv2D)    (None, 56, 56, 256)       68096     
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 28, 28, 256)       0         
_________________________________________________________________
Conv4_1 (SeparableConv2D)    (None, 28, 28, 512)       133888    
_________________________________________________________________
bn3 (BatchNormalization)     (None, 28, 28, 512)       2048      
_________________________________________________________________
Conv4_2 (SeparableConv2D)    (None, 28, 28, 512)       267264    
_________________________________________________________________
bn4 (BatchNormalization)     (None, 28, 28, 512)       2048      
_________________________________________________________________
Conv4_3 (SeparableConv2D)    (None, 28, 28, 512)       267264    
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 14, 14, 512)       0         
```

```
________________________________________________________________
flatten (Flatten)            (None, 100352)            0
________________________________________________________________
fc1 (Dense)                  (None, 1024)              102761472
________________________________________________________________
dropout1 (Dropout)           (None, 1024)              0
________________________________________________________________
fc2 (Dense)                  (None, 512)               524800
________________________________________________________________
dropout2 (Dropout)           (None, 512)               0
________________________________________________________________
fc3 (Dense)                  (None, 2)                 1026
=================================================================
Total params: 104,197,506
Trainable params: 104,194,434
Non-trainable params: 3,072
________________________________________________________________
```

Step 3: Train the weights on the new FC layer.

In [34]:

```
#opt = RMSprop(lr=1e-3, decay=0.9)
#opt = Adam(lr=0.0001, decay=1e-5)
#es = EarlyStopping(patience=10)
#chkpt = ModelCheckpoint(filepath='best_modelvgg.hdf5', save_best_only=True, save_weights_only=True)
#model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer=opt)

batch_size = 16
nb_epochs = 30

# Get a train data generator
train_data_gen = data_gen(data=train_data, batch_size=batch_size)

# Define the number of training steps
nb_train_steps = train_data.shape[0]//batch_size

opt = RMSprop(lr=0.0001, decay=1e-6)
#opt = RMSprop(lr=1e-4, decay=0.9) # SOTA
#opt = Adam(lr=0.0001, decay=1e-5)
#opt = Adam(lr=0.0001, decay=1e-5)
es = EarlyStopping(patience=15)
chkpt = ModelCheckpoint(filepath='best_modelvgg.hdf5', save_best_only=True, save_weights_only=True)
model.compile(loss='binary_crossentropy', metrics=['accuracy'],optimizer=opt)

print("Number of training and validation steps: {} and {}".format(nb_train_steps, len(valid_data)))


# # Fit the model
checkpoint = ModelCheckpoint("PreFineTunebestVGG_StateOfTheArtData.h5", monitor="val_loss", mode="min", save_best_only=True, verbo
se=1)
history = model.fit_generator(train_data_gen, epochs=nb_epochs, steps_per_epoch=nb_train_steps,
                              validation_data=(valid_data, valid_labels),
                              callbacks=[es, checkpoint],
                              class_weight={0:1.0, 1:0.4})
```

```
Number of training and validation steps: 327 and 624
Epoch 1/30
327/327 [==============================] - 63s 192ms/step - loss: 0.4607 - accuracy: 0.5448 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00001: val_loss improved from inf to 0.69315, saving model to PreFineTunebestVGG_StateOfTheArtData.h5
Epoch 2/30
327/327 [==============================] - 59s 181ms/step - loss: 0.4538 - accuracy: 0.5528 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00002: val_loss did not improve from 0.69315
Epoch 3/30
327/327 [==============================] - 61s 186ms/step - loss: 0.4649 - accuracy: 0.5387 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00003: val_loss did not improve from 0.69315
Epoch 4/30
327/327 [==============================] - 59s 182ms/step - loss: 0.4606 - accuracy: 0.5424 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00004: val_loss did not improve from 0.69315
Epoch 5/30
327/327 [==============================] - 59s 181ms/step - loss: 0.4628 - accuracy: 0.5459 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00005: val_loss did not improve from 0.69315
Epoch 6/30
327/327 [==============================] - 58s 179ms/step - loss: 0.4648 - accuracy: 0.5335 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00006: val_loss did not improve from 0.69315
Epoch 7/30
327/327 [==============================] - 58s 178ms/step - loss: 0.4597 - accuracy: 0.5401 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00007: val_loss did not improve from 0.69315
Epoch 8/30
327/327 [==============================] - 58s 177ms/step - loss: 0.4577 - accuracy: 0.5476 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00008: val_loss did not improve from 0.69315
```

```
Epoch 9/30
327/327 [==============================] - 59s 182ms/step - loss: 0.4644 - accuracy: 0.5282 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00009: val_loss did not improve from 0.69315
Epoch 10/30
327/327 [==============================] - 58s 178ms/step - loss: 0.4574 - accuracy: 0.5399 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00010: val_loss did not improve from 0.69315
Epoch 11/30
327/327 [==============================] - 57s 176ms/step - loss: 0.4609 - accuracy: 0.5384 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00011: val_loss did not improve from 0.69315
Epoch 12/30
327/327 [==============================] - 56s 172ms/step - loss: 0.4509 - accuracy: 0.5571 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00012: val_loss did not improve from 0.69315
Epoch 13/30
327/327 [==============================] - 58s 176ms/step - loss: 0.4596 - accuracy: 0.5364 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00013: val_loss did not improve from 0.69315
Epoch 14/30
327/327 [==============================] - 58s 176ms/step - loss: 0.4587 - accuracy: 0.5399 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00014: val_loss did not improve from 0.69315
Epoch 15/30
327/327 [==============================] - 58s 178ms/step - loss: 0.4688 - accuracy: 0.5263 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00015: val_loss did not improve from 0.69315
Epoch 16/30
327/327 [==============================] - 58s 178ms/step - loss: 0.4622 - accuracy: 0.5420 - val_loss: 0.6931 - val_
accuracy: 0.5000

Epoch 00016: val_loss did not improve from 0.69315
```

Step 4: Unfreeze the trainable weights on some of the convolutional layers in the base network.

In [35]:

```
model.trainable = True
set_trainable = False
for layer in model.layers:
    if layer.name in ['block5_conv1']:
            set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

In [36]:

```
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
ImageInput (InputLayer)      (None, 224, 224, 3)       0
_________________________________________________________________
Conv1_1 (Conv2D)             (None, 224, 224, 64)      1792
_________________________________________________________________
Conv1_2 (Conv2D)             (None, 224, 224, 64)      36928
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 112, 112, 64)      0
_________________________________________________________________
Conv2_1 (SeparableConv2D)    (None, 112, 112, 128)     8896
_________________________________________________________________
Conv2_2 (SeparableConv2D)    (None, 112, 112, 128)     17664
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 56, 56, 128)       0
_________________________________________________________________
Conv3_1 (SeparableConv2D)    (None, 56, 56, 256)       34176
_________________________________________________________________
bn1 (BatchNormalization)     (None, 56, 56, 256)       1024
_________________________________________________________________
Conv3_2 (SeparableConv2D)    (None, 56, 56, 256)       68096
_________________________________________________________________
bn2 (BatchNormalization)     (None, 56, 56, 256)       1024
_________________________________________________________________
Conv3_3 (SeparableConv2D)    (None, 56, 56, 256)       68096
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 28, 28, 256)       0
_________________________________________________________________
Conv4_1 (SeparableConv2D)    (None, 28, 28, 512)       133888
_________________________________________________________________
bn3 (BatchNormalization)     (None, 28, 28, 512)       2048
_________________________________________________________________
Conv4_2 (SeparableConv2D)    (None, 28, 28, 512)       267264
_________________________________________________________________
bn4 (BatchNormalization)     (None, 28, 28, 512)       2048
_________________________________________________________________
Conv4_3 (SeparableConv2D)    (None, 28, 28, 512)       267264
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 14, 14, 512)       0
```

```
_________________________________________________________________
flatten (Flatten)            (None, 100352)            0
_________________________________________________________________
fc1 (Dense)                  (None, 1024)              102761472
_________________________________________________________________
dropout1 (Dropout)           (None, 1024)              0
_________________________________________________________________
fc2 (Dense)                  (None, 512)               524800
_________________________________________________________________
dropout2 (Dropout)           (None, 512)               0
_________________________________________________________________
fc3 (Dense)                  (None, 2)                 1026
=================================================================
Total params: 104,197,506
Trainable params: 0
Non-trainable params: 104,197,506
_________________________________________________________________
```

In [37]:

```
# baseModel = VGG16(weights="imagenet", include_top=False,
# input_tensor=Input(shape=(224, 224, 3)))


# headModel = baseModel.output
# headModel = Flatten(name="flatten")(headModel)
# headModel = Dense(512, activation="relu")(headModel)
# headModel = Dropout(0.5)(headModel)
# headModel = Dense(2, activation="softmax")(headModel)


# model = Model(inputs=baseModel.input, outputs=headModel)
```

In [ ]:

```
#opt = RMSprop(lr=1e-4, decay=0.9)
#opt = Adam(lr=0.0001, decay=1e-5)
#es = EarlyStopping(patience=10)
#chkpt = ModelCheckpoint(filepath='best_modelvgg.hdf5', save_best_only=True, save_weights_only=True)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer=opt)

batch_size = 16
nb_epochs = 30

# Get a train data generator
train_data_gen = data_gen(data=train_data, batch_size=batch_size)

# Define the number of training steps
nb_train_steps = train_data.shape[0]//batch_size

print("Number of training and validation steps: {} and {}".format(nb_train_steps, len(valid_data)))


# # Fit the model
checkpoint = ModelCheckpoint("PreFineTunebestVGG_StateOfTheArtData.h5", monitor="val_loss", mode="min", save_best_only=True, verbo
se=1)
history = model.fit_generator(train_data_gen, epochs=nb_epochs, steps_per_epoch=nb_train_steps,
                              validation_data=(valid_data, valid_labels),
                              callbacks=[checkpoint],
                              class_weight={0:1.0, 1:0.4})
```

```
Number of training and validation steps: 327 and 624
Epoch 1/30
327/327 [==============================] - 63s 193ms/step - loss: 0.4573 - accuracy: 0.5545 - val_loss: 0.6931 - val_
accuracy: 0.3750

Epoch 00001: val_loss improved from inf to 0.69315, saving model to PreFineTunebestVGG_StateOfTheArtData.h5
Epoch 2/30
327/327 [==============================] - 62s 189ms/step - loss: 0.4566 - accuracy: 0.5459 - val_loss: 0.6931 - val_
accuracy: 0.3750

Epoch 00002: val_loss did not improve from 0.69315
Epoch 3/30
327/327 [==============================] - 61s 186ms/step - loss: 0.4618 - accuracy: 0.5357 - val_loss: 0.6931 - val_
accuracy: 0.3750

Epoch 00003: val_loss did not improve from 0.69315
Epoch 4/30
327/327 [==============================] - 59s 180ms/step - loss: 0.4536 - accuracy: 0.5436 - val_loss: 0.6931 - val_
accuracy: 0.3750

Epoch 00004: val_loss did not improve from 0.69315
Epoch 5/30
327/327 [==============================] - 60s 184ms/step - loss: 0.4637 - accuracy: 0.5436 - val_loss: 0.6931 - val_
accuracy: 0.3750

Epoch 00005: val_loss did not improve from 0.69315
Epoch 6/30
327/327 [==============================] - 60s 184ms/step - loss: 0.4629 - accuracy: 0.5346 - val_loss: 0.6931 - val_
accuracy: 0.3750

Epoch 00006: val_loss did not improve from 0.69315
Epoch 7/30
188/327 [================>.............] - ETA: 23s - loss: 0.4488 - accuracy: 0.5545
```

In [ ]:

```
# Get the predictions on test set
preds = model.predict(test_data, batch_size=16)
preds = np.squeeze((preds > 0.5).astype('int'))
orig = test_labels.astype('int')
#print(preds)
#print(orig)

# Get predictions
preds = model.predict(test_data, batch_size=16)
preds = np.argmax(preds, axis=-1)

# Original labels
orig = np.argmax(test_labels, axis=-1)

#print(orig)
#print(preds)
```

In [ ]:

```
# Get the confusion matrix
cm  = confusion_matrix(orig, preds)
plt.figure()
plot_confusion_matrix
plot_confusion_matrix(cm,figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.xticks(range(2), ['NORMAL', 'PNEUMONIA'], fontsize=16)
plt.yticks(range(2), ['NORMAL', 'PNEUMONIA'], fontsize=16)
plt.show()
```

In [ ]:

```
# Calculate Precision and Recall
tn, fp, fn, tp = cm.ravel()

precision = tp/(tp+fp)
recall = tp/(tp+fn)


print("Recall of the model is {:.2f}".format(recall))
print("Precision of the model is {:.2f}".format(precision))
```

In [ ]:

```
model.save("SOTA_V_SOTA_STRUCTURE_fine_tuned.h5")
```

In [ ]:

```
```

In [ ]:

```
```