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
from keras import backend as K
color = sns.color_palette()
%matplotlib inline
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

# Input data files are available in the "../input/" directory.
```

```
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input direc
print(os.listdir("input"))
```

```
Using TensorFlow backend.

[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 13442577703570928201
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 5390598144
locality {
  bus_id: 1
  links {
  }
}
incarnation: 7882585801626886672
physical_device_desc: "device: 0, name: GeForce GTX 1060, pci bus id: 0000:01:00.0, compute capability: 6.
1"
]
['xray-best-model', 'chest_xray', 'vgg16']
```

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.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)

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

# Define a tensorflow session with above session configs
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)

# Set the session in keras
K.set_session(sess)

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

In [3]:
```
# Define path to the data directory
data_dir = Path('input/chest_xray')

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

# Path to validation directory
val_dir = data_dir / 'val'

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

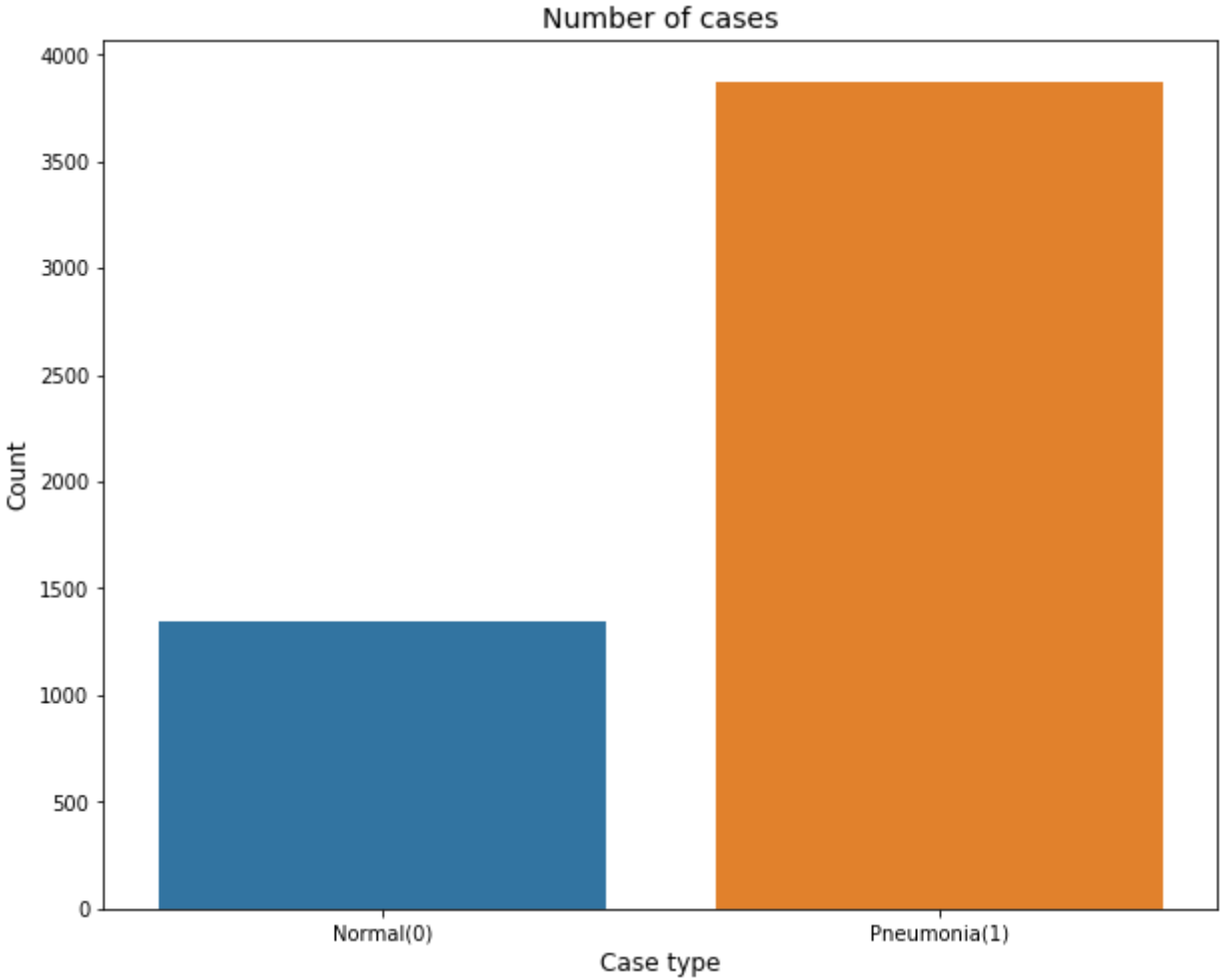
In [4]:
```
# Get the path to the normal and pneumonia sub-directories
normal_cases_dir = train_dir / 'NORMAL'
pneumonia_cases_dir = train_dir / 'PNEUMONIA'

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

# 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()
```

Out[4]:

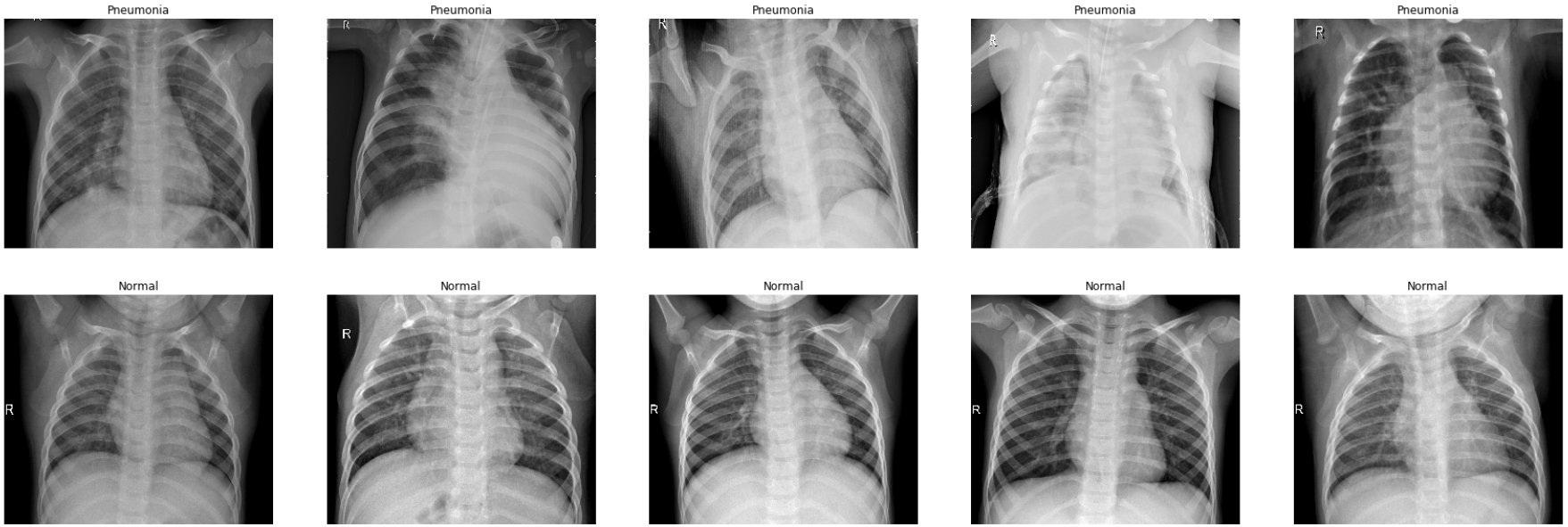
	image	label
0	input/chest_xray/train/NORMAL/NORMAL2-IM-1025-...	0
1	input/chest_xray/train/NORMAL/IM-0348-0001.jpeg	0
2	input/chest_xray/train/PNEUMONIA/person1468_vi...	1
3	input/chest_xray/train/PNEUMONIA/person62_bact...	1
4	input/chest_xray/train/PNEUMONIA/person1503_vi...	1

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(0)', 'Pneumonia(1)'])
plt.show()
```

```
1    3875
0    1341
Name: label, dtype: int64
```


Number of cases
4000
3500
3000
2500
2000
1500
1000
500
0
Count
Normal(0)
Pneumonia(1)
Case type

In [6]:
```
# Get few samples for both the classes
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 [7]:
```
# Get the path to the sub-directories
normal_cases_dir = val_dir / 'NORMAL'
pneumonia_cases_dir = val_dir / 'PNEUMONIA'

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

# 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
# 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:  (16, 224, 224, 3)
Total number of labels: (16, 2)
```

In [8]:
```
# Augmentation sequence
seq = iaa.OneOf([
    iaa.Fliplr(), # horizontal flips
    iaa.Affine(rotate=20), # roatation
    iaa.Multiply((1.2, 1.5))]) #random brightness
```

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(0.7, name='dropout1')(x)
    x = Dense(512, activation='relu', name='fc2')(x)
    x = Dropout(0.5, name='dropout2')(x)
    x = Dense(2, activation='softmax', name='fc3')(x)

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

In [11]:
```
model =  build_model()
model.summary()
```

```
_________________________________________________________________
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 [12]:
```
# Open the VGG16 weight file
f = h5py.File('input/vgg16/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()
```

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

In [ ]:
```

```

In [14]:
```
batch_size = 16
nb_epochs = 20

# 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)))
```

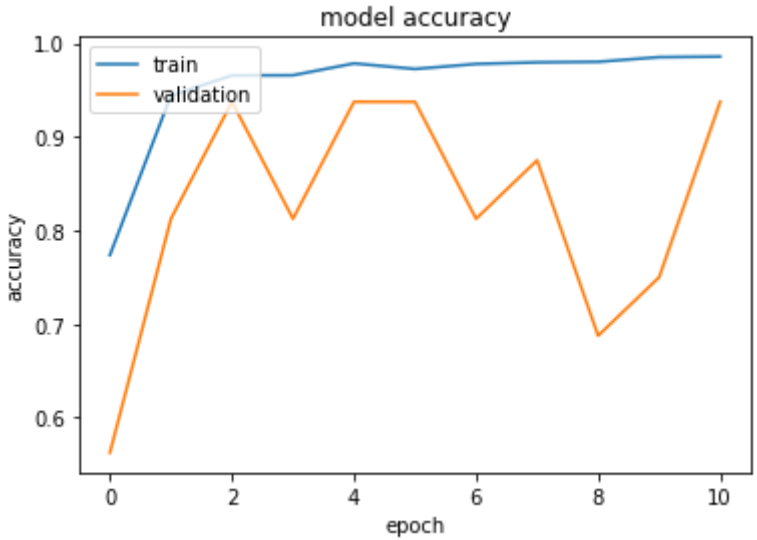
```
Number of training and validation steps: 326 and 16
```

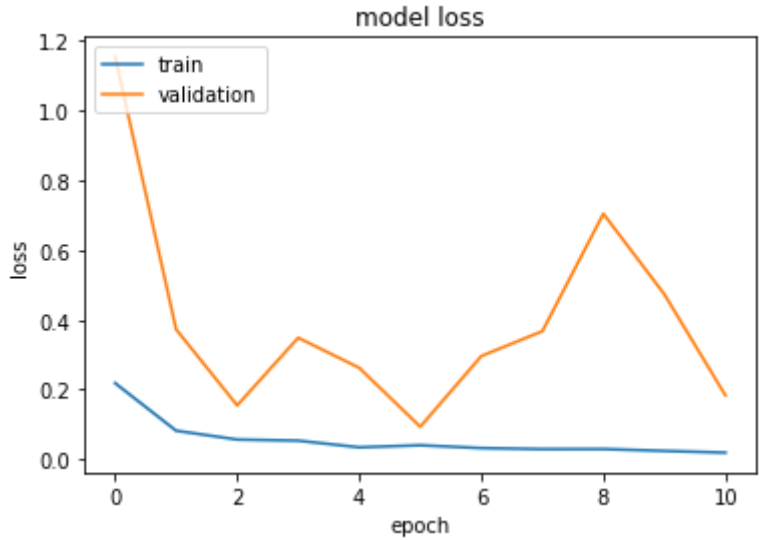
In [15]:
```
# # Fit the model
history = model.fit_generator(train_data_gen, epochs=nb_epochs, steps_per_epoch=nb_train_steps,
validation_data=(valid_data, valid_labels),callbacks=[es, chkpt],
class_weight={0:1.0, 1:0.4})
```

```
Epoch 1/20
326/326 [==============================] - 155s 475ms/step - loss: 0.2187 - acc: 0.7737 - val_loss: 1.1545
- val_acc: 0.5625
Epoch 2/20
326/326 [==============================] - 150s 459ms/step - loss: 0.0824 - acc: 0.9442 - val_loss: 0.3728
- val_acc: 0.8125
Epoch 3/20
326/326 [==============================] - 150s 461ms/step - loss: 0.0572 - acc: 0.9657 - val_loss: 0.1550
- val_acc: 0.9375
Epoch 4/20
326/326 [==============================] - 149s 457ms/step - loss: 0.0539 - acc: 0.9659 - val_loss: 0.3488
- val_acc: 0.8125
Epoch 5/20
326/326 [==============================] - 149s 458ms/step - loss: 0.0349 - acc: 0.9785 - val_loss: 0.2626
- val_acc: 0.9375
Epoch 6/20
326/326 [==============================] - 149s 458ms/step - loss: 0.0408 - acc: 0.9726 - val_loss: 0.0938
- val_acc: 0.9375
Epoch 7/20
326/326 [==============================] - 150s 459ms/step - loss: 0.0326 - acc: 0.9780 - val_loss: 0.2964
- val_acc: 0.8125
Epoch 8/20
326/326 [==============================] - 150s 461ms/step - loss: 0.0298 - acc: 0.9797 - val_loss: 0.3676
- val_acc: 0.8750
Epoch 9/20
326/326 [==============================] - 150s 460ms/step - loss: 0.0300 - acc: 0.9803 - val_loss: 0.7042
- val_acc: 0.6875
Epoch 10/20
326/326 [==============================] - 149s 459ms/step - loss: 0.0246 - acc: 0.9852 - val_loss: 0.4732
- val_acc: 0.7500
Epoch 11/20
326/326 [==============================] - 147s 450ms/step - loss: 0.0196 - acc: 0.9858 - val_loss: 0.1837
- val_acc: 0.9375
```

In [16]:
```
# Load the model weights
#model.load_weights("input/xray-best-model/best_model/best_model.hdf5")
print(history.history.keys())
#  "Accuracy"
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# "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', 'validation'], loc='upper left')
plt.show()
```

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```




model loss
train
validation
loss
epoch
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10

In [17]:
```
# Preparing test data
normal_cases_dir = test_dir / 'NORMAL'
pneumonia_cases_dir = test_dir / 'PNEUMONIA'

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

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 [18]:
```
# 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)
```

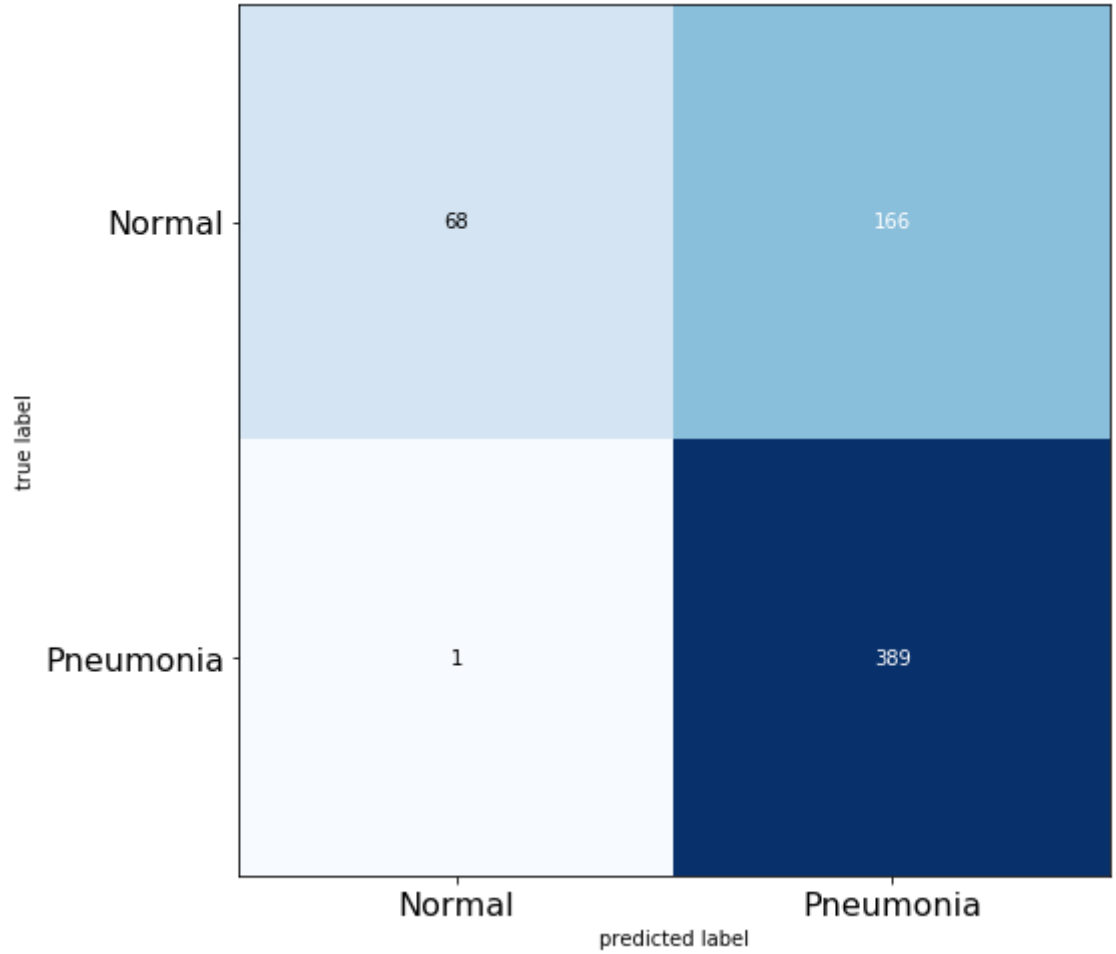
```
624/624 [==============================] - 4s 6ms/step
Loss on test set:  1.6069589958322188
Accuracy on test set:  0.7323717948717948
```

In [19]:
```
# Get predictions
preds = model.predict(test_data, batch_size=16)
preds = np.argmax(preds, axis=-1)

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

print(orig_test_labels.shape)
print(preds.shape)
```

```
(624,)
(624,)
```

In [20]:
```
# Get the confusion matrix
cm  = confusion_matrix(orig_test_labels, preds)
plt.figure()
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 [21]:
```
# 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 1.00
Precision of the model is 0.70
```

In [22]:
```
from PIL import Image
import numpy as np
from skimage import transform
def load(filename):
   np_image = Image.open(filename)
   np_image = np.array(np_image).astype('float32')/255
   np_image = transform.resize(np_image, (224,224, 3))
   np_image = np.expand_dims(np_image, axis=0)
   return np_image

image = load('N1.png')
y = model.predict(image)
if y[0][0] <0.00001:
    print ("Normal")
else:
    print ("Pnemonia")
```

```
/home/anurag/.conda/envs/test/lib/python3.6/site-packages/skimage/transform/_warps.py:105: UserWarning: The
default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
/home/anurag/.conda/envs/test/lib/python3.6/site-packages/skimage/transform/_warps.py:110: UserWarning: Ant
i-aliasing will be enabled by default in skimage 0.15 to avoid aliasing artifacts when down-sampling image
s.
  warn("Anti-aliasing will be enabled by default in skimage 0.15 to "

Normal
```

In [27]:
```
def load(filename):
   np_image = Image.open(filename)
   np_image = np.array(np_image).astype('float32')/255
   np_image = transform.resize(np_image, (224,224, 3))
   np_image = np.expand_dims(np_image, axis=0)
   return np_image

image = load('N2.png')
y = model.predict(image)
if y[0][0] <0.00001:
    print ("Normal")
else:
    print ("Pnemonia")
```

```
Normal
```

In [26]:
```
def load(filename):
   np_image = Image.open(filename)
   np_image = np.array(np_image).astype('float32')/255
   np_image = transform.resize(np_image, (224,224, 3))
   np_image = np.expand_dims(np_image, axis=0)
   return np_image

image = load('P1.png')
y = model.predict(image)
if y[0][0] <0.00001:
    print ("Normal")
else:
    print ("Pnemonia")
```

```
Pnemonia
```

In [25]:
```
def load(filename):
   np_image = Image.open(filename)
   np_image = np.array(np_image).astype('float32')/255
   np_image = transform.resize(np_image, (224,224, 3))
   np_image = np.expand_dims(np_image, axis=0)
   return np_image

image = load('P2.png')
y = model.predict(image)
if y[0][0] <0.00001:
    print ("Normal")
else:
    print ("Pnemonia")
```

```
Pnemonia
```

In [ ]:
```

```