capstone-cnn-01

March 28, 2023

0.0.1 Importing the required packages and libraries*

[1]:
```
import random

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
figure(figsize=(15, 12), dpi=120)
import seaborn as sns
sns.set(style='whitegrid')
%matplotlib inline

import pydicom as dcm
from pathlib import Path
import os
from tqdm.notebook import tqdm
```

[2]:
```
train_class = pd.read_csv('C:
↪\\capstone_data\\Pneumonia_Set_Project\\stage_2_detailed_class_info.csv')
train_class.head()
```

[2]:

	patientId	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity

[3]:
```
train_labels = pd.read_csv("C:
↪\capstone_data\Pneumonia_Set_Project\stage_2_train_labels.csv")
train_labels.head()
```

[3]:

	patientId	x	y	width	height	Target
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0

```
3  003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN       0
4  00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0       1
```

Loading the Images

[4]:
```python
train_path = Path("C:\capstone_data\Pneumonia_Set_Project\stage_2_train_images")
test_path = Path("C:\capstone_data\Pneumonia_Set_Project\stage_2_test_images")
```

0.0.2 As patient-id is unique, we won't need it so dropping is the best option

[5]:
```python
train_meta = pd.concat([train_labels, train_class.drop(columns=['patientId'])],
 axis=1)
train_meta.head()
```

[5]:
```
                              patientId      x      y  width  height  Target  \
0  0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN    NaN    NaN     NaN       0
1  00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN    NaN    NaN     NaN       0
2  00322d4d-1c29-4943-afc9-b6754be640eb    NaN    NaN    NaN     NaN       0
3  003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN       0
4  00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0       1

                          class
0  No Lung Opacity / Not Normal
1  No Lung Opacity / Not Normal
2  No Lung Opacity / Not Normal
3                        Normal
4                  Lung Opacity
```

[6]:
```python
box_df = train_meta.groupby('patientId').size().reset_index(name='boxes')
box_df.head()
```

[6]:
```
                              patientId  boxes
0  0004cfab-14fd-4e49-80ba-63a80b6bddd6      1
1  000924cf-0f8d-42bd-9158-1af53881a557      1
2  000db696-cf54-4385-b10b-6b16fbb3f985      2
3  000fe35a-2649-43d4-b027-e67796d412e0      2
4  001031d9-f904-4a23-b3e5-2c088acd19c6      2
```

[7]:
```python
train_ds = pd.merge(train_meta, box_df, on='patientId')
box_df = box_df.groupby('boxes').size().reset_index(name='patients')
box_df.head()
```

[7]:
```
   boxes  patients
0      1     23286
1      2      3266
2      3       119
3      4        13
```

0.0.3 From our EDA we had seen we have Age, Gender & Image-Path of the patients

[8]:
```
information = ['PatientAge','PatientSex','ImagePath']
```

Again as seen in EDA, we are using Pydicom to process the images Here, in this function we read all files in the specified directory path and loops through each DICOM image file. It extracts patient information from each DICOM image and updates the corresponding rows in the df DataFrame.

[9]:
```
def process_dicom_data(df, path):

    # adding new columns to the imported DataFrame with Null values
    for var in information:
        df[var] = None

    images = os.listdir(path)

    #looping through each dicom image, extract the information from it, and
    # add it to the DataFrame

    for i, img_name in tqdm(enumerate(images)):

        imagePath = os.path.join(path,img_name)
        img_data = dcm.read_file(imagePath)

        idx = (df['patientId'] == img_data.PatientID)
        df.loc[idx,'PatientAge'] = pd.to_numeric(img_data.PatientAge)
        df.loc[idx,'PatientSex'] = img_data.PatientSex
        df.loc[idx, 'ImagePath'] = str.format(imagePath)

process_dicom_data(train_ds,"C:
 ↪\capstone_data\Pneumonia_Set_Project\stage_2_train_images")
```

```
0it [00:00, ?it/s]
```

So this is what our training dataset will look like, with Target being the label*

[10]:
```
train_ds.head()
```

[10]:
```
                              patientId      x      y  width  height  Target  \
0  0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN    NaN    NaN     NaN       0
1  00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN    NaN    NaN     NaN       0
2  00322d4d-1c29-4943-afc9-b6754be640eb    NaN    NaN    NaN     NaN       0
3  003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN       0
4  00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0       1

                          class  boxes PatientAge PatientSex  \
0  No Lung Opacity / Not Normal      1         51          F
```

```
1  No Lung Opacity / Not Normal      1          48           F
2  No Lung Opacity / Not Normal      1          19           M
3                        Normal      1          28           M
4                  Lung Opacity      2          32           F

                                        ImagePath
0  C:\capstone_data\Pneumonia_Set_Project\stage_2…
1  C:\capstone_data\Pneumonia_Set_Project\stage_2…
2  C:\capstone_data\Pneumonia_Set_Project\stage_2…
3  C:\capstone_data\Pneumonia_Set_Project\stage_2…
4  C:\capstone_data\Pneumonia_Set_Project\stage_2…
```

[102]:
```
train_ds.to_csv('pneumonia_ds', index=False)
```

0.0.4 *Pre-processing the Images*

[11]:
```
import cv2
```

[12]:
```
images = []
#converting the images to 128x128
ADJUSTED_IMAGE_SIZE = 128
imageList = []
classLabels = []
labels = []
originalImage = []
```

[13]:
```
# The function reads in DICOM images from the file path specified by dcm_file,␣
↪converts them to RGB format if necessary,
# resizes them to a square of size 128 using bilinear interpolation, and␣
↪appends the resulting images to a list called imageList
def readAndReshapeImage(image):
    img = np.array(image).astype(np.uint8)
    res = cv2.resize(img,(ADJUSTED_IMAGE_SIZE,ADJUSTED_IMAGE_SIZE),␣
↪interpolation = cv2.INTER_LINEAR)
    return res
```

[14]:
```
#Converting the images to arrays along with their corresponding labels
def populateImage(rowData):
    for index, row in rowData.iterrows():
        patientId = row.patientId
        classlabel = row["class"]
        dcm_file = "C:
↪\capstone_data\Pneumonia_Set_Project\stage_2_train_images\\" + '{}.dcm'.
↪format(patientId)
        dcm_data = dcm.read_file(dcm_file)
        img = dcm_data.pixel_array
```

```
        ## Converting the image to 3 channels as the dicom image pixel does not␣
↪have colour classes wiht it
        if len(img.shape) != 3 or img.shape[2] != 3:
            img = np.stack((img,) * 3, -1)

        imageList.append(readAndReshapeImage(img))
        classLabels.append(classlabel)
    tmpImages = np.array(imageList)
    tmpLabels = np.array(classLabels)
    return tmpImages,tmpLabels
```

[21]:
```
images, labels = populateImage(train_meta)
print(images.shape , labels.shape)
```

```
(35675, 128, 128, 3) (35675,)
```

Using LabelBinarizer to convert labels to binary vector, as in Sklearns documentation it suggest to use OneHotencoder for features to feed into model & use Binarizer for the labels

[22]:
```
from sklearn.preprocessing import LabelBinarizer
encode = LabelBinarizer()
y = encode.fit_transform(labels)
```

0.0.5 *Train-Test-Validation Split*

[23]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(images, y, test_size=0.3,␣
↪random_state=50)
X_test, X_val, y_test, y_val = train_test_split(X_test,y_test, test_size = 0.5,␣
↪random_state=50)
```

0.1 CNN-Model

[16]:
```
from tensorflow.keras.layers import Layer, Convolution2D, Flatten, Dense
from tensorflow.keras.layers import Concatenate, UpSampling2D, Conv2D, Reshape,␣
↪GlobalAveragePooling2D, GlobalMaxPooling2D
from tensorflow.keras.layers import Dense,␣
↪Activation,Flatten,Dropout,MaxPooling2D,BatchNormalization

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import losses,optimizers
```

We start with 32 filters with 3,3 kernal and no padding , then 64 and 128 wiht drop layers in between & softmax activaation as the last layer

We are using loss function as categorical crossentropy as we have binary classfication task at hand, and using metrics as accuracy as of now.

[17]:
```
def cnn_model(height, width, num_classes,␣
↪loss='categorical_crossentropy', metrics=['accuracy']):
    batch_size = None

    model = Sequential()

    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                  activation ='relu', batch_input_shape = (batch_size,height,␣
↪width, num_channels)))


    model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                  activation ='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.2))


    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                  activation ='relu'))
    model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'same',
                  activation ='relu'))
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
    model.add(Dropout(0.3))

    model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                  activation ='relu'))
    model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same',
                  activation ='relu'))
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
    model.add(Dropout(0.4))

    model.add(GlobalMaxPooling2D())
    model.add(Dense(256, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = "softmax"))

    optimizer = Adam(lr=0.001)
    model.compile(optimizer = optimizer, loss = loss, metrics = metrics)
    model.summary()
    return model
```

[18]:
```
cnn = cnn_model(ADJUSTED_IMAGE_SIZE,ADJUSTED_IMAGE_SIZE,3,3)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      896       
                                                                 
 conv2d_1 (Conv2D)           (None, 128, 128, 32)      9248      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 64, 64, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 64, 64, 32)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 64, 64, 64)        18496     
                                                                 
 conv2d_3 (Conv2D)           (None, 64, 64, 64)        36928     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)       0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 32, 32, 64)        0         
                                                                 
 conv2d_4 (Conv2D)           (None, 32, 32, 128)       73856     
                                                                 
 conv2d_5 (Conv2D)           (None, 32, 32, 128)       147584    
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0         
 2D)                                                             
                                                                 
 dropout_2 (Dropout)         (None, 16, 16, 128)       0         
                                                                 
 global_max_pooling2d (Globa  (None, 128)              0         
 lMaxPooling2D)                                                  
                                                                 
 dense (Dense)               (None, 256)               33024     
                                                                 
 dropout_3 (Dropout)         (None, 256)               0         
                                                                 
 dense_1 (Dense)             (None, 3)                 771       
                                                                 
=================================================================
Total params: 320,803
Trainable params: 320,803
Non-trainable params: 0
_________________________________________________________________
```

[19]:
```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=4),
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
]
```

[ ]:
```
history = cnn.fit(X_train,
                  y_train,
                  epochs = 20,
                  validation_data = (X_val,y_val),
                  batch_size = 16,
                  callbacks = callbacks)
```

```
Epoch 1/20

2023-03-20 09:48:09.340827: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed:
INVALID_ARGUMENT: Size of values 0 does not match size of permutation 4 @ fanin
shape insequential/dropout/dropout/SelectV2-2-TransposeNHWCToNCHW-
LayoutOptimizer

1323/1323 [==============================] - 35s 19ms/step - loss: 1.1071 -
accuracy: 0.4214 - val_loss: 1.0493 - val_accuracy: 0.4454 - lr: 0.0010
Epoch 2/20
1323/1323 [==============================] - 22s 17ms/step - loss: 1.0046 -
accuracy: 0.4694 - val_loss: 1.0077 - val_accuracy: 0.4710 - lr: 0.0010
Epoch 3/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9885 -
accuracy: 0.4850 - val_loss: 1.0201 - val_accuracy: 0.4490 - lr: 0.0010
Epoch 4/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9713 -
accuracy: 0.4969 - val_loss: 1.0369 - val_accuracy: 0.4088 - lr: 0.0010
Epoch 5/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9665 -
accuracy: 0.4997 - val_loss: 0.9957 - val_accuracy: 0.4556 - lr: 0.0010
Epoch 6/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9647 -
accuracy: 0.5010 - val_loss: 0.9739 - val_accuracy: 0.4714 - lr: 0.0010
Epoch 7/20
1323/1323 [==============================] - 23s 17ms/step - loss: 0.9605 -
accuracy: 0.5059 - val_loss: 0.9841 - val_accuracy: 0.4699 - lr: 0.0010
Epoch 8/20
1323/1323 [==============================] - 24s 18ms/step - loss: 0.9546 -
accuracy: 0.5096 - val_loss: 1.0493 - val_accuracy: 0.4265 - lr: 0.0010
Epoch 9/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9505 -
accuracy: 0.5147 - val_loss: 0.9674 - val_accuracy: 0.4862 - lr: 0.0010
```

```
Epoch 10/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9522 -
accuracy: 0.5118 - val_loss: 0.9976 - val_accuracy: 0.4584 - lr: 0.0010
Epoch 11/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9532 -
accuracy: 0.5101 - val_loss: 1.0142 - val_accuracy: 0.4426 - lr: 0.0010
Epoch 12/20
1323/1323 [==============================] - 22s 17ms/step - loss: 0.9470 -
accuracy: 0.5183 - val_loss: 0.9873 - val_accuracy: 0.4558 - lr: 0.0010
```

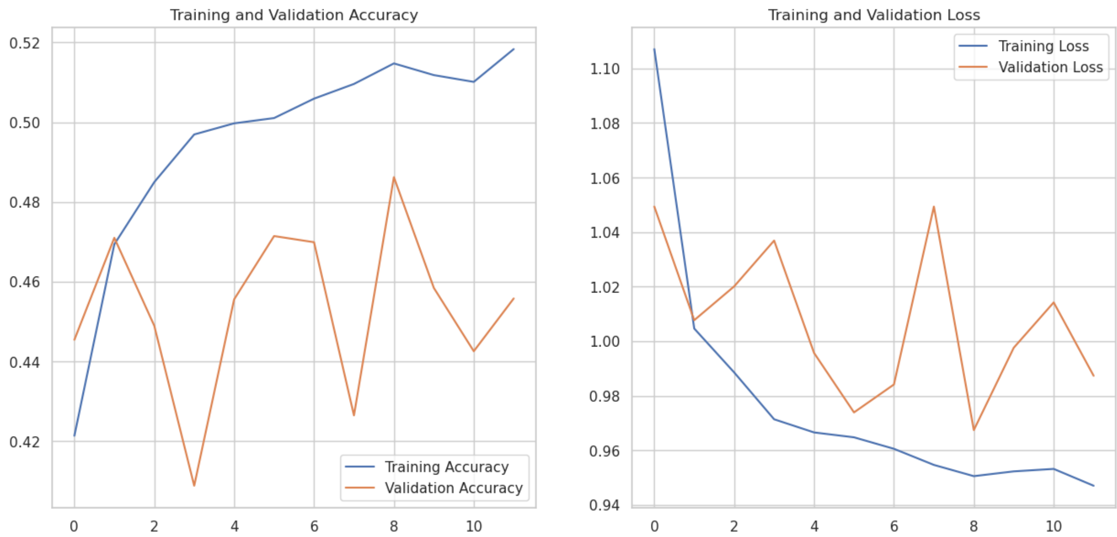
0.1.1 We are getting Training accuracy of around 50 percent and validation of around ~46 percent. Also it seems there our model is not overfitting, but the accuracy is very low.

[ ]:
```
fcl_loss, fcl_accuracy = cnn.evaluate(X_test, y_test, verbose=1)
print('Test loss:', fcl_loss)
print('Test accuracy:', fcl_accuracy)
```

```
142/142 [==============================] - 1s 8ms/step - loss: 0.9635 -
accuracy: 0.4901
Test loss: 0.9635416269302368
Test accuracy: 0.4900749921798706
```

Although our Model is generalized, but the performance is not that good

[ ]:
```
## PLottting the accuracy vs loss graph
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(12)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



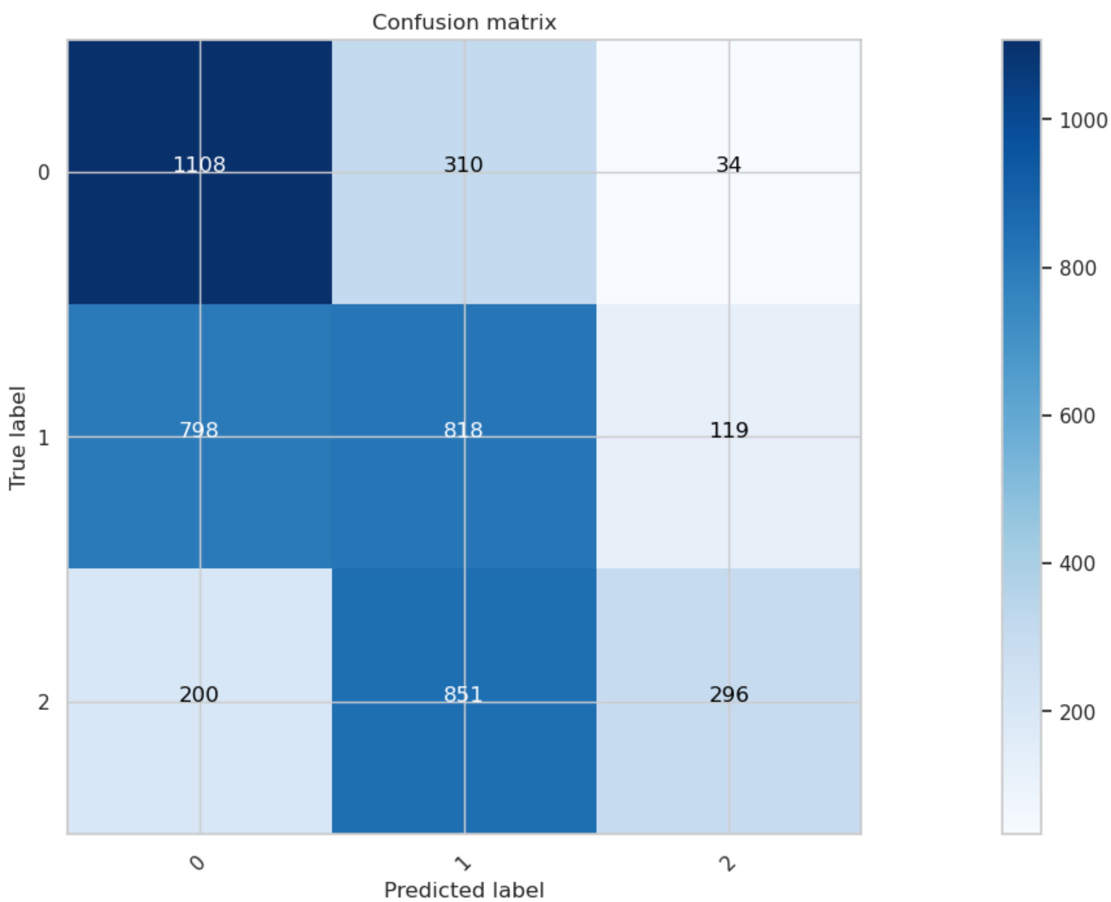
The training and validation loss show similar patterns, but the validation accuracy chart displays a decline in accuracy during the later epochs.

0.1.2 *Confusion Matrix*

[ ]:
```
from sklearn.metrics import confusion_matrix
import itertools
plt.subplots(figsize=(22,7)) #set the size of the plot

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = cnn.predict(X_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(3))
```

```
142/142 [==============================] - 1s 5ms/step
```



- Class 0 is Lung Opacity
- Class 1 is No Lung Opacity/Normal, the model has predicted mostly wrong in this case to

the Target 0. Type 2 error

- Class 2 is Normal