London Metropolitan University

CS7079

Machine Learning – Autumn 2022-23(CS7052)

Coursework

Pneumonia Detection Using CNN

Submitted By:

Krusha shah – 2011176
Divya Prackash Ravi | 22014173
Ankita Kotadia | 21029002

Submitted To:

Dr. Elaheh Homayounvala (Ela)

Submission Date: 9thDecember,2022.

ABSTRACT

Pneumonia has one of the highest death tolls of the deaths caused by lung disease worldwide. Early detection and treatment are crucial for patient recovery from pneumonia. Medical professionals use chest x-rays to detect any inflammation in the lung area so it is a symptom of pneumonia. However, in a massive outbreak to it is time consuming for the doctors to manually view each x-ray to determine if the patient has pneumonia or not. Convolution neural networks can identify and classify pneumonia with good accuracy in a shorter period of time, which is crucial for determining patients right treatment procedure and survival rate on early stage. The pneumonia and pneumonia free lung images are considered as part of the datasets in this report. This report highlight the machine learning model which takes input images to train the model

TABLE OF CONTENTS

1. INTRODUCTION

In research [1] conducted by the professors from St George's University of London, Nottingham University, and Imperial College London, they found out that pneumonia is the cause for the third highest deaths due to lung diseases. In the year 2012 the number of deaths in UK due to pneumonia was 28,952 which was 5.1% of all the deaths in the UK and 25.3% of deaths due to lung disease.

According to research done by Bernadeta [2], in the year 2019 alone pneumonia had a death count of 2.5 million people of which a third were under 5 years old. Pneumonia can be detected using conventional images like X-ray images, Pulse oximetry, and blood tests can be carried out to detect the presence of pneumonia in the patient. This process is time consuming, which may lead to delayed patient treatment. Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate human behavior.

a) Technique used CNN

The topic we have decided to research into, and implement are Convolutional Neural Network (CNN). CNN is a type of Deep Learning which is specifically used to analyse visual information, it is most commonly used for detecting and classifying objects in a given image.

CNN works mainly based on convolution operation where a square filter much smaller than the input is overlayed and convolution multiplication on the overlapping cells are performed to get a value, this process is repeated until all the cells in the input layer have undergone convolution operation this occurs in the first layer called as the Convolution Layer, this layer is usually followed by a relu activation function so that it eliminates any negative values as image data ranges from 0 to 255.

Once the convolution and relu activation is completed the network is generally followed up by two more layers a max pooling layer and a fully connected layer. The max pooling layer reduces the dimensions of the image depending on the filter size, the filter is overlayed on the image then the cell which the maximum is selected as the output for that cell in the output in the same size of the filter. It is then flattened so the images can be given as input to the fully connected layer, the input with m rows and n columns will be converted into (m*n) rows and 1 column. The m*n flattened data points are then fed into the fully connected layers, they assign the suitable weights for each neuron to neuron connection by training on the data multiple times [3].

b) What type of problems does CNN solve

CNN are mainly used for image datasets, It is one of the most popular architecture to perform image classification with. CNN is the go to algorithm when dealing with image based problems such as, automated driving, human activity recognition, object recognition, etc.

CNN are also be used in speech to text (STT) systems, STT uses mel frequency cepstral coefficients (MFCC) which converts the audio frequency into a spectrogram that is suitable to undergo convolution operation [4].

c) Who is using it

CNN is a crucial algorithm used by tech giants like Meta, Google, Amazon, etc. as the tech industry is slowly shifting into augmented reality, In the augmented reality the virtual objects interacts with the physical objects, CNN is used to recognize the objects so that the virtual objects can properly interact with the physical object around it.

CNN is also used in medical field to provide more information to specialists, It can diagnose diseases or abnormalities from x-rays or CT scans [5].

d) What type of data does it require

CNN requires image data or numeric data to perform convolution operation, it cannot work on textual or any other non-image. It could work on frequency data if they are transformed into a spectrogram data.

e) History of CNN

Convolution Neural Network originated from the LeNet-5 designed by Yann LeCun et al [6]. The architecture LeNet-5 consists of 6 layers first the image is supplied to a convolution layer then a max pooling layer to reduce the dimensions, this reduced image will be then fed into another convolution layer and max pooling layer which is reduces the data dimension even more, this reduced image is then flattened in order to be fed into the fully connected layers.

The newer architectures of CNN are inspired by the model winners in the ImageNet large scale visual recognition challenge' that happens every year since 2010 [7], the error rate of the winning model in the 2010 competition was 28.2% this gradually reduced in the following years. In the year 2017 SENet a model developed by Hu et al [8] achieved an error rate of 2.3%.

f) Description of available mathematical/ statistical/ ml techniques

As stated above the CNN architecture works on 3 major layers

- Convolution Layer
- Pooling Layer
- Fully Connected Layer

Convolution Layer

In this layer we perform the convolution operation [9] between the input image n*m and a filter/kernel of much smaller size k*k (usually 3*3 or 5*5) . First the image is padded with floor(k//2) cells, the padding can be 0 or max or same, it then subsamples the image with overlapping into the same size as the filter, the filter is then slid over the main n*m cells and for each cell convolution is performed with its neighbouring k*k - 1 cells.

For example consider a 3*3 matrix and a 3*3 filter

Input	Filter
[1 2 3]	[0 0 0]
[2 1 3]	[0 1 0]
[3 1 1]	[0 0 0]

The input is padding with 1 cell (floor(3//2)) let us consider the padding is 0 padding

Padded Image

[0 0 0 0 0]
[0 1 2 3 0]
[0 2 1 3 0]
[0 3 1 1 0]
[0 0 0 0 0]

Now we can apply the convolution on the padded image with filter to get the output image with the size of the input

Convolution is the sum of the product of opposite ends from subsampled image and filter

For example if the filter size is 3 * 3

For a cell [i,j] we need calculate = image[1,1]*filter[3,3] + image[2,1]*filter[2,3] + image[3,1]*filter[1,3] + image[1,2]*filter[3,2] + image[2,2]*filter[2,2] + image[3,2]*filter[1,2] + image[3,1]*filter[1,3] + image[3,2]*filter[1,2] + image[3,3]*filter[1,1]

This process need to be done for each cell to get the output image.

Pooling layer
In this layer the image is partitioned into n rectangles mentioned by the user and in each partition the maximum value is chosen as the output for the corresponding output cell [10].

Fully Connected layer
The output from the pooling layer need to flattened into a 1 dimensional array and then it can be fed to the fully connected layer, where it trains the neurons and adjusts the weights (i.e.) training the model.

g) Possible software or packages to use
Some of the popular packages for using CNN are tensorflow, pytorch, and keras.

2. DEMONSTRATION OF THE TECHNIQUE

Our proposed system mainly followed by steps such as data preparation, data visualization, data exploration, data augmentation, model training, model evaluation, and performance analysis.

a) Data Preparation
The datasets are collected from Kaggle [11], named as pneumonia detection in form of images. It includes labelled data for 1398 patients, where a label 0 indicates a patient who do have pneumonia and a label 1 indicates a patient who do not have pneumonia. The CT scan consist numerous images of each patient. We got 5872 total number of images out of it we partitioned large part into training set (5232 images), test set (624 images), and smaller part validation set (16 images). Below images show highlight of pneumonia and normal samples.

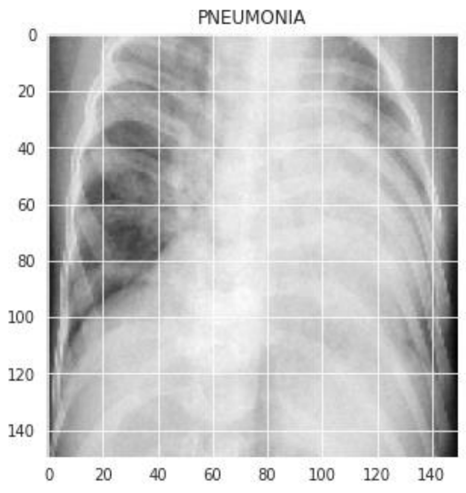


Fig. 1 Lungs infected by Pneumonia

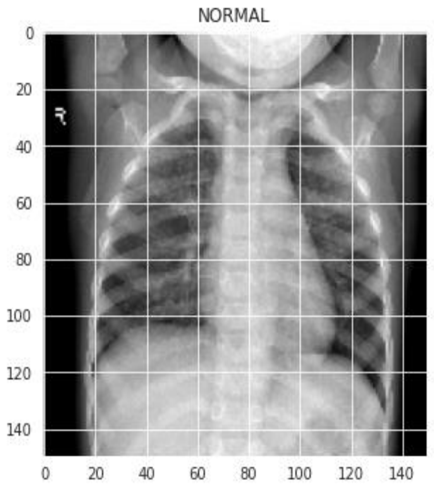


Fig. 2 Lungs not infected by Pneumonia

b) Data Visualization and Exploration
By plotting bar chart of the number of patients with pneumonia and without pneumonia that we have received from Kaggle has significant data discrepancy, which clearly indicates that 70% of the patients are infected with pneumonia and rest 30% of patients are not infected by pneumonia. So, if data splitting were chosen based on only healthy portion, then it would compromise the testing accuracy of CNN model as our data is

not sensitive to splits, we have made. The suggest way is using stratify validation to shuffle the data so each set gets portion of the pneumonia samples as well as samples without pneumonia. To get around this, we first construct our datasets with two classifications (Pneumonia, Normal) being equally likely some portion of pneumonia and normal in each set. Therefore, data visualization plays a key role while splitting the data into train and test. Below image indicates the proportion pneumonia and normal samples:

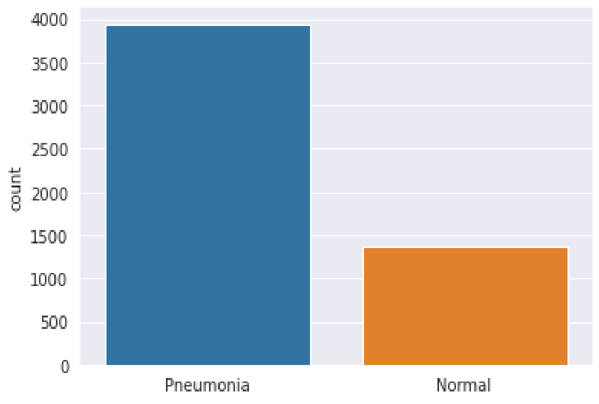


Fig. 3 Number of pneumonia and normal samples

c) Data Augmentation

Before data augmentation, we reshape the image cell values range from 0-255 (usual size of a gray scale image) to the range 0 to 1 (Note this does not convert it into a binary values, it is still continuous value). In this way, the number will be small, and the computation becomes more efficient and quicker. Data augmentation is a technique to artificially create new training data from existing data. Data augmentation is usually required to boost the performance of deep network. Data augmentation process steps to be followed:

- Randomly rotate some training image by 30 degrees.
- Randomly zoom some training image by 20%.
- Randomly shift images horizontally by 10% of the width.
- Randomly shift images vertically by 10% of the height.
- Randomly flip images horizontally.

d) Purpose of Analysis

Nowadays, ecommerce and financial sectors already blessed with the advanced technology called artificial intelligence. However, medical filed has not adopted widely, which inspires us to do research on such field. Successful implementation of pneumonia detection machine learning algorithm would reduce the chance of dying from pneumonia. Moreover, pneumonia can detect on early stage, so patients have best chance for successful treatment.

e) Software and Packages Used

To implement the model we used the following software and packages

- Google Colab (coding environment)
- TensorFlow
- Scikit-learn
- MatplotLib, Numpy & Pandas

f) Model Improvement and Evaluation

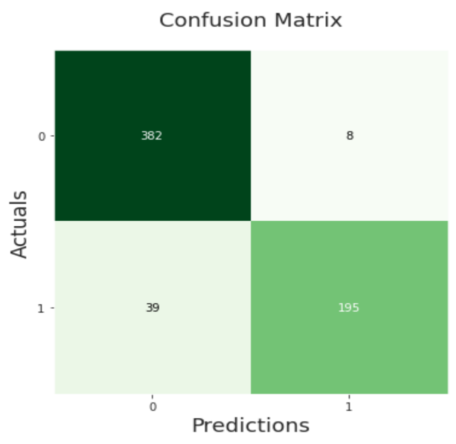
After carefully studying the CNN algorithm first we have tried to train our model using main three layers such as convolution layer followed by max pool and fully connected layer. The table below represents the parameters we have used to train our model and receive different accuracy. At one point we got good result in 94% training accuracy and 93% test accuracy. This model is based on healthcare so thought that's not enough as it is directly associated with someone's life. We have followed some research papers, where mentioned that accuracy can be improved by adding more convolution layers. However, it hasn't worked in our case, so not decided to make model more complex.

CNN	*Layers*	*Learning Rate*	*Epochs*	*Batch Size*	*Training Loss*	*Validation Loss*	*Train Accuracy*	*Validation Accuracy*
CNN	3	0.010	12	32	0.9409	0.6908	65%	45%
CNN	3	0.0010	12	32	0.3281	0. 6724	78%	50%
CNN	3	3.0000e-04	12	32	0.1610	0.6875	94%	68%
CNN	3	9.0000e-05	12	32	0.1423	1.563	94%	75%
CNN	6	2.7000e-05	12	32	0.1665	0.771	94%	68%

The model is evaluated using CNN's evaluate function on test data that we kept separated for testing purpose. The accuracy we have received is 94%, which is quite good. Hence, model is stable neither overfitting nor underfitting as model has calculated good, sweet point.

Confusion Matrix

Confusion matrix in the field of machine learning one of best way to evaluate the performance for classification algorithm. The table below represent the count actual and predicted samples after evaluating our model.



g) Remark On Output

The training and test accuracy we have received is quite good, which clearly indicated that model is working well on the training as well as on unseen data. The below plots represent the training/validation accuracy versus number of epochs and training/validation loss versus epochs.

We can observe in the loss plot that the validation loss is high, therefore the validity accuracy is low when compared to the training and test. The main reason is the supplied images to the validity data are very less, nearly 16 out of 5881. It can be improved by increasing the number of images. The test accuracy is quite good because images are almost noise free and we have invested more time on data preparation and did sensitive data partition, so model worked well on it.

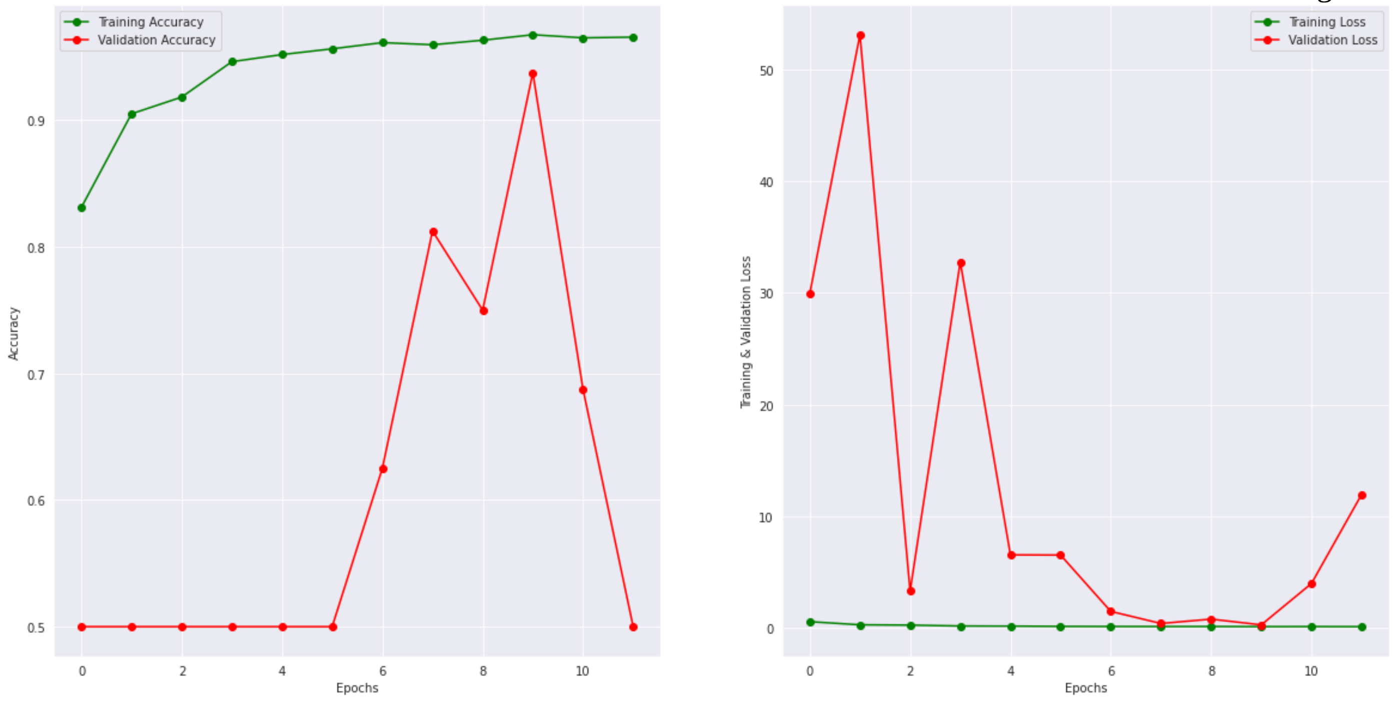


Fig. 4 Training and Validation accuracy/loss versus Epoch

3. FUTURE PLANS

- Pneumonia detection is a binary classification problem, in future same model can be implemented for multiclass classification.
- Due to the limited configuration of the system, we have provided only 5881 images, it may vary the accuracy by feeding more images.
- This model can be combined with other models to detect the complete blood count (CBC) in the blood tests and make use of the pulse oximetry to get a more accurate result (“https://www.nhlbi.nih.gov/health/pneumonia/diagnosis”).
- Implementing methods to remove noise (like blur, salt, and pepper noise, etc.) from the data in the preprocessing step.

4. CONCLUSION

Individual reflection

Prior to working on this model, each team member decided to hone our skills on developing different models and data pre-processing. This coursework has helped me to understand not only CNN but also other algorithms like SVM, Decision tree, and logistic regression. I practiced the mentioned algorithms with “Football Result prediction during half-time” as the goal. In this coursework I was able to grasp the concept of how the weights are allocated in the network, how it can train without explicitly coding it.

Report Conclusion

Through research on CNN and after carefully studying other research papers we were able to understand the working principal of the algorithm and we successfully demonstrated it by implementing pneumonia detection using image datasets retrieved from Kaggle. After understanding the nature of datasets, we have made sensitive partition as well tuning parameter and adjusting number of layers helped us to achieve training (94%) accuracy and test (93%) accuracy, which is quite good. This accuracy could be further improved by supplying more images as neural networks provide greater performance on the larger data.

5. REFFRENCES

[1] https://statistics.blf.org.uk/pneumonia

[2] Dadonaite, B. and Roser, M., Pneumonia. Our World In Data (2018) https://ourworldindata.org/pneumonia)

[3] Choi, K., Fazekas, G. and Sandler, M., 2016. Explaining deep convolutional neural networks on music classification. *arXiv preprint arXiv:1607.02444*.

[4] Lee, K.H., 2020, October. Design of a convolutional neural network for speech emotion recognition. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)* (pp. 1332-1335). IEEE.

[5] Singh, S.P., Wang, L., Gupta, S., Gulyas, B. and Padmanabhan, P., 2020. Shallow 3D CNN for detecting acute brain hemorrhage from medical imaging sensors. *IEEE Sensors Journal*, *21*(13), pp.14290-14299.

[6] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), pp.2278-2324.

[7] Li, F., Johnson, J. and Yeung, S. CS231n course, Stanford, Spring 2019.

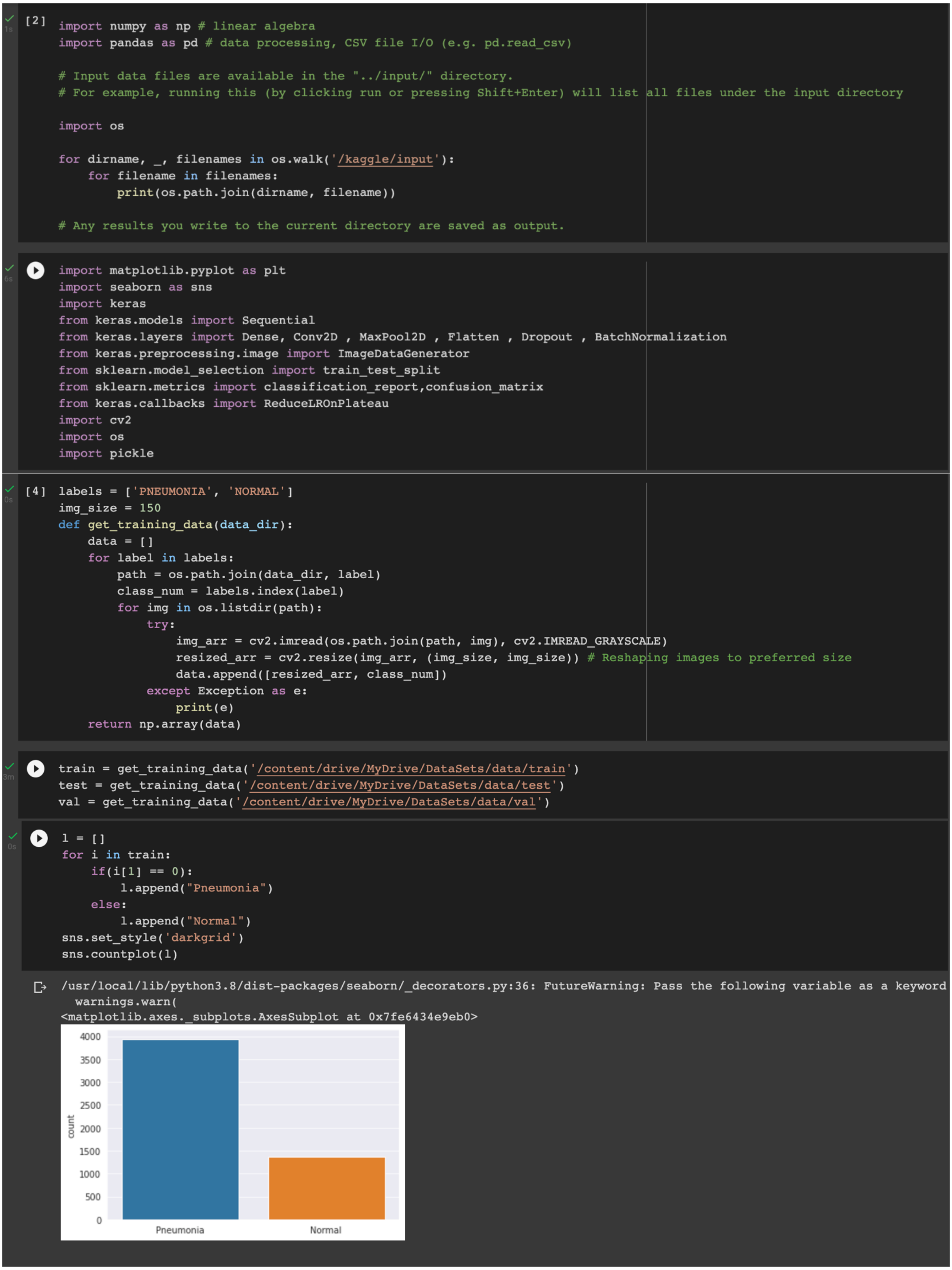
[8] Hu, J., Shen, L. and Sun, G., 2018. Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).

[9] https://en.wikipedia.org/wiki/Kernel_(image_processing)

[10] https://en.wikipedia.org/wiki/Convolutional_neural_network#Pooling_layer

[11] https://www.kaggle.com/datasets/tolgadincer/labeled-chest-xray-images

6. APPENDIX



```
[2]  import numpy as np # linear algebra
     import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

     # Input data files are available in the "../input/" directory.
     # For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

     import os

     for dirname, _, filenames in os.walk('/kaggle/input'):
         for filename in filenames:
             print(os.path.join(dirname, filename))

     # Any results you write to the current directory are saved as output.
```

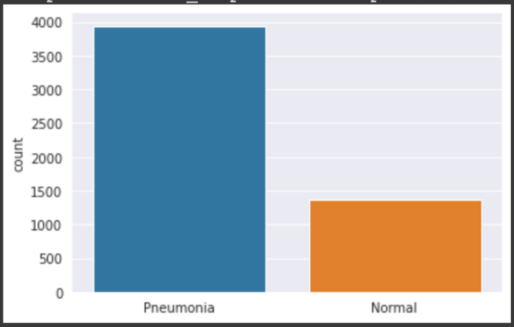
```
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout , BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import ReduceLROnPlateau
import cv2
import os
import pickle
```

```
[4]  labels = ['PNEUMONIA', 'NORMAL']
     img_size = 150
     def get_training_data(data_dir):
         data = []
         for label in labels:
             path = os.path.join(data_dir, label)
             class_num = labels.index(label)
             for img in os.listdir(path):
                 try:
                     img_arr = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                     resized_arr = cv2.resize(img_arr, (img_size, img_size)) # Reshaping images to preferred size
                     data.append([resized_arr, class_num])
                 except Exception as e:
                     print(e)
         return np.array(data)
```

```
train = get_training_data('/content/drive/MyDrive/DataSets/data/train')
test = get_training_data('/content/drive/MyDrive/DataSets/data/test')
val = get_training_data('/content/drive/MyDrive/DataSets/data/val')
```

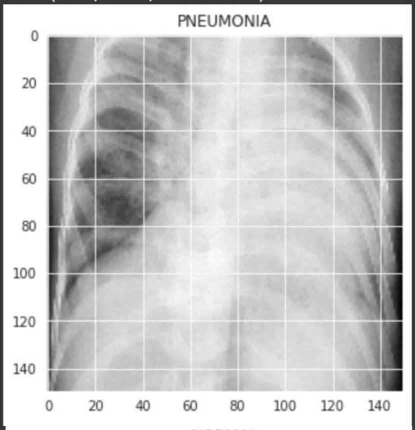
```
l = []
for i in train:
    if(i[1] == 0):
        l.append("Pneumonia")
    else:
        l.append("Normal")
sns.set_style('darkgrid')
sns.countplot(l)
```

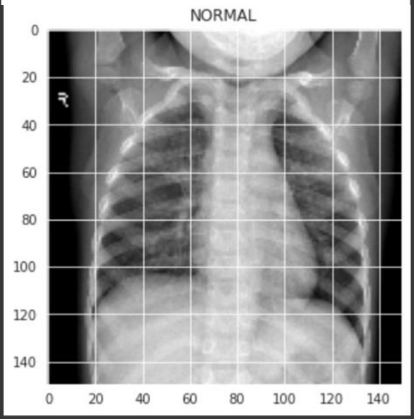
```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7fe6434e9eb0>
```



```
plt.figure(figsize = (5,5))
plt.imshow(train[0][0], cmap='gray')
plt.title(labels[train[0][1]])

plt.figure(figsize = (5,5))
plt.imshow(train[-1][0], cmap='gray')
plt.title(labels[train[-1][1]])
```

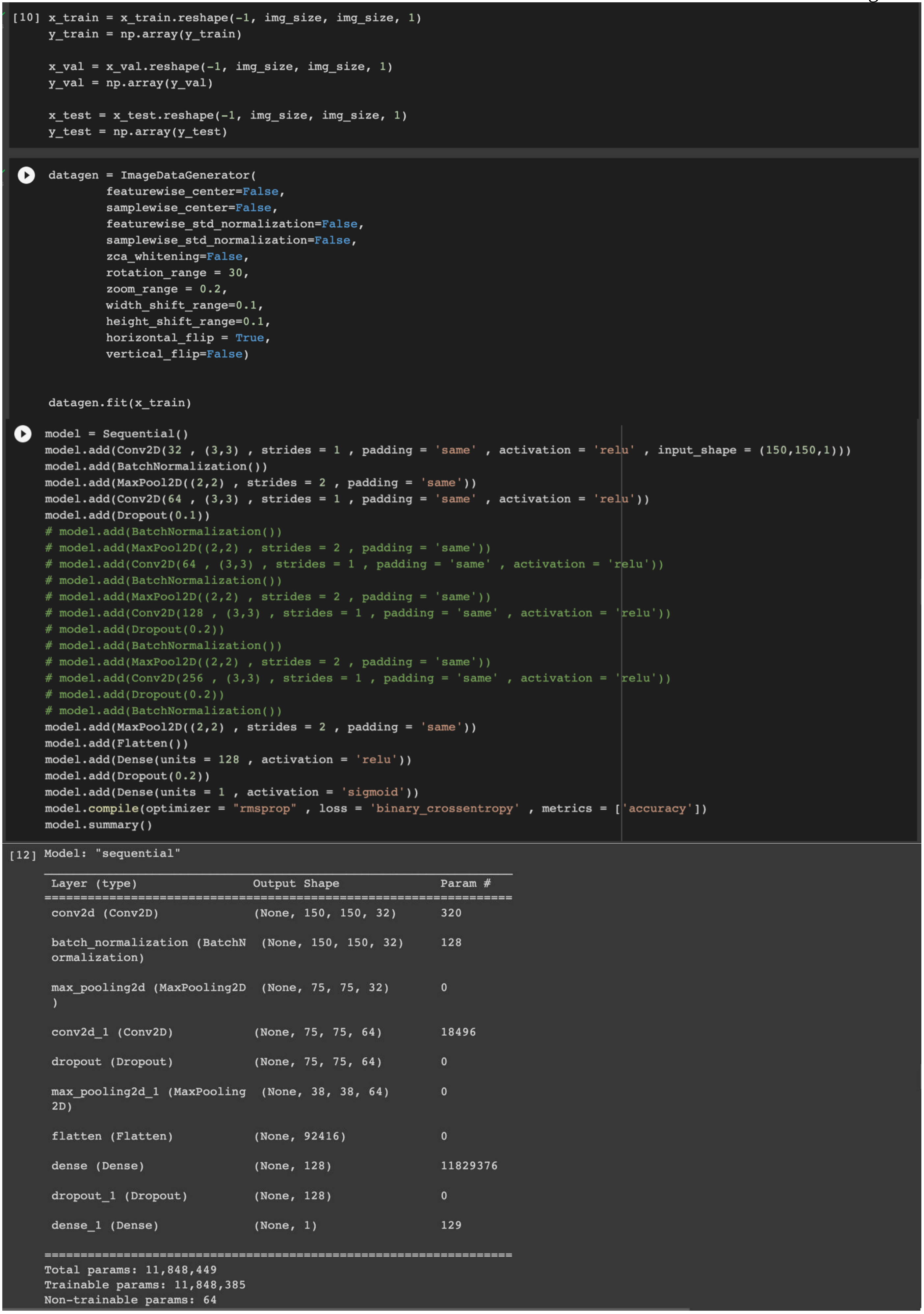
```
Text(0.5, 1.0, 'NORMAL')
```





```
x_train = []
y_train = []

x_val = []
y_val = []

x_test = []
y_test = []

for feature, label in train:
    x_train.append(feature)
    y_train.append(label)

for feature, label in test:
    x_test.append(feature)
    y_test.append(label)

for feature, label in val:
    x_val.append(feature)
    y_val.append(label)
```

```
[9] x_train = np.array(x_train) / 255
    x_val = np.array(x_val) / 255
    x_test = np.array(x_test) / 255
```

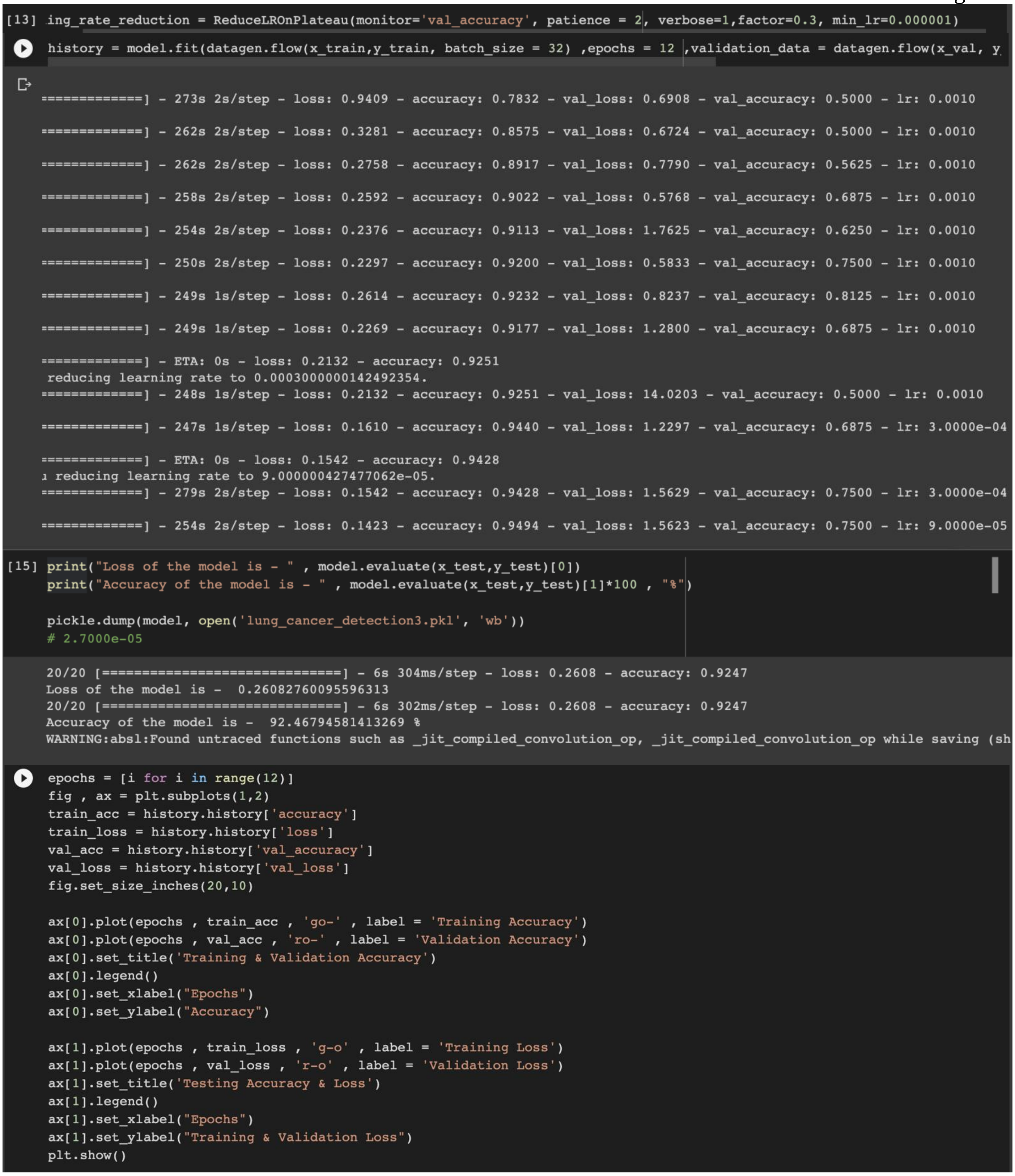


```
[10] x_train = x_train.reshape(-1, img_size, img_size, 1)
     y_train = np.array(y_train)

     x_val = x_val.reshape(-1, img_size, img_size, 1)
     y_val = np.array(y_val)

     x_test = x_test.reshape(-1, img_size, img_size, 1)
     y_test = np.array(y_test)
```

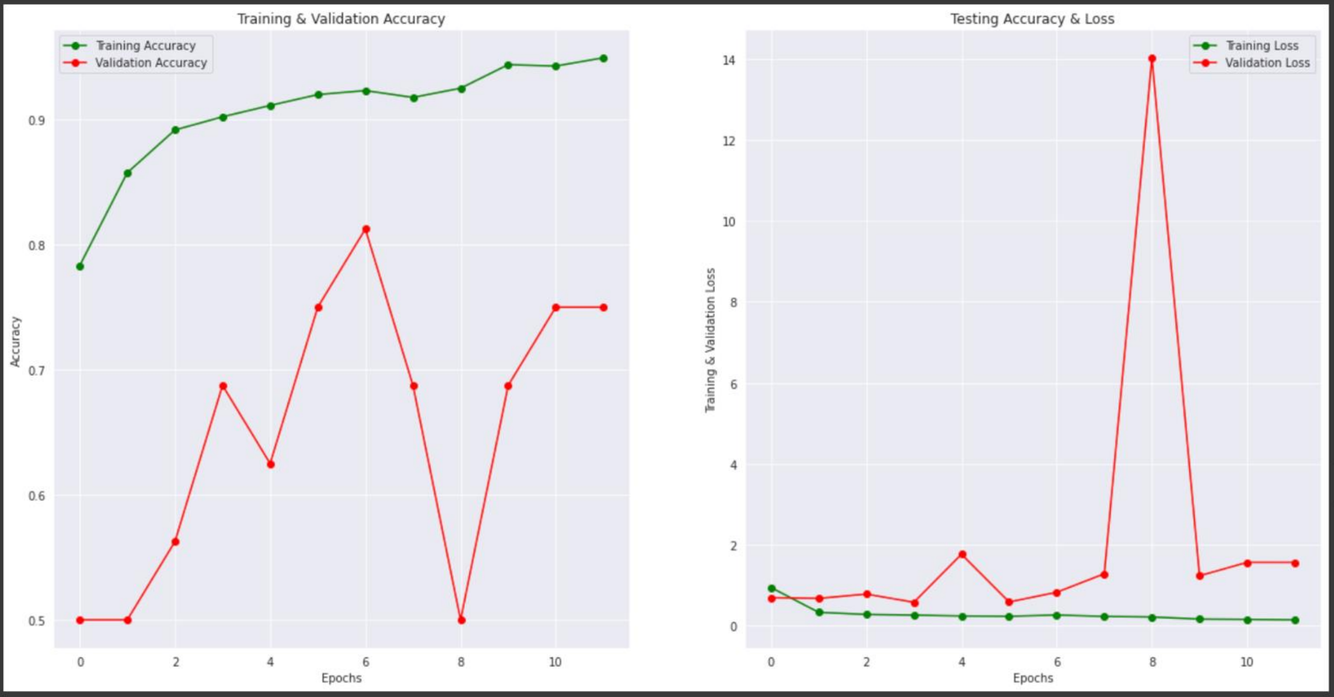
```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range = 30,
        zoom_range = 0.2,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip = True,
        vertical_flip=False)


datagen.fit(x_train)
```

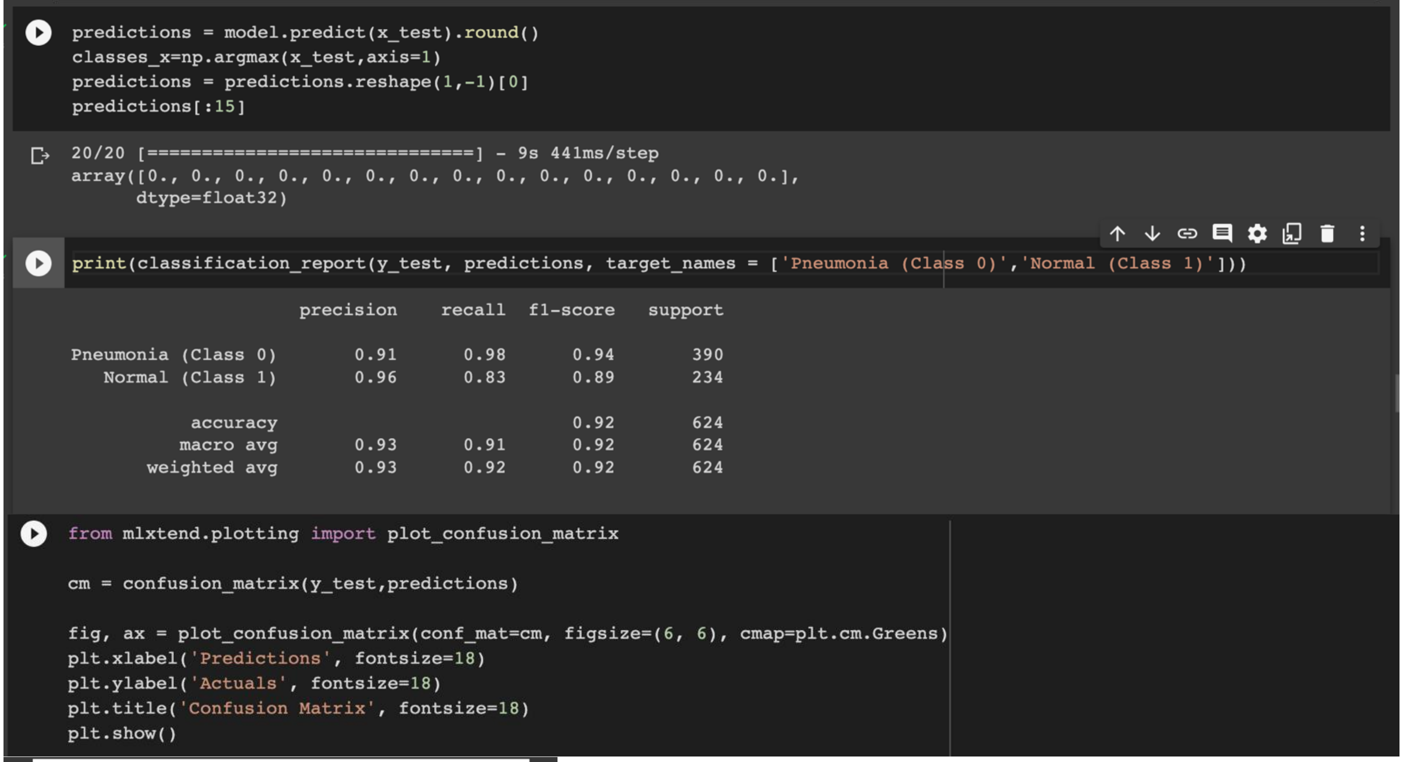
```
model = Sequential()
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.1))
# model.add(BatchNormalization())
# model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
# model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
# model.add(BatchNormalization())
# model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
# model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
# model.add(Dropout(0.2))
# model.add(BatchNormalization())
# model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
# model.add(Conv2D(256 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
# model.add(Dropout(0.2))
# model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128 , activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1 , activation = 'sigmoid'))
model.compile(optimizer = "rmsprop" , loss = 'binary_crossentropy' , metrics = ['accuracy'])
model.summary()
```

```
[12] Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      conv2d (Conv2D)             (None, 150, 150, 32)      320

      batch_normalization (BatchN  (None, 150, 150, 32)     128
      ormalization)

      max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
      )

      conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

      dropout (Dropout)           (None, 75, 75, 64)        0

      max_pooling2d_1 (MaxPooling  (None, 38, 38, 64)       0
      2D)

      flatten (Flatten)           (None, 92416)             0

      dense (Dense)               (None, 128)               11829376

      dropout_1 (Dropout)         (None, 128)               0

      dense_1 (Dense)             (None, 1)                 129

     =================================================================
     Total params: 11,848,449
     Trainable params: 11,848,385
     Non-trainable params: 64
```



```
[13] ing_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1,factor=0.3, min_lr=0.000001)

     history = model.fit(datagen.flow(x_train,y_train, batch_size = 32) ,epochs = 12 ,validation_data = datagen.flow(x_val, y

=============] - 273s 2s/step - loss: 0.9409 - accuracy: 0.7832 - val_loss: 0.6908 - val_accuracy: 0.5000 - lr: 0.0010

=============] - 262s 2s/step - loss: 0.3281 - accuracy: 0.8575 - val_loss: 0.6724 - val_accuracy: 0.5000 - lr: 0.0010

=============] - 262s 2s/step - loss: 0.2758 - accuracy: 0.8917 - val_loss: 0.7790 - val_accuracy: 0.5625 - lr: 0.0010

=============] - 258s 2s/step - loss: 0.2592 - accuracy: 0.9022 - val_loss: 0.5768 - val_accuracy: 0.6875 - lr: 0.0010

=============] - 254s 2s/step - loss: 0.2376 - accuracy: 0.9113 - val_loss: 1.7625 - val_accuracy: 0.6250 - lr: 0.0010

=============] - 250s 2s/step - loss: 0.2297 - accuracy: 0.9200 - val_loss: 0.5833 - val_accuracy: 0.7500 - lr: 0.0010

=============] - 249s 1s/step - loss: 0.2614 - accuracy: 0.9232 - val_loss: 0.8237 - val_accuracy: 0.8125 - lr: 0.0010

=============] - 249s 1s/step - loss: 0.2269 - accuracy: 0.9177 - val_loss: 1.2800 - val_accuracy: 0.6875 - lr: 0.0010

=============] - ETA: 0s - loss: 0.2132 - accuracy: 0.9251
 reducing learning rate to 0.0003000000142492354.
=============] - 248s 1s/step - loss: 0.2132 - accuracy: 0.9251 - val_loss: 14.0203 - val_accuracy: 0.5000 - lr: 0.0010

=============] - 247s 1s/step - loss: 0.1610 - accuracy: 0.9440 - val_loss: 1.2297 - val_accuracy: 0.6875 - lr: 3.0000e-04

=============] - ETA: 0s - loss: 0.1542 - accuracy: 0.9428
ı reducing learning rate to 9.000000427477062e-05.
=============] - 279s 2s/step - loss: 0.1542 - accuracy: 0.9428 - val_loss: 1.5629 - val_accuracy: 0.7500 - lr: 3.0000e-04

=============] - 254s 2s/step - loss: 0.1423 - accuracy: 0.9494 - val_loss: 1.5623 - val_accuracy: 0.7500 - lr: 9.0000e-05

[15] print("Loss of the model is - " , model.evaluate(x_test,y_test)[0])
     print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")

     pickle.dump(model, open('lung_cancer_detection3.pkl', 'wb'))
     # 2.7000e-05

     20/20 [==============================] - 6s 304ms/step - loss: 0.2608 - accuracy: 0.9247
     Loss of the model is -  0.26082760095596313
     20/20 [==============================] - 6s 302ms/step - loss: 0.2608 - accuracy: 0.9247
     Accuracy of the model is -  92.46794581413269 %
     WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (sh

     epochs = [i for i in range(12)]
     fig , ax = plt.subplots(1,2)
     train_acc = history.history['accuracy']
     train_loss = history.history['loss']
     val_acc = history.history['val_accuracy']
     val_loss = history.history['val_loss']
     fig.set_size_inches(20,10)

     ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
     ax[0].plot(epochs , val_acc , 'ro-' , label = 'Validation Accuracy')
     ax[0].set_title('Training & Validation Accuracy')
     ax[0].legend()
     ax[0].set_xlabel("Epochs")
     ax[0].set_ylabel("Accuracy")

     ax[1].plot(epochs , train_loss , 'g-o' , label = 'Training Loss')
     ax[1].plot(epochs , val_loss , 'r-o' , label = 'Validation Loss')
     ax[1].set_title('Testing Accuracy & Loss')
     ax[1].legend()
     ax[1].set_xlabel("Epochs")
     ax[1].set_ylabel("Training & Validation Loss")
     plt.show()
```





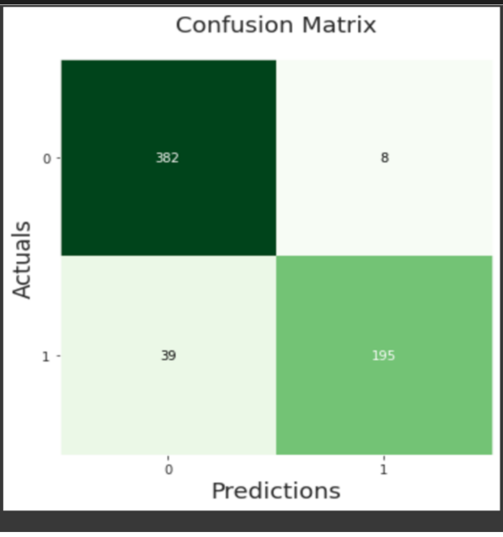
```
predictions = model.predict(x_test).round()
classes_x=np.argmax(x_test,axis=1)
predictions = predictions.reshape(1,-1)[0]
predictions[:15]
```

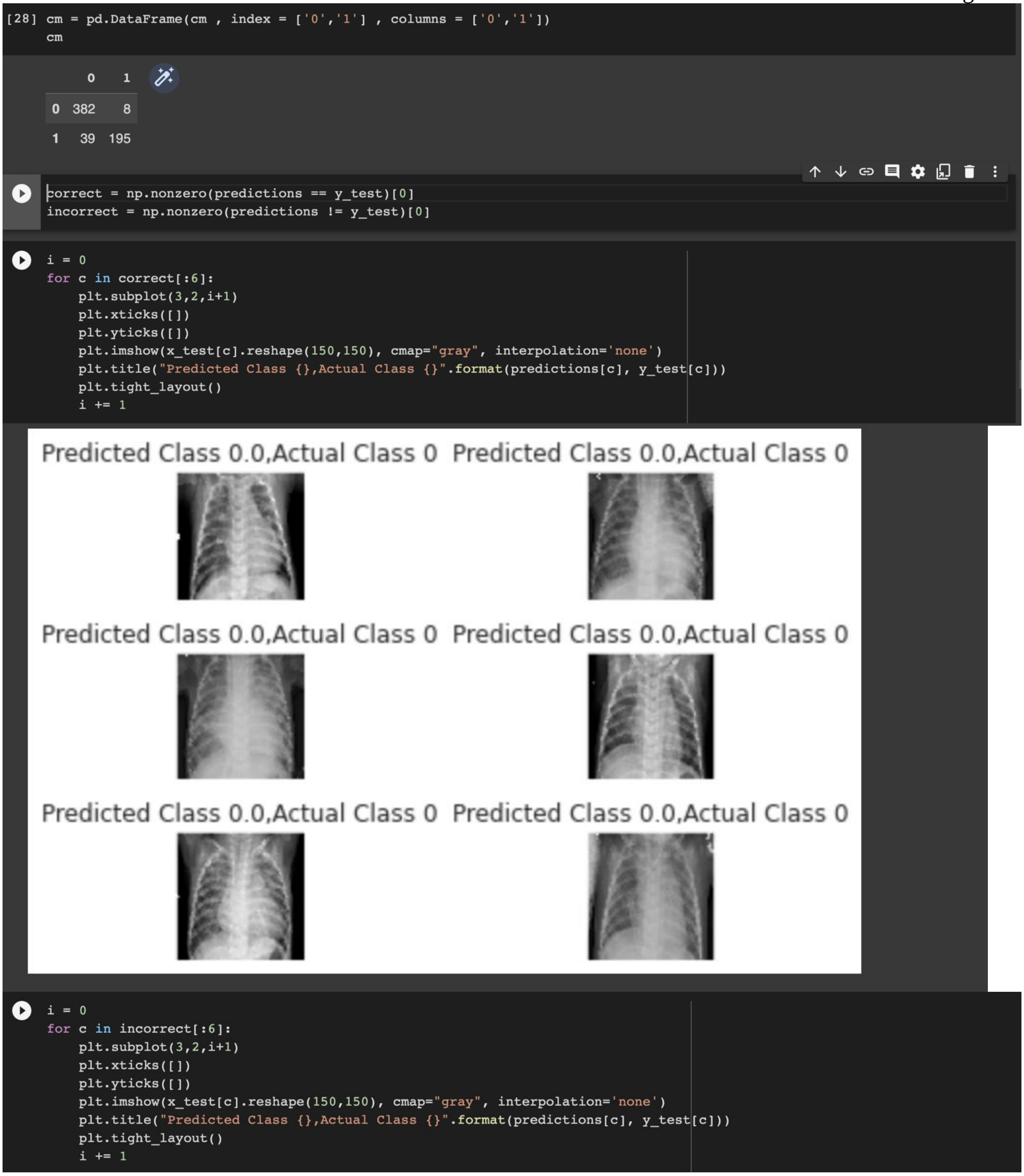
```
20/20 [==============================] - 9s 441ms/step
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
      dtype=float32)
```

```
print(classification_report(y_test, predictions, target_names = ['Pneumonia (Class 0)','Normal (Class 1)']))
```

```
                     precision    recall  f1-score   support

Pneumonia (Class 0)       0.91      0.98      0.94       390
   Normal (Class 1)       0.96      0.83      0.89       234

           accuracy                           0.92       624
          macro avg       0.93      0.91      0.92       624
       weighted avg       0.93      0.92      0.92       624
```

```
from mlxtend.plotting import plot_confusion_matrix

cm = confusion_matrix(y_test,predictions)

fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(6, 6), cmap=plt.cm.Greens)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



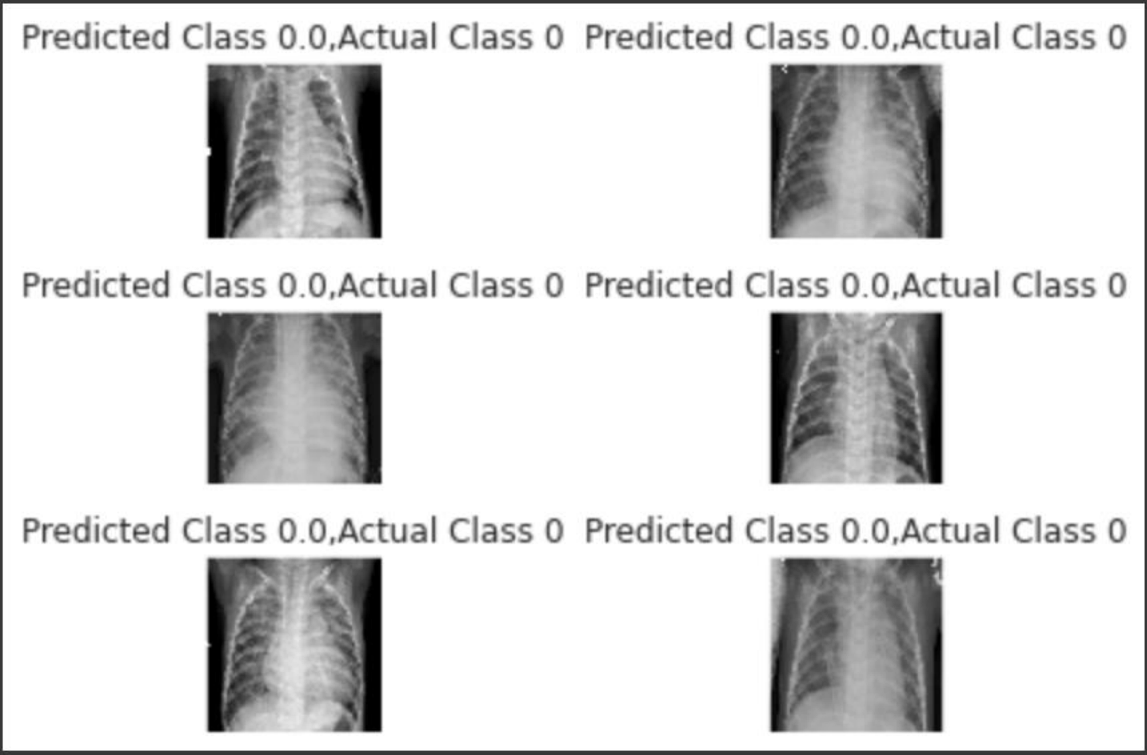


```
[28] cm = pd.DataFrame(cm , index = ['0','1'] , columns = ['0','1'])
     cm
```

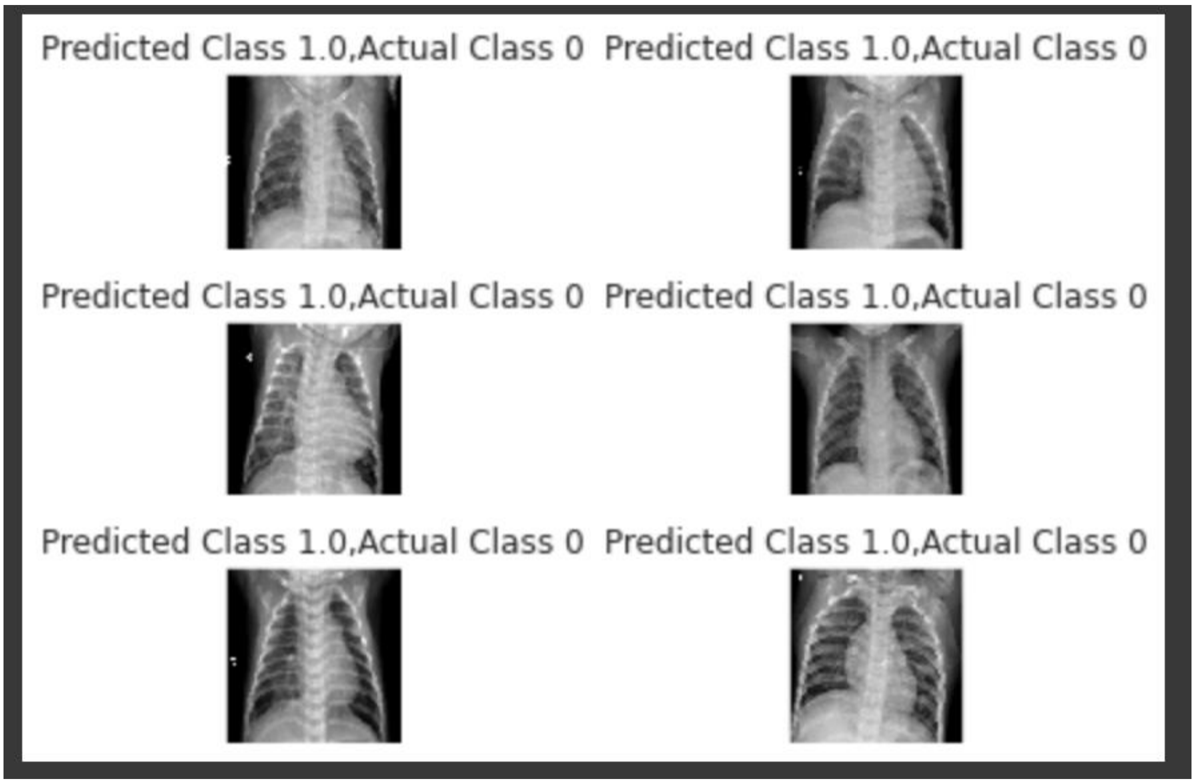
```
       0    1
0    382    8
1     39  195
```

```
correct = np.nonzero(predictions == y_test)[0]
incorrect = np.nonzero(predictions != y_test)[0]
```

```
i = 0
for c in correct[:6]:
    plt.subplot(3,2,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_test[c].reshape(150,150), cmap="gray", interpolation='none')
    plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y_test[c]))
    plt.tight_layout()
    i += 1
```


Predicted Class 0.0,Actual Class 0
Predicted Class 0.0,Actual Class 0
Predicted Class 0.0,Actual Class 0
Predicted Class 0.0,Actual Class 0
Predicted Class 0.0,Actual Class 0
Predicted Class 0.0,Actual Class 0

```
i = 0
for c in incorrect[:6]:
    plt.subplot(3,2,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_test[c].reshape(150,150), cmap="gray", interpolation='none')
    plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y_test[c]))
    plt.tight_layout()
    i += 1
```


Predicted Class 1.0,Actual Class 0
Predicted Class 1.0,Actual Class 0
Predicted Class 1.0,Actual Class 0
Predicted Class 1.0,Actual Class 0
Predicted Class 1.0,Actual Class 0
Predicted Class 1.0,Actual Class 0