Configuration Manual

MSc Research Project
Programme Name

Rahul Selvakumar
Student ID: X20231296

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rahul Selvakumar		
Student ID:	X20231296		
Programme	Msc. In Cybersecurity	**Year:**	2021-2022
Module:	Research Project		
Lecturer:	Niall Heffernan		
Submission Due Date:	15/08/2022		
Project Title:	A Signature based Ransomware detection using CNN		
Word Count:	1816 **Page Count:** 11		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Rahul Selvakumar
Date:	14/08/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1. Introduction

This configuration manual explains the requirements for the proposal "Ransomware early detection in early detection in a corporate network". This proposal is implemented using the latest version of the software and various hardware as listed below. The main aim of this research is to detect ransomware at an early stage using deep learning methods (convolutional neural networks). The ransomware is detected using a signature-based technique. These signatures are available in the RISS dataset. This RISS data set is fed into the CNN model in order to detect if the file contains the ransomware or not. This process is executed through three phases.

Phase I: Pre-processing the data.

Phase II: The model will be trained with the help of the RISS dataset.

Phase III: Testing the model with the help of the RISS dataset, which helps in detecting the ransomware.

2. System Requirements

Convolutional Neural network process requires a high amount of resources. In order to run the process in a seamless manner and with a reduced processing time, the following hardware and software are recommended. These values are not prerequisites. The specs of each program may be found on their respective official websites. The variation in values will have an impact on the project's performance.

2.1.Hardware Requirement:

This process was implemented using Dell Inspiron 5577, with

Hardware	Configurations
Processor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2801 Mhz, 4 Core(s), 8 Logical Processor(s)
RAM	16.0 GB DDR4
GPU	NVIDIA GeForce GTX 1050 Ti Max – Q Design
Hard Disk	1 TB NVMe SSD
OS	Windows 10 Pro 64 – bit

Since we are dealing with a large number of ransomware data in CNN analysis, and neural networks require a large amount of resources for the learning process. Even better hardware is suggested for maximum performance.

2.2. Software requirement:

The software requirement are listed below.

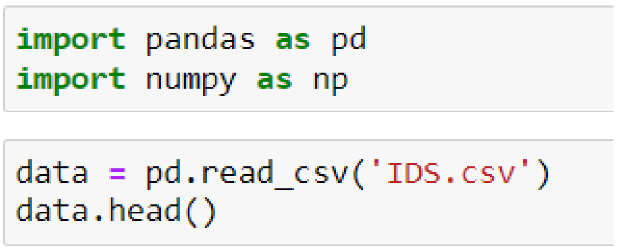
Software	Version
Anaconda Navigator	**2.1.1**
Jupyter Notebook	**6.4.11**
Python	**3.10.5**
Numpy (library)	**1.23.0**
Pandas (library)	**1.4.3**
Tensorflow	**2.9.0**
OpenCV	**4.6.0**
Scikit-learn	**0.24.0**
IBM SPSS	**28.0.0**

3. Data pre-processing

In this process the RISS dataset will be partitioned into two different dataset an then it will be converted into images. Which can be fed as an input to the CNN model.

3.1. Importing libraries:

- In this process, pandas and numpy libraries are used (fig 1). Numpy is used since CNN uses a matrices concept to convert the data into images and it is much quicker in processing the data. Whereas pandas are used for labelling the data in CNN and It has a variety of data structures and operations for working with numbers and time series.

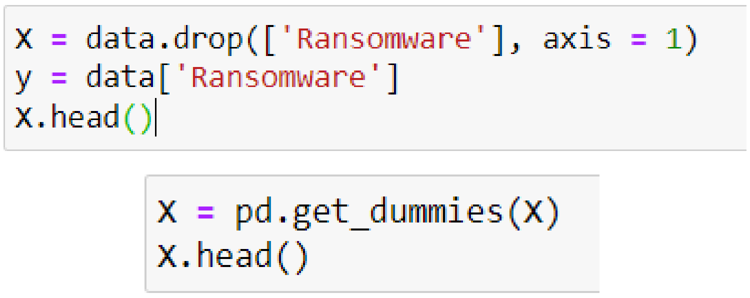


```
import pandas as pd
import numpy as np
```

```
data = pd.read_csv('IDS.csv')
data.head()
```

Fig 1: Importing libraries

- Now the feature 'Label' is dropped (In this case it is ransomware. The label varies depending upon the dataset) from the dataset. The data listed in the 'Label' will be assigned to the variable 'y' and the rest of the data will be stored in the variable 'x' as shown in figure 2.



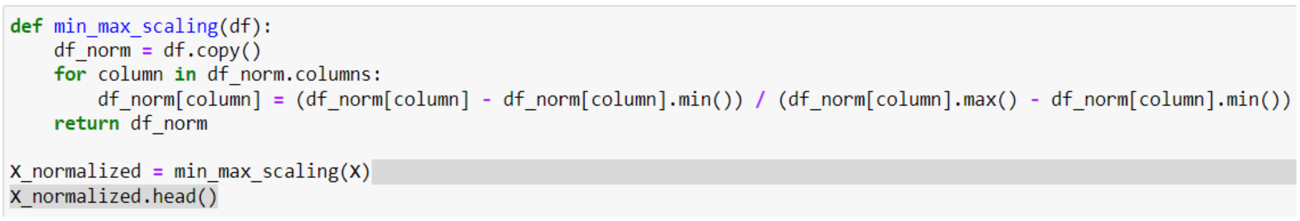
```
X = data.drop(['Ransomware'], axis = 1)
y = data['Ransomware']
X.head()
```

```
X = pd.get_dummies(X)
X.head()
```

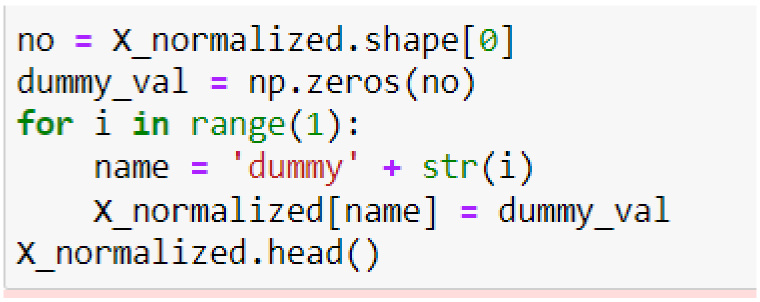
Fig 2: Assigning variables

3.2. Data Normalisation:

- In order to train the CNN, data must be filtered correctly in the network to avoid empty spaces. Which may come from different sources within the same range. If it is not filtered correctly, the learning process will slow down.
- In this case let's say that the data ranges from 0 to 1, 00,000, which is an enormous difference. Now let's divide the whole range by 1 so the result is a decimal number. In my case the label "Ransomaware_Family" has more value so the highest value will be divided with all other values (fig: 3).Which will be procssed by CNN easily.
- And the value in the for loop "1" represents number of dummy columns. In this process the dummy columns are created in order to create a perfect square matrix. In order to make the process easier. If in case there is a rectangular matrix it may lead to cardinality error.



```
def min_max_scaling(df):
    df_norm = df.copy()
    for column in df_norm.columns:
        df_norm[column] = (df_norm[column] - df_norm[column].min()) / (df_norm[column].max() - df_norm[column].min())
    return df_norm

X_normalized = min_max_scaling(X)
X_normalized.head()
```

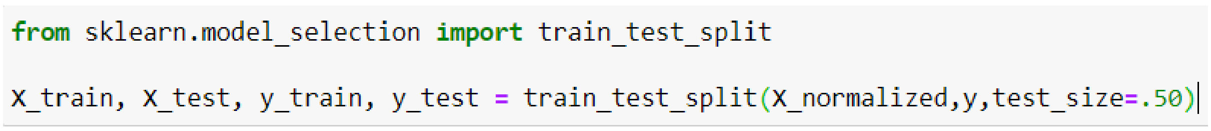


```
no = X_normalized.shape[0]
dummy_val = np.zeros(no)
for i in range(1):
    name = 'dummy' + str(i)
    X_normalized[name] = dummy_val
X_normalized.head()
```

Fig 3: Data Normalisation.

3.3. Splitting of dataset

The CNN should be trained in order to detect ransomware. In order to train the CNN, the dataset is split into two halves i.e., the dataset will be split into train and test with the help of the "train_test_split" function using "sklearn.model_seletion" library. The dataset is split in such a way that 50% of the dataset is used for testing and the remaining 50% is used for training (fig: 4).The dataset is split into 50:50 ratios in order to increase the accuracy. The more CNN gets trained, the better the output.

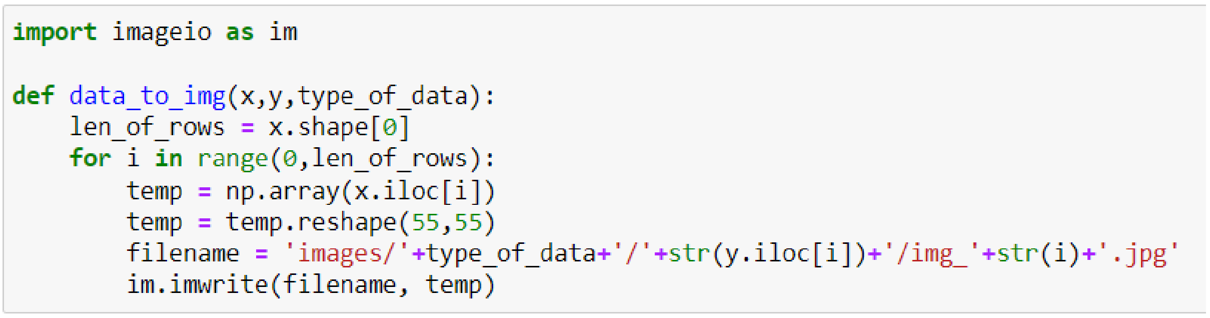


```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_normalized,y,test_size=.50)
```

Fig 4: Splitting of dataset.

3.4. Image conversion

The data that are in the form of array will be converted in to grey scale images as shown in (fig: 6, 7) and these images will be stored to their corresponding folder in my case it is “0” and “1” (0 represents no ransomware, 1 represents the presence of the ransomware). The code is written to form 55x55 matrix which is the perfect square of 3024 columns in order to generate a square matrix. (fig: 5). Note “data_to_image” is a function created on my own so that if I want to call in any other part of the program I can easily call it.



```
import imageio as im

def data_to_img(x,y,type_of_data):
    len_of_rows = x.shape[0]
    for i in range(0,len_of_rows):
        temp = np.array(x.iloc[i])
        temp = temp.reshape(55,55)
        filename = 'images/'+type_of_data+'/'+str(y.iloc[i])+'/img_'+str(i)+'.jpg'
        im.imwrite(filename, temp)
```

Fig 5: Image conversion code.

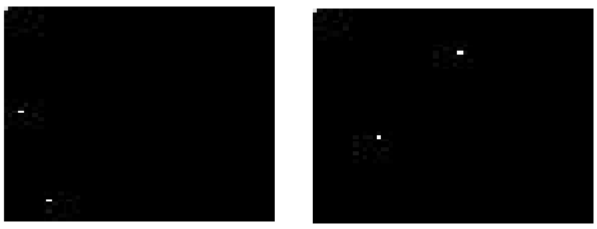


Fig 6: Image stored in 0 folder.

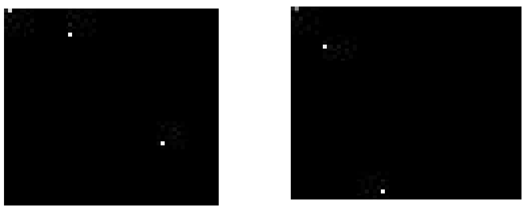


Fig 7: Image stored in 1 folder.

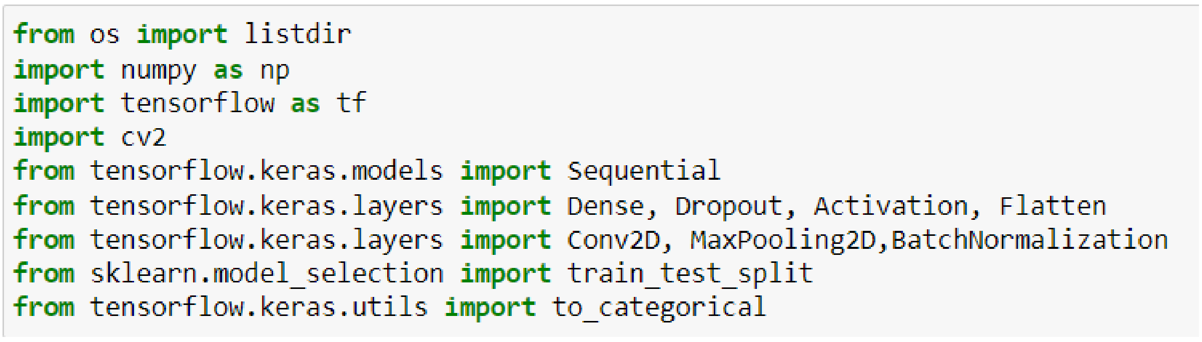
This is the end of the data preprocessing phase, and the converted images from the dataset can now be used for training and testing. Another set of data is passed through CNN in the same manner. The resulted images will be stored in a different place.

4. Training the model

In single CNN training model is created in this phase where 50% of the data are fed into training session. During the training phase, The CNN will be fed with the dataset consisting of a larger set of images that have been labelled with the class labels that correspond to them.

4.1 Importing libraries

Number of libraries are imported in to perform various task in training the CNN. “listdir” is used to fetch the list of files and directories from specific directories. “CV” is used to process and identify the image. From “tensorflow.keras.modles” is imported from “Sequential” function which is used to create model layer by layer. From "tensorflow.keras.layers" ‘Dense’,‘Activation’ and‘Flattern’ functions are imported all these functions are used in different layers of CNN. "tensorflow_keras_utils" function is used to differentiate the labels which are used to different good ware and ransomware (fig: 4).

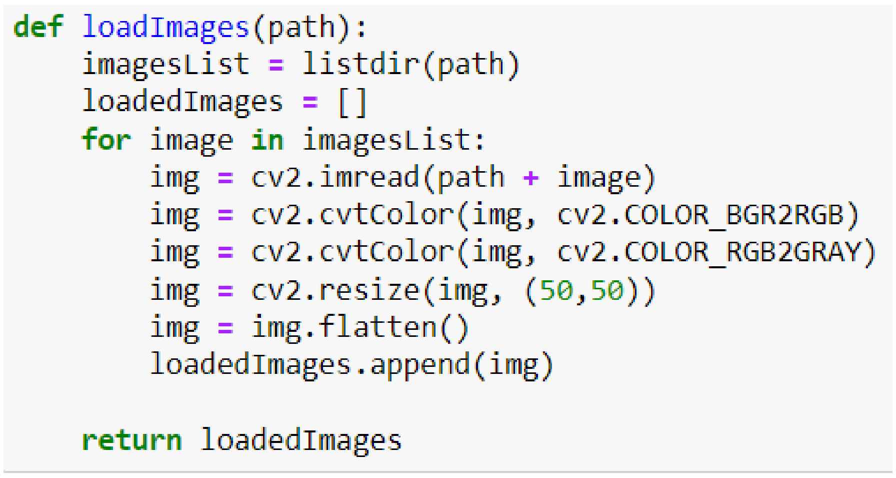


```
from os import listdir
import numpy as np
import tensorflow as tf
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D,BatchNormalization
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
```

Fig: 8 importing libraries

4.2 Grey scale images conversion

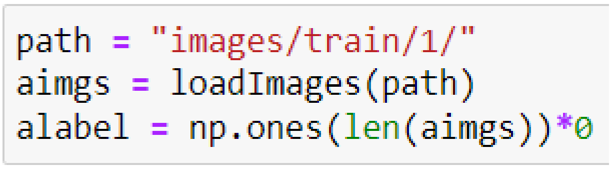
The final image will be converted into grey scale image. The output image will be in the form of BGR which then will be converted into RGB. From RGB the image will be converted into gray scale. This process is done to improve the efficiency of CNN. The image will be resized into 50x50.



```
def loadImages(path):
    imagesList = listdir(path)
    loadedImages = []
    for image in imagesList:
        img = cv2.imread(path + image)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        img = cv2.resize(img, (50,50))
        img = img.flatten()
        loadedImages.append(img)

    return loadedImages
```

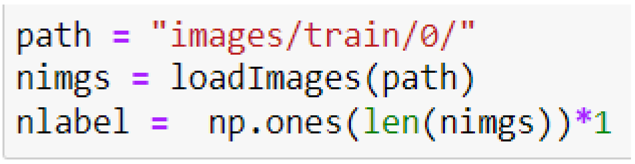
Fig9: Converting grey scale image

In (fig: 10) the anomaly images will be stored in the “images/train/1” path (fig: 10). The benign images will be stored in the “images/train/0” path (fig: 11).



```
path = "images/train/1/"
aimgs = loadImages(path)
alabel = np.ones(len(aimgs))*0
```

Fig10: Anomaly images

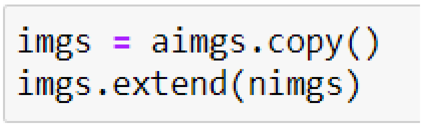


```
path = "images/train/0/"
nimgs = loadImages(path)
nlabel =  np.ones(len(nimgs))*1
```

Fig11: Benign images

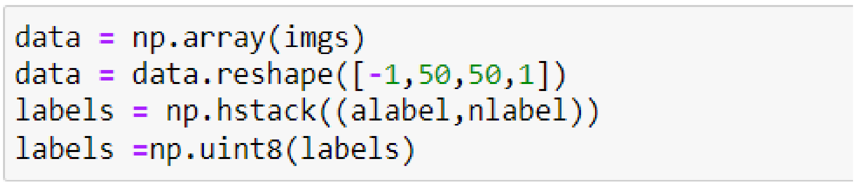
4.3 List conversion:

In the bellow step the anomaly image and the benign images are stored under a under a variable “imgs”



```
imgs = aimgs.copy()
imgs.extend(nimgs)
```

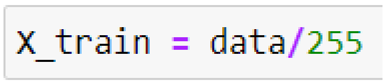
Fig12: New variable



```
data = np.array(imgs)
data = data.reshape([-1,50,50,1])
labels = np.hstack((alabel,nlabel))
labels =np.uint8(labels)
```

Fig13:

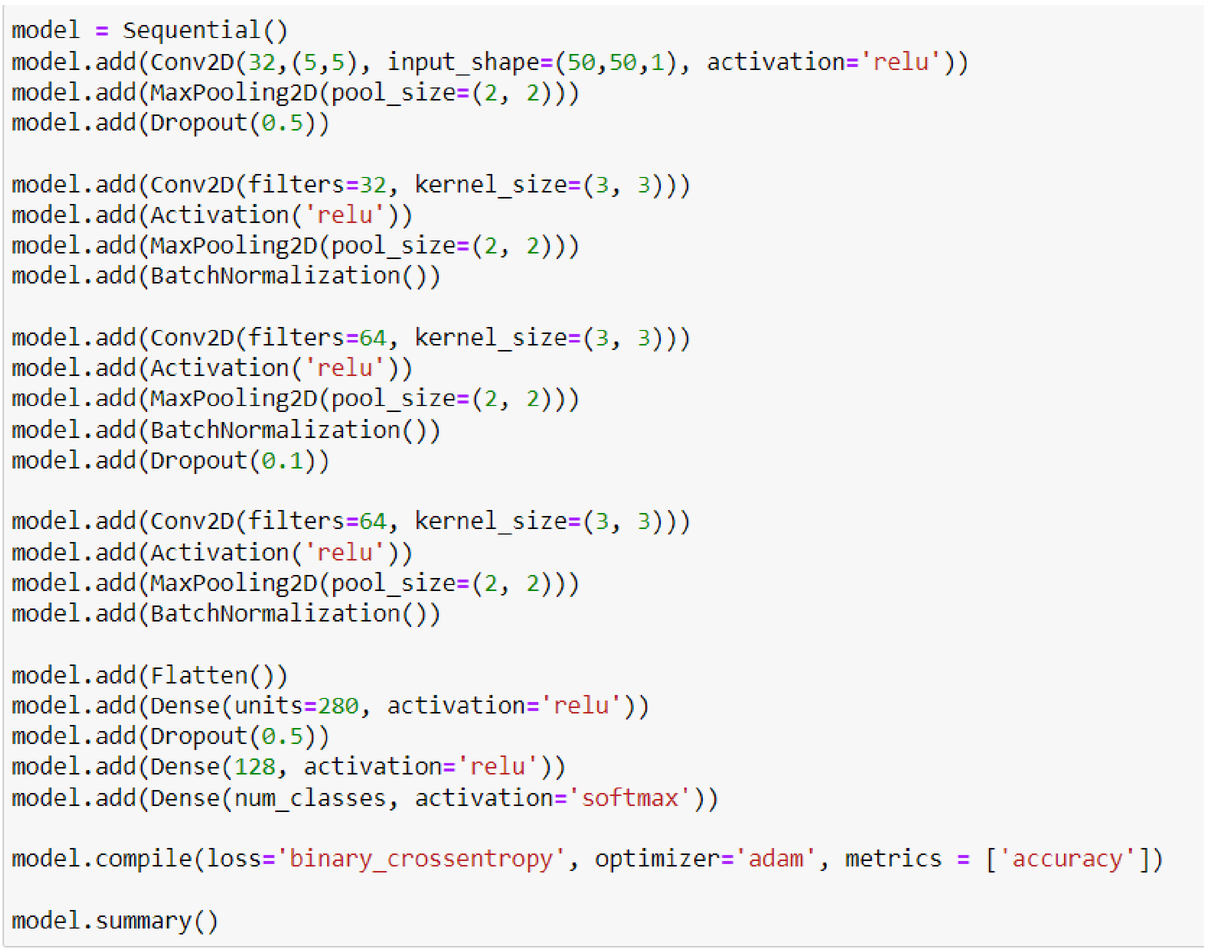
In the (fig 14) the X_train will be divided with 255. The pixel value of each image will be divided by 255 since it is the highest possible value of the pixel.



```
X_train = data/255
```

Fig14: dividing by maximum pixel value

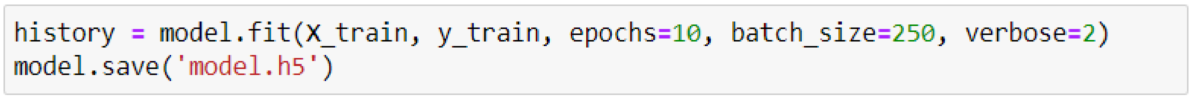
In (fig 15) the convolutional layer the data are filtered using three filters 32 bit, 64 bit and 64 bit filter with the size of 3x3. Each and every layer consist of maxpooling and with the dropout of 0.5 and 0.1.with the kernel size of 3x3. Each and every layer is activated with “Relu” function



```
model = Sequential()
model.add(Conv2D(32,(5,5), input_shape=(50,50,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=32, kernel_size=(3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=64, kernel_size=(3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.1))

model.add(Conv2D(filters=64, kernel_size=(3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(units=280, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])

model.summary()
```

Fig15: Adding filters

In (fig: 16) A function named 'fusion.model.fir' is used at the end of the code to load the function of Figure 15 and specify the input and output variables. During training, the epoch's parameter specifies how many iterations are carried out, while batch size specifies how large each batch is. Models were then saved using fusion model.save.

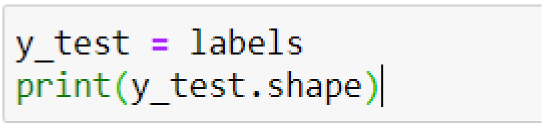


```
history = model.fit(X_train, y_train, epochs=10, batch_size=250, verbose=2)
model.save('model.h5')
```

Fig15: epoch and batch size.

5. Testing the CNN

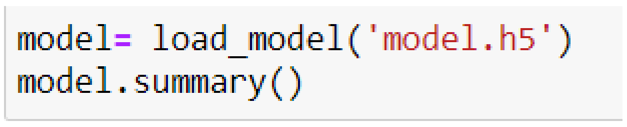
Similar steps will be followed from importing libraries until the X-test from the training section.Now Under Y_test, the label column will be segregated (fig 16).



```
y_test = labels
print(y_test.shape)
```

Fig16: label are segregated

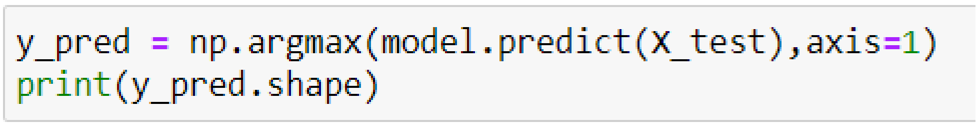
All the images and that are trained will stored in “model” variable under “model.h5”. This file cannot be viewed by the user.



```
model= load_model('model.h5')
model.summary()
```

Fig17: model.h5

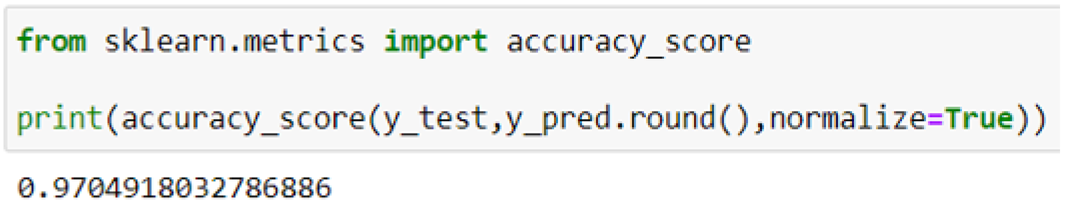
The trained model is loaded for testing using the 'load model' function, as illustrated in (Fig: 17). In (Fig:20), the prediction function is built using'model.predict,' which is given to the variable 'y pred,' which contains the input values supplied to the loaded model.



```
y_pred = np.argmax(model.predict(X_test),axis=1)
print(y_pred.shape)
```

Fig18: prediction using training

The accuracy score is calculated using the function 'accuracy score' from the library 'sklearn.metrics,' as illustrated in fig 19.

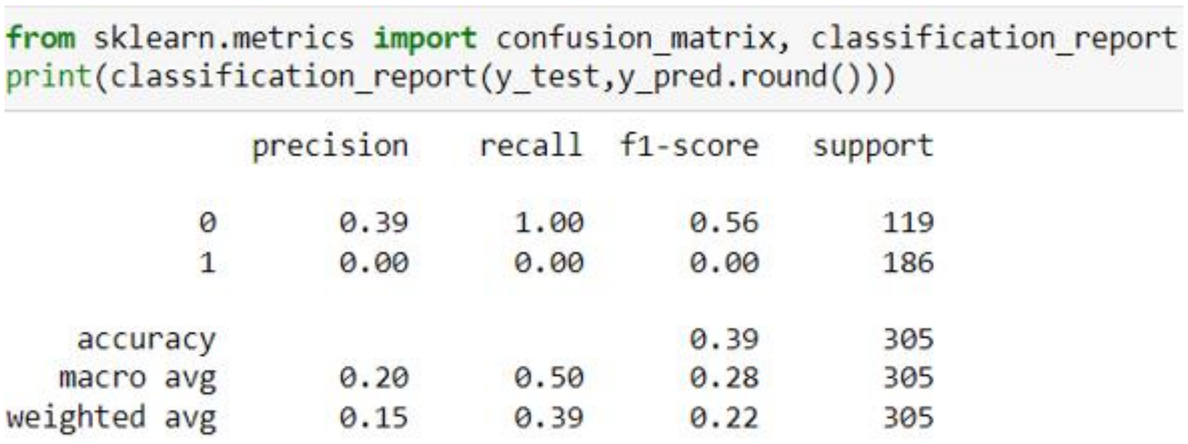


```
from sklearn.metrics import accuracy_score

print(accuracy_score(y_test,y_pred.round(),normalize=True))
```

```
0.9704918032786886
```

Fig19: Accuracy calculation

With the same “Sklearn.metrics” the values such as precesion, recall, and F1 score are calculated.

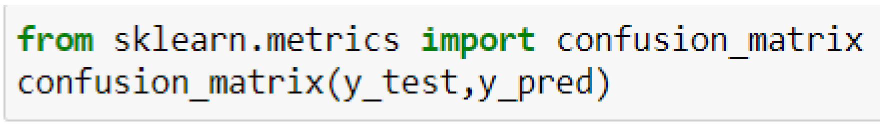


```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test,y_pred.round()))
```

```
              precision    recall  f1-score   support

           0       0.39      1.00      0.56       119
           1       0.00      0.00      0.00       186

    accuracy                           0.39       305
   macro avg       0.20      0.50      0.28       305
weighted avg       0.15      0.39      0.22       305
```

Fig20: precesion, recall, f1 score

With the help of “Sklern. Matrix” the confusion matrix is obtained.



```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

6. Conclusion:

Thus this configuration manual Explains working of CNN with the accuracy of 97.04%. And this program will be able to detect the ransomware the signatures. And this method can be used in early detection.