Configuration Manual

MSc Research Project

Phishing Detection in Emails using multi-Convolutional neural Network fusion

Dharani Kumar Babu
Student ID: X20179197

School of Computing
National College of Ireland

Supervisor: Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Dharani Kumar Babu
Student ID:	X20179197
Programme:	MSc in Cybersecurity **Year:** 2021-2022
Module:	Research Project
Lecturer:	Michael Prior
Submission Due Date:	15.08.2022
Project Title:	Phishing Detection in Emails using multi-Convolutional neural Network fusion
Word Count: 1465	**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: Dharani Kumar Babu

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 report proposes Phishing detection in emails using Multi-Convolutional Neural Network fusion. The various software and hardware requirements are used for implementation. The step-by-step implementation procedure are listed below. The aim of this research is to detect the phishing URLs in emails using Convolutional Neural Network. This research paper implements the methodology that detects the legitimate URLs and filter the phishy URLs using multi-Convolutional neural network. This implementation is executed in three steps. The first step is data pre-processing, second step is training the model and the final step is to test the model.

2. System Requirements

For the smooth processing of the model and reduce the processing time, following hardware and software are required.

2.1. Hardware Requirements

The implementation was performed in Dell XPS 15 laptop, the configuration of the device is as follows

1. Processor – Intel(R) Core i7-8750H CPU @ 2.20GHz
2. RAM - 16.0 GB DDR4
3. GPU – NVIDIA GeForce GTX 1050 Ti Max – Q Design
4. Hard Disk – 1 TB PM981 NVMe SSD
5. OS – Windows 10 Home 64 – bit

2.2. Software Requirements

The software requirements are listed below.

Software	Version
Python	3.8.3
Anaconda Navigator	1.9.12
Jupyter Notebook	6.0.3
Tensorflow	2.4.0
Numpy	1.18.5
OpenCV	4.4.0
Scikit-learn	0.23.1

3. Data pre-processing:

It is the first step to perform some pre-processing on any image data. The phishing dataset are converted into images. The converted images are given as input into the CNN model.

3.1. Importing Libraries:

The ‘pandas’ and ‘numpy’ library are imported for the pre-processing of data. Once the libraries are imported, it is necessary to drop the unwanted columns and segregate the result column. Using ‘Pandas’ function dataset is given as input.

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

```
data = pd.read_csv('Phishing.csv')
data.dropna(inplace = True)
data.head()
```

Fig3: Importing libraries

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

Fig4: assigning variable

3.2. Data Normalisation:

In this process , we convert each row into square matrix and this matrix is converted into grey scale image. The maximum values range from 0 to 1.

```
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()
```

Fig5: normalising the data

3.3. Splitting Train and Test

Using the function ‘train_test_split’ the phishing dataset is separated into train and test from ‘sklearn.model_selection’ library. 50% of data is used for training and the 50% is used for test purpose. The advantage of using 50:50 ratio will reduce loss and increase the accuracy.

```
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)
```

Fig6: splitting the dataset

3.4. Converting into images:

With the help of pre-processing the data are converted into images and saved into respective folder namely ‘Legitimate’ and ‘Phishy’. The 7x7 size images are created and it is shown in fig 7. The converted images are used in training and testing process.

```
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(7,7)
        filename = 'images/'+type_of_data+'/'+str(y.iloc[i])+'/img_'+str(i)+'.jpg'
        im.imwrite(filename, temp)
```

Fig7: converting into image

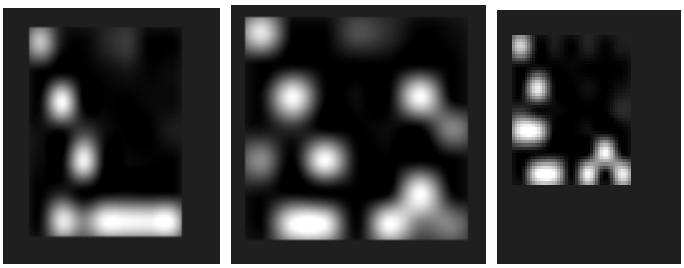

Fig 8: Sample images converted from dataset

4. Training the model:

The multi-Convolutional neural network model is created, and it is trained with 50% of the data.

4.1. Importing Libraries:

Below figure provides the different libraries function that are imported to create the model.

```
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, concatenate
from tensorflow.keras.layers import Conv2D, MaxPooling2D,BatchNormalization
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

Fig 9: importing libraries

The ‘listdir’ function is used to read data from the folder. ‘cv2’ library is used for image processing. ‘Sequential’ function ‘tensorflow.keras.models’ are used to begin the convolutional neural network model. The CNN layers use functions such as ‘Dense’. ‘Activation’, ‘Flattern’, and ‘Concatinate’ and these functions are imported from ‘tensorflow.keras.layers’. ‘concatinate’ function is used for combining the output from each model and create single output. For the categorial classification of lables ‘to_categorial’ function is used from ‘tensorflow.keras.utilis’. This helps in classification between ‘Legitimate’ and ‘phishy’.

4.2. Converting into grey scale images:

```
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
```

Fig 10: Image loading and grey scale conversion

In this conversion process, firstly the image converted to BGR from RGB and to grey scale image, then the image is resized by 50x50.

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

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

Fig 11: Reading the images

By using 'LoadImages' function the images are loaded from the saved path.

4.3. Converting to list:

The listed grey scale images are stored in variable'X_train'. The highest possible pixel value in image is 255, so it has been divided by 255.

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

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

```
X_train = data/255
print(X_train.shape)
```

Fig 12: converting to list and storing as 'X_train'

4.4. Creating the Model:

'model1' and 'model2' are the two models that has been created.

```
model1 = Sequential()

model1.add(Conv2D(32,(5,5), input_shape=(50,50,1), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.5))

model1.add(Conv2D(filters=32, kernel_size=(3, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(BatchNormalization())

model1.add(Conv2D(filters=64, kernel_size=(3, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(BatchNormalization())
model1.add(Dropout(0.1))

model1.add(Conv2D(filters=128, kernel_size=(3, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(BatchNormalization())


model1.add(Flatten())


model1.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])

#model1.summary()
```

Fig 13: Model1 CNN

With the Sequential() function the model has been initiated along to feed into input layer. Once the model has been created it consists of three layers as shown in Fig12. They are Activation, Maxpooling, and Batch Normalisation. Each filter has its size of 32,64,128. In

each layer ‘ReLU’ is the activation function. CNN model stops at flatten layer before hidden layer. With the same procedure the second model2 is created that has been shown below in fig13.

```
model2 = Sequential()
model2.add(Conv2D(32,(5,5), input_shape=(50,50,1), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.5))

model2.add(Conv2D(filters=32, kernel_size=(3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(BatchNormalization())

model2.add(Conv2D(filters=64, kernel_size=(3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(BatchNormalization())
model2.add(Dropout(0.1))

model2.add(Flatten())

model2.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])

#model2.summary()
```

Fig 14: Model2 CNN

4.5. Fusion Model:

In this fusion model, the output of both model1 and model2 are combined. Each model output called by ‘model.output’ function. ‘concatenate’ function is used to combine two models output together. The hidden layer is feed with combined output as input, where hidden layer contains three different dense sizes 512, 256, 128. The ‘ReLU’ function is used to activate this dense layer in the model. ‘softmax’ function is used to get the output which is the final layer of Convolutional neural network.

```
model0 = concatenate([model1.output, model2.output])

x = Dense(512, activation='relu')(model0)
x = Dropout(0.4) (x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.4) (x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.4) (x)
output = Dense(num_classes,activation='softmax')(x)
```

Fig 15: Fusion model

The final model is created using ‘tf.keras.Model’ where the input is feed as values and given to the model and the output from the model is received as an output. ‘RMSprop’ function is used to increase the learning rate. ‘final_model.compile’ is used to compile the fusion model and loss are maintained by ‘ binary_crossentropy’ function.

```
final_model = tf.keras.Model(inputs=[model1.input, model2.input], outputs=[output])

rm = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.8)
rm2 = RMSprop(learning_rate=0.001, rho=0.9)
rm3 = tf.keras.optimizers.Adagrad(learning_rate=0.01)
rm4 = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)

final_model.compile(loss='binary_crossentropy', optimizer=rm, metrics=['accuracy'])
```

Fig 16: compiling the Fusion model

4.6. Training:

```
final_model.fit([X_train,X_train], y_train, epochs=40, batch_size=200, verbose=2)
final_model.save('model.h5')
```

Fig 17: Training the model

‘final_model.fit’ function is used to measure the model performance on how well it produces output on trained data. The number of iterations is produced by epochs value and batch_size provides the size of each batch. By using ‘final_model.save’ function the final model is saved and where the model represented by ‘model.h5’.

5. Testing the model:

In this multi-convolutional neural network model , the 50% of dataset are tested and that data is given into the model that detects the legitimate and phishy with accuracy. In this we load the trained model and the input from test data is stored as ‘X_test’ and ‘X_test’. The ‘y_test’ variable consists of labels that are stored.

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

Fig 18: Loading trained model

‘load_model’ function is used to load the trained model that has been showed in fig16. ‘model.predict' is used to generate the prediction function, which is then given to the variable 'y pred', which contains the input values given to the loaded model.

```
pred = model.predict([X_test, X_test])
```

Fig 19: Trained model prediction

The accuracy is obtained by using ‘accuracy_score’ from library ‘sklearn.metrics’ that has been show below fig18.

```
from sklearn.metrics import accuracy_score

print(accuracy_score(y_test,pred.round()))
```

Fig 20: Obtaining Accuracy

From fig19, it shows that the values are classified using 'classification_report' function from 'sklearn.metrics' library, where we get precision, recall and F1- score.

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

Fig 21: Values classification

Finally, we get confusion matrix as the final step by using 'confusion_matrix' function shown in fig20.

```
confusion_matrix(y_test.argmax(axis=1), pred.round().argmax(axis=1))
```

Fig 22: confusion matrix

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

```
pred = model.predict([X_test, X_test])
```

```
32/32 [==============================] - 3s 65ms/step
```

```
from sklearn.metrics import accuracy_score

print(accuracy_score(y_test,pred.round()))
```

```
0.98
```

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

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       471
           1       0.98      1.00      0.99       529

   micro avg       0.99      0.99      0.99      1000
   macro avg       0.99      0.99      0.99      1000
weighted avg       0.99      0.99      0.99      1000
 samples avg       0.99      0.99      0.99      1000
```

```
confusion_matrix(
    y_test.argmax(axis=1), pred.round().argmax(axis=1))
```

```
array([[425,  46],
       [  0, 529]], dtype=int64)
```

Fig 23: Testing results

6. Conclusion

Thus, the configuration manual consists of the implementation of multi-CNN model along with the figures representing the outcomes. Both Model1 and Model2 are created and combined, and the multi-CNN model obtained 98% accuracy. This method of implementation provides a better efficiency in detecting legitimate and phishing emails.