National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Thokala Poojitha
Student ID: x22230424

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……Thokala Poojitha ……

Student ID: ……x22230424……..……

Programme: …… MSc in Cloud Computing…… **Year:** ……2024……..

Module: …… MSc Research Project ……..……

Lecturer: …… Vikas Sahni……..……

Submission Due Date: ……12-12-2024……..……

Project Title: Harnessing Deep Features and Machine Learning for Malware Image Classification……..……

Word Count: ……1462…… **Page Count:** ……16……..……..……

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: ……Poojitha Thokala……

Date: ……11-12-2024……

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):	

Configuration Manual

Thokala Poojitha
Student ID: x22230424

1 Introduction

This configuration manual provides the steps to replicate the research conducted for malware detection and classification using a hybrid approach of Convolutional Neural Networks (CNNs) for feature extraction and machine learning models, specifically Support Vector Machines (SVM) and Random Forest (RF) for classification. It contains the software, hardware, and implementation details required to replicate the results.

2 System Configuration

2.1 Software Configuration

Platform: Google Colab (preferred for GPU support)
Python: Version 3.10
Libraries: All the required libraries are shown in Table 1, Figure 1 and Figure 2

Library	Version
TensorFlow	2.17.0
NumPy	1.26.4
Pandas	2.1.4
Scikit-learn	1.3.2
Matplotlib	3.7.1
Seaborn	0.13.1

Table 1: Library versions

```
import warnings
warnings.filterwarnings("ignore")
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from tensorflow.keras.utils import to_categorical

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D,  MaxPool2D, Dense, Dropout, Flatten
from tensorflow.keras.optimizers import Adam

import tensorflow as tf
from tensorflow.keras.models import Model
import pandas as pd
```

Figure 1: Libraries for CNN model

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split

import seaborn as sns
import pickle
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

Figure 2: Libraries for CNN model

3 Dataset Preparation

1. Dataset source: The Malimg Dataset[1] is used in this project, which has grayscale images of malware binaries classified into 25 malware families. All the malware families are shown in the table 2.

Table 2: 25 classes

Allaple.A	Cme.B	Fakealert
Allaple.L	Citep.A	Fizzer
Adialer.C	Curlie	Gamarue
Bagle	Dialer.A	Gri
Brontok	Dubr.A	Hupigon
Dumball.A	Luder	MyDoom.A

[1] https://paperswithcode.com/dataset/malimg

Rbot	Redlof	Rimecud
Sasser	Simda	Sobig
Zbot		

2. Preprocessing steps: After downloading the dataset, several preprocessing steps are taken to prepare the dataset for analysis and classification.

    a. The malware binaries are first analysed to ensure proper labelling and organization by their respective malware families as shown in figure 3 and 4.
    b. The binary malware files are converted into grayscale images to enable image-based analysis. This transformation allows the model to detect spatial patterns that are unique to each malware family. This has been already done in the dataset that has been selected.
    c. Then to ensure classes would not have imbalances, the dataset is filtered such that no malware family contains more than 200 images which is shown in figure 5 and figure 6. This makes sure that training process remains unbiased and prevents overfitting to dominant classes.

```
base_dir_test = "malimg_paper_dataset_imgs"
output_dir = "Dataset/ExtDataset"
MAX_IMAGES_PER_LABEL = 200

if os.path.exists(output_dir):
    shutil.rmtree(output_dir)  # Delete the folder and its contents
os.makedirs(output_dir)  # Recreate the empty directory

for label in class_labels:
    modified_label = label.replace('!', '_').replace('.', '_')

    source_path = os.path.join(base_dir_test, label)
    target_path = os.path.join(output_dir, modified_label)

    # Create target directory for the modified class name
    if not os.path.exists(target_path):
        os.makedirs(target_path)

    # Copy up to 200 images
    image_count = 0
    for img in os.listdir(source_path):
        if image_count >= MAX_IMAGES_PER_LABEL:
            break  # Stop after reaching the limit

        try:
            # Source and destination file paths
            src_file = os.path.join(source_path, img)
            dst_file = os.path.join(target_path, img)

            # Copy the file
```

```
                shutil.copy(src_file, dst_file)
                image_count += 1
            except Exception as e:
                print(f"Error copying {img}: {e}")

        print(f"{image_count} images for label '{label}' into '{target_path}'")

    print(f"Dataset with limited images saved to '{output_dir}'")
```

Figure 3: Code for labelling and organization

```
116 images for label 'Agent.FYI' into 'Dataset/ExtDataset/Agent_FYI'
122 images for label 'Adialer.C' into 'Dataset/ExtDataset/Adialer_C'
200 images for label 'Allaple.A' into 'Dataset/ExtDataset/Allaple_A'
200 images for label 'Allaple.L' into 'Dataset/ExtDataset/Allaple_L'
200 images for label 'C2LOP.gen!g' into 'Dataset/ExtDataset/C2LOP_gen_g'
146 images for label 'C2LOP.P' into 'Dataset/ExtDataset/C2LOP_P'
106 images for label 'Autorun.K' into 'Dataset/ExtDataset/Autorun_K'
177 images for label 'Dialplatform.B' into 'Dataset/ExtDataset/Dialplatform_B'
198 images for label 'Alueron.gen!J' into 'Dataset/ExtDataset/Alueron_gen_J'
200 images for label 'Fakerean' into 'Dataset/ExtDataset/Fakerean'
162 images for label 'Dontovo.A' into 'Dataset/ExtDataset/Dontovo_A'
123 images for label 'Lolyda.AA3' into 'Dataset/ExtDataset/Lolyda_AA3'
200 images for label 'Lolyda.AA1' into 'Dataset/ExtDataset/Lolyda_AA1'
200 images for label 'Instantaccess' into 'Dataset/ExtDataset/Instantaccess'
184 images for label 'Lolyda.AA2' into 'Dataset/ExtDataset/Lolyda_AA2'
142 images for label 'Obfuscator.AD' into 'Dataset/ExtDataset/Obfuscator_AD'
80 images for label 'Skintrim.N' into 'Dataset/ExtDataset/Skintrim_N'
158 images for label 'Rbot!gen' into 'Dataset/ExtDataset/Rbot_gen'
136 images for label 'Malex.gen!J' into 'Dataset/ExtDataset/Malex_gen_J'
159 images for label 'Lolyda.AT' into 'Dataset/ExtDataset/Lolyda_AT'
128 images for label 'Swizzor.gen!E' into 'Dataset/ExtDataset/Swizzor_gen_E'
200 images for label 'VB.AT' into 'Dataset/ExtDataset/VB_AT'
97 images for label 'Wintrim.BX' into 'Dataset/ExtDataset/Wintrim_BX'
132 images for label 'Swizzor.gen!I' into 'Dataset/ExtDataset/Swizzor_gen_I'
200 images for label 'Yuner.A' into 'Dataset/ExtDataset/Yuner_A'
Dataset with limited images saved to 'Dataset/ExtDataset'
```

Figure 4: The images after labelling and organization

```
base_dir_test = "Dataset/ExtDataset"

# Initialize lists for filtered labels and counts
class_labels_filtered = []
class_counts_filtered = []

count = 0
index = 1

total_images = 0

for label in os.listdir(base_dir_test):
    path = os.path.join(base_dir_test, label)

    if os.path.isdir(path):  # Ensure it's a directory and not a file
        image_count = len([img for img in os.listdir(path) if os.path.isfile(os.path.join(path, img))])

        # Add to filtered labels and counts if image_count <= 200
        if image_count <= 200:
            class_labels_filtered.append(label)
            class_counts_filtered.append(image_count)
            print(f"{index:>2}. {label:<20}: {image_count}")
            total_images += image_count
        index += 1

print(f"\nTotal images: {total_images}")
```

Figure 5: code for image count to be less then 200

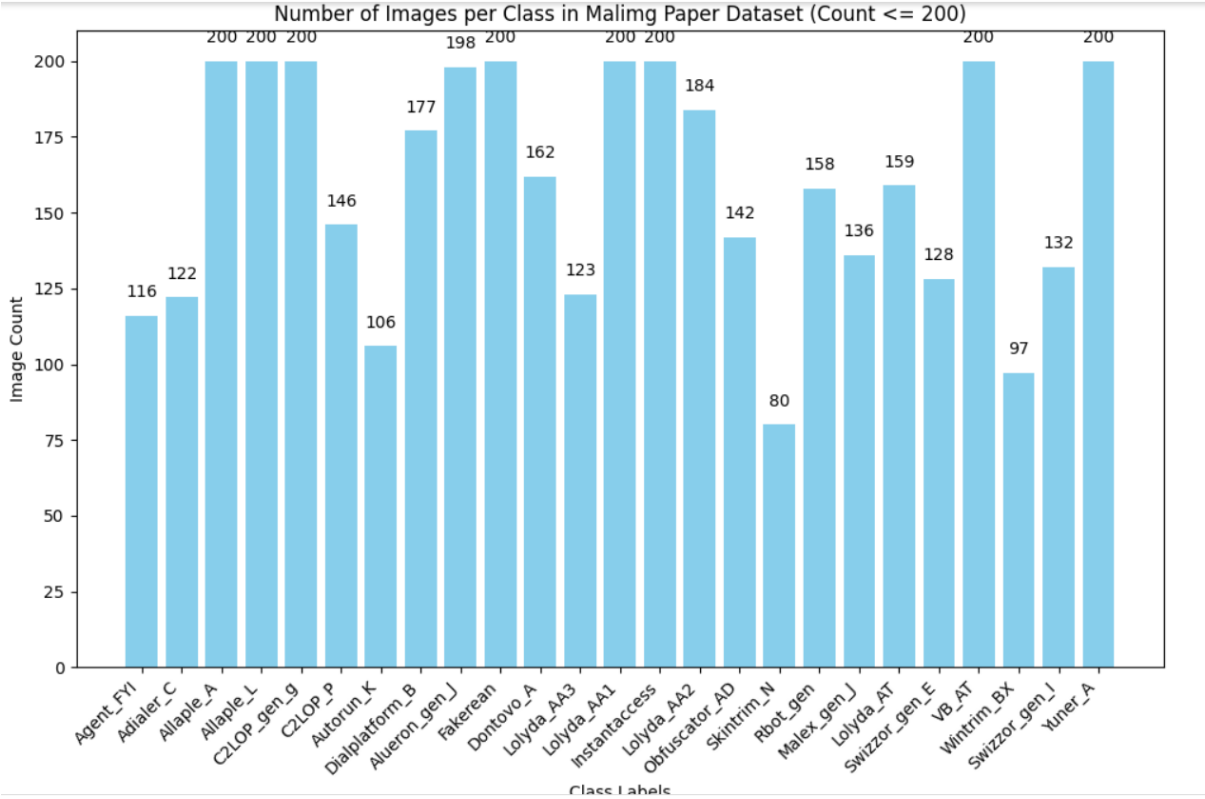

Figure 6: The image count after preprocessing and class balancing

The preprocessing steps are now applied to grayscale images for effective analysis and training. Each image is resized to 256*256 pixels to maintain uniformity across the dataset. This ensures that all the images have same dimensions which is a requirement for input into the CNN. Code is shown in figure 5.

```python
for label in class_labels:
    i=0
    path = os.path.join(base_dir_train, label)
    label_mapping[label] = count
    print(f"Path: {path:<40} | Label: {count:<5} | Class Label: {label}")

    for img in os.listdir(path):
        try:
            image=load_img(os.path.join(path, img), color_mode='rgb', target_size=(256,256))
            image=img_to_array(image)
            image=image/255.0
            train_data.append([image,count])
        except Exception as e:
                print(f"{os.path.join(path, img)}:")
    count=count+1

print(len(train_data))
```

```
Path: Dataset/ExtDataset/Agent_FYI             | Label: 0     | Class Label: Agent_FYI
Path: Dataset/ExtDataset/Adialer_C             | Label: 1     | Class Label: Adialer_C
Path: Dataset/ExtDataset/Allaple_A             | Label: 2     | Class Label: Allaple_A
Path: Dataset/ExtDataset/Allaple_L             | Label: 3     | Class Label: Allaple_L
Path: Dataset/ExtDataset/C2LOP_gen_g           | Label: 4     | Class Label: C2LOP_gen_g
Path: Dataset/ExtDataset/C2LOP_P               | Label: 5     | Class Label: C2LOP_P
Path: Dataset/ExtDataset/Autorun_K             | Label: 6     | Class Label: Autorun_K
Path: Dataset/ExtDataset/Dialplatform_B        | Label: 7     | Class Label: Dialplatform_B
Path: Dataset/ExtDataset/Alueron_gen_J         | Label: 8     | Class Label: Alueron_gen_J
Path: Dataset/ExtDataset/Fakerean              | Label: 9     | Class Label: Fakerean
Path: Dataset/ExtDataset/Dontovo_A             | Label: 10    | Class Label: Dontovo_A
Path: Dataset/ExtDataset/Lolyda_AA3            | Label: 11    | Class Label: Lolyda_AA3
Path: Dataset/ExtDataset/Lolyda_AA1            | Label: 12    | Class Label: Lolyda_AA1
```

Figure 7: Resizing imges for CNN input

The pixel values are then normalized to the range [0,1], which helps pixel intensity values that improves model convergence during training and reduces computational complexity. Each malware class is then limited to maximum of 200 samples to address the class imbalance. This ensures fair representation of classes in training process, that prevents model from being biased towards over-presented classes. Shown in figure 8 and code in figure 7.

```python
plt.show()
```

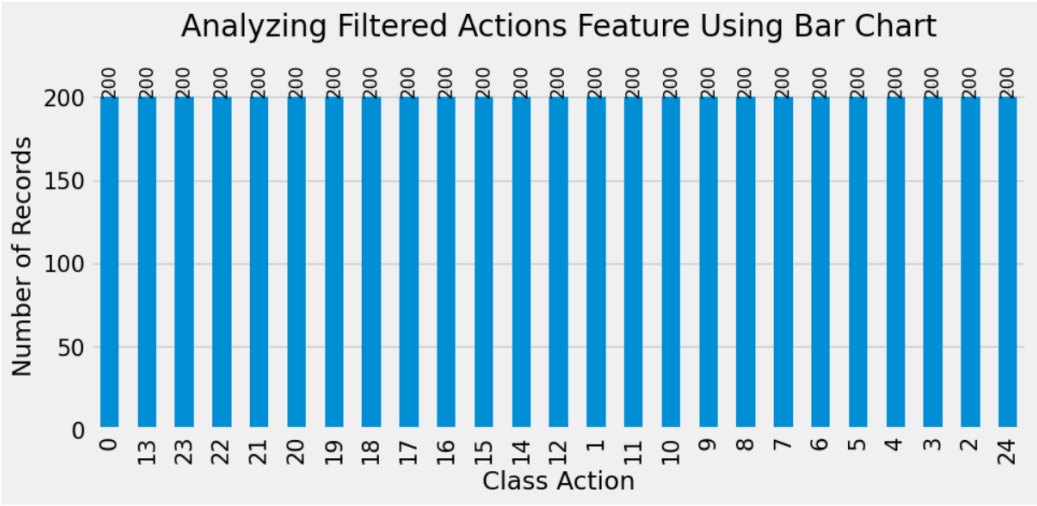

Figure 8: Image count after preprocessing and class balancing

The pre-processed images are saved in an organized directory structure with folders labelled by their respective malware families as shown in figure 9. This helps with easy access and identification during feature extraction and training.

Target-Balanced Data Extraction

```
from imblearn.over_sampling import SMOTE
import pandas as pd

X = features_df.drop('Target', axis=1)  # Feature columns
y = features_df['Target']  # Target column

# Initialize SMOTE
smote = SMOTE(sampling_strategy='auto', k_neighbors=5, random_state=42)

# Apply SMOTE to balance the dataset
X_resampled, y_resampled = smote.fit_resample(X, y)

# Create a new DataFrame with the balanced dataset
balanced_df = pd.DataFrame(X_resampled, columns=X.columns)
balanced_df['Target'] = y_resampled

balanced_df['Target'].value_counts()
```

Figure 9: Code for target balanced data extraction

4 Feature Extraction

4.1 Images preprocessing

Images are pre-processed for the CNN input according to the standards as mentioned in data preprocessing in Section 3 (Data Preparation).

4.2 Model compilation

A CNN model is compiled, and the final classification layer of the model is removed to make sure the output has meaningful vectors instead of class probabilities. Figure 10 shows the removal of the final layer and all the other layers.

CNN Model for Classification

```
[ ] cnn_model=Sequential()

    cnn_model.add(Conv2D(filters=32,kernel_size=3,strides=(1,1),padding="same",activation="relu",input_shape = (256,256,3)))
    cnn_model.add(MaxPool2D(pool_size=(3,3),padding="same"))

    cnn_model.add(Conv2D(filters=64,kernel_size=3,strides=(1,1),padding="same",activation="relu"))
    cnn_model.add(MaxPool2D(pool_size=(3,3),padding="same"))

    cnn_model.add(Flatten())
    cnn_model.add(Dropout(rate=0.4))
    cnn_model.add(Dense(units=512,activation="relu"))
    cnn_model.add(Dense(units=len(class_labels), activation="softmax"))
```

```
[ ] cnn_model.compile(optimizer=Adam(learning_rate=1e-3, decay=1e-4), loss="categorical_crossentropy", metrics=["accuracy"])
```

```
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2D  (None, 86, 86, 32)       0
 )
 conv2d_1 (Conv2D)           (None, 86, 86, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 29, 29, 64)       0
 2D)

 flatten (Flatten)           (None, 53824)             0

 dropout (Dropout)           (None, 53824)             0

 dense (Dense)               (None, 512)               27558400

 dense_1 (Dense)             (None, 25)                12825

=================================================================
Total params: 27,590,617
Trainable params: 27,590,617
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] # Define feature extraction model
    # from tensorflow.keras.models import Model

    feature_extractor = Model(inputs=cnn_model.input, outputs=cnn_model.layers[-2].output)
```

```
[ ] # Extract features from training data
    train_features = feature_extractor.predict(X_train, batch_size=32, verbose=1)
```

```
124/124 [==============================] - 122s 975ms/step
```

Figure 10: CNN architecture with the feature extraction layer highlighted

4.3 Features extraction

The pre-processed images are passed through the CNN to extract high-dimensional feature vectors. The extracted features are saved in CSV format (CNN_features.csv), shown in Figure 11.

```
[ ] # Display shape of extracted features
    print("Extracted feature shape:", train_features.shape)

    Extracted feature shape: (3966, 512)

[ ] # Create column names for the features
    column_names = [f"feature_{i+1}" for i in range(train_features.shape[1])]

[ ] # Convert features to DataFrame for easier handling
    features_df = pd.DataFrame(train_features, columns=column_names)
    features_df['Target'] = Y_train  # Adding labels as a new column

    features_df.to_csv('Dataset/CNN_Features.csv', index=False)

[ ] cnn_model.save("models/CNN_FeatureExtraction_Model.h5")
```

Figure 11: Saved feature vector CSV structure

5 Model Training

5.1 Support Vector Classifier (SVC)

The extracted feature vectors are stored in Dataset/balanced_dataset.csv as shown Figure 12. These features represent spatial patterns of malware families.

Import Libraries

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split

import seaborn as sns
import pickle
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from lib_file import lib_path
```

Import CNN Feature Extracted Dataset"

```
df = pd.read_csv('Dataset/balanced_dataset.csv')
```

```
df.head()
```

Figure 12: Common libraries and feature extracted data

The dataset is split into training and testing sets using train_test_split function as shown in figure 13. The SVC model is trained using default hyperparameters. Then the trained SVC model is saved for future inference using the pickle library.

Data Splitting

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(4000, 512) (1000, 512) (4000, 1) (1000, 1)
```

SupportVectorClassifierAlgorithm

Training SVM Model

```
from sklearn.svm import SVC

svc_model = SVC()
svc_model = svc_model.fit(X_train.values, y_train.values.ravel())
```

Making Predictions with SVM Model

```
svc_prediction = svc_model.predict(X_test.values)
print(svc_prediction.tolist())
```

Define Class Labels

```
class_labels = ['Adialer.C', 'Agent.FYI', 'Allaple.A', 'Allaple.L', 'Alueron.gen!J', 'Autorun.K', 'C2LOP.gen!g',
                'C2LOP.P', 'Dialplatform.B', 'Dontovo.A', 'Fakerean', 'Instantaccess', 'Lolyda.AA1',
                'Lolyda.AA2', 'Lolyda.AA3', 'Lolyda.AT', 'Malex.gen!J', 'Obfuscator.AD', 'Rbot!gen', 'Skintrim.N',
                'Swizzor.gen!E', 'Swizzor.gen!I', 'VB.AT', 'Wintrim.BX', 'Yuner.A']
```

Evaluating SVM Model Accuracy

```
svc_model_accuracy = accuracy_score(y_true=true_labels, y_pred=svc_prediction)
print("Validation accuracy of model is {:.2f}%".format(svc_model_accuracy*100))
```

```
Validation accuracy of model is 94.30%
```

Saving SVM Model

```
with open(file="models/SupportVectorClassifier_model.pkl", mode='wb') as file:
    pickle.dump(obj=svc_model, file=file)
```

Figure 13: Code for SVC Model Training

5.2 Random Forest Classifier (RF)

The features then are loaded from CNN_features.csv. The dataset was split into training and texting sets to validate the model's performance. Then the RF was trained with 500 estimators as shown in Figure 14. Save the trained model as RandomForestClassifier_model.pkl.

RandomForestClassifierAlgorithm

Training RF Model

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=500)
rf_model.fit(X_train.values, y_train.values.ravel())
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=500)
```

Making Predictions with RF Model

```
rf_prediction = rf_model.predict(X_test.values)
print(rf_prediction.tolist())
```

Evaluating RF Model Accuracy

```
rf_model_accuracy = accuracy_score(y_true=true_labels, y_pred=rf_prediction)
print("Validation accuracy of model is {:.2f}%".format(rf_model_accuracy*100))
```

```
Validation accuracy of model is 97.00%
```

Saving RF Model

```
with open(file="models/RandomForestClassifier_model.pkl", mode='wb') as file:
    pickle.dump(obj=rf_model, file=file)
```

Figure 14: Code for RF Model Training

5.3 Ensemble Method

The ensemble method combines predictions from the SVC and RF models using an averaging approach. This method uses the strengths of both classifiers to improve the overall accuracy and robustness of malware classification and malware detection.

First the model is loaded, and the images are pre-processed as shown in Figure 15, each image is then pre-processed and then the features are extracted as shown in Figure 16.

```
model_path = "models/CNN_FeatureExtraction_Model.h5"
cnn_model = load_model(model_path)
```

```
# Create a feature extraction model (excluding the final classification layer)
feature_extractor = Model(inputs=cnn_model.input, outputs=cnn_model.layers[-2].output)
```

```
# Function to preprocess images
def preprocess_image(image_path, target_size=(256, 256)):
    image = load_img(image_path, target_size=target_size)
    image_array = img_to_array(image)
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension
    image_array /= 255.0  # Normalize
    return image_array
```

Figure 15: Code for loading CNN feature extraction

```
image_path = "user_input/Dontovo.A (3).png"
```

```
# Preprocess and extract features from the image
image = preprocess_image(image_path)
features = feature_extractor.predict(image)
# print("features: ", features)
```

Figure 16: Code for input image

Then the features extracted are stored in the CNN_features.csv file as shown in Figure 17. This file will later be used for the following models.

```
# Convert extracted features to DataFrame
features_df = pd.DataFrame(features, columns=[f"feature_{i+1}" for i in range(features.shape[1])])
features_df.head()
```

```
output_csv_path = "cnn_output/CNN_features.csv"
features_df.to_csv(output_csv_path, index=False)

print(f"Features extracted and saved to {output_csv_path}")
```

Figure 17: Output of CNN feature extraction

Then the SVC and RF models previouly trained are loaded and the classed are defined as shown in Figure 18. The output file produced by the CNN is now used as input for the SVC and RF models as shown in Figure 19.

```
with open('models/SupportVectorClassifier_model.pkl', 'rb') as model_file:
    svc_model = pickle.load(model_file)

with open('models/RandomForestClassifier_model.pkl', 'rb') as model_file:
    rf_model = pickle.load(model_file)
```

Define Class Labels

```
class_labels = ['Adialer.C', 'Agent.FYI', 'Allaple.A', 'Allaple.L', 'Alueron.gen!J', 'Autorun.K', 'C2LOP.gen!g', 'C2LOP.P',
                'Dialplatform.B', 'Dontovo.A', 'Fakerean', 'Instantaccess', 'Lolyda.AA1', 'Lolyda.AA2', 'Lolyda.AA3',
                'Lolyda.AT', 'Malex.gen!J', 'Obfuscator.AD', 'Rbot!gen', 'Skintrim.N', 'Swizzor.gen!E', 'Swizzor.gen!I',
                'VB.AT', 'Wintrim.BX', 'Yuner.A']
```

Figure 18: Code for loading model

```
input_file = 'cnn_output/CNN_features.csv'
input_data = pd.read_csv(input_file)
```

SVC Model Prediction and Output

```
svc_prediction = svc_model.decision_function(input_data)
svc_probabilities = np.exp(svc_prediction) / np.sum(np.exp(svc_prediction), axis=1)
print("svc_probabilities : ", svc_probabilities)
```

```
svc_probabilities :  [[3.01712027e-07 1.50310471e-03 7.23604136e-10 1.09793426e-07
  3.29904598e-02 1.32115304e-11 1.98205966e-09 8.04001496e-05
  6.26629654e-06 6.63263919e-01 4.01459283e-08 2.67687139e-10
  1.21011407e-02 1.20803073e-02 2.37141933e-06 2.44124187e-01
  5.41622045e-09 9.83731820e-11 5.89976234e-04 4.03662119e-08
  6.06900966e-06 2.19676597e-04 3.30308002e-02 8.21015940e-07
  3.59120709e-11]]
```

RF Model Prediction and Output

```
rf_predictions = rf_model.predict_proba(input_data)
print("rf_predictions : ", rf_predictions)
```

```
rf_predictions :  [[0.   0.   0.   0.   0.03 0.   0.   0.   0.   0.89 0.   0.   0.02 0.0
1
  0.   0.01 0.   0.   0.02 0.   0.   0.   0.02 0.   0.   ]]
```

Figure 19: Code for Both model Prediction

The combined predictions from SVC and RF using an averaging ensemble approach are shown in figure 20 and the class with the ensemble score shown in figure 21.

```
# Ensemble method
ensemble_scores = (svc_probabilities + rf_predictions) / 2
print("ensemble_scores : ", ensemble_scores)
```

```
ensemble_scores :  [[1.50856014e-07 7.51552356e-04 3.61802068e-10 5.48967132e-08
  3.14952299e-02 6.60576521e-12 9.91029831e-10 4.02000748e-05
  3.13314827e-06 7.76631960e-01 2.00729642e-08 1.33843569e-10
  1.60505704e-02 1.10401537e-02 1.18570966e-06 1.27062093e-01
  2.70811023e-09 4.91865910e-11 1.02949881e-02 2.01831060e-08
  3.03450483e-06 1.09838298e-04 2.65154001e-02 4.10507970e-07
  1.79560355e-11]]
```

Figure 20: Code for ensemble model

```
ensemble_predicted_class_index = np.argmax(ensemble_scores[0])
ensemble_predicted_label = class_labels[ensemble_predicted_class_index]
ensemble_confidence_score = ensemble_scores[0][ensemble_predicted_class_index]

print(f"Ensemble Model predicted class index is -> {ensemble_predicted_class_index}")
print(f"Ensemble Model predicted label is -> {ensemble_predicted_label}")
print(f"Ensemble Model confidence score is -> {ensemble_confidence_score*100:.2f}%")
```

```
Ensemble Model predicted class index is -> 9
Ensemble Model predicted label is -> Dontovo.A
Ensemble Model confidence score is -> 77.66%
```

Figure 21: Code for ensemble prediction and output

References

1. Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016) 'TensorFlow: A system for large-scale machine learning'. doi.org/10.48550/arXiv.1605.08695

2. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020) 'Array programming with NumPy', 357–362. doi.org/10.1038/s41586-020-2649-2

3. Mckinney, W. (2011) 'Pandas: a Foundational python library for data analysis and statistics'. Python High Performance Science Computer.

4. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2018) 'Scikit learn: machine learning in python'. doi.org/10.48550/arXiv.1201.0490