CSE3502: Information Security Management

J-Component Review 3 Final Report

Prof. Dr. Anil Kumar K.

Slot - F2

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Title: Malware Classification using Convolutional Neural Networks

Team Members

19BCT0221- PARTH MAHESHWARI
19BCT0204 – PRAMIT GUPTA
19BCT0217 – AGRIM NAGRANI

Problem Statement

"We worried for decades about WMDs – Weapons of Mass Destruction. Now it is time to worry about a new kind of WMDs – Weapons of Mass Disruption."

— John Mariotti

There exist countless types of malwares attempting to damage companies' valuable information systems. It can also cause substantial damage to personal data servers and other devices. Thus, it is essential for us to detect and prevent them to avoid risk. Malware is a superset term for several malicious software variants which includes ransomwares, spywares, and viruses. Malware is a fancy word for malicious software. Malware classification is a widely used task that, as you probably know, can be accomplished by machine learning models quite effectively. Nowadays, there are countless types of malware attempting to damage companies' information systems. Thus, it is essential to detect and prevent them to avoid any risk. Malware classification is a widely used task that, as you probably know, can be accomplished by machine learning models quite effectively. Our goal in this project is to classify Malware using Convolution Neural Networks. Thus, analysing malware and classifying them will help us to understand them better which in turn will help us to act on them in a proper and efficient manner.

Introduction

Types of malwares and their function:

1. **Dialers:** Dialers, as the name implies, dial to predefined numbers to connect to certain Websites. Many users run dialers without knowing that some of these programs actually dial long distance numbers or connect to pay-per-call sites. As a result, the user incurs unexpected telephone charges and/or accesses unintended and unwanted Web site contents.
2. **Backdoor:** A backdoor is a malware type that negates normal authentication procedures to access a system. As a result, remote access is granted to resources within an application, such as databases and file servers, giving perpetrators the ability to remotely issue system commands and update malware.
3. **Worm:** A computer worm is a type of malware that spreads copies of itself from computer to computer. A worm can replicate itself without any human interaction, and it does not need to attach itself to a software program in order to cause damage.
4. **Trojan:** A Trojan horse or Trojan is a type of malware that is often disguised as legitimate software. Trojans can be employed by cyber-thieves and hackers trying to gain access to users' systems. Users are typically tricked by some form of social engineering into loading and executing Trojans on their systems.
5. **Rogue:** Rogue security software is a form of malicious software and internet fraud that misleads users into believing there is a virus on their computer and aims to convince them to pay for a fake malware removal tool that installs malware on their computer.
6. **PWS:** Password stealers, or PWS, is the specific malware type that attempts to get your passwords and other credentials. Usually, the majority of password stealer injections are done through email spamming.

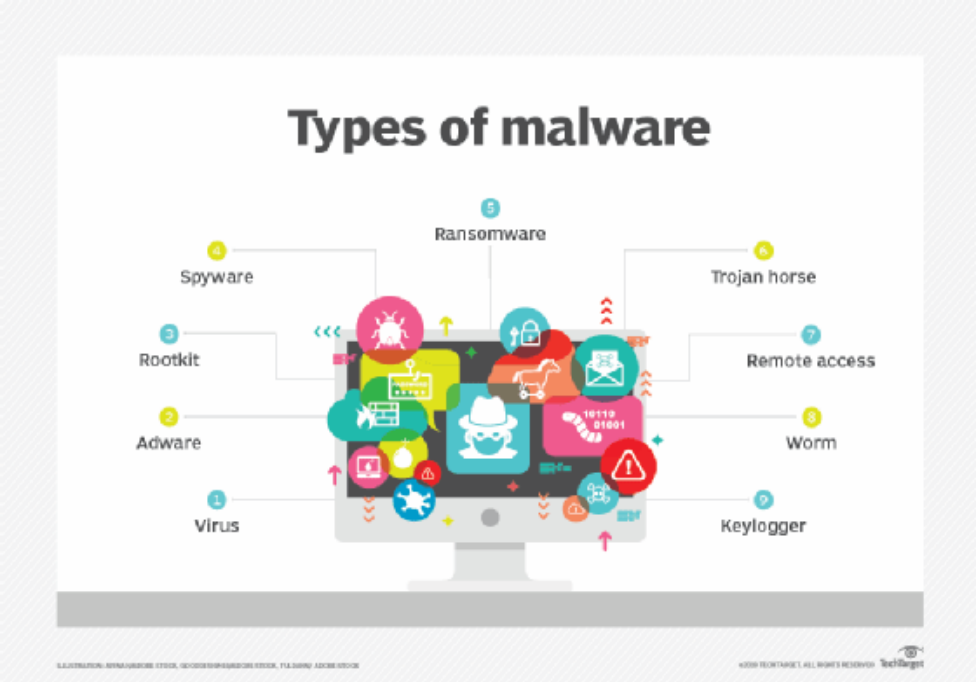


Fig: Types of Malwares.

What are Neural Networks?

In simple terms a 'neural network' is a series of algorithms that endeavours to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural nets take inspiration from the learning process occurring in human brains. They consist of an artificial network of functions, called parameters, which allows the computer to learn, and to fine tune itself, by analysing new data. Each parameter, sometimes also referred to as neurons, is a function which produces an output, after receiving one or multiple inputs. Those outputs are then passed to the next layer of neurons, which use them as inputs of their own function, and produce further outputs. Those outputs are then passed on to the next layer of neurons, and so it continues until every layer of neurons have been considered, and the terminal neurons have received their input. Those terminal neurons then output the result for the model.

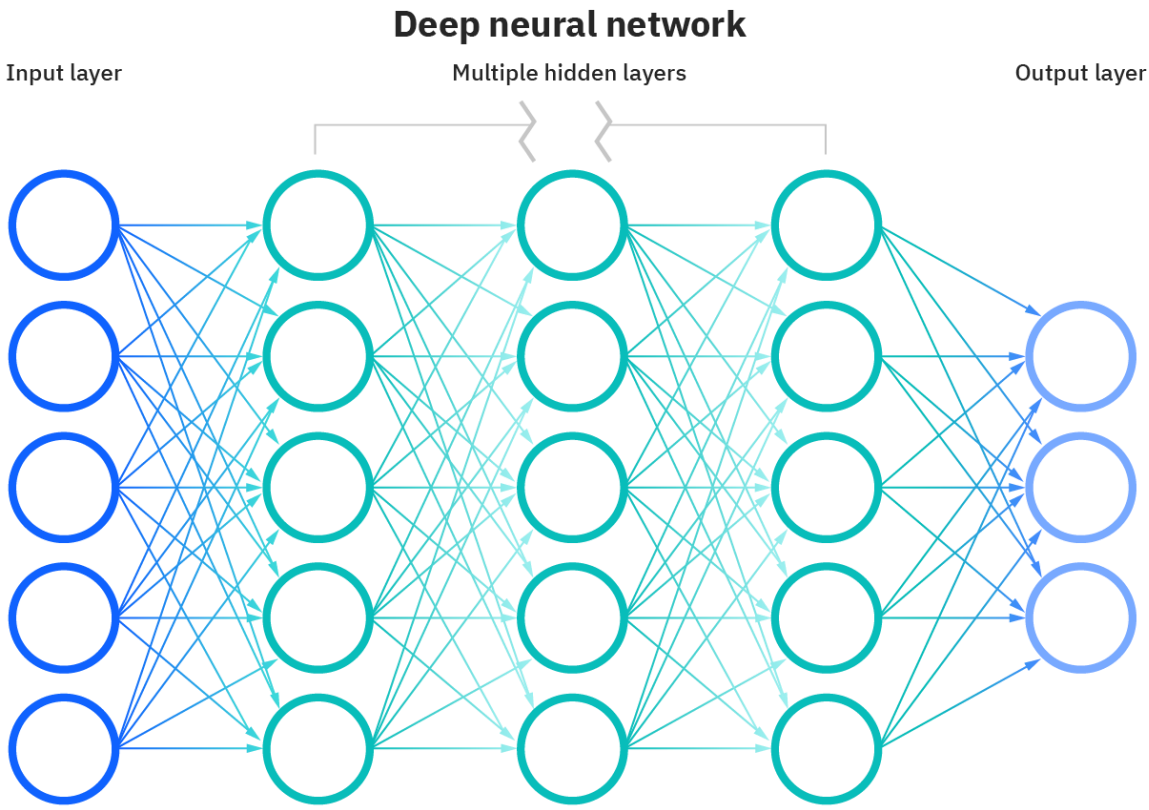


Fig: Neural Networks in Deep Learning.

What are Some Applications of Neural Networks?

There are many applications for machine learning methods such as neural nets. Most of these applications focus on classification of images. Those images could be of anything, from whether or not something is a hot dog, to identifying handwriting. The practical possibilities of such a model are broad, and lucrative. Let's have a look at an example.

Many companies would love to be able to automate a model which would classify articles of clothing. Doing so would allow them to draw insight into fashion trends, buying habits, and differences between cultural, and socio-economic groups. To that end we will use a neural network, to see if an adequate classification model can be constructed, when given a set of 60,000 images, with labels identifying what type of clothing they were.

All those pictures are made up of pixels, which since we will be doing a simple neural network, and not a convolution neural net, will be passed directly into the network as a vector of pixels.

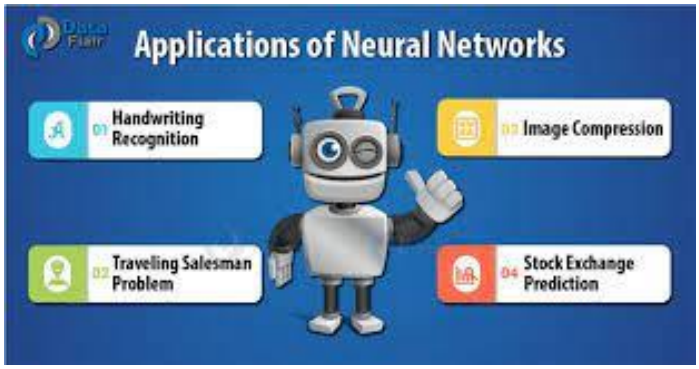


Fig: Applications of Neural Networks

What is a Convolution Neural Network?

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

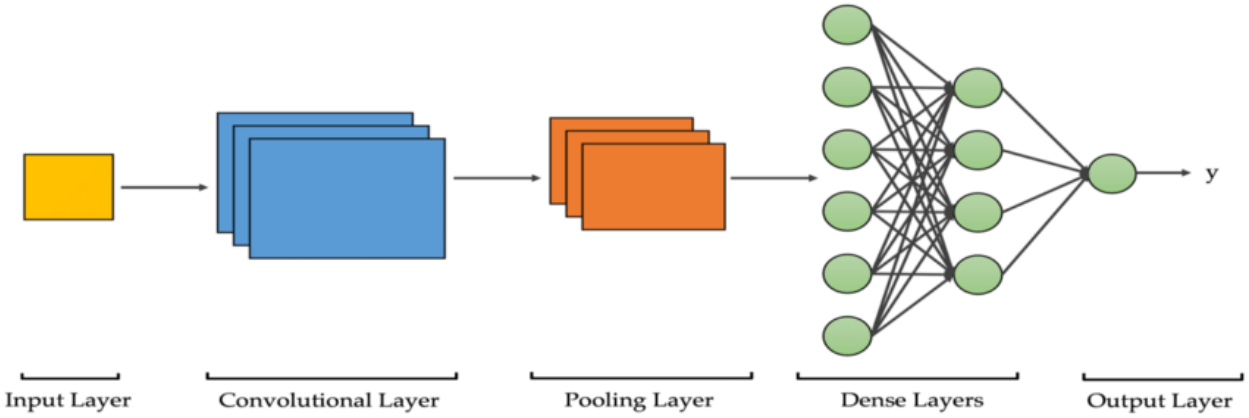


Fig: Overview of CNN.

Literature Review

1. Malware Classification with Deep Convolutional Neural Networks

Mahmoud Kalash†, Mrigank Rochan*†, Noman Mohammed†, Neil D. B. Bruce† , Yang Wang†, Farkhund Iqbal*

2nd April, 2018 - IEEE Xplore

In this paper, they propose a deep learning framework for malware classification. There has been a huge increase in the volume of malware in recent years which poses a serious security threat to financial institutions, businesses and individuals. In order to combat the proliferation of malware, new strategies are essential to quickly identify and classify malware samples so that their behavior can be analyzed.

2. Behavioral Malware Classification using Convolutional Recurrent Neural Networks

Bander Alsulami, Spiros Mancoridis

19th Nov, 2018 - IEEE Xplore

This paper introduces a new behavioral malware classification model for the Microsoft Windows platform. The model extracts features from the Windows Prefetch files. We show the effectiveness of our classification technique on a large malware collection and ground truth labels from 4 majo anti-virus vendors. They also evaluate their models on rare malware families with a small number of malware samples. Despite the increasing number of malware families, the model still outperforms other state-of-the-art models. Moreover, they demonstrate their model's ability to continuously learn the behavior of new malware families, which reduces the time and overhead of the training process.

3. HYDRA: A multimodal deep learning framework for malware classification

Daniel Gibert ∗, Carles Mateu , Jordi Planes

8th May, 2020

HYDRA, a novel framework to address the task of malware detection and classification by combining various types of features to discover the relationships between distinct modalities. An extensive analysis of state-of-the-art methods on the Microsoft Malware Classification Challenge benchmark shows that the proposed solution achieves comparable results to gradient boosting methods in the literature and higher yield in comparison with deep learning approaches.

4. Malware Classification Using Long Short-Term Memory Models

Dennis Dang, Fabio Di Troia, Mark Stamp

5th March, 2021

In this research, we found that malware classification by by family using long-short term memory (LSTM) models is feasible. However, using just a single LSTM layer alone yields poor results. We found that by incorporating techniques from natural language processing (NLP), specifically, word embedding and bidirectional LSTMs (biLSTM), greatly improves the performance. We also discovered that that we could get obtain even better performance by including a convolutional neural network (CNN) layer in our model. Our best model was able to classify samples from 20 different malware families with an average accuracy in excess of 81%. We conjecture that the interplay between the long-term memory of the biLSTM and the local structure found by the CNN are the key to obtaining this strong performance.

5. Malware Classification with GMM-HMM Models

Jing Zhao, Samanvitha Basole†, Mark Stamp‡*

5th March, 2021

Discrete hidden Markov models (HMM) are often applied to malware detection and classification problems. However, the continuous analog of discrete HMMs, that is, Gaussian mixture model-HMMs (GMM-HMM), are rarely considered in the field of cybersecurity. In this paper, we use GMM-HMMs for malware classification and we compare our results to those obtained using discrete HMMs. As features, we consider opcode sequences and entropy-based sequences. For our opcode features, GMM-HMMs produce results that are comparable to those obtained using discrete HMMs, whereas for our entropy-based features, GMM-HMMs generally improve significantly on the classification results that we have achieved with discrete HMMs.

Methodology

Dataset

We will mainly use the Malimg Dataset. The Malimg Dataset contains 9339 malware images, belonging to 25 families/classes. Thus, our goal is to perform a multi-class classification of malware.

Classes in the Dataset:

	Family/Class	Type
0	Adialer.C	Dialer
1	Agent.FYI	Backdoor
2	Allaple.A	Worm
3	Allaple.L	Worm
4	Alueron.gen!J	Worm
5	Autorun.K	Worm:AutoIT
6	C2LOP.P	Trojan
7	C2LOP.gen!g	Trojan
8	Dialplatform.B	Dialer
9	Dontovo.A	Trojan Downloader
10	Fakerean	Rogue
11	Instantaccess	Dialer
12	Lolyda.AA1	PWS
13	Lolyda.AA2	PWS
14	Lolyda.AA3	PWS
15	Lolyda.AT	PWS
16	Malex.gen!J	Trojan
17	Obfuscator.AD	Trojan Downloader
18	Rbot!gen	Backdoor
19	Skintrim.N	Trojan
20	Swizzor.gen!E	Trojan Downloader
21	Swizzor.gen!I	Trojan Downloader
22	VB.AT	Worm
23	Wintrim.BX	Trojan Downloader
24	Yuner.A	Worm

Pre-processing

Our dataset has the malware already in the form of images. In case it was still in binary format, we would have to pre-process the binary file to an image format.

We have to then resize the images to a size that is uniform and to a size we choose [64*64].

After we choose the size, we split the data into training data and test data in the ratio 70:30. Training data is what we train the model on, and testing data is how we test the accuracy of the trained model.

Building the CNN Model- Layers-Based Approach.

Now, that our dataset is ready, we can build our model using Keras. The following structure will be used:

- Convolutional Layer: 30 filters, (3 * 3) kernel size
- Max Pooling Layer: (2 * 2) pool size
- Convolutional Layer: 15 filters, (3 * 3) kernel size
- Max Pooling Layer: (2 * 2) pool size
- DropOut Layer: Dropping 25% of neurons.
- Flatten Layer
- Dense/Fully Connected Layer: 128 neurons, Relu activation function ● DropOut Layer : Dropping 50% of neurons.
- Dense/Fully Connected Layer: 50 neurons, Softmax activation function
- Dense/Fully Connected Layer: num_class neurons, Softmax activation function

The input has a shape of [64 * 64 * 3]: [width * height * depth]. In our case, each Malware is a RGB image.

CNN Model - Handling Unbalanced Data

Several methods are available to deal with unbalanced data. In this project, we chose to give higher weight to the minority class and lower weight to the majority class.

sklearn.utilis.class_weights function uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data. To use this method, yttrian must not be one-hot encoded (single high bit and rest low, ie, one 1 and rest 0's).

Implementation

Link: https://colab.research.google.com/drive/1KzQ7TydKJqlfeeg9Qcr3Uf9b36UwRmM-

Code Snippets (Full Code in Appendix)

```
import sys
import os
from math import log
import numpy as np
import scipy as sp
from PIL import Image
import matplotlib.pyplot as plt
```

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
[ ] !unzip gdrive/MyDrive/malimg_dataset.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/06524ebf396548004410f99a4dde2e54.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/06524f125eece7a543706092871889 80.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/0652f1f4bad2c5c928080bc90db86ed1.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/0654748b4cc3330deef95ab0af4041cf.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/06549b73968ac1fc1e8eb54dddb10833.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/065513aa17b61f57cf793703a725c015.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/06583d83c5278fac7cd20c015780c1d0.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/0658876855197e86e417ba8a84d298ec.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/06598930132128de200841974164a858.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/065a7e3f38d8cc3ca056b4f5fc272643.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/065d1f12c517c06d5a510c100c9b6ed6.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/065efb1c64dfce7a4a584481d86373b3.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/065f39de0853f6348fbaf752663eb146.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/066147d74ecc046916c3fb410f34d9dc.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/06625fad5a72ea3c8319fc357915cfd4.png
  inflating: malimg_dataset/malimg_paper_dataset_imgs/Allaple.L/06642c147f5f48b6b995a0c3d1808a78.png
```

```
[ ] from keras.preprocessing.image import ImageDataGenerator
    from sklearn.model_selection import train_test_split
    #Generating DataSet
    path_root = "./malimg_dataset/malimg_paper_dataset_imgs/"
    batches = ImageDataGenerator().flow_from_directory(directory=path_root, target_size=(64,64), batch_size=10000)
```

```
Found 9339 images belonging to 26 classes.
```

```
[ ] batches.class_indices
```

```
{'Adialer.C': 0,
 'Agent.FYI': 1,
 'Allaple.A': 2,
 'Allaple.L': 3,
 'Alueron.gen!J': 4,
 'Autorun.K': 5,
 'C2LOP.P': 6,
 'C2LOP.gen!g': 7,
 'Dialplatform.B': 8,
 'Dontovo.A': 9,
 'Fakerean': 10,
 'Instantaccess': 11,
 'Lolyda.AA1': 12,
 'Lolyda.AA2': 13,
 'Lolyda.AA3': 14,
 'Lolyda.AT': 15,
 'Malex.gen!J': 16,
 'Obfuscator.AD': 17,
 'Rbot!gen': 18,
```

```
[ ] imgs, labels = next(batches)
```

```
[ ] imgs.shape
```

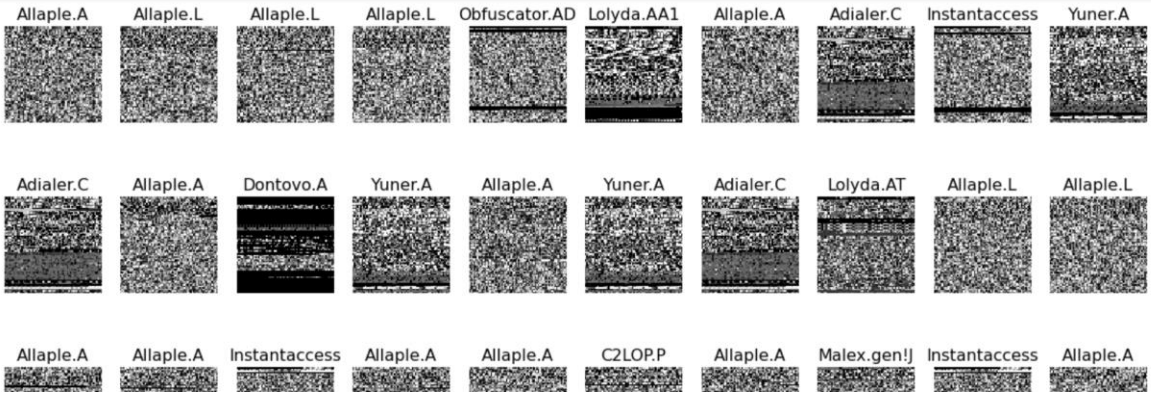
```
(9339, 64, 64, 3)
```

```
[ ] labels.shape
```

```
(9339, 26)
```

```
[ ] # plotting images with labels
    def plots(ims, figsize=(20,30), rows=10, interp=False, titles=None):
      if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
          ims = ims.transpose((0,2,3,1))
      f = plt.figure(figsize=figsize)
      cols = 10
      for i in range(0,50):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
          sp.set_title(list(batches.class_indices.keys())[np.argmax(titles[i])], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')
```

```
[ ] plots(imgs, titles = labels)
```

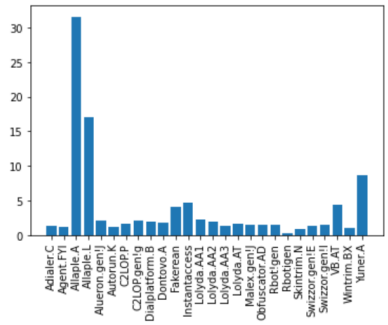



```
[ ] classes = batches.class_indices.keys()
```

```
[ ] perc = (sum(labels)/labels.shape[0])*100
```

```
[ ] plt.xticks(rotation='vertical')
    plt.bar(classes,perc)
```

```
<BarContainer object of 26 artists>
```




```
[ ] from sklearn.model_selection import train_test_split
    #Split into train and test
    X_train, X_test, y_train, y_test = train_test_split(imgs/255.,labels, test_size=0.3)
```

```
[ ] X_train.shape
```

```
(6537, 64, 64, 3)
```

```
[ ] X_test.shape
```

```
(2802, 64, 64, 3)
```

```
[ ] y_train.shape
```

```
(6537, 26)
```

```
[ ] y_test.shape
```

```
(2802, 26)
```

```
[ ] import keras
    from keras.models import Sequential, Input, Model
    from keras.layers import Dense, Dropout, Flatten
    from keras.layers import Conv2D, MaxPooling2D
    from tensorflow.keras.layers import BatchNormalization
```

```
[ ] num_classes = 26
```

```
[ ] def malware_model():
      Malware_model = Sequential()
      Malware_model.add(Conv2D(30, kernel_size=(3, 3),
      activation='relu',
      input_shape=(64,64,3)))
      Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
      Malware_model.add(Conv2D(15, (3, 3), activation='relu'))
      Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
      Malware_model.add(Dense(num_classes, activation='softmax'))
      Malware_model.compile(loss='categorical_crossentropy', optimizer ='adam', metrics=['accuracy'])
      return Malware_model
```

```
[ ] Malware_model = malware_model()
```

```
[ ] Malware_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 30)        840

 max_pooling2d (MaxPooling2D  (None, 31, 31, 30)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 15)        4065

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 15)       0

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 50)                6450

 dense_2 (Dense)             (None, 26)                1326

=================================================================
Total params: 389,129
Trainable params: 389,129
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] y_train
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
```

```
[ ] y_train_new
```

```
array([21,  2,  2, ...,  2,  3,  5])
```

```
[ ] from sklearn.utils import class_weight
    class_weights = class_weight.compute_class_weight(class_weight = 'balanced', classes = np.unique(y_train_new), y = y_train_new)
    class_weight_dict = dict(enumerate(class_weights))
    class_weight_dict
```

```
{0: 3.1039886039886038,
 1: 3.1427884615384616,
 2: 0.12367096749782436,
 3: 0.21843881574550558,
 4: 1.8903990746096009,
 5: 3.0661350844277675,
 6: 2.591990483743061,
 7: 1.9047202797202798,
 8: 1.9192601291837934,
 9: 2.2054655870445345,
 10: 0.8915711947626841
```

```
[ ] Malware_model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 20, class_weight = class_weight_dict)
```

```
Epoch 1/20
205/205 [==============================] - 30s 142ms/step - loss: 2.1652 - accuracy: 0.3145 - val_loss: 1.1056 - val_accuracy: 0.5064
Epoch 2/20
205/205 [==============================] - 30s 146ms/step - loss: 0.9600 - accuracy: 0.5151 - val_loss: 0.9462 - val_accuracy: 0.5371
Epoch 3/20
205/205 [==============================] - 29s 141ms/step - loss: 0.7591 - accuracy: 0.5686 - val_loss: 0.6409 - val_accuracy: 0.7780
Epoch 4/20
205/205 [==============================] - 31s 153ms/step - loss: 0.6255 - accuracy: 0.6062 - val_loss: 0.7003 - val_accuracy: 0.6677
Epoch 5/20
205/205 [==============================] - 29s 139ms/step - loss: 0.5317 - accuracy: 0.6485 - val_loss: 0.6596 - val_accuracy: 0.7045
Epoch 6/20
205/205 [==============================] - 29s 140ms/step - loss: 0.4908 - accuracy: 0.6720 - val_loss: 0.5782 - val_accuracy: 0.6763
Epoch 7/20
205/205 [==============================] - 31s 152ms/step - loss: 0.4345 - accuracy: 0.7190 - val_loss: 0.6072 - val_accuracy: 0.6767
Epoch 8/20
205/205 [==============================] - 29s 140ms/step - loss: 0.3981 - accuracy: 0.7375 - val_loss: 0.5455 - val_accuracy: 0.7181
Epoch 9/20
205/205 [==============================] - 29s 139ms/step - loss: 0.3593 - accuracy: 0.7597 - val_loss: 0.4616 - val_accuracy: 0.7891
Epoch 10/20
```

```
[ ] scores = Malware_model.evaluate(X_test, y_test)
```

```
88/88 [==============================] - 4s 40ms/step - loss: 0.2616 - accuracy: 0.8715
```

```
[ ] print('Final CNN accuracy: ', scores[1])
```

```
Final CNN accuracy:  0.8715203404426575
```

```
[ ] y_pred = Malware_model.predict(X_test)
    classes_y=np.argmax(y_pred,axis=1)
    # y_pred = Malware_model.predict_classes(X_test, verbose=0)
```

```
[ ] classes_y
```

```
array([14, 17,  0, ...,  5,  6,  5])
```

```
[ ] y_test2 = np.argmax(y_test, axis=1)
```

```
[ ] y_test2
```

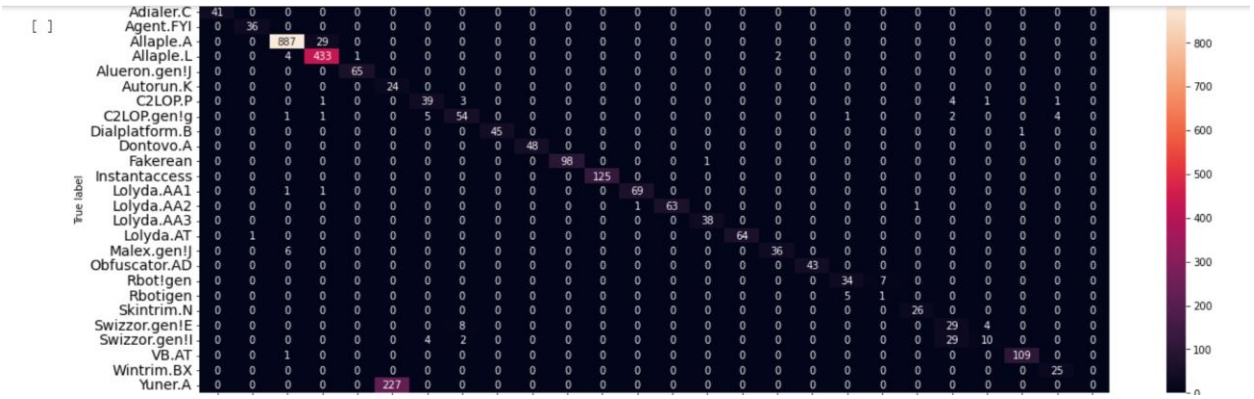
```
array([14, 17,  0, ..., 25, 22, 25])
```

```
[ ] from sklearn import metrics
    c_matrix = metrics.confusion_matrix(y_test2, classes_y)
```

```
[ ] import seaborn as sns
    def confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=14):
      """Prints a confusion matrix, as returned by
      sklearn.metrics.confusion_matrix, as a heatmap.
      Arguments
      ---------
      confusion_matrix: numpy.ndarray
      The numpy.ndarray object returned from a call to
      sklearn.metrics.confusion_matrix.
      Similarly constructed ndarrays can also be used.
      class_names: list
```

```
[ ]   df_cm = pd.DataFrame(
      confusion_matrix, index=class_names, columns=class_names,
      )
      fig = plt.figure(figsize=figsize)
      try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
      except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
      heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(),
      rotation=0, ha='right', fontsize=fontsize)
      heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(),
      rotation=45, ha='right', fontsize=fontsize)
      plt.ylabel('True label')
      plt.xlabel('Predicted label')
```

```
[ ] class_names= batches.class_indices.keys()
    import pandas as pd
    confusion_matrix(c_matrix, class_names, figsize = (20,7), fontsize=14)
```



Results:

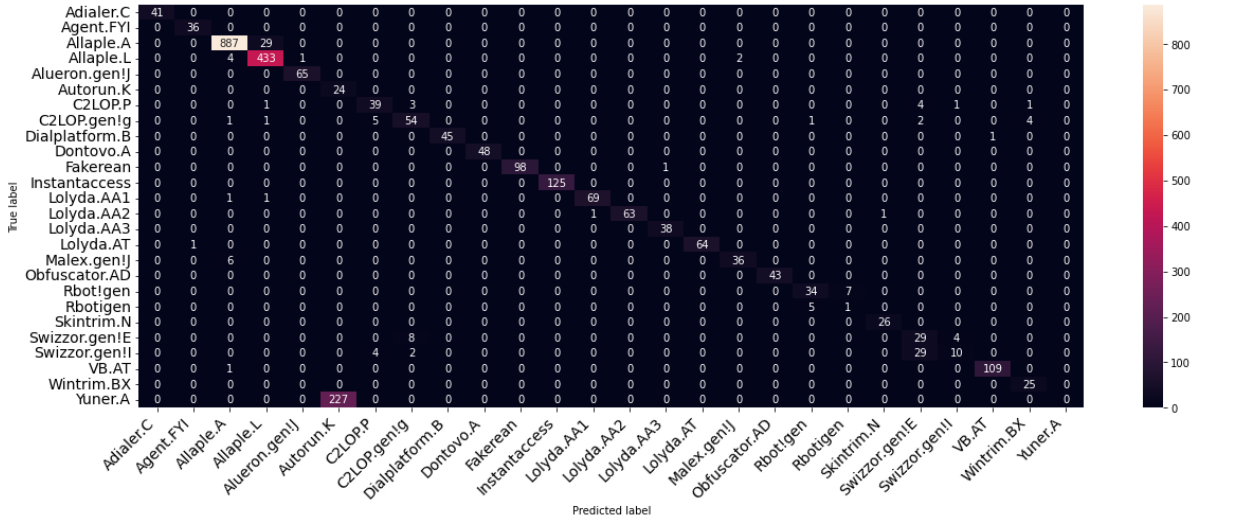
Accuracy of Model

```
[ ] print('Final CNN accuracy: ', scores[1])

    Final CNN accuracy:  0.8715203404426575
```

Accuracy of the model is 87%.

Confusion Matrix



Analysis and Inference (Explaination):

We can observe in the confusion matrix that although most of the Malwares were well classified, Autorun.K is always mistaken for Yuner.A. This is probably because we have very few samples of Autorun.K in our dataset and that both are part of a close Worm type. Moreover, Swizzor.gen!E is often

mistaken with Swizzor.gen!I, which can be explained by the fact that they come from really close kind of families and types and thus could have similarities in their code.

Conclusion

It can be observed from this project that using this multi-class classification-based CNN model, it is possible to differentiate between various classes of Malware with a relatively high level of accuracy.

Companies' Information Security Departments are likely to have access to larger Datasets which will enhance the percentage of the accuracy.

Appendix

Full Code:

```
import sys
import os
from math import log
import numpy as np
import scipy as sp
from PIL import Image
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/gdrive')
!unzip gdrive/MyDrive/malimg_dataset.zip

from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
#Generating DataSet
path_root = "./malimg_dataset/malimg_paper_dataset_imgs/"
batches = ImageDataGenerator().flow_from_directory(directory=path_root, targe
t_size=(64,64), batch_size=10000)

batches.class_indices

imgs, labels = next(batches)

imgs.shape

labels.shape

# plotting images with labels
```

```
def plots(ims, figsize=(20,30), rows=10, interp=False, titles=None):
  if type(ims[0]) is np.ndarray:
    ims = np.array(ims).astype(np.uint8)
    if (ims.shape[-1] != 3):
      ims = ims.transpose((0,2,3,1))
  f = plt.figure(figsize=figsize)
  cols = 10
  for i in range(0,50):
    sp = f.add_subplot(rows, cols, i+1)
    sp.axis('Off')
    if titles is not None:
      sp.set_title(list(batches.class_indices.keys())[np.argmax(titles[i])],
fontsize=16)
    plt.imshow(ims[i], interpolation=None if interp else 'none')

classes = batches.class_indices.keys()

perc = (sum(labels)/labels.shape[0])*100

plt.xticks(rotation='vertical')
plt.bar(classes,perc)

from sklearn.model_selection import train_test_split
#Split into train and test
X_train, X_test, y_train, y_test = train_test_split(imgs/255.,labels, test_si
ze=0.3)

X_train.shape

X_test.shape

y_train.shape

y_test.shape

import keras
from keras.models import Sequential, Input, Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization

num_classes = 26

def malware_model():
```

```
  Malware_model = Sequential()
  Malware_model.add(Conv2D(30, kernel_size=(3, 3),
  activation='relu',
  input_shape=(64,64,3)))
  Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
  Malware_model.add(Conv2D(15, (3, 3), activation='relu'))
  Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
  Malware_model.add(Dropout(0.25))
  Malware_model.add(Flatten())
  Malware_model.add(Dense(128, activation='relu'))
  Malware_model.add(Dropout(0.5))
  Malware_model.add(Dense(50, activation='relu'))
  Malware_model.add(Dense(num_classes, activation='softmax'))
  Malware_model.compile(loss='categorical_crossentropy', optimizer ='adam', m
etrics=['accuracy'])
  return Malware_model

Malware_model = malware_model()

Malware_model.summary()

y_train

y_train_new = np.argmax(y_train, axis=1)

y_train_new

from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight(class_weight = 'balanced',
classes = np.unique(y_train_new), y = y_train_new)
class_weight_dict = dict(enumerate(class_weights))
class_weight_dict

Malware_model.fit(X_train, y_train, validation_data = (X_test, y_test), epoch
s = 20, class_weight = class_weight_dict)

scores = Malware_model.evaluate(X_test, y_test)
print('Final CNN accuracy: ', scores[1])

      y_pred = Malware_model.predict(X_test)
classes_y=np.argmax(y_pred,axis=1)
# y_pred = Malware_model.predict_classes(X_test, verbose=0)

classes_y
```

```
y_test2 = np.argmax(y_test, axis=1)

y_test2

from sklearn import metrics
c_matrix = metrics.confusion_matrix(y_test2, classes_y)

import seaborn as sns
def confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsiz
e=14):
  """Prints a confusion matrix, as returned by
  sklearn.metrics.confusion_matrix, as a heatmap.
  Arguments
  ---------
  confusion_matrix: numpy.ndarray
  The numpy.ndarray object returned from a call to
  sklearn.metrics.confusion_matrix.
  Similarly constructed ndarrays can also be used.
  class_names: list
  An ordered list of class names, in the order they index the given
  confusion matrix.
  figsize: tuple
  A 2-long tuple, the first value determining the horizontal size of
  the ouputted figure,
  the second determining the vertical size. Defaults to (10,7).
  fontsize: int
  Font size for axes labels. Defaults to 14.
  """
  df_cm = pd.DataFrame(
  confusion_matrix, index=class_names, columns=class_names,
  )
  fig = plt.figure(figsize=figsize)
  try:
    heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
  except ValueError:
    raise ValueError("Confusion matrix values must be integers.")
  heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(),
  rotation=0, ha='right', fontsize=fontsize)
  heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(),
  rotation=45, ha='right', fontsize=fontsize)
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
```

```
class_names= batches.class_indices.keys()
import pandas as pd
confusion_matrix(c_matrix, class_names, figsize = (20,7), fontsize=14)
```

References

1. Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D. B., Wang, Y., & Iqbal, F. (2018). Malware Classification with Deep Convolutional Neural Networks. 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). doi:10.1109/ntms.2018.8328749
2. Alsulami, B., & Mancoridis, S. (2018). Behavioral Malware Classification using Convolutional Recurrent Neural Networks. 2018 13th International Conference on Malicious and Unwanted Software (MALWARE). doi:10.1109/malware.2018.8659358
3. Gibert, D., Mateu, C., & Planes, J. (2020). HYDRA: A Multimodal Deep Learning Framework for Malware Classification. Computers & Security, 101873. doi:10.1016/j.cose.2020.101873
4. Dang, Dennis & Di Troia, Fabio & Stamp, Mark. (2021). Malware Classification Using Long Short-Term Memory Models.
5. Zhao, Jing & Basole, Samanvitha & Stamp, Mark. (2021). Malware Classification with GMM-HMM Models.
