



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Winter Semester (2022-23)

Information Security Management (CSE3502)

Malware Detection using Artificial Intelligence

J Component Review-3

Report

By

Darsh Kumar(20BCE0301)

Mir Mohammed Zain(20BDS0211)

Khyati Pareek(20BDS0299)

Srishti Sinha(20BDS0329)

Under the guidance of: Prof. Ruby D.

ABSTRACT

As we find ourselves immersed in the digital age, the specter of malware looms large over our technological landscape. With the increasing advancement and complexity of these malicious programs, the standard methods of detection have proven inefficient. As such, this project represents a bold foray into a new frontier of malware detection, leveraging the power of AI-driven techniques to combat these emerging threats. Our exploration delves into innovative methods of analysis, charting the impact of these techniques upon the ever-evolving digital landscape. By forging a path toward deep learning-based malware detection, we offer a glimpse into the future of cybersecurity. Drawing upon a veritable library of both malicious and benign files, we construct a visual lexicon of these programs, using them as the basis for our training data. By harnessing the incredible power of deep learning, our model is able to deftly distinguish between malware and non-malicious files, delivering a novel and efficient approach to detection. Foregoing the need for static or dynamic analysis, we are able to rapidly and accurately identify malware through the use of our unique methodology. This project, in its very essence, contains a major step forward in the ongoing fight against malicious software. It holds the promise of a safer, more secure digital realm, where threats are identified and neutralized with the utmost efficacy.

Keywords: Malware detection, Neural networks, Deep Learning, Signature intrusion detection .

INTRODUCTION

Malware is a growing concern for individuals and organizations, with cyberattacks becoming more sophisticated and frequent. Traditional malware detection methods, such as signature-based detection, have been shown to be inadequate in detecting new or unknown variants of malware. To address this, researchers turned to convolutional neural networks as it is better for malware detection.

CNN is specifically designed for image classification and classification tasks. They use convolution, clustering, and activation functions to extract relevant features from an image. When used for image classification in a malware dataset, a convolutional layer is applied to detect edges or textures in the image, allowing the CNN to learn underlying patterns in the malware dataset. malicious software. The down-sampling features of the clustering layer reduce the computation required, and the activation function allows learned models to become increasingly complex.

Malware detection using the CNN model is a promising approach because it exploits the strengths of neural networks and signature-based detection. The CNN model can analyze a program's behavior and functionality, and then compare it with predefined signatures to detect known malware. This approach can help identify new or unknown malware variants by discriminating patterns in new entries. The proposed system visualizes the malware as an image and trains the classifier based on deep learning methods to maximize accuracy. This approach ensures improved performance, high classification accuracy, reduced computational costs, faster response to threats, and resilience to cloaking techniques. By representing malware as an image, it becomes easier to apply image processing techniques such as feature extraction and data enhancement.

The proposed method using CNN models and image processing provides a promising solution to handle the limitations of existing detection techniques. It provides a more efficient and accurate way to detect malware by visualizing it as an image and leveraging deep learning techniques to extract meaningful features. This approach provide stronger defenses against cyberattacks.

Malware, short for malware, is a major threat to information security. Hackers use malware to gain unauthorized access to computer systems, steal sensitive data, and damage computer networks. As a result, malware detection has become an important task for any organization that uses computers and networks. Lately, there is an increase in use of deep learning techniques for malware detection. One such approach is to use a convolutional neural network to detect malware based on visual features extracted from images of program behavior. CNN is specifically designed for image classification and analysis tasks. They are used to extract relevant features from an image by applying several convolution, clustering and activation functions. The convolution layer helps detect edges or textures in images, allowing the neural network to learn underlying patterns in malware datasets. The down-sampling features of the clustering layer reduce the computation required, and the activation function allows learned models to become increasingly complex.

Traditional malware detection techniques rely on signature-based detection systems that use predefined signatures or hashes to identify known malware. However, these approaches can be

circumvented by new or unknown malware variants. Neural networks, on the other hand, can analyze the behavior and functionality of a program and compare them with predefined signatures. This approach leverages the strengths of neural networks and signature-based detection to distinguish patterns in new entries and detect known malware.

Malware detection is crucial for any organization that wants to protect its information systems against threats from malicious hackers. The prevalence of malware on the Internet has grown exponentially, making it essential to have an accurate classifier to identify and prevent malware attacks. Malware detection works as an early warning system for computer security related to malware and cyber attacks, preventing hackers from accessing your computer and preventing information from being stolen. violate.

Existing malware analysis technologies include the use of static and dynamic code analysis, which have their specific strengths. However, static analysis suffers from code obfuscation due to the need to decompress and decode the code, while dynamic analysis can ignore malicious behavior due to inappropriate virtual environments. Both approaches are computationally heavy and time consuming.

To overcome these limitations, the proposed system uses a new approach which is to first visualize the malware as an image and then train the classifier based on deep learning methods to Maximize accuracy. This approach ensures improved performance, high classification accuracy, reduced computational costs, faster response to threats, and resilience to cloaking techniques.

The CNN model is implemented by the suggested system using TensorFlow. The proposed method also makes use of fastai, a library that offers users high-level components that may be utilized swiftly and efficiently to produce the most cutting-edge outcomes in the ML domains. Additionally, it provides low-level components that may be combined and matched to produce new processes.

LITERATURE SURVEY

Sr. No.	Name of the transaction/journal/conference	Major technologies used	Results/ Outcome	Drawbacks
[1]	Malware Images: Visualization and Automatic Classification (2017)	To categorize malware photos, the authors employed a deep-learning strategy based on CNNs. To visualize malware pictures and their related class labels, they created the Malware Image Grid (MIG) visualization approach. They evaluated the effectiveness of their suggested strategy using accuracy, precision, recall, the F1-score, and a confusion matrix.	Using a CNN-based classifier, the authors' studies on a sizable collection of malware photos yielded an accuracy of 97.86%. Additionally, they demonstrated how their suggested visualization method (MIG) might offer helpful insights into the properties of malware pictures and the classes to which they belong.	The quality of the virus photos, which may differ depending on the source and the technique used to acquire them, is one possible disadvantage of the suggested strategy. Another drawback is that because the suggested method is based on a pre-defined collection of classes, it might not be able to handle novel or undiscovered forms of malware.
[2]	Use of Data Visualization for Zero-Day Malware Detection (2018)	For machine learning system, the researchers employed a collection of malicious and good files. Each file's characteristics were generated by the algorithm and then shown as scatter plots and heatmaps. To find patterns	Both known and undiscovered malware files may be identified with high accuracy using the method. The visualisations made it easier to spot key malware-indicating	The method is computationally demanding, and creating the visualisations and analysing the data both need a sizable amount of computer power. The method depends on the availability of a

		that might differentiate between malicious and good files, the visualizations were examined.	patterns and characteristics, such high entropy and odd code structure.	dataset containing benign and malicious files for the machine learning algorithm's training, which might be difficult to come by in real life.
[3]	Malware Detection Based on Code Visualization and Two-Level Classification (2021)	A new malware detection system is proposed that utilizes a two-level Artificial Neural Network (ANN) combined with Ensemble Bagged Trees method for improved classification accuracy.	The ANN method is faster, up to 30 times faster, than the ensemble classification method and accuracy gets better with bigger and comprehensive datasets. This makes it a promising solution for implementation in antivirus software,	The continuously growing pool of available knowledge presents a challenge for the complexity and efficiency of methods, as it greatly affects their execution time.
[4]	An Evaluation of Image-based Malware Classification Using Machine Learning (2020)	This study includes the evaluation of multiple classifiers including k-NN, Naive Bayes, Support Vector Machine, and GIST and Convolutional Neural Network along with the Gabor wavelet-kNN method.	Among the test taken, k-NN had the highest accuracy on the datasets while Naive Bayes showed limitations in its performance for image-based malware classification.	K-NN relies on the Euclidean distance to determine the nearest neighbors, but when the number of pixels is too high it can lead to confusion rather than improving accuracy.

[5]	Visualized Malware Classification Based-on Convolutional Neural Network (2019)	CNN and image converter are the proposed malware classifier, which uses a combination of image processing and machine learning techniques to accurately identify and classify malicious code.	The study suggests that the proposed malware family classification model is effective in providing effective defense techniques. By quickly classifying malware families without executing malicious code or conducting in-depth code analysis, the model is well-suited for the fast-paced reality.	The proposed method relies on the use of (CNNs), which can become computationally heavy as number of malicious code categories increases. This means that higher hardware performance is required, making it challenging to use the method for large-scale analysis.
-----	--	---	--	--

Table 1: Literature Survey

[6] We examine artificial intelligence (AI) methods for identifying and preventing malware, underscoring the importance of looking to the future for solutions. Researchers benefit from this information as they investigate AI-powered malware detection and prevention.

CNN, RNN, LSTM, and autoencoders are only some of the deep learning techniques discussed in this article's [7] discussion of malware detection. Since LSTM and autoencoders have shown some success in Android malware detection, it is clear that more study of deep learning is necessary to make progress in this area.

Using picture modification and machine learning classifiers, [8] researchers present a computer vision-based method for malware detection. The method outperforms other methods by a wide margin, with an accuracy of 97.01 percent.

Using static analysis, dynamic analysis, and the EfficientNet B3 model for malware detection, [9] a two-layer hybrid malware detection system (2-MaD) is presented for IoT malware identification.

In [10] we present IoT-IDCS-CNN, an IoT attack detection and classification system based on

convolutional neural networks. With the help of convolutional neural networks (CNNs), HPC, and parallel processing, it is able to get an accuracy of 99.3% in binary classification and 98.2% in multiclass classification.

To better understand the mechanics behind machine learning algorithms and their conclusions, [11] a framework is offered to analyse the generalizability and functionality of the MalConv architecture.

[12] For those just getting started with malware detection using deep learning, a survey of the literature reveals that convolutional neural networks (CNNs), long short-term memories (LSTMs), deep belief networks (DBNs), and autoencoders are some of the most popular methods.

Through malware visualisation, feature selection, and preprocessing, [13] researchers use deep learning techniques like CNNs and LSTM to create a highly accurate malware detection model.

[14] We present a location-based hashing, Apache Spark-based, big data and machine learning malware detection solution. The model improves detection speed while maintaining 99.8 percent accuracy.

[15] In this paper, we offer an Android malware detection approach based on autoencoders and five feature sets, and we show that this method can accurately identify malware.

[16] Using grey-scale image representation and autoencoder networks, we describe a novel malware detection strategy that achieves 96% accuracy and an F-score of roughly 96%.

With the help of AI planning, the authors of Paper [17] get to the bottom of the hypothesis exploration problem in malware detection. The authors frame the issue of hypothesis generation as one of AI planning, with time-dependent objectives and the associated costs of possible courses of action. In this study, we describe a strategy for integrating a modern planner into grounded research and establish the idea of "plausibility" for hypotheses in the context of flawed observations. Experimentally, the proposed method outperforms prior art by a wide margin.

The methods of SVM, KNN, LDA, LSTM, CNN-LSTM, and autoencoders, among others, are used in paper [18] to detect malicious Android-based assaults. SVM and LSTM models show great accuracy in identifying malware on Android applications, and the cybersecurity solution is assessed using two benchmark datasets.

In Paper [19], the authors present a methodology for improving the accuracy of malware detection systems that use machine learning. The study recommends employing adversarial training and producing adversarial malware pictures to strengthen classification models' resistance to attacks from malicious actors.

A literature review of deep learning and machine learning techniques for ransomware detection is presented in paper [20]. The study's goal is to find the holes in current methods and investigate how they could be patched. This study also explores how machine learning and deep learning can be used

to identify zero-day threats and decipher ransomware's tactics.

In order to make malware detection more robust, paper [21] describes a method for creating adversarial malware images and retraining classification systems. The work improves the attack resilience of multiple categorization models by applying adversarial training.

In Paper [22], the authors use a variety of machine learning and deep learning algorithms to identify malicious attacks on Android mobile devices, including SVM, KNN, LDA, LSTM, CNN-LSTM, and autoencoders. The results show that the SVM, LSTM, and CNN-LSTM algorithms are efficient at detecting malware on Android.

This paper [23] examines the state of malware research, feature extraction methods, and classification/clustering algorithms utilised in advanced malware detection systems. Data mining approaches for malware detection are discussed, along with the difficulties and complications that arise from applying them, and predictions are made regarding the development of malware in the future.

Using kNN classification and a VP tree with a similarity hash, the authors of Paper [24] offer a new method for detecting malicious files. The method was designed to improve upon existing pattern-based antivirus and AI-based malware detection systems by speeding up the detection process and increasing the detection rate.

In order to improve the safety of Data Centres and Smart Cities, the authors of paper [25] introduce pAElla, an AI-driven edge computing and high-resolution power usage-based technique. The technique uses autoencoders and power spectral density measurements to detect malware in real time on an IoT-based monitoring system operating outside of the band. According to the findings, pAElla outperforms State-of-the-Art methods in both accuracy and malware detection breadth.

In paper [26], the authors propose combining permission and API calls with machine learning approaches to identify potentially malicious Android applications. The strategy is based on elementary static analysis and is universally applicable to mobile programmes.

Using both static and dynamic malware analysis techniques, the authors of Paper [27] provide a framework for AI-based ransomware detection and introduce a detection tool (AIRaD). Maximum accuracy was 99.54 percent, with a false-positive rate of just 0.005 percent.

In order to detect malware, the authors of paper [28] use LSTM to model traffic data from networks as a series of flows. Both new families of malware and malware inside 50 connection flows were detected with a high degree of accuracy.

In their paper, [29], the authors probe the viability of using LightGBM to categorise malware statically. The suggested method outperforms the standard log loss function on the EMBER dataset, with an AUC of 0.979, thanks to the use of a modified log loss function to optimise learning.

In paper [30], the authors offer a method for detecting ransomware using artificial intelligence and a combination of static and dynamic malware analysis. This research uses NLP, ARM, and ML classifiers to process characteristics extracted from dynamic link libraries (DLLs), functions (function calls), and assemblies. Using support vector machines and the Adaboost with J48 algorithms, the proposed model obtained a remarkable 99.54% accuracy with a false-positive rate of 0.005%.

SYSTEM DESIGN

Architecture

The proposed system consists of two major script files: `cnn-model` and `exe-to-img`, with an auxiliary script file: `extract-exe`. The main function of each script file are as follows:

1. **extract-exe:** It is an auxiliary script that copies exe files present in a directory over to a target directory. Its functioning is done in 3 major steps that loop over the directory recursively:
 - a. Browse directory to get file list
 - b. Iterate through file list to get executables list
 - c. Copy executables in the list to target directory
2. **exe-to-img:** It is a main script that converts all executables into images of equal sizes that can be used as dataset for the CNN model. The steps of execution of the script are as follows:
 - a. Browse directory to get file list
 - b. For each executable in file list, execute the following steps:
 - i. Get bytecode data of executable through standard python file handling functions
 - ii. Re-structure data into groups of 3 bytes each to form pixel data
 - iii. Re-shape grouped data into 512x512 dimension array. Ignore all overflowing bytes, and pad underflowing bytes with zeros
 - iv. Create png image using standard python png library functions
 - v. Save png image to target directory
3. **cnn-model:** It is the primary script that hosts the CNN model. There are three main environment variables that store the directory locations of the training dataset, testing dataset and saved model respectively. In order to make training and handling the model with ease, two types of execution are implemented:
 - a. **Refresh mode:** The script ignores the existing model in the directory and overwrites it with a new untrained model. This mode is useful when experimenting with model architectures and dataset sizes. It is recommended to run the training option as the first operation to get accurate results during predictions.
 - b. **Continue mode:** The script loads the existing model in the directory to “continue” the operation of the program as if no exit command was given in the middle. This mode is useful in dividing longer training sequences over separate execution passes, and in running the program to directly get prediction results.

The script consists of a main menu that branches into 3 execution paradigms:

- a. **Train:** This option is used to train the CNN model. It follows these steps to complete one pass of training cycle:
 - i. Load training dataset
 - ii. Read number of epochs from the user

- iii. For each epoch, follow these steps:
 1. Iterate through the entire training dataset and execute the following steps for each image in dataset:
 - a. Make a prediction for the image
 - b. Compare result to actual values in the dataset
 - c. Modify model weights according to error difference between predicted actual values
 2. Report epoch progress to user
- b. **Predict:** This option is used to get a prediction from the CNN Model. The following main stages of the prediction cycle are completed in this option:
 - i. Parsing testing dataset
 - ii. Predicting on images present in the dataset
 - iii. Comparing predictions with actual values in the dataset to get error points
 - iv. Calculating performance metrics
- c. **Edit Environment Variables:** There are three main environment variables used by the script file. This option is used to edit the variables to desired values. The variables are as follows:
 - i. Training Dataset directory
 - ii. Testing Dataset directory
 - iii. Saved Model directory

The interactions between the scripts are shown in the following figure:

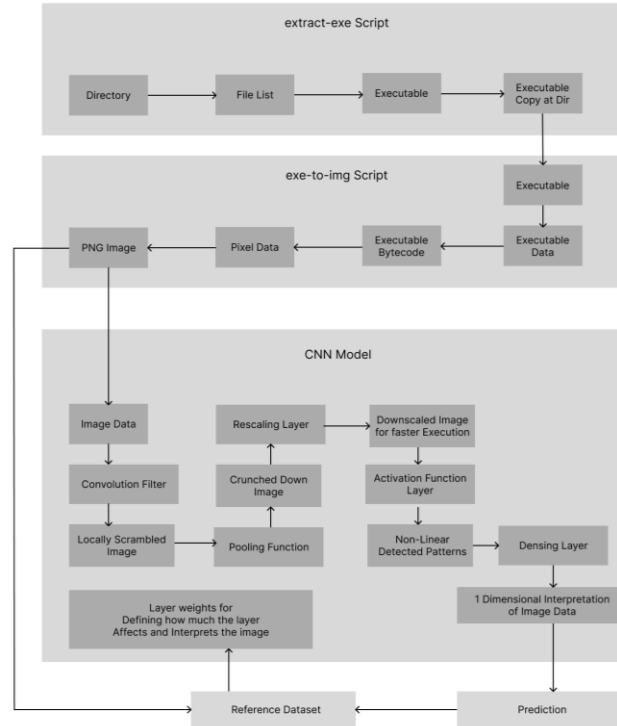


Fig 1 : Architecture Diagram of the proposed model

Libraries used

1. **TensorFlow**: TensorFlow is a freely available software library that empowers developers to create and train machine learning models. It was originally created by Google and has found widespread use in diverse fields, ranging from image and speech recognition to natural language processing and robotics. It is popular both in research and industry for its ability to handle complex computations and its ease of use.
2. **Keras**: Keras is a comprehensive library that offers an array of pre-designed layers to developers, including convolutional layers suitable for processing images, recurrent layers ideal for sequential data, and dense layers for general-purpose computation. In addition, Keras provides an array of training tools, including various optimization algorithms, loss functions, and metrics, that developers can specify while building their models to achieve the desired training outcomes.
3. **ArgParse**: ArgParse is a python standard library used for parsing command line arguments and options. It is very useful in creating command line interfaces (CLI) that accept various options, arguments and sub-commands. In this project, ArgParse is used to give a CLI to exe-to-img, and specify startup mode for cnn-model.

Control Flow

In our proposed workflow, the user first converts the executables in his specified directory to images using the exe-to-img script provided in our solution. Segregation of the executables into training, validation and testing dataset is made easier using the auxiliary extract-exe script, which copies the executable files in a user-defined directory to a target directory. The segregation of data using extract-exe script, if applied, needs to be applied before conversion to images. The finally segregated dataset is used by the cnn-model script that trains the CNN Model on the training and validation datasets. The user can verify the operation of the model through the cnn-model script's "Make Prediction" option.

A step-by-step detailed process is as follows:

1. Extract exe files using the extract-exe script. Command:
python3 extract-exe.py
 2. Convert exe files to images using the exe-to-img script. Command:
python3 exe-to-img --folder [Source Directory] --target [Target Directory] [--benign | --malware]
 3. Run cnn-model script
python3 model.py [--refresh | --continue]
- The following options can be selected in cnn-model main menu in no particular order:

- a. Train model using train model option, execute it first if running in refresh mode
- b. Test model performance using make prediction option
- c. Edit environment variables if images are stored in locations differing from defaults:
 - i. Model location: ./model/
 - ii. Training and validation dataset location: ../Generated Images/

iii. Testing dataset location: ../Equalised Images/

The flow of control can be visualised using the following figure:

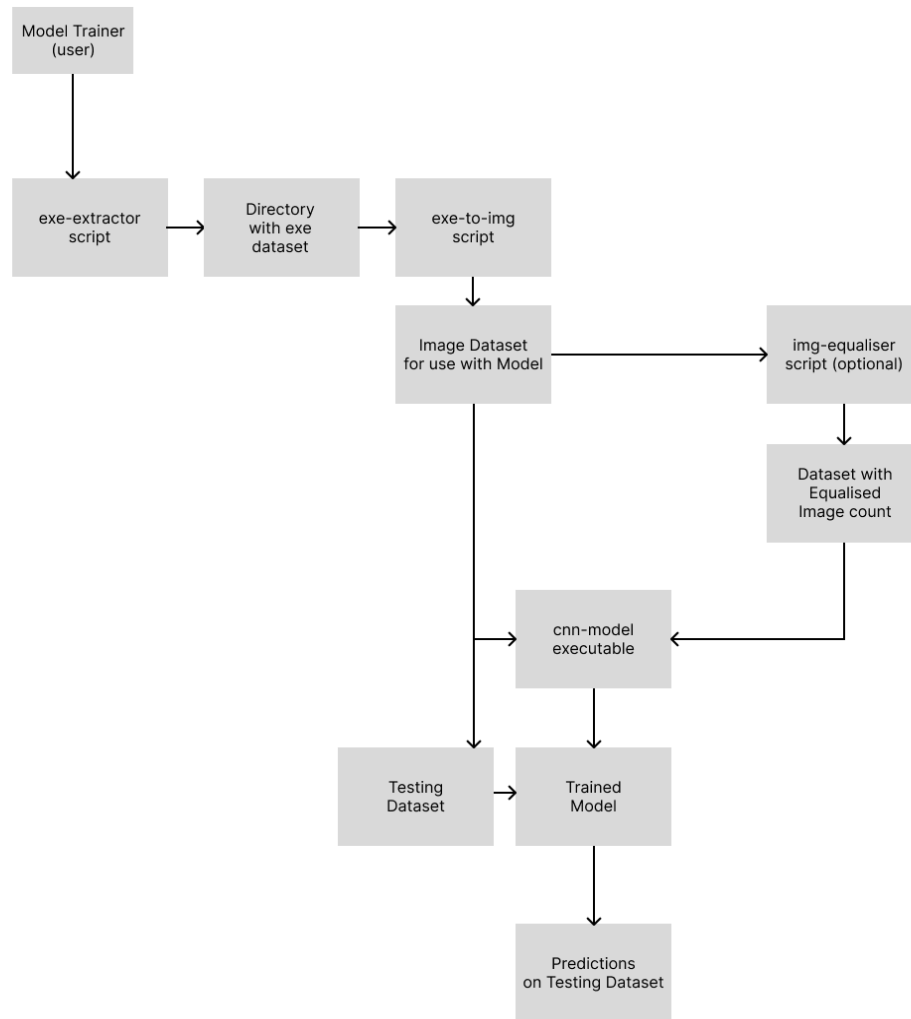
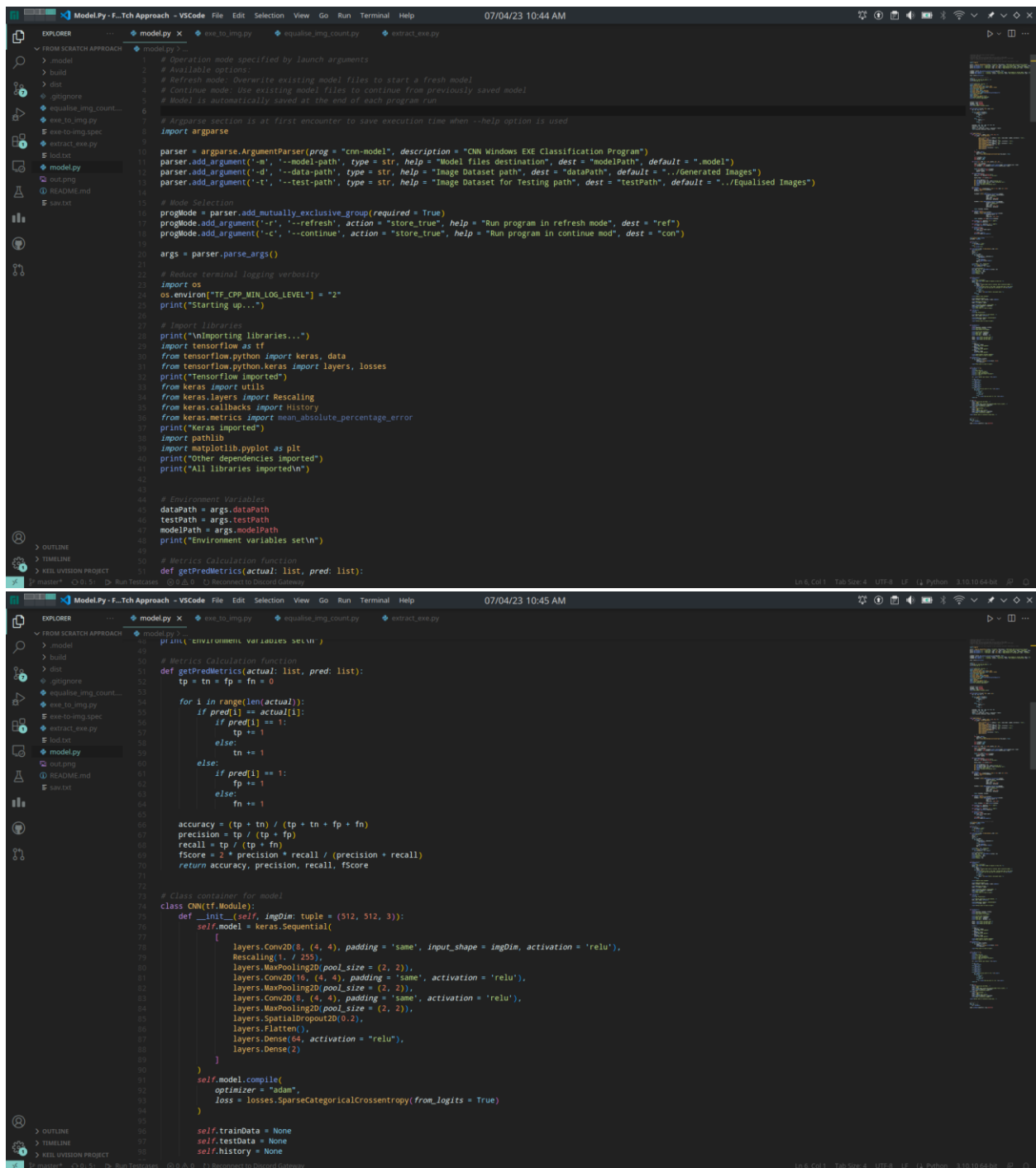


Fig 2 : Control Flow

IMPLEMENTATION

cnn-model



```
1 # Operation mode specified by launch arguments
2 # Available options:
3 # Refresh mode: Overwrite existing model files to start a fresh model
4 # Continue mode: use existing model files to continue from previously saved model
5 # Model is automatically saved at the end of each program run.
6
7 # Argparse section is at first encounter to save execution time when --help option is used
8 import argparse
9
10 parser = argparse.ArgumentParser(prog = "cnn-model", description = "CNN Windows EXE Classification Program")
11 parser.add_argument('-m', '--model-path', type = str, help = "Model files destination", dest = "modelPath", default = ".\\model")
12 parser.add_argument('-d', '--data-path', type = str, help = "Image Dataset path", dest = "dataPath", default = "../Generated Images")
13 parser.add_argument('-t', '--test-path', type = str, help = "Image Dataset for Testing path", dest = "testPath", default = "../Equalised Images")
14
15 # Mode Selection
16 progMode = parser.add_mutually_exclusive_group(required = True)
17 progMode.add_argument('-r', '--refresh', action = "store_true", help = "Run program in refresh mode", dest = "ref")
18 progMode.add_argument('-c', '--continue', action = "store_true", help = "Run program in continue mod", dest = "con")
19
20 args = parser.parse_args()
21
22 # Reduce terminal logging verbosity
23 import os
24 os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
25 print("Starting up...")
26
27 # Import Libraries
28 print("\nImporting libraries...")
29 import tensorflow as tf
30 from tensorflow.python.keras import data
31 from tensorflow.python.keras import layers, losses
32 print("Tensorflow imported")
33 from keras import utils
34 from keras.layers import Rescaling
35 from keras.callbacks import ModelHistory
36 from keras.metrics import mean_absolute_percentage_error
37 print("Keras imported")
38 import pathlib
39 import matplotlib.pyplot as plt
40 print("Other dependencies imported")
41 print("All libraries imported\n")
42
43
44 # Environment Variables
45 dataPath = args.dataPath
46 testPath = args.testPath
47 modelPath = args.modelPath
48 print("Environment variables set\n")
49
50 # Metrics Calculation Function
51 def getPredMetrics(actual: list, pred: list):
52     tp = tn = fp = fn = 0
53     for i in range(len(actual)):
54         if pred[i] == actual[i]:
55             if pred[i] == 1:
56                 tp += 1
57             else:
58                 tn += 1
59         else:
60             if pred[i] == 1:
61                 fp += 1
62             else:
63                 fn += 1
64
65     accuracy = (tp + tn) / (tp + tn + fp + fn)
66     precision = tp / (tp + fp)
67     recall = tn / (tn + fn)
68     fscore = 2 * precision * recall / (precision + recall)
69     return accuracy, precision, recall, fscore
70
71
72 # Class container for model
73 class CNN(tf.Module):
74     def __init__(self, imgDim: tuple = (512, 512, 3)):
75         self.model = keras.Sequential(
76             [
77                 layers.Conv2D(8, (4, 4), padding = 'same', input_shape = imgDim, activation = 'relu'),
78                 Rescaling(1. / 255),
79                 layers.MaxPooling2D(pool_size = (2, 2)),
80                 layers.Conv2D(16, (4, 4), padding = 'same', activation = 'relu'),
81                 layers.MaxPooling2D(pool_size = (2, 2)),
82                 layers.Conv2D(8, (4, 4), padding = 'same', activation = 'relu'),
83                 layers.MaxPooling2D(pool_size = (2, 2)),
84                 layers.SpatialDropout2D(0.2),
85                 layers.Flatten(),
86                 layers.Dense(64, activation = "relu"),
87                 layers.Dense(2)
88             ]
89         )
90         self.model.compile(
91             optimizer = "adam",
92             loss = losses.SparseCategoricalCrossentropy(from_logits = True)
93         )
94
95         self.trainData = None
96         self.testData = None
97         self.history = None
```

```
ModelPy - F...Tch Approach - VSCode File Edit Selection View Go Run Terminal Help 07/04/23 10:45 AM
EXPLORER
FROM SCRATCH APPROACH
> model
> build
> test
> gptignore
> equalize_img_count.py
> exe_to_img.py
> exe_to_img.spec
> extract_exe.py
> load_exe
> model.py
> out.png
> README.md
> save.txt

> OUTLINE
> TERMINAL
> KILL SESSION PROJECT
? master* C:\S:\S\ Run Testcases @ 0.0.0.0 F) Reconnected to Discord Gateway

190 self.trainData = None
191 self.testData = None
192 self.history = None
193
194 def train(self, seed: int = 1542, epochs: int = 10):
195     """Configure datasets for better loading performance"""
196     tuner = data.AUTOTUNE
197     self.trainData = self.trainData.cache().shuffle(seed).prefetch(buffer_size = tuner)
198     self.testData = self.testData.cache().prefetch(buffer_size = tuner)
199
200     # Training the model
201     return self.model.fit(self.trainData,
202                           validation_data = self.testData,
203                           epochs = epochs)
204
205 def trainSummary(self):
206     loss = self.history.history['loss']
207     val_loss = self.history.history['val_loss']
208
209     epochs_range = self.history.epoch
210
211     plt.figure(figsize=(8, 8))
212     plt.plot(epochs_range, loss, label='Training Loss')
213     plt.plot(epochs_range, val_loss, label='Validation Loss')
214     plt.legend(loc='upper right')
215     plt.title('Training and Validation Loss')
216     plt.show()
217
218 @tf.function
219 def loadData(self, trainTestSplit: float = 0.35, seed: int = 1574):
220     dataDir = pathlib.Path(dataPath)
221     batchSize = 32
222
223     # Creating dataset loaders
224     trainData = utils.image_dataset_from_directory(dataDir,
225                                                    validation_split = trainTestSplit,
226                                                    subset = "training",
227                                                    seed = seed,
228                                                    image_size = (512, 512),
229                                                    batch_size = batchSize)
230
231     testData = utils.image_dataset_from_directory(dataDir,
232                                                  validation_split = trainTestSplit,
233                                                  subset = "validation",
234                                                  seed = seed,
235                                                  image_size = (512, 512),
236                                                  batch_size = batchSize)
237
238     return trainData, testData
239
240 def rawPredict(self):
241
242     return trainData, testData
243
244 def rawPredict(self):
245     testDir = pathlib.Path(testPath)
246     testData = utils.image_dataset_from_directory(testDir,
247                                                    validation_split = 0,
248                                                    seed = 0,
249                                                    image_size = (512, 512))
250
251     return testData, self.model.predict(testData)
252
253 def saveModel(self, modelFile: str = ".model"):
254     self.model.save(modelFile)
255
256 def loadModel(self, modelFile: str = ".model"):
257     self.model.load_weights(modelFile)
258
259 def debugInfo(self):
260     print(self.model)
261     for layer in self.model.layers:
262         print(layer)
263
264     self.model.summary()
265     print(self.model.get_weights())
266
267 # Define the OS currently running the model
268 from platform import system
269 currentOs = system()
270
271 # Helper Functions
272 def clear():
273     if currentOs == 'Linux':
274         os.system("clear")
275     else:
276         os.system("cls")
277
278 def interpretPred(data: tf.data.Dataset, pred: list):
279     print("Results:")
280     idx = 0
281     actual = []
282     res = []
283     for x, y in data:
284         for tensor in y:
285             if(pred[idx][0] < pred[idx][1]):
286                 res.append(1)
287             else:
288                 res.append(0)
289             actual.append(tensor.numpy())
290             idx += 1
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2
```



```
ModelPy - F...Tch Approach - VSCode File Edit Selection View Go Run Terminal Help 07/04/23 10:46 AM

EXPLORER
  FROM SCRATCH APPROACH
    > model
    > build
    > dist
    > gptignore
    > equalize_img_count...
    > exe_to_img.py
    > exe_to_img.spec
    > extract_exe.py
    > load.txt
    > model.py
    > out.png
    > README.md
    > save.txt
  > OUTLINE
  > TIMELINE
  > KIEL VISION PROJECT

model.py
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000

def manualTrain():
    clrscr()
    print("Model Training\n")
    epoch = int(input("Enter number of epochs to train for: "))
    while True:
        if epoch <= 1:
            epoch = ("Given epoch value is invalid, enter a positive number: ")
        elif epoch > 20:
            print("The epoch value is very high, training may take a long time")
            e = input("Are you sure you want to continue? (Y / N)")[0].lower()
            if e == "y":
                break
            elif e == "n":
                epoch = int(input("Enter a new epoch value: "))
            else:
                break
    print("\nEpoch value accepted")

    print("Importing dataset...")
    model.trainData, model.testData = model.loadData()

    print("Beginning training...")
    model.train(epochs = epoch)

    print("\nTraining complete, saving model...")
    model.saveModel(modelFile = modelPath)
    print("Model saved\n")
    input("Press enter to return to menu")

def rawPredict():
    print("Model saved\n")
    input("Press enter to return to menu")

    def rawPredict():
        clrscr()
        print("Raw Prediction\n")
        print("Making predictions on test dataset...")
        data, pred = model.rawPredict()
        print("Predictions complete, interpreting\n")
        interpretPred(data, pred)
        input("\nPress enter to return to menu")

    def editEnvVars():
        clrscr()
        global modelPath, dataPath, testPath
        print("Environment Setup\n")
        print("Leave blank if no change")

        print("Current path variables:")
        print("Model path:", modelPath)
        print("Data path:", dataPath)
        print("Test path:", testPath, "\n")

        mPath = input("Enter new model path: ")
        dPath = input("Enter new data path: ")
        tPath = input("Enter new test path: ")

        print("")

        if mPath != "":
            modelPath = mPath
            print("Model path updated")
        if dPath != "":
            dataPath = dPath
            print("Data path updated")
        if tPath != "":
            testPath = tPath
            print("Test path updated")

        print("\nEnvironment variables updated")
        input("Press enter to return to menu")

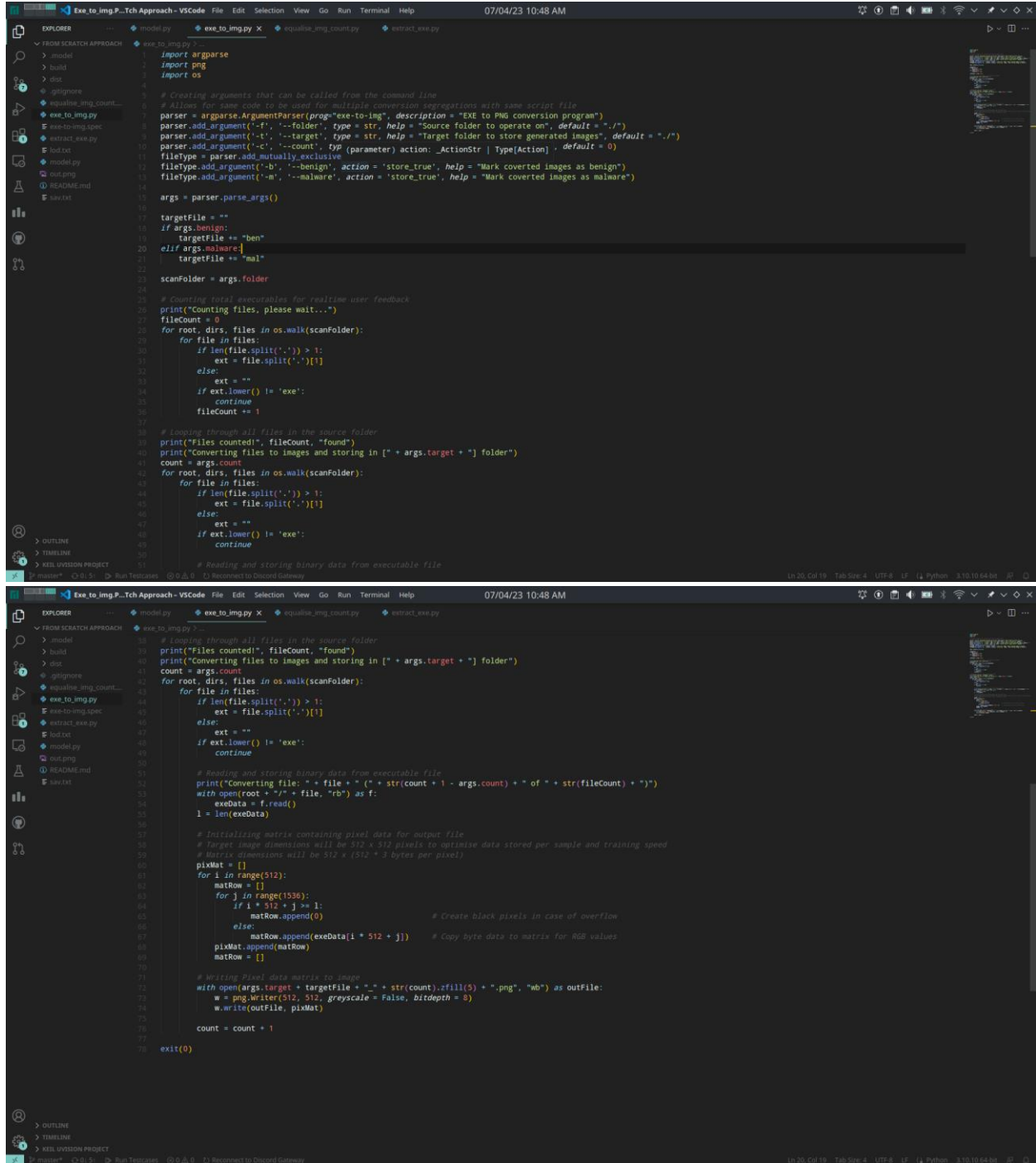
    def debugChecks():
        for x, y in model.trainData:
            for tensor in x[5]:
                newTensor = tf.math.divide(tensor, 255.0)
```

```
ModelPy - F...Tch Approach - VSCode File Edit Selection View Go Run Terminal Help 07/04/23 10:47 AM

EXPLORER
  FROM SCRATCH APPROACH
    > model
    > build
    > dist
    > ignore
    > equalise_img_count...
    > exe_to_img.py
    > exe_to_img.spec
    > extract_exe.py
    > load.txt
    > model.py
    > out.png
    > README.md
    > save.txt
  > KEEL DIVISION PROJECT
  > master* C:\S:\S\ Run TestCases @ 0.0.0.0 Reconnect to Discord Gateway

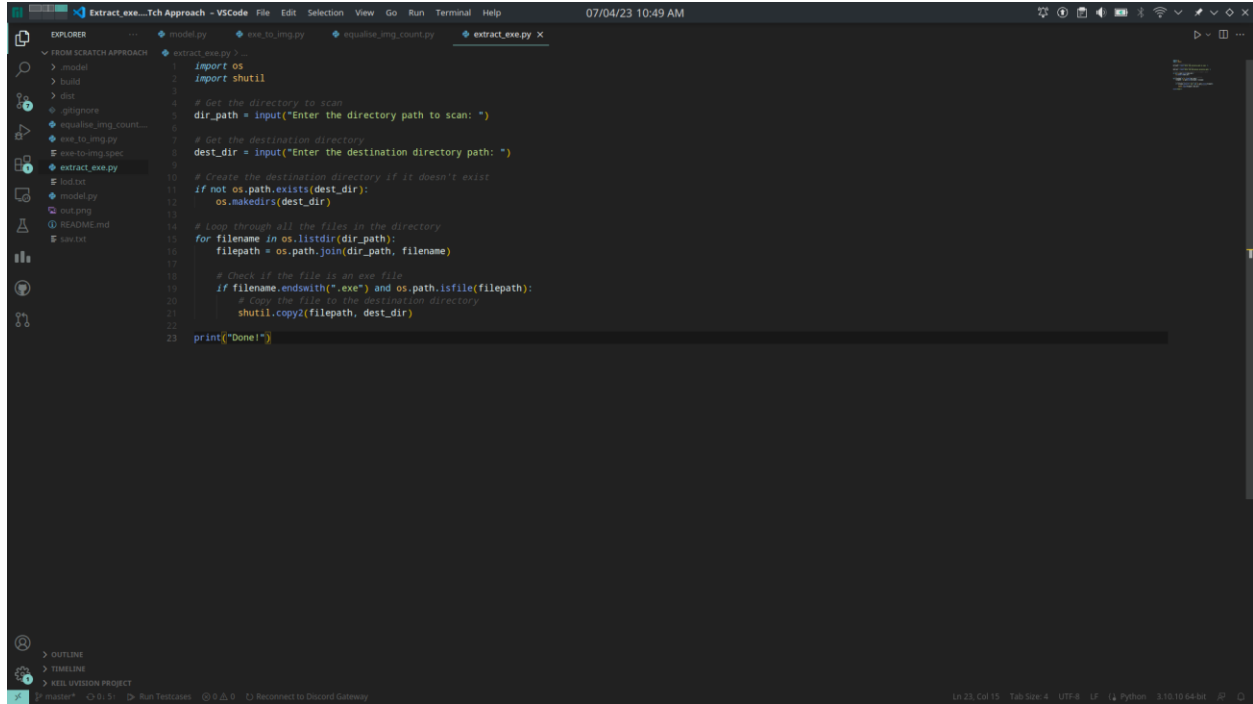
model.py
  290 input("Press enter to return to menu")
  291
  292 def debugChecks():
  293     for x, y in model.trainData:
  294         for tensor in x[:5]:
  295             newTensor = tf.math.divide(tensor, 256.0)
  296             print(newTensor)
  297
  298     input("Press enter to return to menu")
  299
  300 # Main menu Function
  301 def mainMenu() -> str:
  302     clrscr()
  303     print("Main Menu")
  304     print("T -> Train the model")
  305     print("P -> Make a prediction")
  306     print("E -> Edit Environment Variables")
  307     print("Q -> Quit")
  308
  309     ch = input("\nEnter your choice: ")[0].lower()
  310
  311     if ch == "t":
  312         manualTrain()
  313     elif ch == "p":
  314         ranPredict()
  315     elif ch == "e":
  316         editEnvVars()
  317     elif ch == "q":
  318         debugChecks()
  319     elif ch == "q":
  320         e = input("Are you sure? (Y / N): ")[0].lower()
  321         while True:
  322             if e == "y":
  323                 break
  324             elif e == "n":
  325                 return e
  326             else:
  327                 e = input("Are you sure? (Y / N): ")[0].lower()
  328         return ch
  329
  330 if args.ref:
  331     print("Initialising new Model...")
  332     model = CNN()
  333     print("Saving model (overwrites existing model files in path)...")
  334     model.saveModel(modelFile = modelPath)
  335     print("Model successfully saved!")
  336
  337 elif args.con:
  338     print("Creating Model...")
  339     model = CNN()
  340
  341     print("Loading model values")
  342     model.loadModel(modelFile = modelPath)
  343     print("Model successfully loaded!")
  344
  345     input("\nPress enter to continue to main menu...")
  346
  347 # Main menu loop
  348 ex = 'a'
  349 while ex != 'q':
  350     ex = mainMenu()
  351
  352 model.saveModel(modelFile = args.modelPath)
```

exe-to-img



```
1 import argparse
2 import png
3 import os
4
5 # Creating arguments that can be called from the command line
6 # Using argparse module for multiple conversion operations with some script file
7 parser = argparse.ArgumentParser(prog="exe-to-img", description="EXE to PNG conversion program")
8 parser.add_argument('-f', '--folder', type=str, help="Source folder to operate on", default=".")
9 parser.add_argument('-t', '--target', type=str, help="Target folder to store generated images", default=".")
10 parser.add_argument('-c', '--count', type=int, help="Number of files to convert", default=0)
11 filetype = parser.add_mutually_exclusive_group()
12 filetype.add_argument('-b', '--benign', action='store_true', help="Mark converted images as benign")
13 filetype.add_argument('-m', '--malware', action='store_true', help="Mark converted images as malware")
14
15 args = parser.parse_args()
16
17 targetFile = ""
18 if args.benign:
19     targetFile = "ben"
20 elif args.malware:
21     targetFile = "mal"
22
23 scanFolder = args.folder
24
25 # Counting total executables for realtime user feedback
26 print("Counting files, please wait...")
27 fileCount = 0
28 for root, dirs, files in os.walk(scanFolder):
29     for file in files:
30         if len(file.split('.')) > 1:
31             ext = file.split('.')[1]
32             if ext.lower() != 'exe':
33                 continue
34             fileCount += 1
35
36 # Looping through all files in the source folder
37 print("Files counted!", fileCount, "found")
38 print("Converting files to images and storing in [" + args.target + "] folder")
39 count = args.count
40 for root, dirs, files in os.walk(scanFolder):
41     for file in files:
42         if len(file.split('.')) > 1:
43             ext = file.split('.')[1]
44             if ext.lower() != 'exe':
45                 continue
46
47 # Reading and storing binary data from executable file
48 print("Converting file: " + file + " (" + str(count + 1 - args.count) + " of " + str(fileCount) + ")")
49 with open(root + "/" + file, "rb") as f:
50     exeData = f.read()
51     l = len(exeData)
52
53 # Initializing matrix containing pixel data for output file
54 # Target image dimensions will be 512 x 512 pixels to optimise data stored per sample and training speed
55 # Matrix dimensions will be 512 x (512 * 3 bytes per pixel)
56 pixMat = []
57 for i in range(512):
58     matRow = []
59     for j in range(1536):
60         if i * 512 + j >= l:
61             matRow.append(0) # Create black pixels in case of overflow
62         else:
63             matRow.append(exeData[i * 512 + j]) # Copy byte data to matrix for RGB values
64     pixMat.append(matRow)
65     matRow = []
66
67 # Writing Pixel data matrix to image
68 with open(args.target + targetFile + "." + str(count) + ".png", "wb") as outFile:
69     w = png.Writer(512, 512, greyscale=False, bitdepth=8)
70     w.write(outFile, pixMat)
71
72 count = count + 1
73
74 exit(0)
```

extract-exe



```
1 import os
2 import shutil
3
4 # Get the directory to scan
5 dir_path = input("Enter the directory path to scan: ")
6
7 # Get the destination directory
8 dest_dir = input("Enter the destination directory path: ")
9
10 # Create the destination directory if it doesn't exist
11 if not os.path.exists(dest_dir):
12     os.makedirs(dest_dir)
13
14 # Loop through all the files in the directory
15 for filename in os.listdir(dir_path):
16     filepath = os.path.join(dir_path, filename)
17
18     # Check if the file is an exe file
19     if filename.endswith(".exe") and os.path.isfile(filepath):
20         # Copy the file to the destination directory
21         shutil.copy2(filepath, dest_dir)
22
23 print("Done!")
```

RESULTS AND DISCUSSION

Our Model Performance

Measure	Value
Accuracy	0.952
Precision	0.981
Recall	0.976
F-Score	0.978

Table 2(a): Model Performance when Using 32x32x3 resolution images

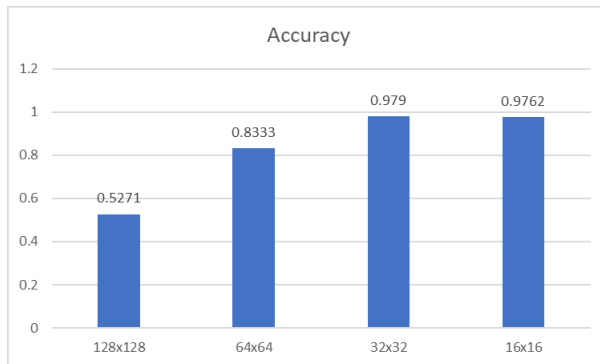
Measure	Value
Accuracy	0.472
Precision	0.472
Recall	0.265
F-Score	0.339

Table 2(b): Model Performance when using 512x512x3 resolution images

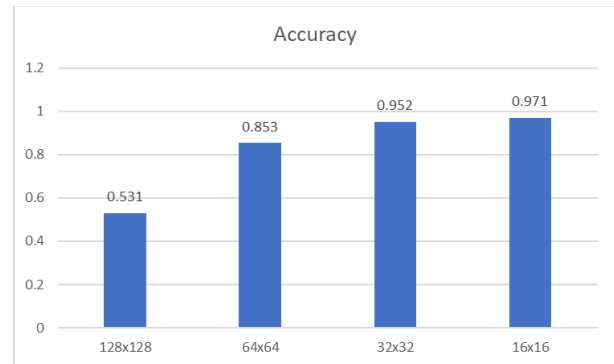
Comparison with Other Models

Model	Accuracy (256x256 Images)	Accuracy (32x32 Images)
k-NN	0.527	0.979
Naive Bayes	NaN	0.955
Support Vector Model	NaN	0.974
Reference CNN	NaN	0.962
CNN	0.531	0.976

Table 3: Model Comparison with Reference k-NN Model



(a)



(b)

Figure 3: Model Performance (a) Reference Model, (b) Our Model

In the base paper referred for the project, the k-Nearest Neighbours algorithm had very high accuracy of 97.9% when the supplied images of 32x32 pixels resolution were used. This is very high performance, but the major trade-off is that only the first 1024 bytes. When images of 128x128 pixels resolution (implying the first 16 kilobytes are read) were used, the accuracy fell down to 52.71%. This implies that in the modern age, where executables are present in sizes of range 10 to 100 megabytes, malware will be easier to pass through when packaged within larger executable files.

In comparison, the CNN used by the base paper had a lower accuracy of 96.29% when 32x32 pixel resolution images are used, compared to the k-NN itself. This discrepancy can be attributed to several factors including the number of epochs and the structure of the neural network itself. Where the CNN gained its advantage was in getting test results, with improvements by a factor of as much as 15.

Coming to the performance metrics of our CNN model, we see that an accuracy of 47.2% was attained when 512x512 pixel images were used. The difference in images is not just in the pixel size, but information packed within a pixel itself. The images used in the base paper used one pixel to store one byte of data, while our images can store 3 bytes of data in each pixel. This implies that our model achieved an accuracy score which was lowered by approximately 5% while processing 48 times more data in each image.

Training Times

Model	128x128 Images	64x64 Images	32x32 Images
Reference k-NN	NaN	NaN	0.33 s
Reference CNN (10 epochs)	NaN	NaN	88.49 s
Our CNN (CPU run, 10 epochs)	840 s	520 s	195 s
Our CNN (GPU Accelerated, 10 epochs)	79.8 s	49.4 s	24.7

Table 4: Model Training Time Comparison

The models specified by the base paper were lacking in training times required for images of resolutions 128x128 pixels and 64x64 pixels, and did not specify the model specifics and code for us to actively run and test the training times. However, it is clearly visible that the increased complexity of our CNN model, demonstrated by a 2-fold increase in CPU bound training times of similarly sized image datasets, was sharply reduced by implementing our proposed approach using NVidia CUDA framework and Tensorflow Keras libraries.

This increased performance from CPU bound training to GPU accelerated training is evident in larger image sizes, with improvements of up to 1052.63%. The faster GPU execution of the model training algorithm can allow for higher epoch counts and model complexities to be used

CONCLUSION AND FUTURE SCOPE

In conclusion, using image processing with CNN models for malware detection offers a promising solution to the challenges faced by traditional signature- and behavior-based detection methods. system must face. By converting malware into an image representation and using deep learning techniques to extract relevant features, this method has shown significant improvements in accuracy, speed, and accuracy. resilience to cloaking techniques.

In addition, the flexibility and scalability of CNN models enables the detection of many different types of malware, including previously unknown variants. As the use of machine learning in cybersecurity continues to grow, the potential for integrating image processing with CNN models in malware detection is enormous.

The scope for future research in this area is enormous. One potential direction is to explore the use of transformation and aggregation learning methods to further improve model accuracy. Transfer learning can leverage pre-trained CNN models on large data sets to learn features relevant to malware detection, while ensemble methods can combine multiple models. to improve overall performance.

Additionally, investigating the use of explainable AI (XAI) techniques can improve the interpretability of the model, allowing security professionals to better understand how the model performs predictions and capable of identifying new features that can help detect malware.

In summary, the use of image processing with CNN models in malware detection has shown significant improvements in accuracy, speed, and resiliency. Through continuous research and development, this method has the potential to be a powerful tool in the fight against malware and cyber threats.

REFERENCES

- [1] Faruk, M. J. H., Shahriar, H., Valero, M., Barsha, F. L., Sobhan, S., Khan, M. A., ... & Wu, F. (2021, December). Malware detection and prevention using artificial intelligence techniques. In 2021 IEEE International Conference on Big Data (Big Data) (pp. 5369-5377). IEEE.
- [2] Majid, A. A. M., Alshaibi, A. J., Kostyuchenko, E., & Shelupanov, A. (2021). A review of artificial intelligence based malware detection using deep learning. *Materials Today: Proceedings*.
- [3] Qasem Abu Al-Haija and Saleh Zein-Sabatto (2020, December) An Efficient Deep-Learning-Based Detection and Classification System for Cyber-Attacks in IoT Communication Networks 10.3390/9122152
- [4] Baek, S., Jeon, J., Jeong, B., & Jeong, Y. S. (2021). Two-stage hybrid malware detection using deep learning. *Human-centric Computing and Information Sciences*, 11(27), 10-22967.
- [5] Syed Shakir Hameed Shah ,Abd Rahim Ahmad , Norziana Jamil and Atta ur Rehman Khan(2022, July) Memory Forensics-Based Malware Detection Using Computer Vision and Machine Learning 11, 2579.
- [6] Bose, S., Barao, T., & Liu, X. (2020, July). Explaining ai for malware detection: Analysis of mechanisms of malconv. In 2020 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.
- [7] Rahman Ali , Asmat Ali,Farkhund Iqbal,Mohammed Hussain ,and Farhan Ullah (2022,September)Deep Learning Methods for Malware and Intrusion Detection: A Systematic Literature Review 10.1155/2022/2959222
- [8] Huang, L. C., Chang, C. H., & Hwang, M. S. (2020). Research on malware detection and classification based on artificial intelligence. *International Journal of Network Security*, 22(5), 717-727.
- [9] Manish Kumar (2021, October) Scalable malware detection system using big data and distributed machine learning approach. Springer-Verlag GmbH Germany. doi.org/10.1007/s00500-021-06492-9(
- [10] Abdelmonim Naway, Yuancheng Li (2019, Jan) Android Malware Detection Using Autoencoder doi.org/10.48550/arXiv.1901.07315
- [11] Xing, X., Jin, X., Elahi, H., Jiang, H., & Wang, G. (2022). A malware detection approach using autoencoder in deep learning. *IEEE Access*, 10, 25696-25706.
- [12] Sohrabi, S., Udreă, O., & Riabov, A. (2013, June). Hypothesis exploration for malware detection using planning. In Twenty-Seventh AAAI Conference on Artificial Intelligence.
- [13] Alkahtani, H., & Aldhyani, T. H. (2022). Artificial intelligence algorithms for malware detection in android-operated mobile devices. *Sensors*, 22(6), 2268.
- [14] Patil S, Varadarajan V, Walimbe D, Gulechha S, Shenoy S, Raina A, Kotecha K. Improving the Robustness of AI-Based Malware Detection Using Adversarial Machine Learning. *Algorithms*. 2021; 14(10):297.

- [15] Fernando DW, Komninos N, Chen T. A Study on the Evolution of Ransomware Detection Using Machine Learning and Deep Learning Techniques. *IoT*. 2020; 1(2):551-604
- [16] Patil, S., Varadarajan, V., Walimbe, D., Gulechha, S., Shenoy, S., Raina, A., & Kotecha, K. (2021). Improving the robustness of AI-based malware detection using adversarial machine learning. *Algorithms*, 14(10), 297.
- [17] Alkahtani H, Aldhyani THH. Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices. *Sensors*. 2022; 22(6):2268
- [18] Ye, Y., Li, T., Adjero, D., & Iyengar, S. S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3), 1-40.
- [19] Choi S. Combined kNN Classification and Hierarchical Similarity Hash for Fast Malware Detection. *Applied Sciences*. 2020; 10(15):5173.
- [20] Libri, A., Bartolini, A., & Benini, L. (2020). pAElla: Edge AI-based real-time malware detection in data centers. *IEEE Internet of Things Journal*, 7(10), 9589-9599.
- [21] Peiravian, N., & Zhu, X. (2013, November). Machine learning for android malware detection using permission and api calls. In 2013 IEEE 25th international conference on tools with artificial intelligence (pp. 300-305). IEEE.
- [22] Poudyal, S., & Dasgupta, D. (2020, December). Ai-powered ransomware detection framework. In 2020 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1154-1161). IEEE.
- [23] Tanjie Wang, Yueshen Xu, Xinkui Zhao, Zhiping Jiang, Rui Li, Android malware detection via efficient application programming interface call sequences extraction and machine learning classifiers, *IET Software*, 10.1049/sfw2.12083.
- [24] Gao, Y., Hasegawa, H., Yamaguchi, Y., & Shimada, H. (2022). Malware detection using LightGBM with a custom logistic loss function. *IEEE Access*, 10, 47792-47804.
- [25] S. Poudyal and D. Dasgupta, "AI-Powered Ransomware Detection Framework," 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 2020, pp. 1154-1161, doi: 10.1109/SSCI47803.2020.9308387.
- [26] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2017, July). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security* (pp. 1-7).
- [27] Venkatraman, S., & Alazab, M. (2018). Use of data visualisation for zero-day malware detection. *Security and Communication Networks*, 2018, 1-13.
- [28] Moussas, V., & Andreatos, A. (2021). Malware detection based on code visualization and two-level classification. *Information*, 12(3), 118.
- [29] Son, T. T., Lee, C., Le-Minh, H., Aslam, N., Raza, M., & Long, N. Q. (2020). An evaluation of image-based malware classification using machine learning. In *Advances in Computational Collective Intelligence: 12th International Conference, ICCCI 2020, Da Nang, Vietnam, November 30–December 3, 2020, Proceedings 12* (pp. 125-138). Springer International Publishing.
- [30] Seok, S., & Kim, H. (2019). Visualized malware classification based-on convolutional neural network. *Journal of The Korea Institute of Information Security & Cryptology*, 26(1), 197-208.