Malware Classification using Dynamic Analysis with Deep Learning

Asad Amin, Muhammad Nauman Durrani, Nadeem Kafi, Fahad Samad, Abdul Aziz
k173066@nu.edu.pk , muhammad.nouman@nu.edu.pk , nadeem.kafi@nu.edu.pk , fahad.samad@nu.edu.pk, abdulaziz@nu.edu.pk
FAST-National University of Computer and Emerging Sciences, Pakistan

Abstract
There has been a rapid increase in the creation and alteration of new malware samples which is a huge financial risk for many organizations. There is a huge demand for improvement in classification and detection mechanisms available today, as some of the old strategies like classification using mac learning algorithms were proved to be useful but cannot perform well in the scalable auto feature extraction scenario. To overcome this there must be a mechanism to automatically analyze malware based on the automatic feature extraction process. For this purpose, the dynamic analysis of real malware executable files has been done to extract useful features like API call sequence and opcode sequence. The use of different hashing techniques has been analyzed to further generate images and convert them into image representable form which will allow us to use more advanced classification approaches to classify huge amounts of images using deep learning approaches. The use of deep learning algorithms like convolutional neural networks enables the classification of malware by converting it into images. These images when fed into the CNN after being converted into the grayscale image will perform comparatively well in case of dynamic changes in malware code as image samples will be changed by few pixels when classified based on a greyscale image. In this work, we used VGG-16 architecture of CNN for experimentation.

Keywords:
Malware Classification, SimHash, Hashing, Opcodes, Deep Learning Models, CNN, VGG 16.

1. Introduction

Malicious or malware software is a major problem in modern-day computing devices, especially containing the data. Many anti-malware (anti-virus) softwares detect and delete malwares from the computer but the malwares nowadays are programmed in such a way that it can mutate itself to attack again without being detected by the anti-malware software. The creation of new types of malware on daily basis is affecting many companies and organizations financially and also affecting many individual user's security which was also reported by McAfee's annual report [1]. However, there are reverse engineering procedures like signature-based or heuristics-based detection and analysis but the creation of different unique versions of the same malware using different polymorphic and metamorphic algorithms [2] make it difficult to automate the reverse engineering procedures using these techniques.

Malware is one of the major challenges to Internet security. The report from Symantec in 2016 claimed that more than 430 million unique malware were discovered in 2015[3]. Today, malware is increasing with many different patterns and new families for different malign purposes. The purpose of malware is to interrupt the flow of normal operations, gather sensitive data or information. Many types of malware are divided according to their functionality and purpose. Some of the types are backdoor worms, Spyware, Adware, Rootkit, Trojan-horse, etc. The history of malware starts from the 1970s when the malware was used to spread offline using Floppy Disks and other external devices. As the age of networks and the internet matured in the late 90s, the malware authors used this platform to quickly transfer malware from one computer to many different systems. The malware types varied from email worms to rootkit and SQL injections became one of the popular types to infiltrate and take advantage of the lack of security protocols in many websites. After 2010, there is a significant evolution in the sophistication and power of malware which is being developed by authors to bypass many anti-malware systems and also used to attack many government institutes and the type of malware grew rapidly with the growth in ransomware and illegal schemes. Today, many unique malwares are getting created at a rapid speed using techniques that allow malwares to be not detected by classic signature-based or heuristic-based techniques [4].

Different detection techniques were introduced, which include signature-based malware detection. These techniques try to match the unique signature of malware, and also behavioral/pattern-based malware detection [5]. This requires the malware to be detected in a runtime environment or virtual environment that is more time and resource-consuming. Also, these approaches were later exploited by malware makers by making malicious software more dynamic and unique each time with new attacks which makes it impossible for these methods to detect malware.

With the introduction of machine learning approaches [10] in many domains, malware detection also benefits by quickly detecting using fewer resources than previous techniques. The different algorithms that are used for this purpose require efficient feature extraction techniques to be used based on the classification algorithm used for the problem. Malware authors have now developed different techniques to bypass anti-malware software like packing, encryption, or obfuscation techniques [6]. As obfuscation techniques became more popular, the dynamic analysis of

Manuscript received August 5, 2023
Manuscript revised August 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.8.7

malware became necessary to get rid of this type of malware. As dynamic analysis executes malware in a sandbox virtual environment to get traces of its behavior. It allows the classification model to detect malware based on its behavior similarity between new and known ones. Now to circumvent this advancement the malware author inserts meaningless codes or tries to shuffle the sequence of programs [8]. To get rid of these changes to create new variants of the same malware family, the use of malware in the form of grayscale images [9] can be used. As malware code changes small parts of the original source code to produce a new variant of malware, images can capture small changes yet retain the global structure. Hence, using images we can detect samples belonging to the same family because they appear very similar to the captured image and the image structure.

Previously for the new changes in malware creation, the machine learning algorithms became popular for malware detection. These machine learning approaches are based on heuristic feature engineering which is expensive and un-scalable [36]. A lot of effort is spent in feature engineering, which requires domain expertise to know the characteristics that make the objects belong to different categories of malware. Also because of constant evolution and changes in patterns of malware using different obfuscation techniques, these machine learning algorithms faced a drift problem due to lack of ability to understand features automatically [10]. Further, these algorithms were unable to maintain good accuracy by increasing the number of datasets and variations. For the domain of malware classification, advanced machine learning techniques based on AI, specifically deep learning. The main reason for using the deep learning approach was the more robust and scalable approach of these techniques, as the architecture of deep learning algorithms are designed in a way that it can easily be extended for the new data because the use of neurons and layers in the deep learning architecture can easily be increased by increasing the number of arguments in the code methods [11].

For this problem, many machine learning algorithms have been utilized for a variety of malware samples. Different researchers have used different techniques of machine learning like Hidden Markov Models were used to model system call sequences [12], Support Vector Machine and random forest trees were used for different feature extraction techniques [13][14]. Several other machine learning algorithms were also used for malware detection but the main problem is that many of these algorithms are based on domain knowledge of malwares and its feature analysis. Many researchers used feature engineering, where features are used to train a machine learning model to classify and make decisions but due to the change in the malware type from the existing types (and hence features), these models are unable to classify the new malware based on the modified features. This results in miss classification. To reduce the inefficiency of feature engineering in a constantly changing feature environment and to extract useful information from raw data and enable models to be able to self-learn new features to improve malware classification is our main focus.

Deep learning models have shown great results with rapid improvements in areas like object recognition, speech recognition [15]. It uses the neural network architecture which consists of many hidden layers to learn new features of the training set. As compared to traditional approaches, we have used Convolutional Neural Network [16] approach which has not been explored by many researchers for malware classification problems. CNN performed well in different problems related to image classification like image recognition. For this purpose, the malware files were converted into greyscale images, later on, used to train the CNN model. The main advantage of classifying malware based on images is that the malwares of the same family are similar in visualization that helps in identifying different variants by comparing few pixel changes of the same family which was difficult to identify using previous approaches.

There is a huge demand for improvement in malware classification and detection mechanisms as malwares now can affect computers without getting identified using different obfuscation techniques. For this purpose, the dynamic malware analysis is used to detect the changed behavior of malware by running the malware inside a virtual sandbox environment [7] called cuckoo. Malware Classification has been a very famous research topic because of many gaps that still exist in classification techniques. After many problems from different domains are being applied on deep learning architecture for better accuracy and scalable solutions, the malware classification problem has also been applied on different CNN architectures, but not much work has been done on more robust and new architectures like VGG-16 [17]. Some of the work was found using malware classification with deep learning algorithms using static analysis [19], because of easier conversion of already prepared binaries into images.

In this research, the dynamic analysis is done not only on already prepared malware files collection but also on generally available files from personal computers to test the model for all types of datasets. The behavior of malware files after getting identified from the sandbox environment in the form of API system calls [20] will be converted into an opcode sequence. This allows us to develop a mechanism that can represent the useful features of malware in a way that it can be represented to automatically analyze small changes. The SimHash hashing [21] techniques used previously have been used to get more weight of important features as the hashing technique is a very important part of the conversion of malware into useful grayscale images.

The paper is organized as follows: Section 2 is the literature review, Section 3 explains the methodology, and

results are presented and discussed in Section 4. Section 5 concludes the paper and specifies the future work.

2. Literature

Much work has been done for malware classification using the classical signature and heuristic-based approaches. Also, some of the machine learning approaches have been used for classification but the latest work is being done using deep learning algorithms to get better results using images.

2.1 Graph-Based Approaches

T. Wuchner et al. [22] introduced a compression-based graph mining technique for dynamic malware detection which scans unknown graphs for characteristics of malware behavior patterns and compares it with the repository of already labeled malign patterns. The approach followed by the author is to first retrieve data under the sandbox environment by identifying the system call traces for each program sample and convert that system calls into a quantitative data flow graph. After data retrieval, the subgraphs are extracted from QDFGs to identify the pattern which is then stored in a repository to capture the basic pattern of known malwares. Now to train the classifier the author used the mined patterns on the training set and recorded which sub-pattern belongs to malware and benign software respectively. This helps classifiers understand different relationships of mined patterns. Now to identify the new malware- whether it is malign or benign the program sample has to through the same step to generates subgraphs (pattern), after that the QDFGs of this unknown sample is matched with previously generated detection patterns which are then converted into a feature vector using the same process used for generating training feature. Finally, this test feature vector is then passed to the classifier which results in the classification of the sample test program as malware or benign software.

2.2 API call Analysis Approach

Youngjoon Ki et al [23] worked on dynamic analysis techniques using the API call analysis. For this purpose, the Detours hooking library was used to trace the API call sequence of malware under a virtual environment which is a hooking library used to intercept the call sequence to target function and provide a user to insert a detour function to analyze the API calls between the start and completion of the program. The tracing of API calls is done using 23,080 malware samples. For creating the signature of API call sequence all API call sequences which are labeled as malware are extracted and stored in the database along with multiple sequence alignment which helps in finding out the longest common sequence. After that in the dynamic analysis process, the call sequence of a randomly generated program is compared with the signature and longest common sequence of malware program APIs which were stored in the database.

Alazab et al. [50] provide a major improvement in detecting zero-day malware by improving the efficiency of previous zero-day classification techniques. The author classified zero-day malware with high levels of accuracy and efficiency based on the frequency of Windows API calls after disassembling the malware file which is then stored into a signature database after passing through similarity measures to generate similarity reports. After that, the mutual information-based maximum relevance filter is applied to get the ranking of API functions to get relevant features before being given as input to the training classifier. Using Naive Bayes and KNN, they got 98.5 % accuracy with less than 0.25 FPR.

Iwamoto et al. [51] worked on a simple classification methodology to compare the pairs of consecutive API calls which are maintained using a graph data structure. They are then compared with the executable API calls, already identified as malware to determine the similarity using the dice coefficient. The similarity matrix produced by the analysis system considers many features using modules like control flow analyzer, API call extractor, etc. To analyze the similarity factor of the malware files the hierarchical cluster analysis is done to do visualization using similarity values.

2.3 Machine Learning Approaches

Schultz et al. [24] mainly used some static features like PR head, sequence of string, and sequence of bytes. Also, different methods like signature-based were used to differentiate malicious executables from one another. A ripper method [25] was used as a rule creation algorithm and Naive Bayes and Muti Naive Bayes were used as a learning algorithm that helped in obtaining classification accuracy of 97.11%. These data mining methods at that time doubled the decision rates for new malicious executables.

A much better result was then achieved by Kolter and Maloof et al. [26] by using a byte code of N-gram as a feature. Different types of machine learning algorithms like SVM, Decision Trees algorithms, and boosting were used with features selected as the most important n-grams for classification. Among all the methods the decision tree gave better results with a ROC curve of 0.996.

A new method was introduced to detect the malware which is unknown or not based on the signature-based method which was used previously [4][7][39]. This method was based on the frequency of operational codes which was introduced along with classification algorithms like KNN,

Bayesian Network, SVM and it gives the 92.2% frequency with one opcode and 95.90% frequency with two opcode sequence lengths. This method was a major step towards the dynamic identification of malware, which mostly occurs using different code obfuscation techniques.

Shabtai et al. [24] introduced a new feature extraction technique called OpCode n-gram patterns. The opcode sequence was obtained after disassembling process of malware file and analyzing it which methods like Document frequency, G-mean, using this technique many learning algorithms like SVM, logistic regression, naive Bayes, random forest was used but the most frequency of 95.14% was obtained using Random Forest [14].

The major advancement in malware classification came after the Microsoft Kaggle challenge with a dataset of malwares in hex and assembly code format. For this format of malware, many features were used which were introduced by Ahmedi et al. [28] including image representation and entropy methods. These features are then combined using feature fusion techniques which combine all the feature categories sequentially in a single feature vector to be able to run a classifier on them which then gives the accuracy of 99.8% using the XGBoost classification algorithm. To overcome the difficulty of correct feature extraction the deep learning algorithm performed this task with more efficiency without analyzing the data available before running classification tasks. Following is the research work on malware classification carried out using deep learning algorithms.

2.4 Deep Learning Approaches

The first step towards a deep learning model for malware classification was taken by Nataraj et al. [29]. The authors came up with the approach of visualizing and classifying greyscale images using image processing techniques. The classification was made based on the observation that images from the same malware family have the same type of texture and sequence of pixels in images. In this research, they represented images in the range of 0-255 (black-white) and observed that the obtained image presented different sections of information about malware. The main breakthrough of this research was that there was no need for disassembly or code execution for classification and image features were computed using GIST descriptor and KNN for classification.

A deep feed-forward neural network was presented by Saxe and Berlin [30]. The dataset for classification was binary which contains malware samples as benign or malicious. The features used were mostly static-like system library imports, ASCII format strings, metadata of the executable, and bytes sequence from raw code. These four features are then aggregated to produce a single 1024 dimensional feature vector. After making a consolidated feature vector, the authors used the virus total online analysis portal to analyze the files and so the labeling which is malware. The experiments performed by the authors were performed on one of the largest datasets of 400,000 software binaries with a detection rate of 95% and FPR of 0.1.

Tobiyama et al. [31] use process behavior as an API call sequence which represents the operations of the process. Features of process behavior are extracted to be used to train the RNN model. After training using RNN the features got transformed into feature vectors which then fed into CNN after transforming malware into images which gives the result of around 0.96 AUC score.

George E. Dahl [32] worked on the malware classification with the main objective to minimize a large number of potential features. For this purpose different feature selection techniques like random projections can feed large data to the neural network system. The proposed solution achieved classification results with an error rate from 0.42%-0.49% in which an ensemble of neural networks gives a smaller error rate.

Deep learning algorithms achieve greater accuracy for classification because of a higher number of diverse layers. Kolosnjaji [33] then achieved greater improvement in performance by modeling system calls sequence using the architecture with two combined computations of CNN and recurrent neural network. The result of using this architecture was that convolution of n-gram with full sequential modeling was achieved with hierarchical feature extraction. Using this architecture, 85.6% accuracy was achieved.

In another advancement towards deep learning malware classification, Wenyi Huang [34] proposed the deep learning multi-task architecture for binary malware classification. The models were trained from data after dynamic analysis of malicious and benign files. The improvement was achieved on a very large dataset of around 4.5 million files which is one of the largest studies on malware classification. They also worked on the malware family classification architecture and then combined it with binary classification architecture to build multi-task architecture.

Agarwal et al. [35] proposed the solution of malware classification using deep learning techniques which starts from parameter pre-processing in which the parameters associated with API calls are a very important aspect of malware behavior. Since the malware parameters can be of any format and values are also not specific. For this purpose, the author created the algorithm in which each malware after disassembling evaluates a tokenized sequence of characters which are then built into an n-gram sequence

based on characters and also frequency distribution These parameters helps the author to gain important information. This use of parameters was used in sequential models to evaluate vertical and horizontal relationships after deriving parameters embedding. After all these, the classification technique was the linear model with multi-layer architecture which contained the hidden dense RELu layers followed by the sigmoid layer. The experiment was performed on 75,000 files which were evenly distributed.

2.5 Dynamic Analysis Approaches

Galal et al. [36] proposed a dynamic analysis method for the classification of malware. It uses a heuristic function that selects the sequence of API calls based on the API category created by the author, to create action traces. These action traces are converted into feature sets by extracting them into a benign and malign dataset which are then accumulated into a global action list. After this, every sample program is represented into a binary vector such that binary feature f is set to 1 if the sample has done any action otherwise 0. The dataset used in this work has 2000 samples from 50 different families.

Pascanu [37] proposed a recurrent neural network solution to dynamically classify malware using a deep learning approach. Using this approach the author proposed an approach that was similar to natural language modeling using Echo State Network and Recurrent Neural Network to extract the features. After the projection stage using these RNN and ESN the max-pooling layer is also introduced to increase invariance and half-frame which increases the memory capacity of our final sequence representation. The dataset with around 297500 and 150000 samples was used for training and testing respectively which achieved an AUC of 95%.

Yuan et al. [38] used a hybrid approach for malware analysis and applied a deep learning algorithm for the classification of malware. The author used the static analysis to extract sensitive API and required permissions by uncompressing the .apk file and parsing AndroidManifest .xml and classes.dex file. The AndroidManifest.xml file was used to get all the android permission and class.dex file was parsed to extract which API function is getting called. In the dynamic phase, the author installed and ran the application in DroidBox which is an android application sandbox that extends the TaintDroid hooking system. Using this sandboxing process the authors were able to analyze 13 different app functions and were able to obtain a total of 192 different features for each app. After generating features using static and dynamic analysis the author made an android deep-learning-based method (DroidDetector) which then gives the classification result for the app using an in-depth examination of the application along with additional information obtained from the analysis phase. Droid Detector deep learning methods were able to accomplish 96.7% classification accuracy.

Chen et al. [39] uses dynamic analysis to detect ransomware as compared to static signature-based analysis which can easily be avoided using different obfuscation techniques. The author used the dynamic analysis technique with different data mining techniques like Random Forest, SVM, Naive Bayes, etc. The proposed architecture consists of the dynamic analysis part using API calls sequences which are monitored using software called API Monitor which analyzes the working of the application using API calls sequence and generates a call flow graph (CFG) to show the program flow and behavior. After feature extraction, the data normalization and feature selection are done using Correlation and Gain Ratio to improve the performance of the classifier by reducing the dimensionality and selecting the relevant features for model creation. For classification, different learning algorithms like SVM, Naive Bayes, Random Forest, and Simple Logistic with k-fold cross-validation were used on 168 different ransomware samples using original and normalized data. After the experiment, the results provided by the author show better performance using a simple logistic algorithm with normalized data with 97.6% accuracy with the lowest false positive rate as compared to other classifiers.

Alsulami et al. [40] introduced dynamic malware detection using the deep learning technique by generating a pre-fetch file of each malware which is a summary of the behavior of the windows-based application. For this experiment around 100000 malware samples were obtained from the malware repository which was then installed and executed into the virtual environment of windows 10 to collect the pre-fetch file of each malware sample.

Kolosnjaji et al. [33] classified malware using dynamic analysis and by constructing a neural network based on a combination of the convolutional and recurrent networks to obtain the best features. The process of classification starts with the extraction of calls sequence of malware programs which are fetched by running the PE files in a virtual environment.

Table 1: Advantage and Disadvantages of Dynamic Malware Analysis

Advantages	***Disadvantages***
Detecting new types of malware attacks	Storage complexity for behavioral patterns
Dependency detector	Time Complexity
Detecting the polymorphic malwares	Detected after the malware has infected the victim machine.

The author used the one-hot encoding mechanism to create a feature vector of length equal to the number of distinct API calls. After preprocessing the author forwarded

the API calls vector into the proposed neural network in which a convolutional neural network is used for feature extraction by capturing the correlation between neighboring input vectors which are then fed into recurrent neural network layer explicitly model the sequential dependencies from the API traces. The layers are then followed by a mean pooling layer to extract the highly important features and then drop out a softmax layer is used to prevent overfitting and get label probabilities. The performance of this model was significant as compared to previous hybrid machine learning models with an overall accuracy of 95%.

Xiao et al. [41] proposed the solution to dynamically identify malware in the IoT environment using behavior graph construction and SAE-based malware detection (deep learning method). In the proposed Solution the cloud platform transforms the program into a behavior graph using API calls that are then converted into binary vectors to be given as input to the SAE-based malware detection model. The cloud platform after collecting runtime activities creates a behavior graph or executes sample activities inside the cuckoo sandbox to extract API calls for malicious files which are categorized based on some predefined API calls. After the creation of a behavior graph from API calls the SAE model transforms it into a binary vector that uses one-hot encoding to identify unique behavior for every API call graph. The SAE model's main goal is to reduce the number of features and describe the features in compact high-level expression. The author experimented on 1760 samples in which the ratio of benign and malware samples was 50%. The experiment was conducted using different SAE-based deep learning algorithms and basic machine learning algorithms also but the SAE-based algorithm performed relatively well with the SAE-Decision tree algorithm achieving the precision of 0.98 percent which is 1.5% better than the average precision of malware detection.

2.6 Gaps found and Contributions

The methods discussed above can be divided into classical and modern approaches. The classical approaches include graph-based and API call-based approaches which were compared and classified using signature-based approaches. These approaches were groundbreaking in the domain of malware detection and even getting used currently in new approaches. But the main drawback of using these classification approaches was the pattern creation and comparison complexity which impacts the computational efforts that were needed to perform these classical approaches. Keeping the computational drawback in mind, machine learning approaches were introduced basically to improve the performance of classification with algorithms like naive Bayes, SVM, decision tree, etc which gives robustness and scalability to classical approaches and identify unknown nor new malware easily. But the process of feature extraction in growing malware families was difficult. Due to new obfuscation and encryption techniques used by malware authors, it was very difficult to analyze new features automatically from files.

To overcome this problem, new advanced convolutional neural network (CNN) based approaches were used which were mainly used to evaluate new features in scalable problems to identify new features in scalable problems for the identification of new malwares. These approaches include CNN architectures like SAE, RNN, etc. which were used for classification using multiple deep learning layers. The work done using CNN techniques was mostly using a static approach, as can be seen in Table 2, but due to increasing obfuscation and encryption/hiding techniques used by the author at runtime even deep learning approaches cannot correctly classify malware without a dynamic approach.

In this research, we have used advanced CNN architecture like VGG-16, which are rarely used for malware classification problems but have got many recommendations by performing incredibly well with datasets of other domains.

The other contribution of this paper is the use of classical approaches like opcode hashing for feature extraction and representation with modern approaches like deep learning in a dynamic environment. The combination of opcode sequence with image creation can perform well as images can detect small changes of code. The other work done in this research is the use of SimHash [21] technique for the creation of similarity comparison vectors which can improve similarity between hash vectors and the pattern to be created using grayscale images.

The use of dynamically extracting features and using hashing techniques with images creation has been used for the first time and advanced new CNN architectures like VGG-16 which can perform quick robust classification tasks with high precision. Following are the main contributions of this paper.

3. Methodology

One of the major issues is to select the best opcode pairing to get features for classification. For this purpose, two different opcode generation methods will be used which will be compared later based on accuracy and relevancy in model classification. One of the opcode generation techniques is to create a series of opcodes and another one is to create a pair of two opcodes and then create opcode vectors for other processing for images.

The malware represented in formats like vector and file for classification is useful in case of static analysis of

malware. To get a more detailed analysis of malware which is created in a way to change its representation during execution to be able to bypass detection mechanism, an image representation can help in identifying little changes in the malware and little pattern mismatch from different trained families of malware.

The use of the deep learning approach is better to overcome the problem of the creation of different malware created daily as in deep learning architecture if the dataset increases the classifier performance typically keeps on increasing. In comparison, in the machine learning approach performance of the model stops improving with the increasing size of data. In case of any increase in the size of training data which causes the performance of deep learning networks to plateau, we can also increase the performance by increasing the capacity of the neural network by increasing the number of neurons or layers and repeating the continuously improving cycle.

Table 2: Comparison of different work done with different Analysis, Feature extraction, and classification techniques

Research work	***Analysis***	***Features Extraction***	***Classification***	***Datasets***
Nataraj et al. [29]	Static	Greyscale Image	KNN	Large scale malware from VxHeaven and different sources
Berlin et al. [30]	Static	PE import features	DNN	Samples from VirusTotal and private Repos
Tobiyama et al. [31]	Static	API call sequence	CNN	Samples from different sources
Dahl et al. [32]	Static	API call sequence	DNN	Samples from different sources
Kolosnjaji et al. [33]	Dynamic	API call sequence	CNN + RNN	Samples from Virus Share, Maltrieve and private collections.
Huang et al. [34]	Dynamic	API call sequence	DNN	Sample from Microsoft.
Agarwal et al. [35]	Static	Opcode frequency	ML + DNN	Samples from Virustotal and Malicia-Project
Alsulami et al. [40]	Dynamic	Metadata	RNN	Samples from VirusShare
Xaio et al. [41]	Dynamic	API call sequence	SAE model	1760 samples files from Vx Heaven
Kang et al. [39]	Dynamic	API call sequence	ML	Samples from private collection
Pascanu et al. [37]	Static + Dynamic	API call sequence	RNN	Samples from private collection

The other main problem due to the ever-increasing size and variation of malware format is to extract features that can be handled using our approach of using CNN architecture, in which different layers of the neural network are used to automatically learn features at different levels.

3.1. Feature Extraction:

3.1.1. Dynamic Disassembling

Dynamic analysis is performed in a safe and virtual environment to observe the action and behavior performed by the file. To tackle the problem of obfuscation and packing of malware by malware authors the dynamic analysis is necessary to get analysis of malware while being executed. To get the analysis while running malware in a controlled environment cuckoo sandbox [42] will be used to disassemble malware and get opcode sequence.

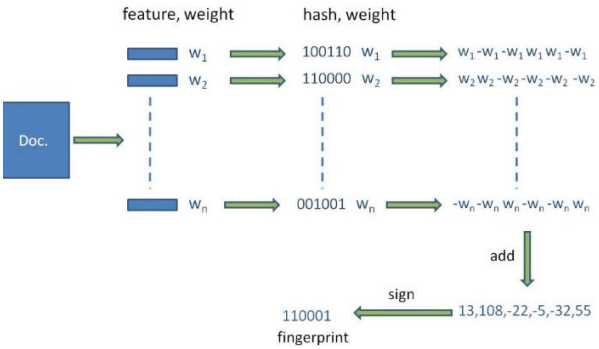
The process to perform this consists of three steps which are to disinfect the environment, then execute the malicious file, and then monitor logs and operation traces. To analyze the behavior as it would run on any computer the sandbox-based tool Cuckoo is used by analyzing it automatically in a virtual environment. The PE (Program Executable) [43] files inserted for analysis can be monitored by analyzing function call monitoring or information flow tracking. These API functions can be divided into 6 different types based on the functionality which they can affect like network management, registry operations, 10 file operation, thread and processing, memory management, etc. To get access to the system resources such as file system, processes, windows registry, etc. the malicious files use API functions. To analyze the behavior, patterns and data flow from a malicious file, it is necessary to extract and analyze the results and inputs of API functions. By analyzing the malicious file dynamically, the API-call sequence can be traced from API functions which give us information about each step in which the file is affecting the system.

The PE files which are used as input for malware classification contain several different headers and section parts. For disassembling the cuckoo sandbox environment will be used to trace API calls and find the behavior of the file. This cuckoo-based sandbox environment is useful to identify different malware which can change its patterns using different obfuscation techniques. The PE file which will be inserted into the sandbox environment consists of header and body sections of the file which is used to map the file into memory. During dynamic analysis using a sandbox environment, the OllyDbg v2 debugger will be used in a virtual machine to trace assembly calls to get the

opcode sequence at run time. This process gives a code section for each file to get the details of features that explain different operations that a PE file can perform if executed on the system.

3.1.2 Feature Extraction

After disassembling the next step is to extract opcode sequences which are like push, mov, lea, call, etc. These opcode sequences can be in a 1-tuple or 2-tuple opcode sequence [44]. One of the major issues is to select the best



opcode pairing to get features for classification. For this purpose, 2 different opcode generation methods will be used which will be compared later based on accuracy and relevancy in model classification. One of the opcode generation techniques is to create a series of opcodes and another one is to create a pair of two opcodes and then create opcode vectors for other processing for images.

Zhang et al. [45] using the IRMD algorithm generated the opcodes in a single sequence like push, lea, push, mov, call which gives more importance to all the features. Other techniques like 2-tuple opcode sequences <mov,mov> and <mov, call> are also used in the experiment to get the results and see the difference in images created using these sequences.

After creating the opcode sequence the next step is to convert these opcodes into some meaningful values which can help in creating malware images. For this purpose, a new algorithm is used which creates a hash value of a given opcode sequence. This algorithm helps in comparing sequence similarity which is used to make sure that there is no collision between hash values.

3.1.3 Binary Hash Conversion

After creating an opcode sequence from one of the techniques, the next step is to generate a hash of these opcodes sequences. The hashing function used in the experiments is a variation of different sizes and types of Secure hashing algorithm (SHA)[46] which is used to generate secure data using a cryptographic technique by converting data that is given as input to fix the size string of numbers and alphabets using different operations like bitwise, modular addition and compression, etc. The different algorithms used in experiments are SHA-256, SHA-512[47], SHA-768, SHA-896, and SHA-1024 [48]. The difference between each encryption technique is the stronger encryption methods and the size of hash value generated increase as the number of SHA functions increases.

One of the main reasons to use an SHA hash function is the ability to create totally different hash values even a slight change in the input value. In our experiments, the sequence of machine operation codes (opcodes) are created. The next step is to create the hash vector using one of the similarity comparison algorithms like SimHash [21]. The main reason for using a hashing algorithm to generate a hash vector is to make the comparison between repeated patterns of malware or similar malware because SimHash creates the same hash of similar malware.

The algorithm which is used to create a hash of the malware files takes as input the series of opcode values and converts it into a binary vector to represent the file. The SimHash algorithm starts by converting all the opcodes into hash values. Based on these hash values weight vectors are generated for each hash code which is then added up to cumulative vectors which again get converted into binary vectors for the generation of grayscale images.

The process of converting the opcode into a binary vector as defined in Fig. 1 including the first step in which the opcode sequences will be assigned to variables w which will be converted into n-bit binary hash values using the hash function. Now for each bit of hash value, if the bit is 1 the value of the corresponding bit of SimHash is increased by 1 otherwise the value is decreased by 1. Now for converting the SimHash vector into a binary format, we have used the normalization technique and will set the value to 1 if the bit is greater than 1 or else set it to 0.

Fig 1: Algorithm process of SimHash

3.2 Malware Image Generation

After converting each malware sample into a hash value with equal length. The next step is to convert these hash values to images by converting each hash value into a pixel value. If the hash bit value is equal to 0 then the pixel value will be 0 and if the hash bit value is 1 then the pixel value will be 255. This conversion enables greyscale malware images [49] to be generated using which we can differentiate between different malware of different families as shown in Fig [figure number], a malware has different patterns between different families of malware.

As shown in the binary array has been given as a parameter to function which will be converted into a proper NumPy array for further manipulations and changes using the NumPy library. After this, the array elements with the value of 1 will be replaced by 255 and the remaining elements will be 0 which is necessary for creating grayscale images.

After changing the element values in the array the array will get reshaped to create a proper image that can be processed by the CNN model and patterns inside the image can also be observed. According to the size of the array which was 1600 because of using SHA-896 which produces hashes of length approximately 1600, the array will be reshaped into 40*40 which will create more sub-arrays due to which the image can be easily created depends upon our preferred size of the image. Here the standard malware images size of width=250 and height=250 is used which was used in many previous types of research using malware images [9].

3.3 Convolutional Neural Network

The use of the deep learning approach is better to overcome the problem of the creation of different malware created daily as in deep learning architecture if the dataset increases the classifier performance typically keeps on increasing. In comparison, in the machine learning approach performance of the model stops improving with the increasing size of data. In case of any increase in the size of training data which causes the performance of deep learning networks to plateau, we can also increase the performance by increasing the capacity of the neural network by increasing the number of neurons or layers and repeat the continuously improving cycle.

The other main problem due to the ever-increasing size and variation of malware format is to extract features that can be handled using our approach of using CNN architecture, in which different layers of the neural network are used to automatically learn features at different levels. The architecture that we have used has rarely been used in the domain of malware classification. Following is the CNN architecture used during experimentation.

The architecture of our Convolutional Neural Network model is based on VGG-16[17] which is considered as one of the best performing architecture on ImageNet dataset competitions. The model is trained on more than a million images from the ImageNet database. In convolutional neural networks the three main layers which are used to make any architecture are fully connected, convolutional and pooling layers. The full overview of VGG-16 CNN architecture can be seen in figure.2.

This network is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully connected layers, each with 4,096 nodes are then followed by a softmax classifier.

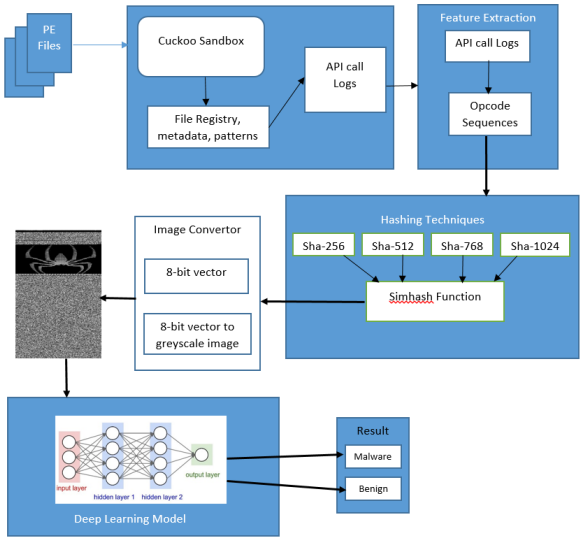


Fig 2: Detailed view of disassembly, feature extraction and image creation process

3.4 Proposed Solution for Malware Classification:

The architecture of our proposed method is divided into 5 parts, as can be seen in Fig 2. The classification process starts from dynamic analysis using a cuckoo sandbox environment which will generate API call logs after running malware or benign files inside a virtual machine. After that, the opcode sequence is generated in a feature extraction module which is then converted into a unique value generated by different hash techniques applied on every word of the opcode. After generating hash for all values the SimHash function is applied which is used for generating similarity matrices which are then divided into right vector format for grayscale image generation which is then given as input to CNN algorithm for classification.

3.5 Key Requirements

3.5.1 Dataset

The other samples used in this research were taken from VirusShare.com which contains a repository of malware PE files and also some windows PE files were used from our personal computers for experimentations. The other sources of the dataset were the benign executable (.exe) files of software setups taken from filehippo.com and various software downloading websites. The dataset consists of 500 PE executable files which contain malwares of 5 different families which include Locker, Mediyes, Zbot, WinWebsec, and Zeroaccess.

The malware was downloaded from the virus total after requesting the author to use their virus-infected executable files for research purposes which were given after some validation.

3.5.2 Software Requirements.

The malware disassembly has been done using

- Cuckoo sandbox
- PyCharm IDE
- Python 2.7, 3.7
- Virtual Box

Software and the development of hash algorithms have been done using Python 3.7 on PyCharm IDE.

The analysis using sandbox was done only on the Ubuntu 18.0 operating system with Windows 10 used as OS in virtual machines. The development of deep learning algorithms has also been done using python with additional Keras and Tensorflow libraries.

3.5.3 Hardware Requirements

During the research, many different hardware resources have been used for powerful training computation of deep learning algorithms on large datasets. For this purpose university computers with hardware specifications of Core i7 processor with 16GB RAM and 1070 T Nvidia GPU were used. Online cloud platforms like google colab, Google cloud were also used for training deep learning models.

3.6 Evaluation Matrix

The evaluation matrix used in the experiments is first to find the accuracy of the model after applying it with one of 5 different hashing techniques and also to find accuracy after applying 1-gram, 2-gram, and 3-gram opcode sequence generation. Along with accuracy, the validation accuracy was also evaluated which tells us about how much accurate result our model will be able to give in case if new data is evaluated using the put model.

The other evaluation factor which is used is log loss which gives us the factor of the uncertainty of our model based on predictions and tells us how much it will deviate from the actual labels. The same will be applied to the validation data and log loss will be evaluated for the validation set also.

The other main evaluation matrix is the graph containing training and validation accuracy and loss sync graph which will show whether the model is not overfitting if the graph of training and validation accuracy keeps on increasing with sync readings and keeps on decreasing with sync readings.

1. Results and Discussion

For the conversion of opcode sequence into a binary form, the hashing technique was used in which several hashing functions were used like Sha256, Sha512, Sha768, Sha896, Sha1024. These hash functions created the difference in image creation as greater hash functions produced a greater length of binary number for image creation. As a result, the images created using greater hash functions were more complex and contained more patterns as compared to lower hash functions as can be seen in Fig 3.

Fig 3: Malware greyscale image samples using SHA-256, SHA-896 and SHA-1024 hashing

4.1 Results with different Hash Functions

The images created by different hash functions also affected the accuracy and log loss when applied to the Convolutional neural network model as can be seen in Table 3a. As can be seen that the training accuracy increases as

the hashing algorithm of greater length are used. But the best result was given by the SHA-768 and SHA-1024 hashing functions.

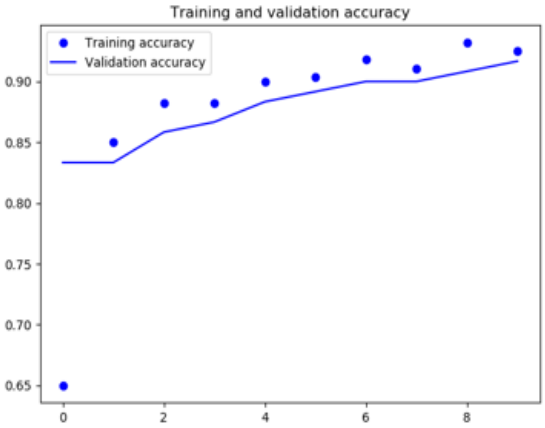
Also, it can be seen from Fig 3 that greyscale images created after opcode hashing using these hashing function is different and as the length of output from hashing function increases with an increasing number of hash functions like SHA-1024 and SHA-896, the greyscale images generated are more complex and contain more patterns as compared to the lower length of hash functions.

Table 3a: Results using different hash functions

Hash Functions	*Accuracy*	*Validation Accuracy*	*Loss*	*Validation Loss*
Sha-256	0.9438	0.8625	0.1333	0.1945
Sha-768	0.9607	0.9167	0.0887	0.1886
Sha-896	0.9536	0.9167	0.1191	0.1737
Sha-1024	0.9571	0.93338	0.1342	0.1699

4.2 Results with different Opcode Sequence

The difference in the accuracy of models was also observed due to the use of different opcode sequence methods. In this research, the 1-gram, 2-gram, and 3-gram opcode sequence methods were used and the difference in model performance shown in Table 3b**.** As can be seen, the model performed relatively well on a 3-gram opcode sequence.



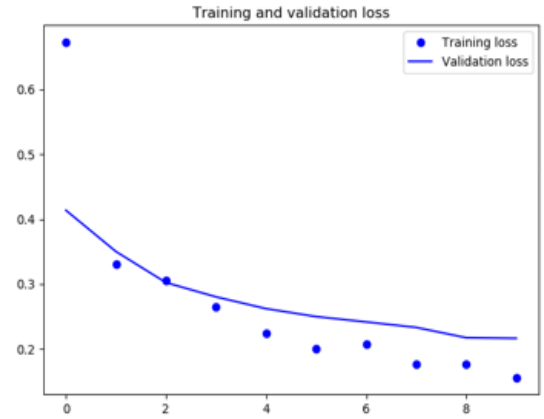


Figure 4: Training and validation a

Table 3b: Results using different opcode sequences

Opcode Sequence	*Accuracy*	*Validation-Accuracy*	*Loss*	*Validation-Loss*

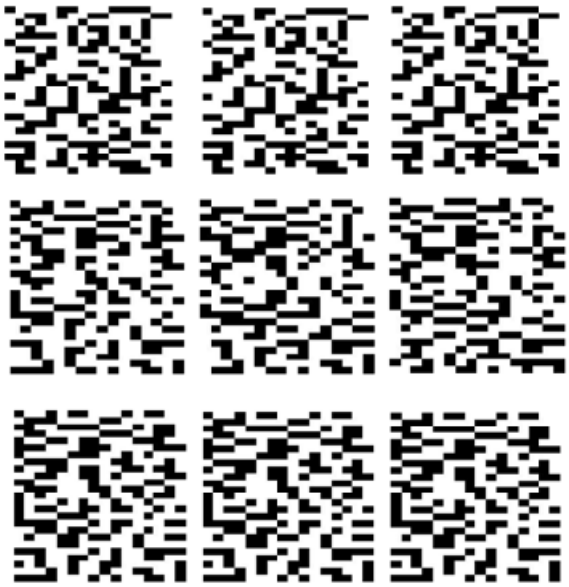


Fig 5: Similarity in pattern of same families

2-gram	0.9429	0.9000	0.14	0.2123
3-gram	0.9679	0.9250	0.09	0.2076

In Fig 4, we can see the training loss and accuracy using the SHA-256 hashing function is in sync with validation loss and accuracy. The validation loss and accuracy lines are not linear but it shows that the model is not overfitting because the validation loss is decreasing and not increasing and also accuracy is increasing for both validation and training data. Also, there is not much gap between training and validation accuracy.

It is due to this similarity as can be seen in Fig. 5 and the difference between different malware family patterns that malware greyscale image can give much better results because obfuscation or cryptographic techniques to change the behavior of software will result in the creation of similar malware image with only few pixel changes.

4.3 Results Using Hash Functions with opcode sequences

i. 2-gram

The images created by different hash functions also affected the accuracy and log loss when applied to the Convolutional neural network model as can be seen in the Table 4. As can be seen that the training accuracy increases when the hashing algorithms with greater length are used. But the best result was given by the SHA-768 and SHA-1024 hashing functions.

ii. 3-gram

Also, the result of a 3-gram opcode sequence with different hash functions can be seen in Table 5. According to the result, the CNN model performs best when used with Sha-256 and Sha-768 techniques based on accuracy and loss.

The performance evaluated from 2-gram and 3-gram opcode sequences show us that lower n-gram sequence gives best results with a high number of hashing techniques and greater n-gram sequences give better result with lower hash functions.

iii. VGG-16 Model:

As shown in Table 6, SHA-768 performed better when applied on VGG-16 model for classification with an input size of 150 x 150 (width x height) as compared to other sizes.

Table 4: Results using different hash functions with 2-gram opcode sequence

Hash Function	*Accuracy*	*Validation Accuracy*	*Loss*	*Validation-Loss*
Sha-256	0.9536	0.8833	0.122	0.2435
Sha-768	0.9464	0.9000	0.142	0.2478
Sha-896	0.9286	0.9083	0.136	0.2275
Sha-1024	0.9179	0.8917	0.157	0.1990

Table 5: Results using different hash functions with 3-gram opcode sequence

Hash Function	*Accuracy*	*Validation-Accuracy*	*Loss*	*Validation-Loss*
Sha-256	0.9679	0.9333	0.1021	0.1966
Sha-768	0.9607	0.9250	0.0987	0.1916
Sha-896	0.9357	0.9167	0.1374	0.2311
Sha-1024	0.9429	0.8833	0.1193	0.2747

Table 6: Results using different hash functions with 3-gram opcode sequence

Accuracy	*Loss*	*Validation Accuracy*	*Validation Loss*	*Input Size*
0.9594	0.1483	0.9000	0.1920	150,150
0. 8375	0.4525	0. 8750	0.5179	28,28

2. Limitations

Learning of good robust features is an active area of research in AI. For malware detection, the hope for these techniques is that they will be less prone to the time drift problem. With an increase in the size of a dataset, the size of the deep learning model (CNN) also increases which becomes very complex and computationally very hard for the system to compute malware on CNN techniques. This requires powerful hardware with more GPUs.

As malware analysis technologies become known, malware authors begin to utilize anti-analysis techniques, which include anti-virtualization and anti-debugging, to detect and disturb the analysis [6]. A study has shown that deploying a decoy executable in online dynamic malware sandbox analysis systems for analysis can provide information to the attacker about the analysis system. A new technique called Decoy Sample Injection (DSI), which can be conducted against a public malware analysis system using an Internet-connected sandbox. The technique points out that IP addresses of Internet-connected sandboxes in the public malware sandbox analysis systems which can be disclosed by an attacker who submits a decoy sample designed for this purpose. The disclosed address can then be shared among attackers as a blacklist to detect and disturb the analysis.

3. Future Work

With these feature extraction techniques performing relatively different than other techniques in the greyscale generation, they can be further used with other hashing similarity techniques like cosine similarity, sketching algorithms to check differences with our proposed approach. One advantage of using deep learning is that it can also tell us which portions of a file are malicious by looking at receptive fields of neurons that get activated. This opens the possibility of a malware analyst using deep learning as a tool to assist in the task of classifying difficult cases. This could also lead to finding out the specific purpose or specific data that the malware is intended to carry out from a specific system which can allow the antimalware stakeholders to take necessary security measures and modify antimalware software according to the specific problem which the organization is facing rather than having general standards and rules set to analyze and classify malware.

4. Conclusion

In this paper, we have introduced the malware classification framework with dynamic analysis and using different hashing techniques and opcode sequences to generate varying patterns of malware images with different complexity and got different accuracies in which SHA-768 gave the best performance and 3-gram opcode sequence gave us better results as compared to 1-gram and 2-gram sequences. Also, we applied these images generated by SHA-768 to VGG-16 using which we got the best accuracy of 96% with greater image size.

References:

[1] McAfee LabsTreats Report in June 2017," https://www.mcafee.com/us/resources/reports/rp quarterly threats jun 2017.pdf

[2] Cesare, S., Xiang, Y. and Zhou, W., 2012. Malwise an effective and efficient classification system for packed and polymorphic malware. IEEE Transactions on Computers, 62(6), pp.1193 1206.

[3] Wood, P., Nahorney, B., Chandrasekar, K., Wallace, S. and Haley, K., 2016. Symantec internet security threat report. Symantec Corporation, Tech. Rep., 21.

[4] Griffin, K., Schneider, S., Hu, X. and Chiueh, T.C., 2009, September. Automatic generation of string signatures for malware detection. In International workshop on recent advances in intrusion detection (pp. 101 120). Springer, Berlin, Heidelberg.

[5] Burguera, I., Zurutuza, U. and Nadjm Tehrani, S., 2011, October. Crowdroid: behavior based malware detection system for android. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (pp. 15 26). ACM.

[6] You, I. and Yim, K., 2010, November. Malware obfuscation techniques: A brief survey. In 2010 International conference on broadband, wireless computing, communication and applications (pp. 297 300). IEEE.

[7] Greamo, C. and Ghosh, A., 2011. Sandboxing and virtualization: Modern tools for combating malware. IEEE Security & Privacy, 9(2), pp.79 82.

[8] You, I. and Yim, K., 2010, November. Malware obfuscation techniques: A brief survey. In 2010 International conference on broadband, wireless computing, communication and applications (pp. 297 300). IEEE.

[9] Nataraj, L., Karthikeyan, S., Jacob, G. and Manjunath, B.S., 2011, July. Malware images: visualization and automatic classification. In Proceedings of the 8th international symposium on visualization for cyber security (p. 4). ACM.

[10] Shaid, S.Z.M. and Maarof, M.A., 2014, August. Malware behavior image for malware variant identification. In 2014 International Symposium on Biometrics and Security Technologies (ISBAST) (pp. 238 243). IEEE.
[11] Alazab, Mamoun, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. "Zero day malware detection based on supervised learning algorithms of API call signatures." In Proceedings of the Ninth Australasian Data Mining Conference Volume 121, pp. 171-182. Australian Computer Society, Inc., 2011.
[12] Iwamoto, Kazuki, and Katsumi Wasaki. "Malware classification based on extracted api sequences using static analysis." In Proceedings of the Asian Internet Engineering Conference, pp. 31 38. ACM, 2012.

Asad Amin received the B.S and M.S. degree in Computer Science from FAST-National University of Computer and Emerging Sciences in 2015 and 2020 respectively. During 2015-2020, he worked as a software engineer on projects related to Fintech, Banking and E-Commerce domains in both project and product based software companies. Currently working as a software engineer for Netpace inc.

Nouman M. Durrani is an Assistant Professor in the Department of Computer Science, FAST National University of Computer and Emerging Science, Karachi. He has received his Ph.D. (CS) from FAST NUCES in 2017. He is a member of Systems Research Laboratory and Center for Research in Ubiquitous Computing (CRUC), and Co-PI at the HEC National Center in Big Data and Cloud Computing (NCBC), Smart Video Surveillance Lab FAST-NUCES. His research interests include Heterogeneous Devices Volunteer Computing, Distributed Systems, IoTs, Computer Vision, and Big Data Analytics.

Nadeem Kafi Khan is an Assistant Professor in the Department of Computer Science, FAST National University of Computer and Emerging Sciences, Karachi. He has received his Ph.D (EE) from FAST NUCES in 2020. He is a member of Systems Research Laboratory and Center for Research in Ubiquitous Computing (CRUC). His research interests include Software Defined Networks, Wireless Sensor Networks, Parallel/Distributed Computing, Software Engineering and Crowdsourced Human Computations.

Fahad Samad received his Ph.D. from RWTH Aachen University Germany in 2011. He is working as an Assistant Professor in the FAST School of Computing since 2017. His research interests include Network Analysis through Crowdsourcing, Network Security, Software Defined Networks and Internet of Things. Additionally, he has a number of international publications in different IEEE and ACM conferences and journals.

Abdul Aziz, an academician and researcher serving as an Assistant Professor at the Department of Computer Science, National University of Computer & Emerging Sciences– FAST, Karachi, Pakistan. He has over a decade of experience with academia & industry in the field of Information Technology. His research interests are in the area of emerging trends in Computer Science, Education Management and Disaster Management.