MACHINE LEARNING AND DEEP LEARNING APPROACH FOR MALWARE DETECTION

INNOVATIVE PRODUCT DEVELOPMENT REPORT

Submitted by

Y. Bhanu Lekha	M. Sai Shreya	M. Laxmi Priya
20RH1A6660	20RH1A6634	20RH1A6639

Under the Esteemed Guidance of

Mrs. P. Gayatri

Assistant Professor

in partial fulfillment of the Academic Requirements for the Degree of

BACHELOR OF TECHNOLOGY

CSE-AI & ML

MALLA REDDY ENGINEERING COLLEGE FOR WOMEN

(Autonomous Institution-UGC, Govt. of India)

Accredited by NBA & NAAC with 'A' Grade, UGC, Govt. of India

NIRF Indian Ranking–2018, Accepted by MHRD, Govt. of India

Permanently Affiliated to JNTUH, Approved by AICTE, ISO 9001:2015 Certified Institution AAAA+

Rated by Digital Learning Magazine, AAA+ Rated by Careers 360 Magazine,

6th Rank CSR, Platinum Rated by AICTE-CII Survey, Top 100 Rank band by ARIIA, MHRD, Govt. of India

National Ranking-Top 100 Rank band by Outlook, National Ranking-Top 100 Rank band by Times Ne ws Magazine

Maisammaguda, Dhullapally, Secunderabad, Kompally-500100

May- 2023

MALLA REDDY ENGINEERING COLLEGE FOR WOMEN

(Autonomous Institution-UGC, Govt. of India)

Accredited by NBA & NAAC with 'A' Grade, UGC, Govt. of India

NIRF Indian Ranking–2018, Accepted by MHRD, Govt. of India
Permanently Affiliated to JNTUH, Approved by AICTE, ISO 9001:2015 Certified Institution
AAAA+ Rated by Digital Learning Magazine, AAA+ Rated by Careers 360 Magazine,
6th Rank CSR, Platinum Rated by AICTE-CII Survey, Top 100 Rank band by ARIIA, MHRD, Govt. of India National Ranking-Top 100 Rank band by Outlook, National Ranking-Top 100 Rank band by Times Ne ws Magazine
Maisammaguda, Dhullapally, Secunderabad, Kompally-500100

DEPARTMENT OF CSE - AI &ML

CERTIFICATE

This is to certify that the Innovative Product Development work entitled **Machine Learning and Deep Learning Approach for Malware Detection** is carried out by **Y. Bhanu Lekha (20RH1A6660), M. Sai Shreya(20RH1A6634), M. Laxmi Priya(20RH1A6639)** in partial fulfillment for the award of degree of **BACHELOR OF TECHNOLOGY** in CSE - AI & ML, Jawaharlal Nehru Technological University, Hyderabad during the academic year 2022-2023.

Supervisor's Signature

Mrs. P. Gayatri
Assistant Professor

Head of the Department

Dr. G. Kalpana
Professor

External Examiner

MALLA REDDY ENGINEERING COLLEGE FOR WOMEN

(Autonomous Institution –UGC,Govt. of India)
(Permanently Affiliated to JNTU, Hyderabad, Approved by AICTE - ISO 9001:2015 Certified)
Accredited by NBA & NAAC with 'A' Grade
NIRF India Ranking 2018, Accepted by MHRD, Govt. of India
Maisammaguda, Dhulapally (Post Via Hakimpet), Secunderabad – 500100

Department of CSE –AI & ML

DECLARATION

We hereby declare that the Innovative Product Development entitled **MACHINE LEARNING AND DEEP LEARNING APPROACH FOR MALWARE DETECTION** submitted to Malla Reddy Engineering College For Women affiliated to Jawaharlal Nehru Technological University, Hyderabad (JNTUH) for the award of the Degree of Bachelor of Technology in CSE-AI & ML is a result of original research work done by us. It is further declared that the Innovative Product Development report or any part thereof has not been previously submitted to any University or Institute for the award of Degree.

Y. Bhanu Lekha (20RH1A6660)

M. Sai Shreya(20RH1A6634)

M. Laxmi Priya(20RH1A6639)

ACKNOWLEDGEMENT

We feel ourselves honored and privileged to place our warm salutation to our college **Malla Reddy Engineering College for Women** and **Department of CSE-AI &ML** which gave us the opportunity to have expertise in engineering and profound technical knowledge.

We would like to deeply thank our Honorable Minister of Telangana State **Sri**.**Ch. Malla Reddy Garu,** founder chairman MRGI, the largest cluster of institutions in the state of Telangana for providing us with all the resources in the college to make our project success.

We wish to convey gratitude to our **Principal Dr. Y. Madhavi Latha,** for providing us with the environment and mean to enrich our skills and motivating us in our endeavor and helping us to realize our full potential.

We would like to thank **Prof. A. Radha Rani**, Director of Computer Science and Engineering & Information Technology for encouraging us to take up a project on this subject and motivating us towards the Project Work.

We express our sincere gratitude to **Dr. G. Kalpana, Head of the Department** of CSE–AI&ML for inspiring us to take up a project on this subject and successfully guiding us towards its completion.

We would like to thank our guide **Mrs. P. Gayatri, Assistant Professor of CSE-AI&ML** Dept and all the Faculty members for their valuable guidance and encouragement towards the completion of our project work.

With Regards and Gratitude

Y. Bhanu Lekha(20RH1A6660)

M. Sai Shreya(20RH1A6634)

M. Laxmi Priya(20RH1A6639)

ABSTRACT

Now-a-days to detect cyber-attack are using static and dynamic analysis of request data. Static analysis is based on signature which we will match existing attack signature with new request packet data to identify packet is normal or contains attack signature. Dynamic analysis will use dynamic execution of program to detect malware/attack, but dynamic analysis is time consuming. To overcome from this problem and to increase detection accuracy with old and new malware attacks, we are using machine learning algorithms and evaluating prediction performance of various machine learning algorithms such as Support Vector Machine (SVM), Random Forest, Decision Tree, Naïve Bayes, Logistic Regression, K-Nearest Neighbors and Deep Learning Algorithms such as Convolution Neural Networks (CNN) and LSTM (Long Short-Term Memory). Among those, various models Deep learning CNN resulted in superior performance compared to other models.

To implement this work and to evaluate machine learning algorithms performance this work using binary malware dataset called 'MALIMG'. This dataset contains 25 families of malware and application will convert this binary dataset into gray images to generate train and test models for machine learning algorithms. This algorithm converting binary data to images and then generating model, so they are called as MalConv CNN and MalConv LSTM and another algorithm refers as EMBER. Application convert dataset into binary images and then used 80% dataset for training model and 20% dataset for testing. Whenever we upload new test malware binary data then application will apply new test data on train model to predict malware class.

INDEX

CHAPTER 1

INTRODUCTION

Researchers create cutting-edge methods for detecting malware, and criminals create cunning methods for evading discovery in the game of "cat and mouse" known as malware detection. Antivirus software (AVS) typically detects malware using a trail or behavior analysis. One of the most popular methods for identifying business viruses is signature-based identification. Program executables' signatures are compared with previously saved malware traces using signature-based detection. However, because the relevant signature is missing from the database, signature-based AVS is useless against unverified zero-delay malware. Malware with known modularity or metamorphic characteristics is not immune to AVS based on indicators, in fact. These infections either have a mutation engine or use different software obfuscation strategies to update themselves with each version.

PROJECT DEFINITION:

The objective of this project is to develop a machine learning model for malware detection that can identify the type of malware present in a given dataset. The system will be trained using different machine learning algorithms such as KNN, Random Forest, SVM, CNN, and LSTM to achieve the highest possible accuracy**.**To implement this work, application will convert binary malware dataset 'MALIMG' into gray images and use 80% dataset for training and 20% dataset for testing. Whenever application uploads new test malware binary data, application will apply new test data on train model to predict malware class.

Data Pre-processing: In this phase, the dataset will be cleaned and processed to eliminate irrelevant data, duplicates, and outliers. The data will be preprocessed to convert it into a suitable format for machine learning algorithms.

Feature Extraction: In this phase, we will extract the features of the dataset using various techniques, such as static and dynamic analysis. We will use features such as file size, file type, file behaviour, and network activity to build a robust machine learning model.

Model Development: In this phase, we will develop a machine learning model using different algorithms such as KNN, Random Forest, SVM, CNN, and LSTM to train the dataset. We will

evaluate the performance of each algorithm and select the most accurate one.

Model Testing: In this phase, we will test the model using a separate test dataset to ensure that the model can accurately classify malware samples.

Model Deployment: In the final phase, the machine learning model will be deployed as a web application that can accept input files and classify them based on the type of malware present.

PROJECT OVERVIEW:

To distinguish between helpful and dangerous products, machine learning algorithms (MLAs) use feature engineering, feature selection, and feature encoding techniques. Dynamic analysis is the process of watching malware behavior at run time in an isolated setting, whereas static analysis is a way of collecting data from the binary program without actually running it. Because it requires time to analyze the behavior of the malware, dynamic analysis cannot be used for end-point real-time malware identification, but it can be an effective long-term solution. Due to security and privacy concerns, there aren't as many openly accessible standard data sets for malware research studies as MLAs demand. These issues are the main drawbacks behind developing generic machine learning based malware analysis system that can be deployed in real time.

A more advanced neural network model called deep learning has recently beaten the traditional MLAs in a variety of tasks related to speech processing, computer vision, natural language processing (NLP), and other fields. During the training process, it attempts to record higher level representations of characteristics in concealed deep levels and is able to learn from errors. Deep learning increases productivity by finding new patterns and connecting them to previously found patterns, but MLAs produce less work as they examine more and more data. There haven't been many studies on the application of deep learning methods for malware identification to enhance internet security. Malware has become more common as a result of Industry 4.0, but the current fixes are limited.

SOFTWARE REQUIREMENTS:

Operating system	:	Windows, Ultimate, Linux, Mac.
Front-End	:	Python.
Coding Language	:	Python.
Software Environment	:	Anaconda (Spyder).

HARDWARE REQUIREMENTS:

System	:	Intel i Processor.
Hard Disk	:	500 GB.
Floppy Drive	:	1.44 Mb.
Monitor	:	14’ Colour Monitor.
Mouse	:	Optical Mouse.
RAM	:	2GB.

CHAPTER 2
LITERATURE SURVEY

EXISTING SYSTEM:

The existing approaches are not scalable with very high requirements for storage and time in making efficient decisions. The absence of scalable and distributed architectures in solving malware analysis motivated the current research to investigate the algorithms and develop a scalable architecture, namely ScaleMalNet.

DISADVANTAGES:

- Accuracy is less
- The model is built either by using ML based techniques or Deep Learning based.

PROPOSED SYSTEM:

The novel image processing technique with optimal parameters for MLAs and deep learning architectures. A comprehensive experimental evaluation of these methods indicates that deep learning architectures outperform classical MLAs. Overall, this work proposes an effective visual detection of malware using a scalable and hybrid deep learning framework for real-time deployments. The visualization and deep learning architectures for static, dynamic and image processing-based hybrid approach in a big data environment is a new enhanced method for effective zero-day malware detection.

ADVANTAGES:

- Accuracy is high
- More reliable
- Includes both ML and Deep Learning Techniques.

CHAPTER 3

METHODOLOGY

ALGORITHMS

- Run ember SVM algorithm: This module is used to read malware dataset and generate train and test model and then apply SVM algorithm to calculate its prediction accuracy, Precision and Recall.
- Run ember KNN algorithm: this module is used to get its performance.
- Run ember naive bayes algorithm: This module is used to get its performance details.
- Run ember decision tree algorithm: This module is used to get its performance details.
- Run ember logistic regression algorithm: this module is used to get its performance details.
- Run ember random forest algorithm: this module is used to get its performance.
- Run malConv CNN: This module is used button to get its performance details.
- Run malConv LSTM: This module is used to run LSTM algorithm.
- Precision, recall and Fscore graph Acscuracy graph: This module is used to get comparison graph for all metrics and all algorithms.
- Predict malware family: This module is used upload binary file to get or predict class of malware.

Convolutional Neural Network (CNN):

A CNN is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

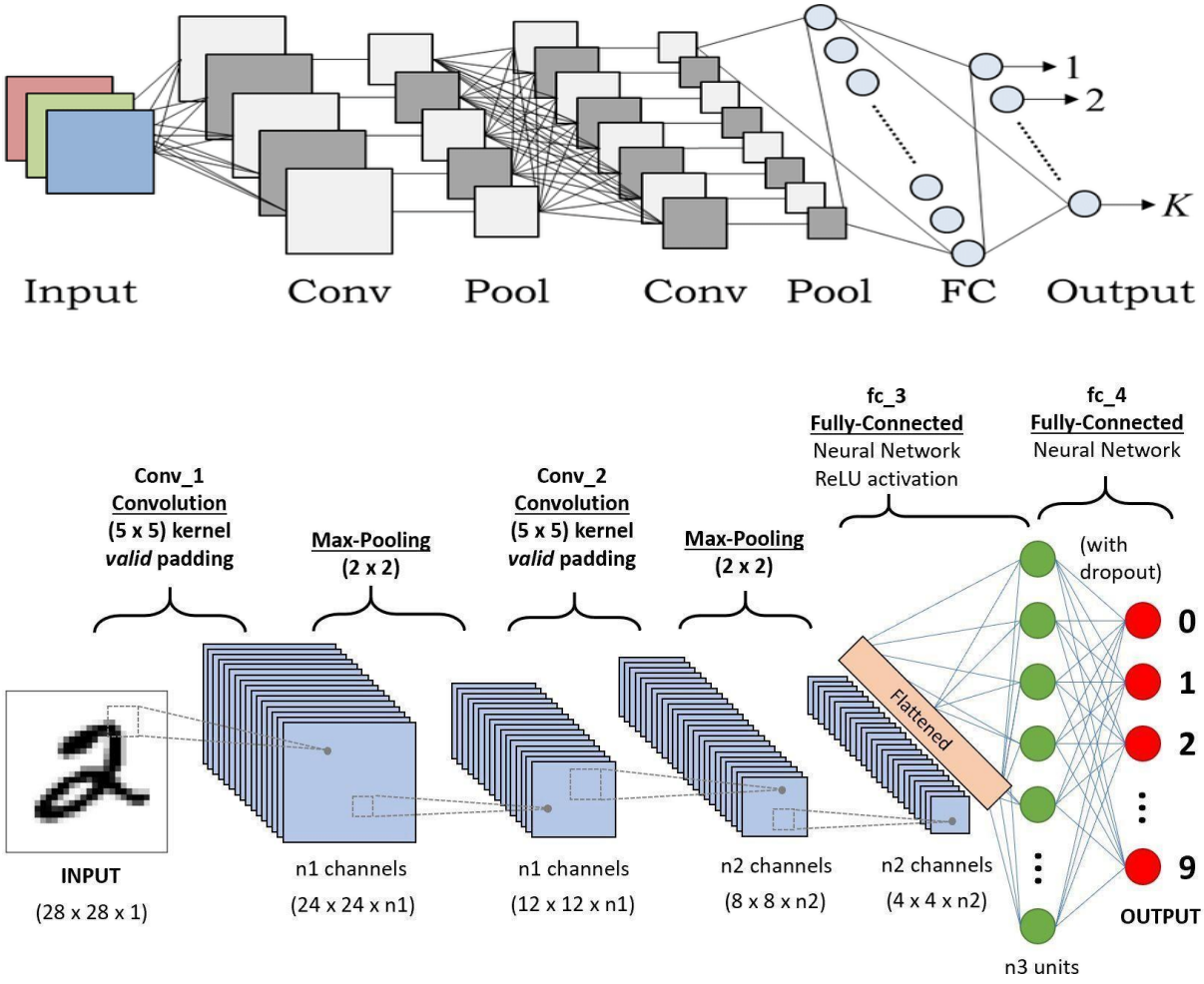
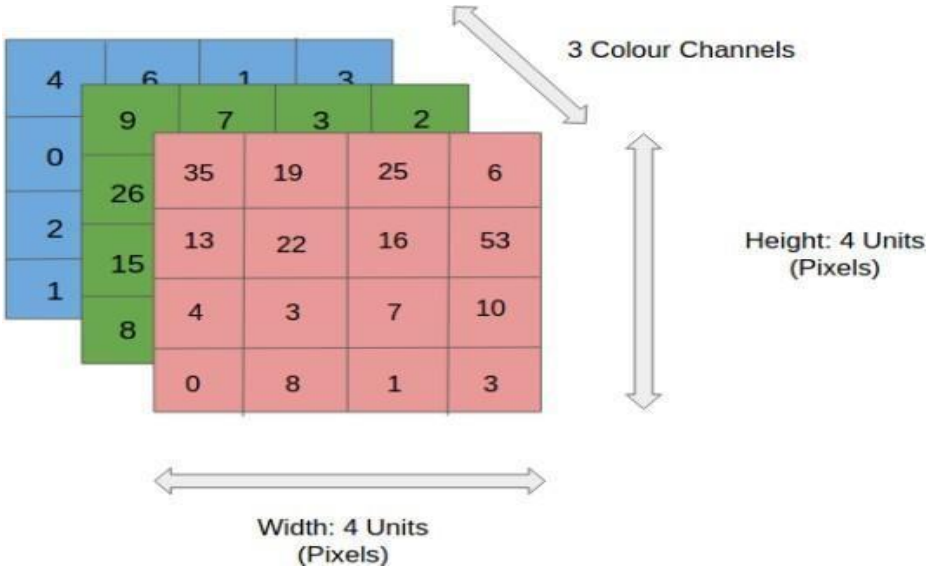


Fig 3.1: Architecture of CNN

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes? not really. In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

INPUT IMAGE:



You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

Convolution Layer –The Kernel:

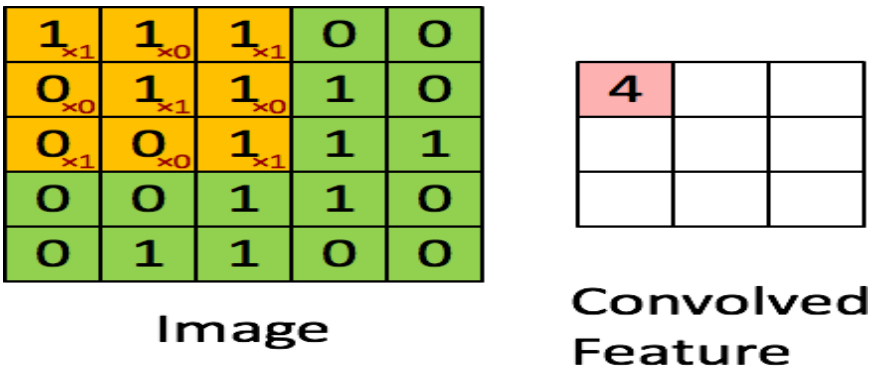


Fig 3.2: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature.

Image Dimensions = 5 (Height) x 5 (Breadth) x 1(No.of channels, eg. RGB).

In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is

called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

Kernel/Filter, K = 1 0 1
0 1 0
1 0 1

The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strides)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.

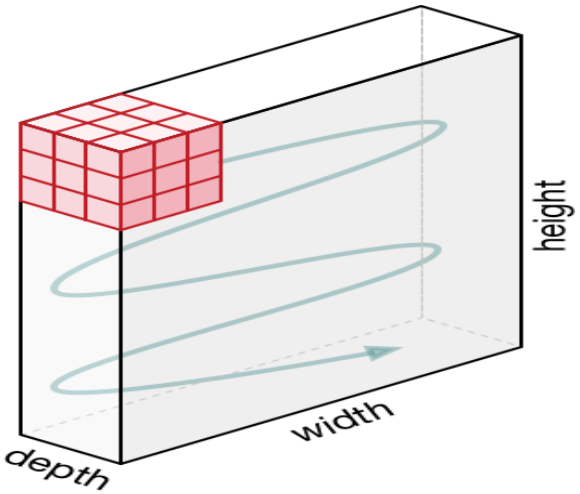


Fig 3.3: Movement of the Kernel

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

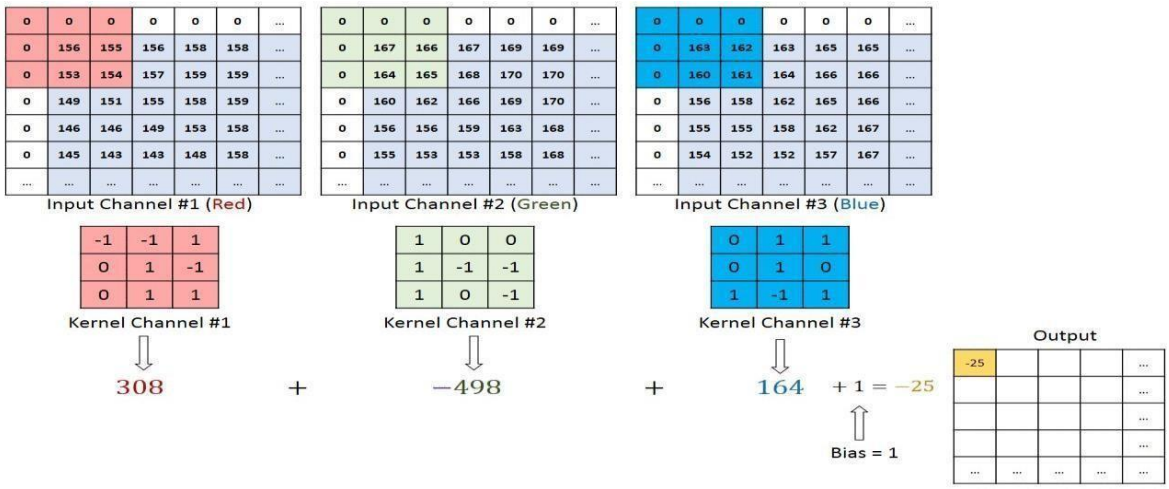


Fig 3.4: Convolution operation on a MxNx3 image matrix with 3x3x3 Kernel

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

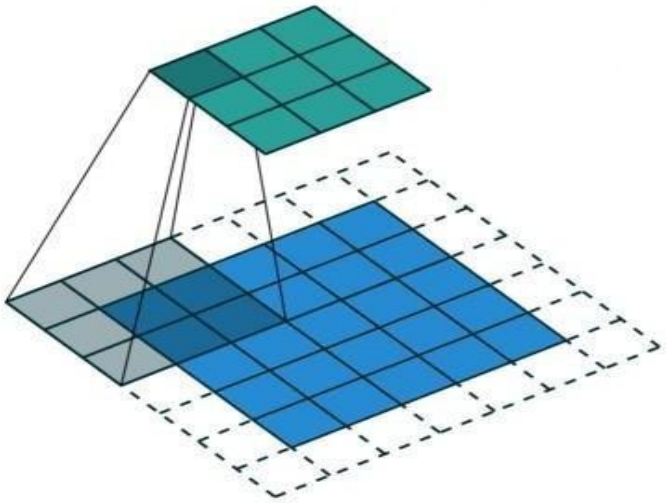


Fig 3.5: Convolution Operation with Stride Length=2

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low- Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network, which has the wholesome understanding of images in the dataset, similar to how we would. There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.

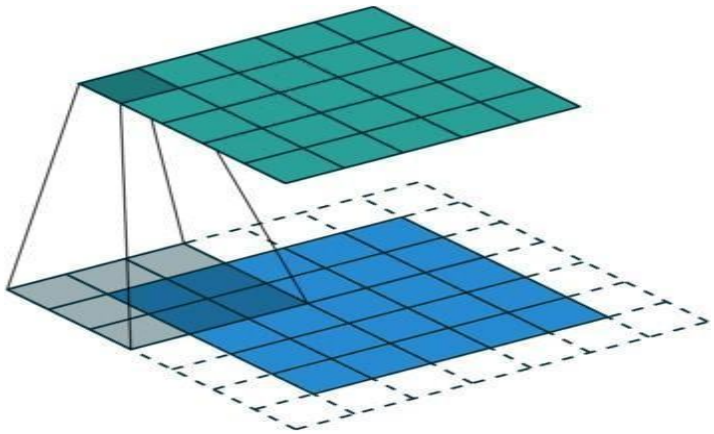


Fig 3.6: Padding Operation

SAME padding: 5x5x1 image is padded with 0s to create a 6x6x1 image When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — **Same Padding.**

On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — **Valid Padding.**

The following repository houses many such GIFs which would help you get a better understanding of how Padding and Stride Length work together to achieve results relevant to our needs.

POOLING LAYER:

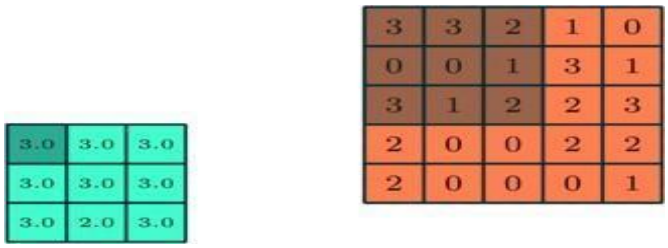


Fig 3.7: 3x3 pooling over 5x5 convolved feature

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

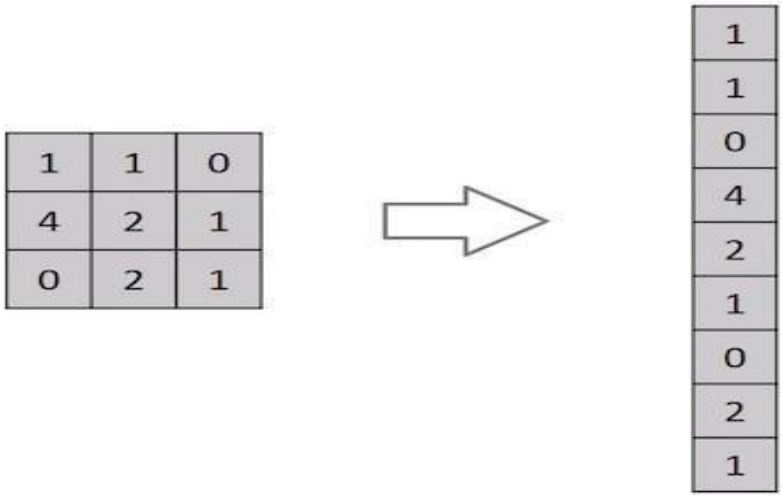


Fig 3.8: Flattening of 3x3image matrix into 9x1 vector

CLASSIFICATION – FULLY CONNECTED LAYER (FC LAYER):

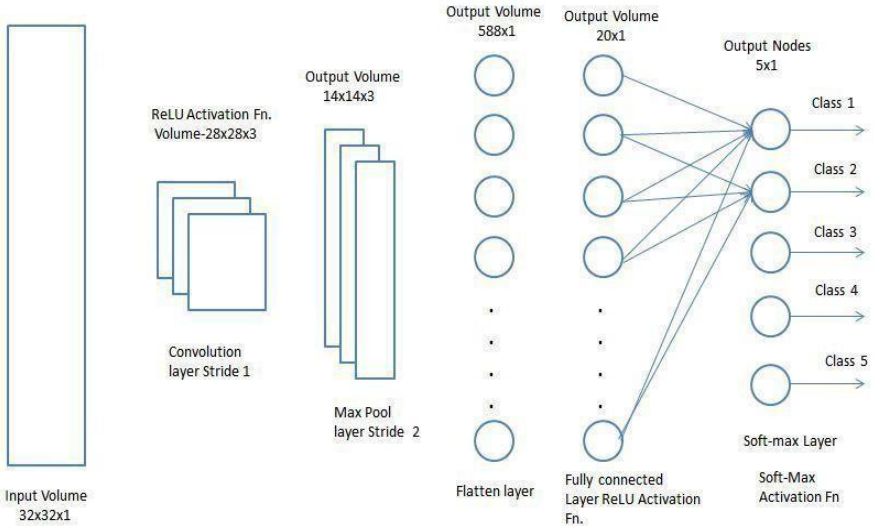


Fig 3.9: FC Layer

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high- level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low- level features in images and classify them using the **SoftMax Classification** technique.

CHAPTER 4

SYSTEM ARCHITECTURE

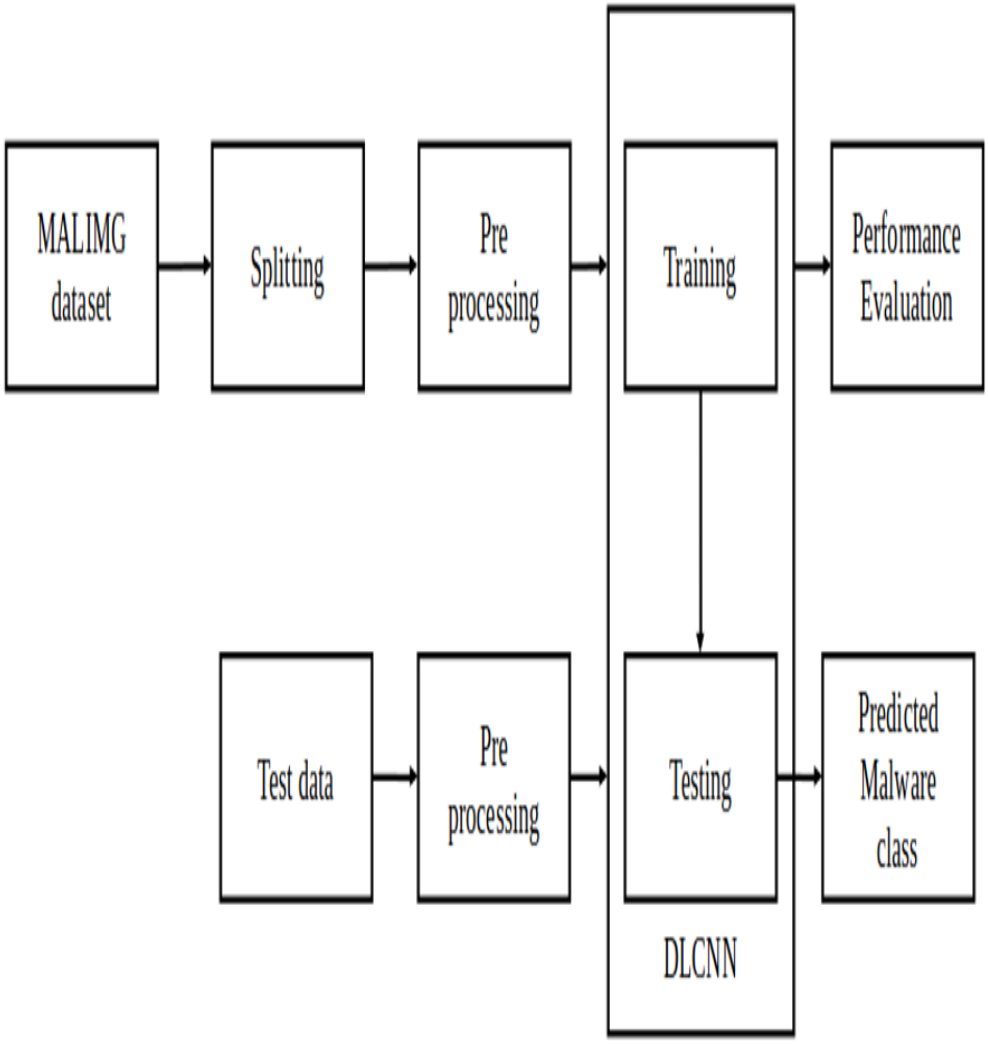
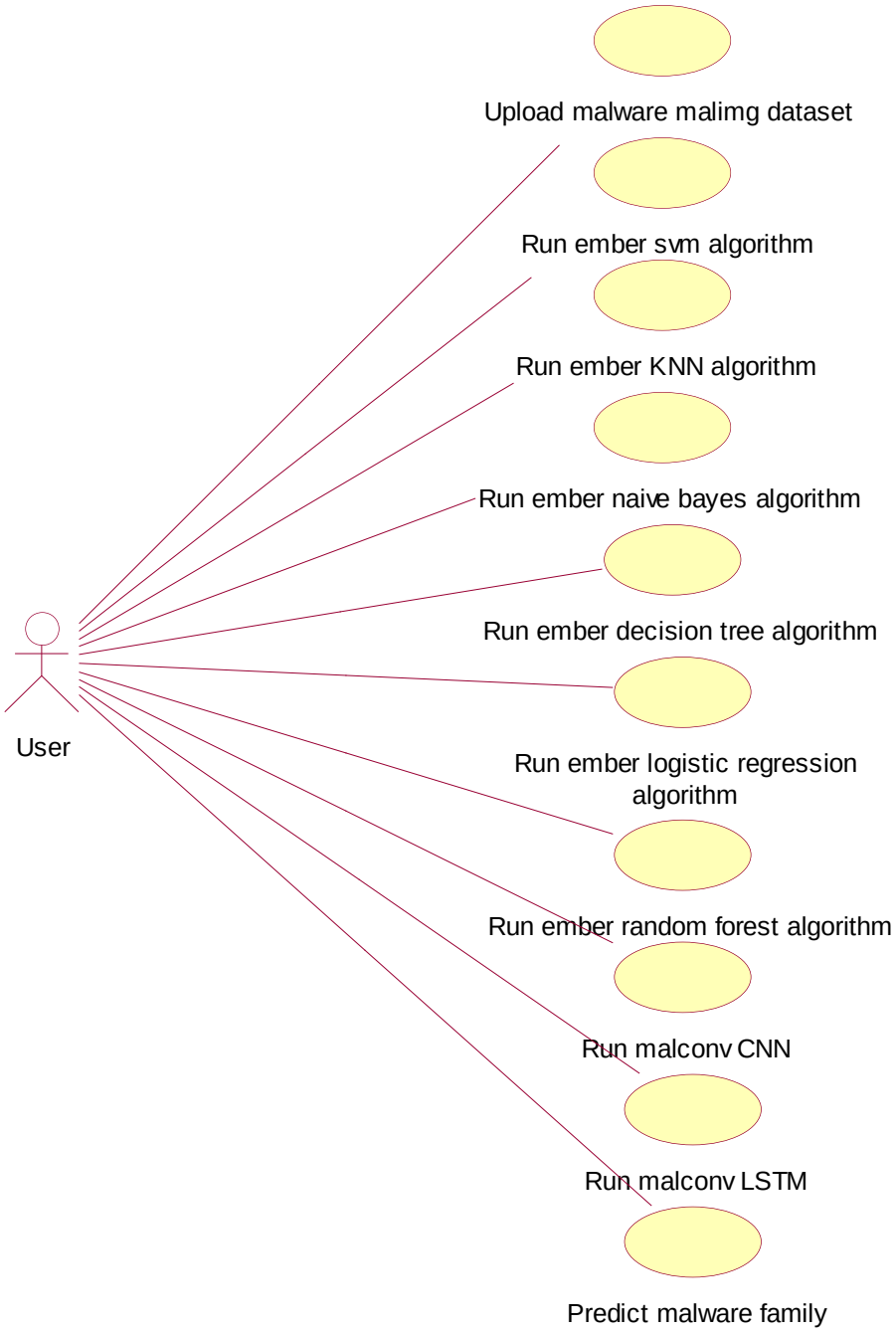


Fig 4.1: Block Diagram of Malware Detection using DLCNN

USE CASE DIAGRAM:



.

Fig 4.2: Use Case diagram of Malware Detection using DLCNN

SEQUENCE DIAGRAM:

Fig 4.3: Sequence diagram of Malware Detection using DLCNN

CHAPTER 5

IMPLEMENTATION

MODULES USED:

TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and Ipython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provide a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python.

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

SOURCE CODE:

```
def prediction(X_test, cls):
    y_pred = cls.predict(X_test)
    for i in range(len(X_test)):
      print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred
def CNN():
    global cnn_acc
    global cnn_precision
    global cnn_recall
    global cnn_fmeasure
    text.delete('1.0', END)
    data, labels = load_lstmcnn(np.load(filename,allow_pickle=True))
    labels = labels.reshape((9339,1))
    print(labels)
    data = data.reshape((9339, 32, 32))
    X_train1, X_test1, y_train1, y_test1 = train_test_split(data, labels, test_size=0.101)
    enc = OneHotEncoder()
    enc.fit(y_train1)
    print(y_train1.shape)
    y_train1  = enc.transform(y_train1)
    y_test1   = enc.transform(y_test1)
    #rehsaping traing
    print("X_train.shape before  = ",X_train1.shape)
    X_train1 = X_train1.reshape((8395, 32, 32,1))
```

```
    print("X_train.shape after  = ",X_train1.shape)
    print("y_train.shape  = ",y_train1.shape)
    #rehsaping testing
    print("X_test.shape before  = ",X_test1.shape)
    X_test1 = X_test1.reshape((944, 32, 32,1))
    print("X_test.shape after  = ",X_test1.shape)
    print("y_test.shape  = ",y_test1.shape)
    classifier = Sequential()
    classifier.add(Convolution2D(32, (3, 3), border_mode='valid', input_shape=(32, 32, 1)))
    classifier.add(BatchNormalization())
    classifier.add(Activation("relu"))
    classifier.add(Convolution2D(32, (3, 3),  border_mode='valid'))
    classifier.add(BatchNormalization())
    classifier.add(Activation("relu"))
    classifier.add(MaxPooling2D(pool_size=(2, 2)))
    classifier.add(Flatten())
       classifier.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    classifier.fit(X_train1, y_train1,epochs = 10,batch_size=64)
    prediction_data = classifier.predict(X_test1)
    prediction_data = np.argmax(prediction_data, axis=1)
    y_test1 = np.argmax(y_test1, axis=1)
    cnn_precision = precision_score(y_test1, prediction_data,average='micro') * 100
    cnn_recall = recall_score(y_test1, prediction_data,average='micro') * 100
    cnn_fmeasure = f1_score(y_test1, prediction_data,average='micro') * 100
    cnn_acc = accuracy_score(y_test1,prediction_data)*100
    text.insert(END,"CNN Prediction Results\n")
```

```
    text.insert(END,"CNN Precision : "+str(cnn_precision)+"\n")
    text.insert(END,"CNN Recall : "+str(cnn_recall)+"\n")
    text.insert(END,"CNN FMeasure : "+str(cnn_fmeasure)+"\n")
    text.insert(END,"CNN Accuracy : "+str(cnn_acc)+"\n")
def predict():
    filename = filedialog.askopenfilename(initialdir = "images")
    text.delete('1.0', END)
    text.insert(END,filename+" loaded\n\n")
    with open('model.json', "r") as json_file:
        loaded_model_json = json_file.read()
        loaded_model = model_from_json(loaded_model_json)
    loaded_model.load_weights("model_weights.h5")
    loaded_model._make_predict_function()
    print(loaded_model.summary())

    img = np.load(filename)
    im2arr = img.reshape(1,32,32,1)
    preds = loaded_model.predict(im2arr)
    print(str(preds)+" "+str(np.argmax(preds)))
    predict = np.argmax(preds)
    text.insert(END,'Uploaded file contains malware from family : '+malware_name[predict])
```

CHAPTER 6

RESULTS

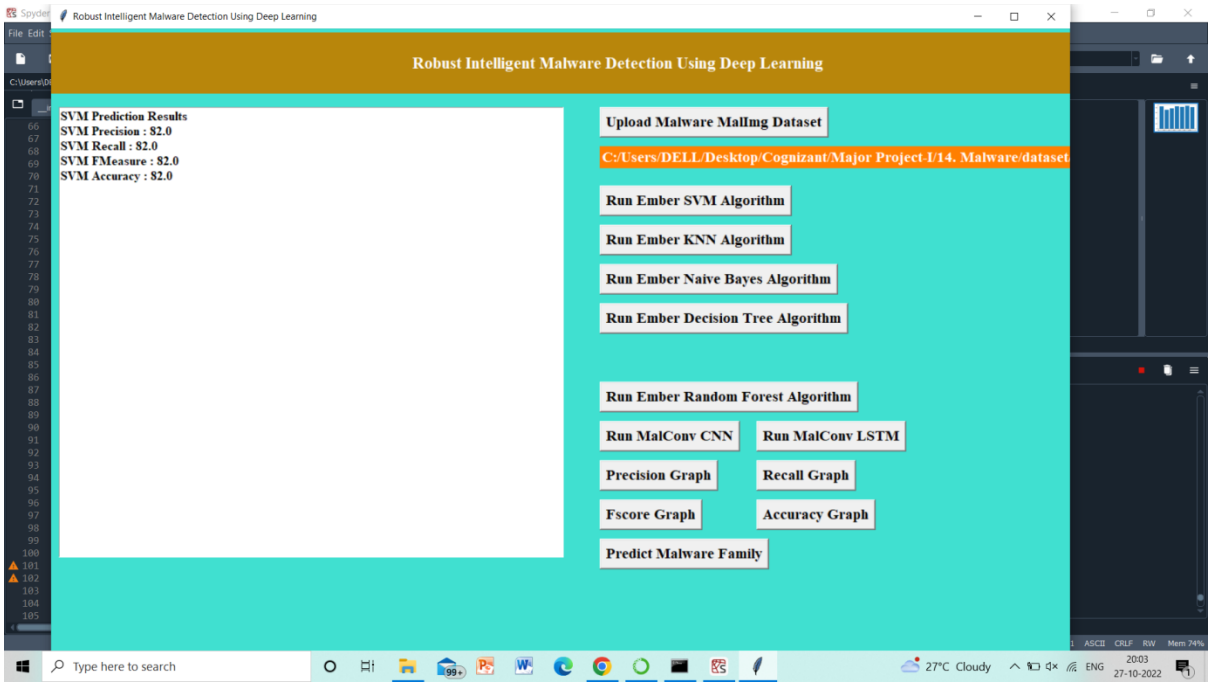


Fig 6.1: Performance evaluation using SVM

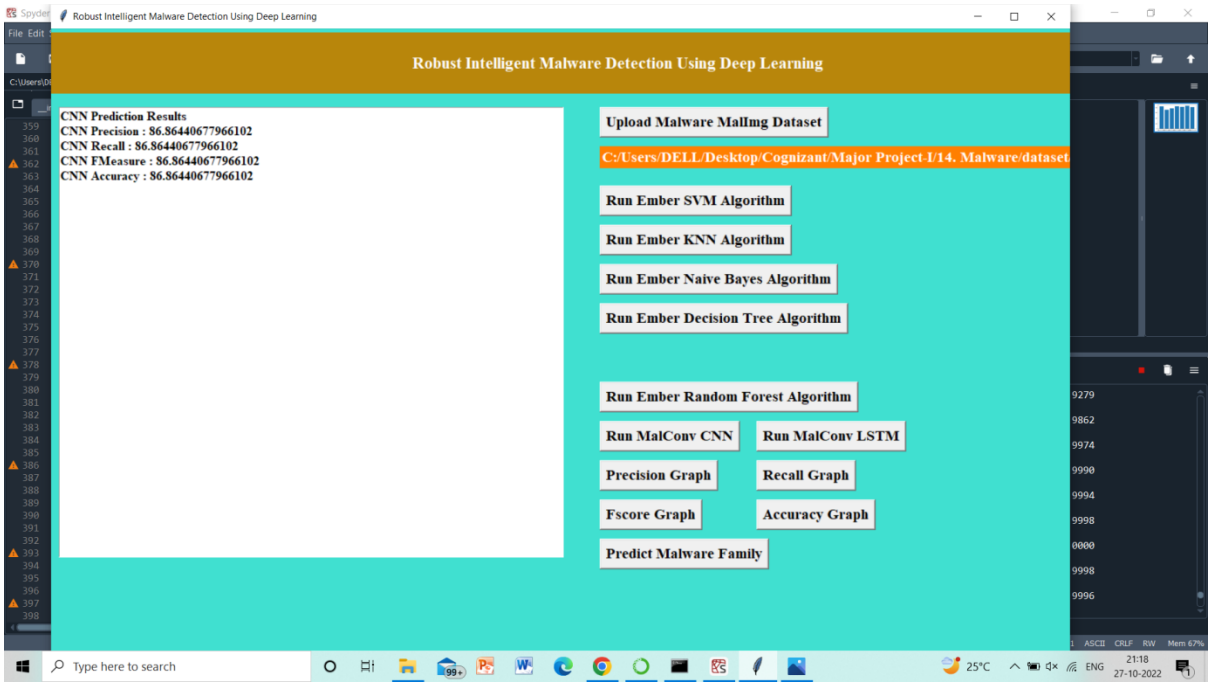


Fig 6.2: Performance evaluation using CNN

DETECTING MALWARE FAMILY:

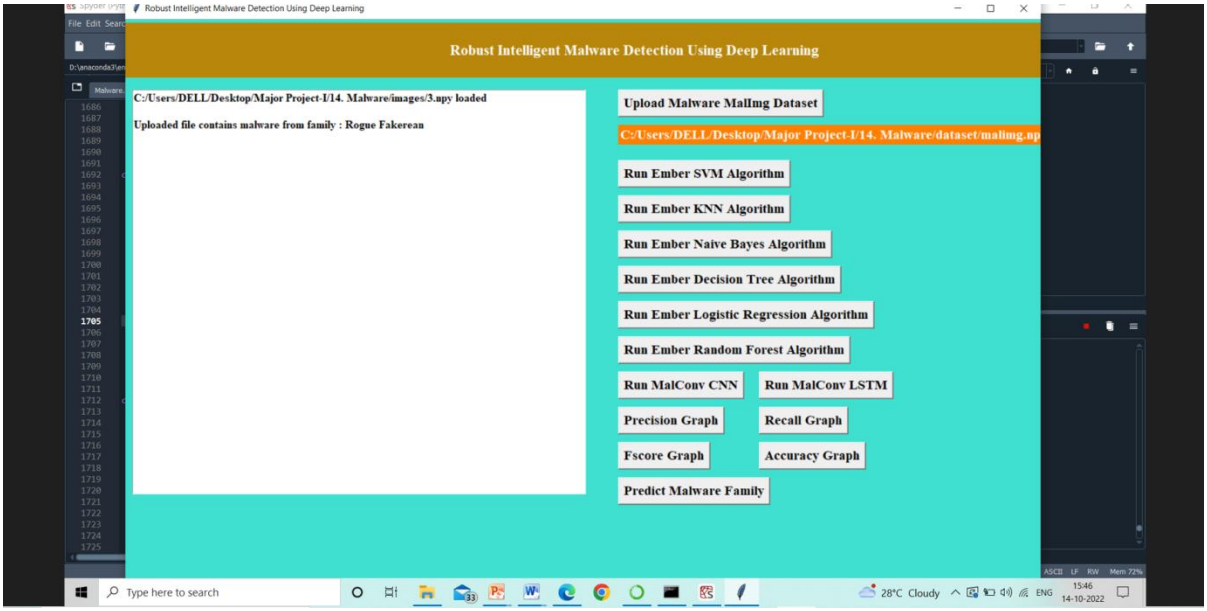


Fig 6.3: Prediction of malware by uploading the malware file

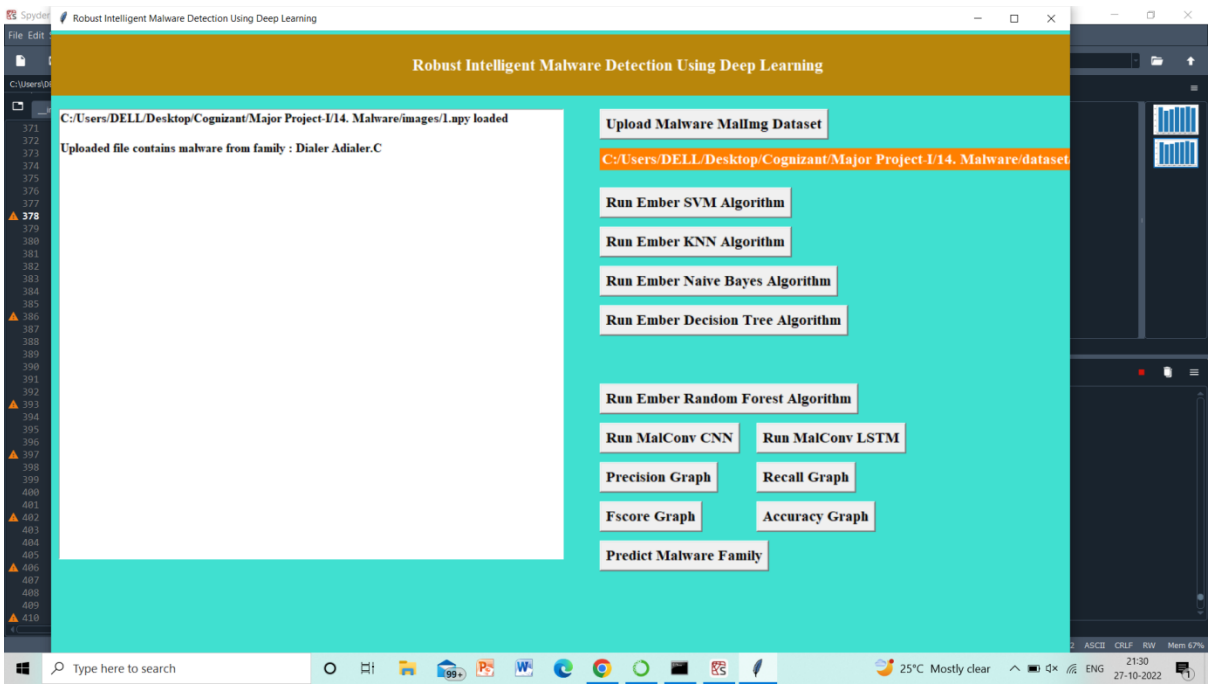


Fig 6.4: Prediction of malware by uploading the malware file

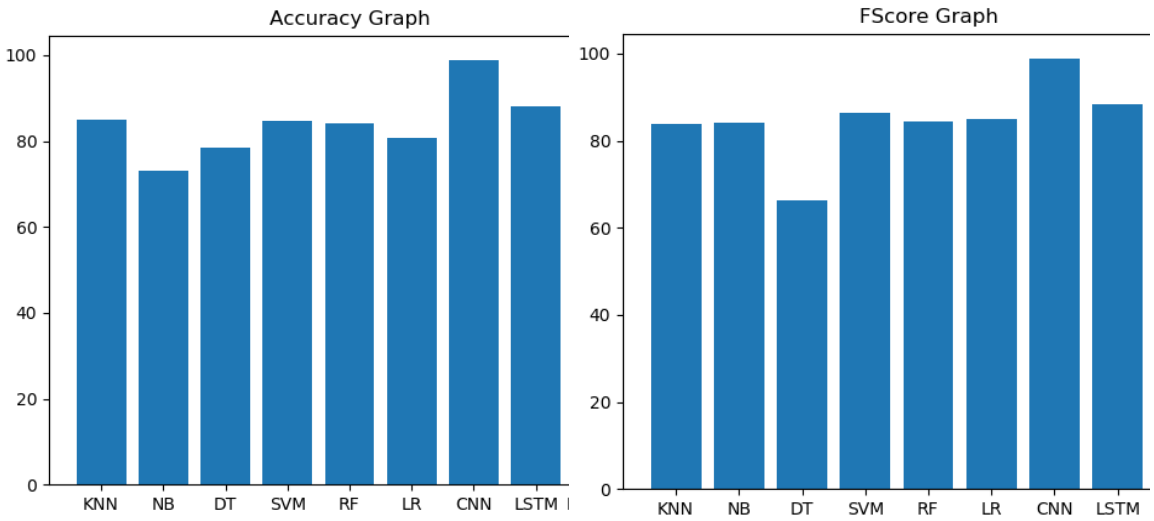


Fig 6.5: Performance comparison of accuracy and F-score obtained using existing and proposed models.

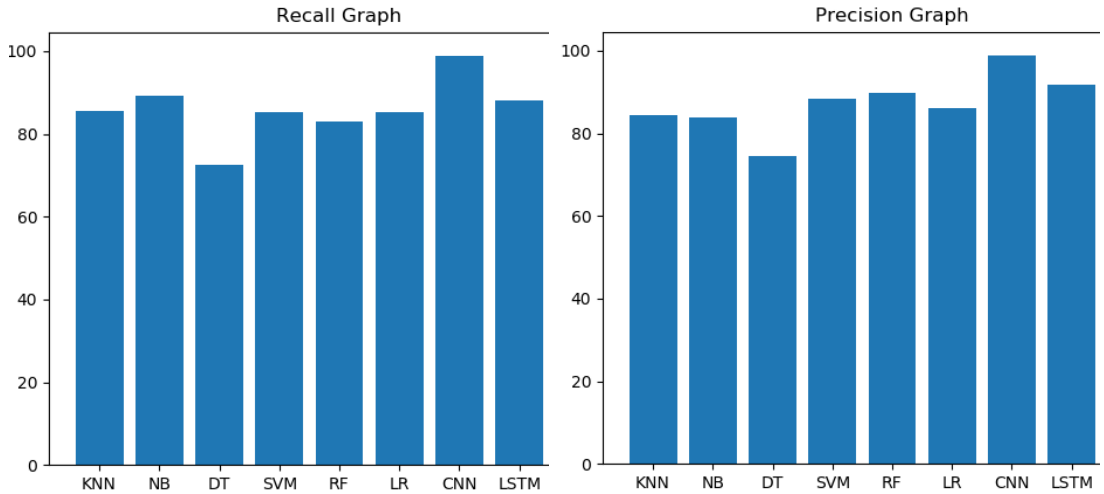


Fig 6.6: Performance comparison of recall and precision obtained using existing and proposed models.

CHAPTER 7
CONCLUSION

This project proposed an efficient malware detection and designed a highly scalable framework to detect, classify and categorize zero-day malwares. This framework applies neural network on the collected malwares from end user hosts and follows a two-stage process for malware analysis. In the first stage, a hybrid of static and dynamic analysis was applied for malware classification. In the second stage, malwares were grouped into corresponding malware categories using image processing approaches. Various experimental analysis conducted by applying variations in the models on publicly available benchmark dataset and indicated the proposed model outperformed classical MLAs. The developed framework is capable of analysing large number of malwares in real-time and scaled out to analyse even larger number of malwares by stacking a few more layers to the existing architectures. Future research entails exploration of these variations with new features that could be added to the existing data.

REFERENCES

[1]. B. Li, K. Roundy, C. Gates, and Y. Vorobeychik, ‘‘Large-scale identification of malicious singleton files,’’ in Proc. 7th ACM Conf. Data Appl. Secur. Privacy. New York, NY, USA: ACM, Mar. 2017, pp. 227–238.

[2]. S. Huda, J. Abawajy, M. Alazab, M. Abdollalihian, R. Islam, and J. Yearwood, ‘‘Hybrids of support vector machine wrapper and filter based framework for malware detection,’’ Future Gener. Comput. Syst., vol. 55, pp. 376–390, Feb. 2016.

[3]. E. Raff, J. Sylvester, and C. Nicholas, ‘‘Learning the PE header, malware detection with minimal domain knowledge,’’ in Proc. 10th ACM Workshop Artif. Intell. Secur. New York, NY, USA: ACM, Nov. 2017, pp. 121–132.

[4]. E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas. (2017). ‘‘Malware detection by eating a whole exe.’’ [Online]. Available: https://arxiv.org/abs/1710.09435

[5]. M. Krcál, O. Švec, M. Bálek, and O. Jašek. (2018). Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only. [Online]. Available: https://openreview.net/forum?id=HkHrmM1PM

[6]. M. Rhode, P. Burnap, and K. Jones, ‘‘Early-stage malware prediction using recurrent neural networks,’’ Comput. Secur., vol. 77, pp. 578–594, Aug. 2018.

[7]. H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, Evading Machine Learning malware Detection. New York, NY, USA: Black Hat, 2017.

[8]. R. Verma, ‘‘Security analytics: Adapting data science for security challenges,’’ in Proc. 4th ACM Int. Workshop Secur. Privacy Anal. New York, NY, USA: ACM, Mar. 2018, pp. 40–41.

[9]. S. H. Ebenuwa, M. S. Sharif, M. Alazab, and A. Al-Nemrat, ‘‘Variance ranking attributes selection techniques for binary classification problem in imbalance data,’’ IEEE Access, vol. 7, pp. 24649–24666, 2019.

[10]. E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. de Geus, ‘‘Malicious software classification using VGG16 deep neural network’s bottleneck features,’’ in Information Technology-New Generations. Cham, Switzerland: Springer, 2018, pp. 51–59.