

EMAIL BASED PHISHING WEBSITES AND EMAIL DETECTION USING MACHINE LEARNING

J. M. LINDAMULAGE

(IT20222840)

BSc (Hons) in Information Technology
Specializing in Cyber Security

Department of Computer Systems Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

September 2023

DETECTING PHISHING WEBSITES USING THE VISUAL SIMILARITY

J. M. LINDAMULAGE

(IT20222840)

Dissertation submitted in partial fulfillment of the requirements for the
Bachelor of Science (Hons) in Information Technology

Specializing in Cyber Security

Department of Computer Systems Engineering


Sri Lanka Institute of Information Technology

Sri Lanka

September 2023

DECLARATION

I declare that this is my own work, and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
J.M.Lindamulage	IT20222840	

The above candidate has carried out research for the bachelor's degree Dissertation under my supervision.

Signature of the Supervisor

Date

ABSTRACT

With the forever evolving technology there are several downsides. Phishing attacks are one of them. Phishing attacks are carried out through emails that send victims fraudulent links to websites. Victims of these attacks often suffer financial, reputational, and other losses. If users can identify suspicious websites before entering their sensitive information, it can help prevent financial and reputational damage.

To address this issue there are many ways and techniques that are introduced by many researchers. For this study the graph neural network will be used to identify a phishing site with visual similarity of a website. The usage of graph neural network to identify an image is a new area which was recently introduced by a group of researchers [1]. In this report the attempt and process taken to identify a phishing website using a screenshot with a graph neural network model will be presented. For this purpose, the VISUALPHISHNET data set was used, and we were able to achieve a good accuracy from the model in classifying the phishing and normal websites.

The report will first discuss the background literature on phishing detection. Then, it will describe the process of finding the best model with the highest accuracy. Finally, it will discuss how the model was integrated into a mobile application.

ACKNOWLEDGEMENT

First of all, I would like to express my sincere gratitude to Ms. Jenny Krishara of Faculty of computing in Sri Lanka Institute of Information Technology (SLIIT) for her invaluable support and guidance throughout the project and for sharing her experience and knowledge with us to complete the project. I am grateful for the guidance given by the CDAP panel and lecturers for their advice in our project work. I would also like to thank my team members for the support given through the project. Finally, I would like to express my heartfelt appreciation to my parents and my family members for their guidance and prayers throughout all these years.

CONTENTS

DECLARATION	i
ABSTRACT.....	ii
ACKNOWLEDGEMENT	iii
Table of figures	vi
LIST OF TABLES	viii
1 INTRODUCTION	1
1.1 Background Literature.....	1
1.1.1 Phishing attacks	1
1.1.2 Techniques for detecting phishing websites	4
1.1.3 Visual similarity to detect phishing websites.....	6
1.1.4 Data sets	10
1.1.5 Graph Neural Networks(GNN).....	10
1.1.6 Graph Neural Networks for computer vision tasks.....	12
1.1.7 Research Gap	16
1.1.8 Research problem.....	19
1.1.9 Research Objectives.....	19
a) Main objectives.....	19
b) Specific Objectives.....	20
2 Methodology	21
3 Testing and Implementation	24
3.1 Mobile application.....	27
4 Commercialization aspects of the product.....	28
4.1 Results and Discussion.....	29

4.1.1	Results.....	29
4.1.2	Confusion metrices	34
4.1.3	Mobile Application	37
5	Research Findings.....	41
6	Conclusion	42
7	References.....	43
8	Appendices.....	47

TABLE OF FIGURES

Figure 1.1 2022 APWG report (source [4])	1
Figure 1.3 phishing email that used fear of the pandemic	2
Figure 1.4-Phishing Attack Scenario(Source- [5])	3
Figure 1.5Real vs Fake web page instances.....	4
Figure 1.6 Overview of Phishing Detection methods(source [12])	5
Figure 1.7 Visual similarity approaches(source - [6]).....	6
Figure 1.8VisualPhishNet zero-day attack detection capability(Source [16]).....	8
Figure 1.9 Simple directional graph representation between 3 nodes(Source [29]).....	10
Figure 1.10Graph representation of people in social media (source [21])	11
Figure 1.11 The steps of how a graph is constructed Source [25]	13
Figure 1.12 Vision GNN model Architecture.....	15
Figure 2.1 Screenshots of same site from VISUALPHISHNET dataset	22
Figure 2.2 Screenshots of different websites from the VISUALPHISHNET dataset	22
Figure 3.1Pytorch to android	27
Figure 3.2 PyTorch mobile in the android bundle	27
Figure 4.1 tiny model Mean Squared Error vs Epochs.	30
Figure 4.2 Tiny model validation accuracy	30
Figure 4.3 Small model Mean Squared Error vs Epochs.	31
Figure 4.4 Small model validation accuracy.....	31
Figure 4.5Medium model Mean Squared Error vs Epochs.....	32
Figure 4.6Medium model validation accuracy	32
Figure 4.7Big model mean squared error vs Epochs.	33
Figure 4.8Big model validation accuracy	33
Figure 4.9 Confusion Metrix of the model big	34
Figure 4.105Confusion Metrix of the model tiny	35
Figure 4.11Confusion Metrix of the model small.....	35
Figure 4.12Confusion Metrix of the model medium	36
Figure 4.13 Mobile app UI.....	37
Figure 4.14 Select screenshot from gallery.....	38
Figure 4.15 Benign site detected correctly.	39

Figure 4.16 Phishing site detected correctly.	40
Figure 8.1Layers(Source[[26]]	47
Figure 8.2Weights.....	48
Figure 8.3Nueral network	48

LIST OF TABLES

Table 1.1 Research Gap 17

Table 1.2 Research Gap 18

Table 5.1 Model accuracies and number of parameters 41

1 INTRODUCTION

1.1 Background Literature

1.1.1 Phishing attacks

With the fast-growing technology phishing remains as the most popular method of stealing sensitive data, money and distributing malware. By using social engineering tactics attackers persuade a victim by posing as a trustworthy person and sent emails which contains the link to the phishing site. The goal of this type of attack is to deceive the victim into revealing sensitive information or downloading malware onto their device. When the victim clicks on the link or open the attachment they will be redirected to a phishing site, or a malware will be installed inside the user's machine [1]. The attacker will monitor it and if the user enters their sensitive details to the fake website, then the attacker can gain them easily [2]. These emails can be described as phishing emails. Not only emails phishing scams can be done in calls or text messages also [3]. Typically, phishing attempts take the form of messages soliciting charitable donations, urging recipients to change their passwords, cancel credit card payments, or even disseminate pandemic-related instructions [3]. When successful, these attacks can lead to dire consequences, including data breaches, identity theft, malware infections, or loss of critical services. According to the report issued for the 3rd quarter of 2022 by Anti-Phishing Working Group (APWG), there were 1,270,883, total phishing attacks. They state that this is a new record and the worst quarter to ever observed [4] (Figure 1.1).

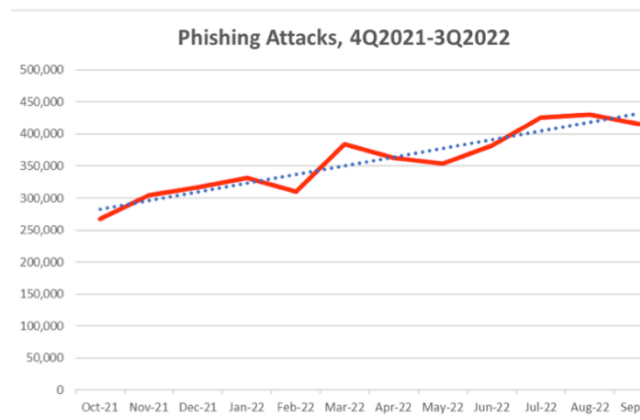


Figure 1.1 2022 APWG report (source [4])

The following figure show how the attackers send phishing emails as instructions for Covid-19 pandemic with the target of gaining user credentials or distributing the malware.

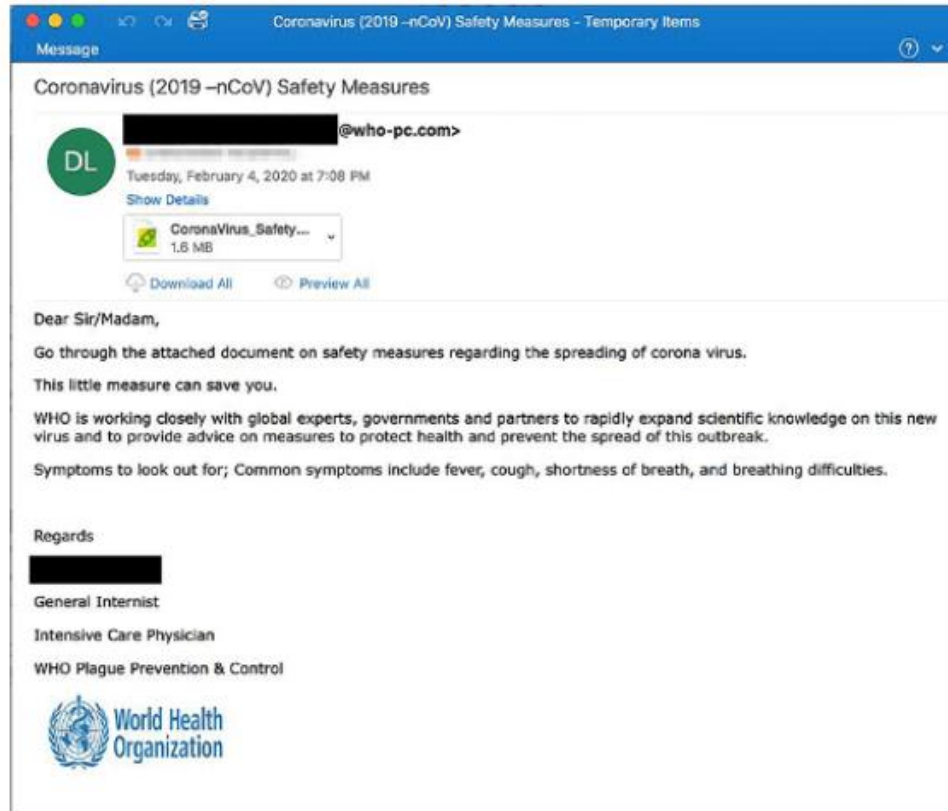


Figure 1.3 phishing email that used fear of the pandemic

In a typical phishing attack an attacker generates fake website pages and sends suspicious URLs to targeted victims via unwanted texts, emails, or online social networking forums.

The attacker tempts the victim to include highly sensitive personal or financial data such as login, passwords, bank details, government savings numbers, and so on, using these credentials to steal money or data [5].

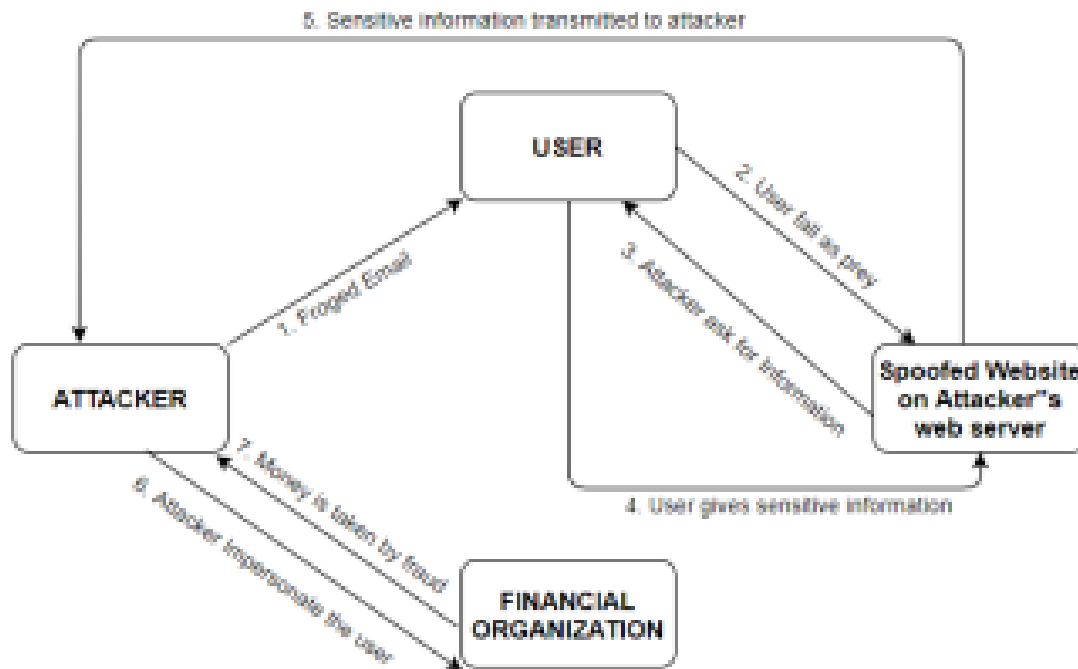


Figure 1.4-Phishing Attack Scenario(Source- [5])

When a phishing website closely mimics the visual elements of its legitimate counterpart, including page layouts, images, text content, font size, and font color, it becomes significantly more deceptive and increases the likelihood of users falling victim to a phishing attack. An illustration of this is provided in the figure below, where both counterfeit and authentic pages share an indistinguishable visual appearance, despite having distinct URLs. This likeness is achieved by cyber attackers who replicate the HTML source code of a genuine website to craft fraudulent web pages [6]. In a survey [8] on recognising phishing sites, 90% of the participants were unable to tell the difference between a phishing website from a legal one. The percentage of persons who skip checking the URL is 23% [7].



Figure 1.5 Real vs Fake web page instances

1.1.2 Techniques for detecting phishing websites

The most frequent way for mitigating cyber-attacks is to use a firewall or antivirus software. However, these are insufficient to halt the attack. Over the years, there have been a few methods for detecting phishing sites [8]. Aside from user awareness, software-based detection approaches are shown in Figure 1.6. In the list-based approach, a list of blacklists containing dangerous URLs is updated and maintained by continuously adding new URLs. This method is easy and accurate, but it is not effective at detecting new rogue websites. Several content-based and non-content-based aspects have been utilized in the heuristic-based approach [9] [10] [11] [8].

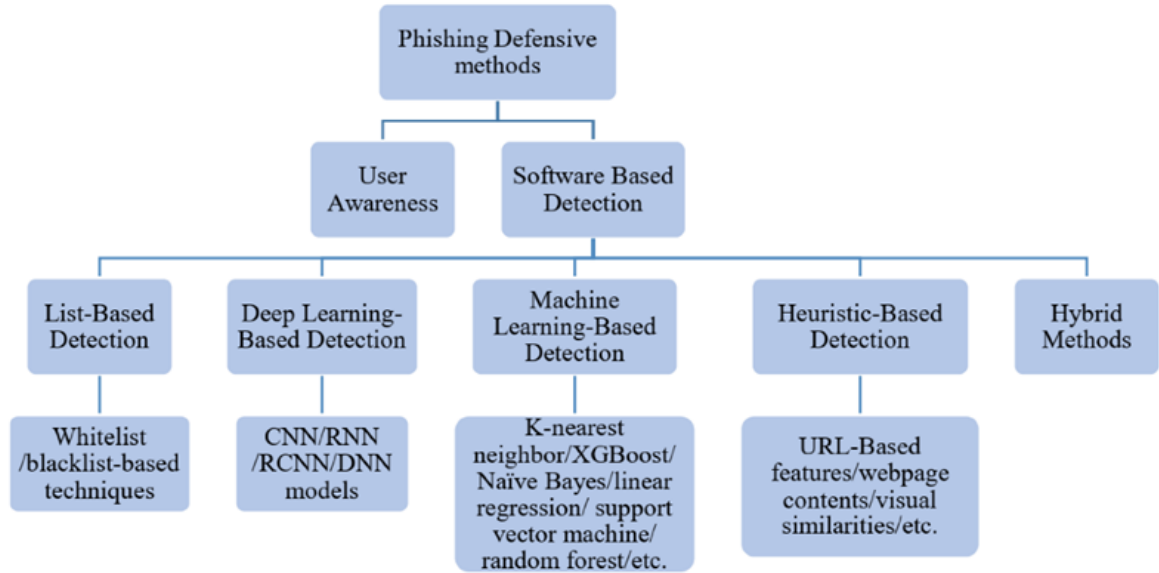


Figure 1.6 Overview of Phishing Detection methods(source [12])

Deep learning and machine learning-based technologies have recently been applied to achieve high accuracy in recognizing these sites. Researchers have utilized approaches such as K-nearest neighbour, XGBoost, Nave Bayes, linear regression, and support vector to detect phishing websites using URL features and a bag of words. Although deep learning approaches have not been widely used to detect phishing websites, previous research has used CNN, recurrent neural networks, recurrent convolutional neural networks, and deep neural networks (DNN) with URLs. Hybrid detection systems combine many strategies from the list above to detect phishing websites efficiently. To identify phishing websites, the researchers used machine learning and deep learning to combine URL characteristics, network information, and visual features [12].

1.1.3 Visual similarity to detect phishing websites.

There are numerous types of visual similarity-based methodologies, including pixel-based analysis, visual perception, cascading style sheet similarity, and hybrid approaches (as shown in Figure 1.7). These methods rely on a variety of indicators, including text content and text-related variables including font color, font size, background color, and font family [6].

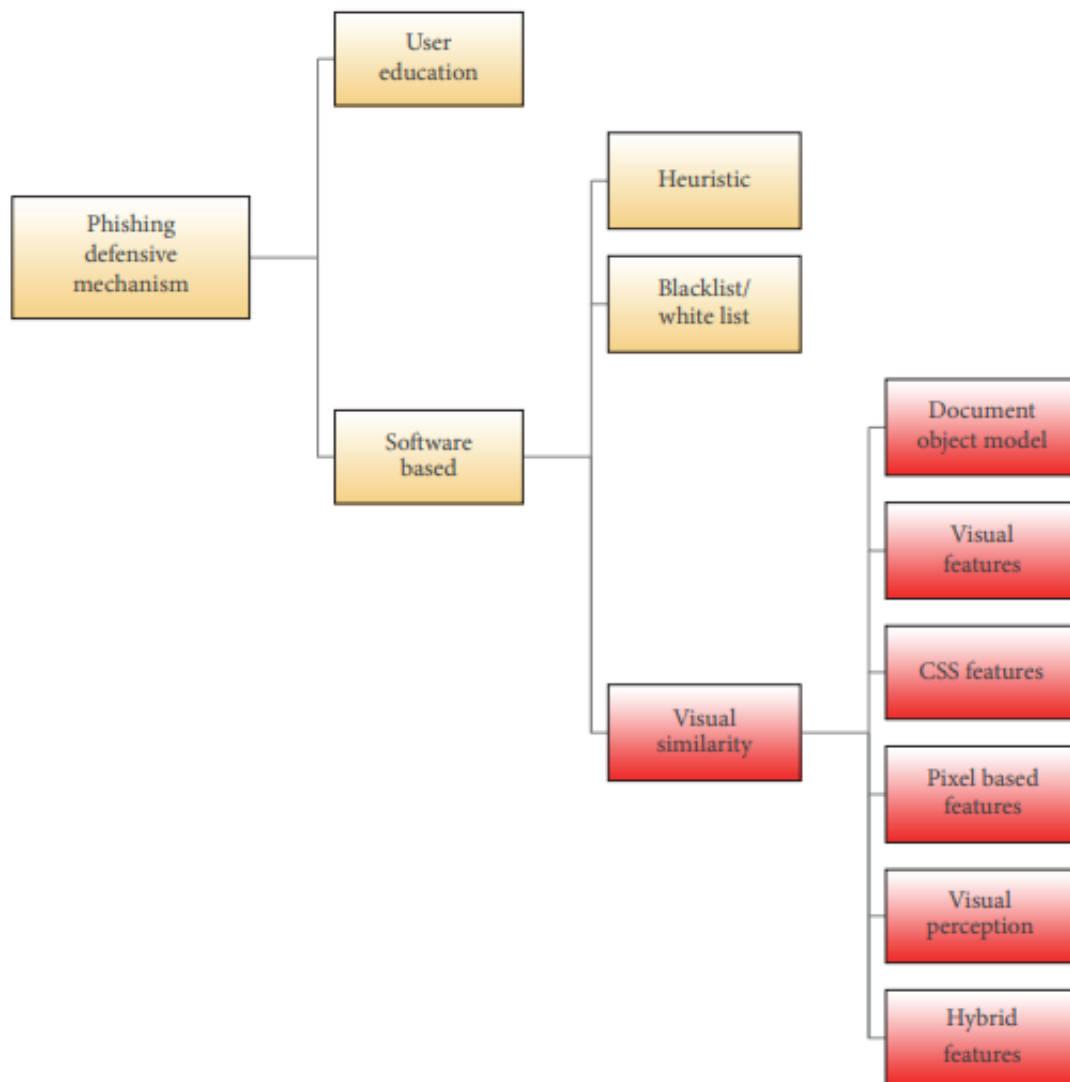


Figure 1.7 Visual similarity approaches(source - [6])

Machine Learning (ML) models have lately been used to detect phishing websites, and much research is being conducted to detect fraudulent URLs [3]. To train an ML model to detect phishing websites, datasets are required. There are much ongoing research conducted regarding detecting phishing websites utilizing characters in the URL, IP address, email header data, text in the email, and the aesthetic similarity of the phishing website to the real site [8]. Machine learning is being used to detect phishing websites using these visual cues and visual similarities. Some related work will be covered in the following section.

Eric Medvet et al. [13] suggest a visual similarity-based approach for evaluating a webpage's visual attributes to those of known legitimate webpages in their research. They utilized computer vision techniques to extract visual features such as layout, color, and texture from the suspect webpage as well as a set of reference web pages. Text fragments, images incorporated in the page, and the overall visual aspect of the web page as rendered by the browser are among the features they have explored. The similarities between the suspected webpage's features and those of the reference web pages were then used to determine if the webpage was real or phishing.

In 2017, Igino Corona et al. [14] published a paper that introduces a method named "DeltaPhish" that can identify HTML code and visual changes between pages. Their system is suitable for usage as part of a web application firewall. They test these on over 5500 webpages and claim that it can recognize 70% more of them. They have created their own dataset which has 5511 distinct webpages and 1012 are phishing pages. The Histogram of oriented Gradients and color histograms were used as image classification.

F.C. Dalgic [15] did a study on the detection of phishing websites using picture classification in 2018. They presented a novel approach that integrates machine learning with visual analysis, with the goal of extracting and classifying succinct visual features from website screenshots. The researchers used a variety of compact visual descriptors based on MPEG7 and similar principles, including Joint Composite Descriptor (JCD), Scalable Color Descriptor (SCD), Color and Edge Directivity Descriptor (CEDD), Color Layout Descriptor (CLD), and FCTH. These descriptors were used to identify various visual patterns associated with color and edges. They used support vector machine and

random forest models to classify photos.

Their data set included 2852 samples, and their method produced an outstanding F1 score of 90.5 utilizing SCD.

A study done by Sahar Abdelnabi et al. [16] introduces a similarity-based phishing detection framework named “VisualPhishNet” using Convolutional Neural Network(CNN) to detect zero-day phishing websites.

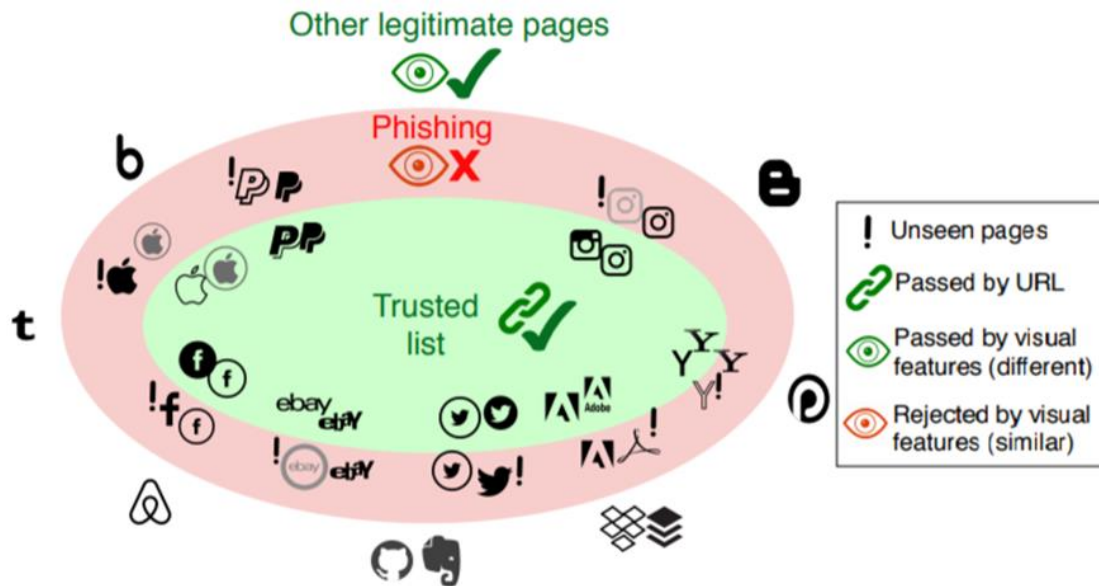


Figure 1.8 VisualPhishNet zero-day attack detection capability (Source [16])

According to the Figure, they were able to classify URLs that were not included in the dataset used to train the model using their approach. They also provided a new dataset called "VisualPhish" that may be used to detect visual phishing. This is the first study to use deep learning to detect visual similarity phishing based on pixels. The model captures relevant website elements such as picture content, text layout, and code structure using a combination of computer vision and natural language processing approaches. To assess whether a website is a phishing site, these features are matched to a database of legitimate websites. VisualPhishNet has a high detection rate of 96.42% when evaluated on a dataset of over 4,500 phishing websites, according to the authors. The model was also able to detect previously unknown phishing websites, showing its potential as a zero-day detection tool.

The research "Visual Similarity-based Phishing Detection Using Deep Learning" [17] offers a system that analyses the visual resemblance between a suspected phishing website and a group of legitimate websites using a Convolution Neural Network (CNN). The likeness between the sites is determined by analyzing numerous visual aspects such as color schemes, logos, and layouts. For their experiments, the researchers used two data sets: Phish-IRIS and CityU. They investigated three alternative CNN architectures in their study: Inception-V3, ResNet-50, and DarkNet19. Following the training process, the findings showed that ResNet-50 performed the best, with an accuracy of 97.37% for the Phish-IRIS data set and 97.56% for the CityU data set.

Padmalochan Panda et al. [18] describe a new approach for logo identification that involves extracting logos from internet screenshots or photos and then converting them to grayscale images. Following that, several feature extraction approaches, such as the scale-invariant feature transform (SIFT) and speeded-up robust features (SURF), are used to distinguish distinct logo features. The researchers use machine learning algorithms such as decision trees, Support Vector Machine, K-Nearest Neighbour, and Gaussian Naive Bayes classifiers to improve the logo recognition process. These algorithms efficiently classify logos as authentic or fraudulent. The establishment of a new metric known as the Logo Identification Index (LII) is a novel contribution of this work.

Saad Al-Ahmadi et al [19] suggested a technique to detect phishing websites utilizing both the website picture and the URL in their investigation. They extracted URL and visual features using two convolutional neural networks (CNN). They then integrated them to obtain the results. Their model is 99.7% accurate. Their future work will be to discover a mechanism to automatically determine the shortest URL length and screenshot size in order to optimize performance.

1.1.4 Data sets

There are several numbers of datasets that were used in different researches. VisualPhish [16] Contain 9363 screenshots of PhishTank phishing pages that target 155 websites and 1195 phishing pages. The owners of this dataset [16] have also created a trusted list from those 155 websites by collecting legitimate pages from those . DeltaPhish [14] [16] data set consist with a group having one phishing page in addition to a few other safe websites from the hosting website that has been hijacked. Phish-IRIS dataset [15] was created using phishing pages from the PhishTank website that target 14 websites and a "other" class gathered from the top 300 Alexa websites that represent trustworthy examples outside the trusted list. CityU database [17] consist of 5711 phishing pages for every eight legitimate pages. The database is made up of images of various sizes of web pages. The dataset used in [18] have a combined total of 48 different classes of logo graphics across 21 different well-known websites, there are 538 logos in total.

1.1.5 Graph Neural Networks(GNN)

A graph is a data structure made up of nodes and edges that represent the connections between them. Edges explain the link between nodes, which can be items, people, or places. The direction of edges is governed by the direction of dependencies between nodes, which determines whether they are directed or undirected. The blue circles in the figure represent nodes, while the arrows represent edges. The orientation of the arrows indicates the reliance between two nodes [20].

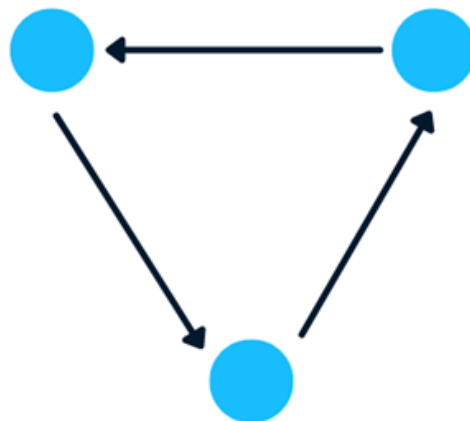


Figure 1.9 Simple directional graph representation between 3 nodes(Source [29])

GNNs can capture graph structure information by considering the connectivity between nodes and their neighbors. This enables them to forecast individual nodes or the entire graph. GNNs can, for example, be used to forecast missing links in a graph, categorize nodes based on their attributes, or perform graph-level tasks like graph classification [20]. GNNs, for example, can be used to describe interpersonal interactions in social media (e.g., name, gender, age, city) [21].

There are many uses for GNNs, but the essential need for GNNs to address a problem is that it be able to represent nodes and edges. Node classification is one of them. Node categorization implies that GNN can add missing information to a node or fill gaps between nodes if information is missing. For example, on a social network, if an account is discovered as a bot, then can use GNNs to find other accounts that may be bot accounts based on the details of the bots we know about. A GNN may be taught to determine whether a node is a bot or not depending on how near the graph embedding of the unknown nodes is to the known nodes [21]. Predicting traffic patterns in cities is another example, which involves modelling the road network as a graph and predicting traffic flow and congestion [22]. GNNs can also be utilized in natural language processing tasks as text categorization, sentiment analysis, and named entity recognition. GNNs can be utilized for computer vision applications such object detection, tracking, and image segmentation [23].



Figure 1.10 Graph representation of people in social media (source [21])

GNNs (Graph Neural Networks) are neural networks that use graph data as input. Unlike other neural network architectures, GNNs can manage complex object relations in non-Euclidean data. The majority of GNNs have a message-passing architecture (MPNN) and can be thought of as a graph-specific generalization of convolution neural networks (CNN). This message transmission technique accepts input in the form of a graph, which can be directed or undirected and has n nodes ($v_i \in V$) and m edges ($(v_i, v_j) \in E$) [24].

$$G = (V, E)$$

Each node and edge can have a vector of features called node and edge features, respectively. The architecture for message passing is divided into four stages. The first three phases are carried out by a single GNN layer, and these stages are repeated for each layer. The fourth stage is the final phase, which should be performed after completing all of the GNN layers [24].

- 1) MESSAGE: Using its own node features, each node generates a message and delivers it to all of its neighbors.
- 2) AGGREGATE: Nodes collect messages from their neighbors and use an aggregation mechanism to combine these messages.
- 3) UPDATE: Existing node attributes are updated by blending them with newly aggregated characteristics, resulting in the development of node embeddings.
- 4) READOUT: This is the process of combining all of the node embeddings into a representation. This model can be used to create predictions by Machine Learning algorithms.

1.1.6 Graph Neural Networks for computer vision tasks

An image is normally considered as a 3-channel array for RGB channels and based on the size of the image Ex- $244 \times 244 \times 3$. To convert an image as a graph the pixels of an image will represent as nodes. The study conducted in 2022, titled "Vision GNN: An Image is Worth Graph of Nodes [25]," introduces a Graph Neural Network (GNN) architecture tailored for image recognition tasks. The proposed Vision GNN architecture depicts an

image as a graph of nodes, with each node representing a local image patch and edges connecting neighboring patches. The GNN analyses the graph by sending messages to update node representations and aggregate node information to build a final image-level representation for classification. The authors tested Vision GNN on different benchmark datasets and found that it performs well on picture classification tasks while being computationally inexpensive. Overall, the research proposes a promising picture recognition approach that combines the strengths of graph neural networks. Furthermore, they demonstrated that this method is more computationally efficient than using CNNs for image classification, implying that if GNN is employed in computer vision, it will be significantly quicker than CNN.



Figure 1.11 The steps of how a graph is constructed Source [25]

In the vision GNN [25] the authors have built two kinds of architectures called isotropic architecture and pyramid architecture to provide a comprehensive comparison with different neural network types. The term "isotropic architecture" refers to the fundamental structure's ability to preserve elements of uniform size and shape over the whole network. The isotropic neural network maintains the same feature size in its core computational body, allowing it to grow easily. The multi-scale nature of the photographs is taken into account by the pyramid design, which captures qualities with progressively lower spatial dimensions as the layers increase in depth. They constructed four different models with varying numbers of hyperparameters for each architecture: tiny, small, medium, and large.

For converting an image into a graph an image with the size of $H \times W \times 3$, it will be divided into N number of patches. Then each patch will be transferred into a feature vector $x_i \in \mathbb{R}^D$, $X = [x_1, x_2, \dots, x_N]$ where D is the feature dimension and $i = 1, 2, \dots, N$. These features are a set of unordered s which are denoted by $V = v_1, v_2, \dots, v_N$. For each node, its K

Nearest value will be found $N(v_i)$. And then add an edge e_{ji} directed from v_j to v_i for all $v_j \in N(v_i)$. Finally, the graph can be denoted as $G = G(X)$ as shown in the Figure 1.11. A detailed diagram of the Vision GNN architecture is illustrated in the Figure 1.12.

Even though no previous author has attempted it, these data demonstrate that GNNs can be used to categorize phishing websites based on their visual characteristics.

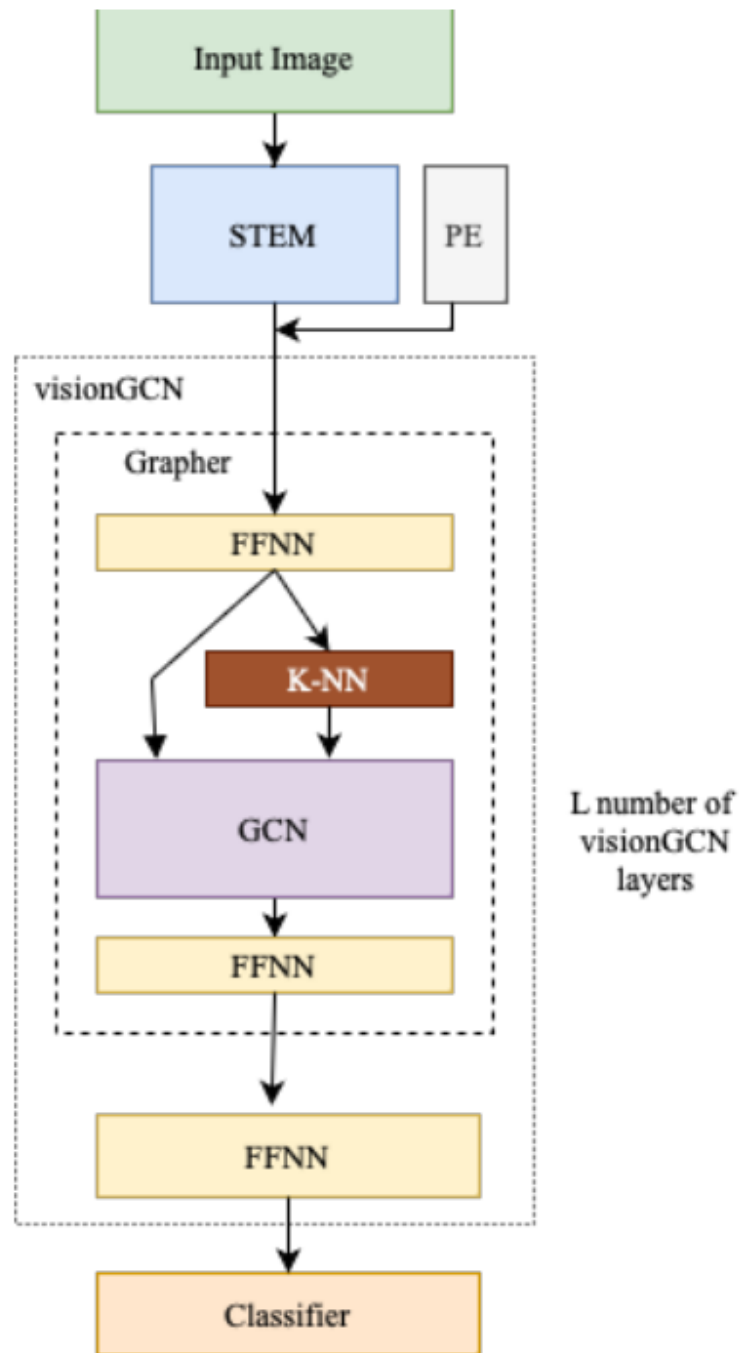


Figure 1.12 Vision GNN model Architecture

1.1.7 Research Gap

The literature review conducted reveals a notable gap in research concerning the utilization of visual features for detecting phishing websites through machine learning techniques. Although there are several methods that leverage visual similarity-based approaches for phishing site detection, a comprehensive exploration of various machine learning algorithms is limited. Past studies have employed decision trees, K-Nearest Neighbor, Support Vector Machine, Gaussian Naive Bayes classifiers, and Convolutional Neural Networks (CNN) for training models with visual features. However, there remains an unaddressed challenge: the development of a method capable of detecting phishing websites beyond the confines of existing datasets. Furthermore, the availability of updated and expansive datasets featuring visual website attributes is lacking. A significant demand exists for high-quality datasets that transcend temporal limitations and specific URLs, enabling researchers to extract visual features more effectively.

Beyond model development, there is a deficiency in the integration of phishing detection into practical applications. Presently, no mobile applications have been developed to seamlessly integrate these phishing detection models into the daily digital experiences of users.

Lastly, optimizing these models for edge computing is a novel avenue that has not yet been explored. Edge computing can potentially enhance the real-time detection of phishing threats, and further research in this direction is warranted.

There appears to be no study currently supporting the use of Graph Neural Networks (GNN) for identifying phishing websites based on photographs of web pages. This highlights a significant gap in the existing research environment, as GNNs have shown promise in a variety of machine learning and graph-based data processing applications. Investigating the potential relevance of GNNs in this context could be a worthwhile scientific endeavor. GNNs are well-known for their capacity to capture complex linkages and dependencies within graph-structured data, making them a potentially ideal alternative for analyzing web page structural aspects and visual attributes. Researchers could pave the path for more advanced and accurate phishing detection approaches that combine image

analysis by building unique GNN-based models tailored to this problem. The features, ML methods, and final implementation used by prior researchers are mentioned in the table below.

According to the above table the researchers have used the text of the webpage, images of the web page, overall appearance, the source code of the website and the website logo had used as features. Several studies had utilized convolutional neural network. There were no studies that had used graph neural network for detecting phishing websites with visual features.

Table 1.1 Research Gap

Reference	Used CNN	Used GNN	Text	Embedded images	Overall appearance/color	Utilized the Source code	Logo	Mobile application
[13]			Yes	Yes	Yes			
[14] <i>DeltaPhish</i>						Yes		
[16] <i>Visualphishnet</i>	Yes		Yes	Yes				
[17]	Yes		Yes	Yes	Yes	Yes		
[18]	Yes				Yes		Yes	
[19]	Yes				Yes		Yes	

Table 1.2 Research Gap

Work	Research Gap
[13]	<p>Limited dataset was used to do testing.</p> <p>Lack of real-world testing.</p> <p>Not defined how the features were extracted.</p> <p>Have not compare other techniques.</p>
[14]	<p>Testing was done using their own custom dataset with limited data.</p>
[16]	<p>Their data set was collected for a specific time so that the performance of the model for new sites is a problem.</p> <p>They have used a specific dataset and the performance are done under controlled situations.</p> <p>Other methods can be used to identify phishing websites using visual features to improve accuracy.</p>
[17]	<p>Used datasets are specific and more generalized datasets are needed for testing.</p> <p>Only have used three types of CNN and can do more testing with CNN and ML.</p> <p>Did not address how they going to detect a website.</p>
[18]	<p>They have only focused on detecting logos and the dataset is a custom dataset that they made for companies in south Asia region.</p> <p>They have not focused on other features that can use as visual similarity based in a website.</p>
[19]	<p>There is lack of ability to detect new websites.</p> <p>More machine learning techniques could have used for testing.</p>

1.1.8 Research problem

The main research problem is how to detect a phishing website using a screenshot by analyzing the visual similarity of the site using GNN. Then the next research question is to determine which elements have the most prominent visual features, such as logos, texts, element placement, distances between elements of the site, and so on, and how to enhance existing datasets with those features, or, if datasets are insufficient, how to collect data with the new proposed features. Another research goal is to investigate how to express the visual aspects of a webpage as a graph.

Either use the complete image of the web page as pixels and represent them as a graph or partition the web page into parts and represent them as graphs, and then experiment to see which option provides the most accuracy. Then for each model the accuracy should be calculated and need to get the best model with the highest accuracy.

After that need to train the model by using techniques such as hyperparameter tuning. Also need to find out compared to earlier CNN-based approaches, will the GNN methodology be more accurate and computationally efficient than CNN-based approaches? The final research objective is to determine how to incorporate the produced model into the mobile app and install the model on the device itself without running the model on a cloud server, so protecting the user's privacy to the greatest extent possible.

1.1.9 Research Objectives

a) Main objectives

To develop an optimized model for mobile application and a web extension which can notify a user about a phishing website. The URL, website features, visual similarity and the email header and the content can be taken as approaches to fulfil this. The characteristics in the URL can be taken to identify if the website is a legitimate one. Visual elements are used to identify the website in a visual-based approach. Text analysis can be used to detect grammar and spelling errors in emails sent by attackers with urgent requests. Email headers can be examined to determine whether the sender's address is genuine.

Finally, website features , such as address bar-based features abnormal behavior features and domain-related features will be used to determine whether the site is phishing or not. The final tool will be able to detect and notify the user about phishing websites based on email content, website URL, visual similarity features, and website features using all the methods described above.

b) Specific Objectives

1. To develop a deep learning model that can detect phishing websites using visual similarity of a websites using GNNs.
2. To compare performance and inference time of GNN.
3. To improve model to be able to detect new phishing websites that are not in the data set that was used for training purposes.
4. To further optimize the size and the inference time of the model for edge computing.
5. To integrate the developed models to a mobile application.

2 METHODOLOGY

In this section the process of training the model to implementing the mobile app will be discussed.

1. The VISUALPHISHNET Dataset which contains screenshots of website that contains both legitimate and phishing websites was used.
2. The dataset was preprocessed by Horizontal flip and vertical flipping were used as image augmentations to reduce over-fitting.
3. The experiments were conducted with graph neural network four models.
4. Models were trained using extracted visual features to distinguish between legitimate and phishing websites.
5. The performance of the trained models was evaluated using appropriate metrics such as accuracy, precision, recall, and F1-score.
6. The model with the most accuracy was selected out of the four models.
7. Optimized the developed model using model pruning, weight clustering and quantization.
8. Then the trained model was integrated with a mobile app using appropriate programming languages and tools such as Python and Native Android.
9. User-friendly interface was designed for the mobile app that displays the classification results and provides appropriate feedback to the user.
10. Test the integrated system on a sample dataset of website screenshots and validate its performance using appropriate metrics.
11. Regularly update the model with new datasets and features to improve its accuracy and effectiveness in detecting phishing websites. Also, maintain the app to ensure its functionality and compatibility with the latest mobile devices and operating systems.

Under this research a novel deep learning architecture for classifying websites as either phishing or benign, utilizing a Graph Neural Network approach will be introduced. This classification is based on analyzing screenshots of the websites. We employ all four versions of the Pyramid architecture of the Vision GNN backbone for the initial experiments until the best accuracy model is picked for our goal of categorizing phishing websites and legal websites using photos of the websites.

The dataset that was used for this study is VISUALPHISH [16]. This data set contains 1195 phishing pages for 155 different websites. This data set also consists of legitimate pages from those 155 websites and can be used as the trusted list but not contain legitimate pages for all the websites. The overall number of screenshots obtained for each of these 155 websites is determined by the total number of hyperlinks recognized on the homepage. The data set's owners searched Alexa's top 500 rated websites, Similar Web4's top 100 websites, and the top 100 websites in phishing-prone industries like banking, finance, and government services, obtaining a total of 400 websites from Similar Web. The 155 websites acquired from the phishing pages' targets were then removed from these lists, and screenshots of the top 60 websites (non-overlapping) from each list were downloaded. The data is divided into three types: a training trusted list of authentic pages, false pages targeting trusted-list websites, and legitimate/harmless test cases from websites outside the trusted list (other domains)

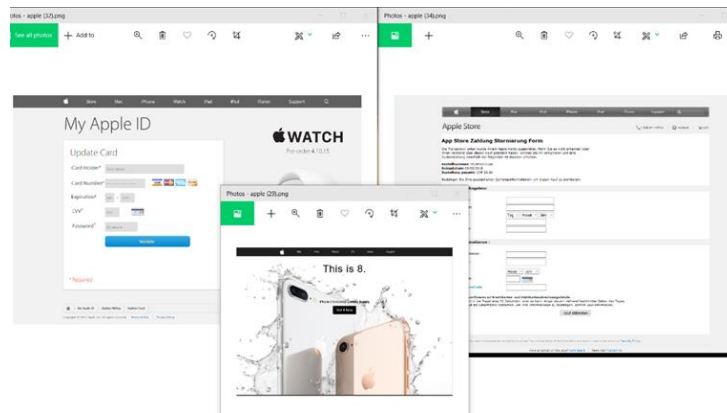


Figure 2.1 Screenshots of same site from VISUALPHISHNET dataset

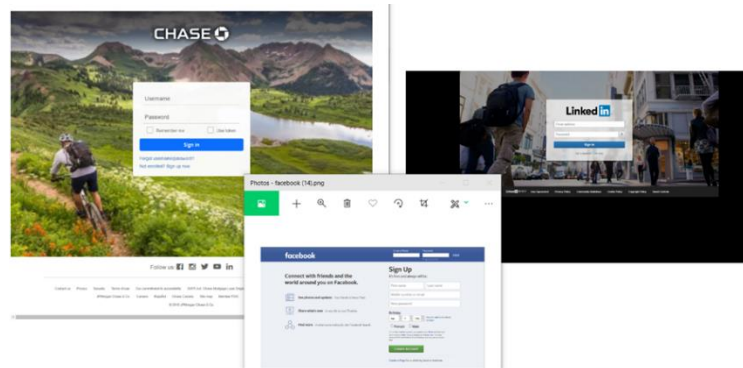


Figure 2.2 Screenshots of different websites from the VISUALPHISHNET dataset

For experimenting RGB photos were scaled to 224*224. The batch size was set to 64. Image augmentations such as horizontal flipping and vertical flipping were utilized to reduce over-fitting. We train the model for 100 epochs with a 25% split. An epoch is the number of times the entire dataset needs to be worked through to learn an algorithm. The total number of data points used in the experiments was 4072, with 3054 train data points and 1018 test data points. Throughout all tests, the total number of epochs was kept constant at 100.

The VISIONGNN [25] model was developed inside the **Timm** model which is a deep learning library in pytorch. This is a collection of computer vision models, layers, utilities, optimizers, schedulers, data loaders. VisionGNN code was implemented to run on a GPU 8 processor. So that by going through the original code the necessary functions and files were extracted and the code was developed.

There are four models in Vision GNN and they are tiny, small, medium and big. These models differentiate from each other from the number of blocks that are in the backbone which is the number of layers K-Nearest neighbors and the number of channels.

Once the model with the highest accuracy was selected it was integrated into an android mobile application.

3 TESTING AND IMPLEMENTATION

Each of the four models were trained by using the VISUALPHISHNET data set and calculate the accuracy.

Model definition

```
def _cfg(url='', **kwargs):
    return {
        'url': url,
        'num_classes': 2, 'input_size': (3, 224, 224), 'pool_size':
None,
        'crop_pct': .9, 'interpolation': 'bicubic',
        'mean': IMAGENET_DEFAULT_MEAN, 'std': IMAGENET_DEFAULT_STD,
        'first_conv': 'patch_embed.proj', 'classifier': 'head',
        **kwargs
    }

default_cfgs = {
    'vig_224_gelu': _cfg(
        mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5),
    ),
    'vig_b_224_gelu': _cfg(
        crop_pct=0.95, mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5),
    ),
}

@register_model
def pvig_ti_224_gelu(pretrained=False, **kwargs):
    class OptInit:
        def __init__(self, num_classes=1000, drop_path_rate=0.0,
**kwargs):
            self.k = 9 # neighbor num (default:9)
            self.conv = 'mr' # graph conv layer {edge, mr}
            self.act = 'gelu' # activation layer {relu, prelu,
leakyrelu, gelu, hswish}
            self.norm = 'batch' # batch or instance normalization
{batch, instance}
            self.bias = True # bias of conv layer True or False
            self.dropout = 0.0 # dropout rate
            self.use_dilation = True # use dilated knn or not
            self.epsilon = 0.2 # stochastic epsilon for gcn
            self.use_stochastic = False # stochastic for gcn, True
or False
```

```

        self.drop_path = drop_path_rate
        self.blocks = [2,2,6,2] # number of basic blocks in the
backbone
        self.channels = [48, 96, 240, 384] # number of channels
of deep features
        self.n_classes = num_classes # Dimension of out_channels
        self.emb_dims = 1024 # Dimension of embeddings

    opt = OptInit(**kwargs)
    model = DeepGCN(opt)
    model.default_cfg = default_cfgs['vig_224_gelu']
    return model

```

Data Loading and Training

Data were trained with a split of 25% which shows in the following code.

```

def train_val_dataset(dataset, val_split=0.25):
    train_idx, val_idx = train_test_split(list(range(len(dataset))),
test_size=val_split)
    datasets = {}
    datasets['train'] = Subset(dataset, train_idx)
    datasets['val'] = Subset(dataset, val_idx)
    return datasets

```

- **vision_gnn_phishing_pyramid_tiny.ipynb**

For this model the blocks and channels that were used are as follows.

```

self.blocks = [2,2,6,2] # number of basic blocks in the backbone
self.channels = [48, 96, 240, 384] # number of channels of deep
features

```

- **vision_gnn_phishing_pyramid_medium.ipynb**

For this model the blocks and channels that were used are as follows.

```

self.blocks = [2,2,16,2] # number of basic blocks in the backbone
self.channels = [96, 192, 384, 768] # number of channels of deep
features

```

- **vision_gnn_phishing_pyramid_small.ipynb**

For this model the blocks and channels that were used are as follows.

```
self.blocks = [2,2,6,2] # number of basic blocks in the
backbone
self.channels = [80, 160, 400, 640] # number of channels of
deep features
```

- **vision_gnn_phishing_pyramid_big.ipynb**

For this model the blocks and channels that were used are as follows.

```
self.blocks = [2,2,18,2] # number of basic blocks in the
backbone
self.channels = [128, 256, 512, 1024] # number of channels of
deep features
```

The accuracy and the implementation of the mobile application will be discussed in the results section. Once the training was done, we were able to find out that the highest accuracy rate is for small model. The accuracy was calculated with $\text{Accuracy} = C/N$ equation. For the small model with the highest accuracy hyperparameter tuning was conducted by hanging following parameters.

- optimizer - AdamW
- starting learning rate - 0.01
- learning rate decay - cosine decay

3.1 Mobile application

Pytorch model was converted in to pytorch mobile version using the following code snippet.

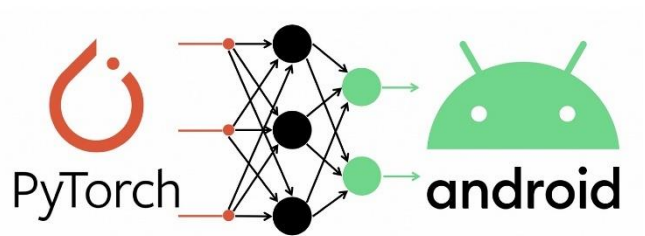


Figure 3.1 Pytorch to android

```
from torch.utils.mobile_optimizer import optimize_for_mobile
example = torch.randn(64, 3, 224, 224)

traced_script_module = torch.jit.trace(model, example)
traced_script_module_optimized =
optimize_for_mobile(traced_script_module)
traced_script_module_optimized._save_for_lite_interpreter("model.pt1")
```

The model was saved as model.pt1 and integrated into android mobile application. The saved model is saved in the app resources. Since the model is running inside the mobile application itself it does not need an internet connection.

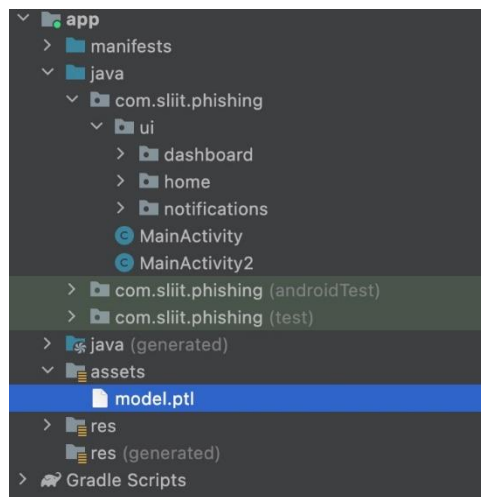


Figure 3.2 PyTorch mobile in the android bundle

4 COMMERCIALIZATION ASPECTS OF THE PRODUCT

Our mobile app is designed to cater to the needs of both basic and advanced mobile users. It is particularly useful for users who deal with many emails and engage in frequent web browsing, as they are often the target of phishing attacks. For basic mobile phone users, our app provides a user-friendly interface and simple functionalities that are easy to navigate. It can be easily downloaded and installed on any basic mobile phone without the need for a high-end device. The app is designed to help users identify and avoid phishing scams, which are becoming increasingly common in today's digital age.

Advanced mobile users, on the other hand, require more robust security features to stay safe from phishing attacks. Our app provides advanced security features such as anti-phishing filters, real-time scanning of emails and websites for potential threats, and proactive alerts that notify users of potential phishing attacks.

Overall, our mobile app is a comprehensive security solution that caters to the needs of both basic and advanced mobile phone users. By providing robust security features and a user-friendly interface, we aim to make mobile phone usage safer and more secure for all users.

We plan to offer our users two different subscription packages for our mobile app.

The first package is a basic subscription, which we plan to charge an annual fee of \$5. This package provides users with the option to upload fifty screenshots per month and 50 emails per month. The second package is our advanced subscription, which includes all the features of the basic package, but unlimited number of scans. We plan to charge an annual fee of \$10 for this package. We believe that both subscription packages offer a great value to users who are concerned about their online security. The basic package provides essential security features that can help protect users from phishing attacks, while the advanced package offers more robust security features for those who require additional protection.

We also understand the importance of transparency when it comes to subscription pricing. Therefore, we will clearly outline the pricing and features of each package on our app's website and within the app itself. We aim to provide our users with the information they need to make an informed decision about which subscription package is right for them.

4.1 Results and Discussion

4.1.1 Results

When trained the four models for each of the model the accuracy curve and mean squared error with epochs were taken to check the accuracy, loss of training and validation loss.

```
import matplotlib.pyplot as plt
num_epochs = num_epochs
def loss_curves(epochs, training_epoch_loss, valid_epoch_loss):
    plt.title('Mean Squared Error v Epochs', fontsize=20)
    plt.xlabel('No. of Epochs', fontsize=18)
    plt.ylabel('Mean Squared Error', fontsize=16)

    plt.plot(epochs, training_epoch_loss, label="Training Loss")
    plt.plot(epochs, valid_epoch_loss, label="Validation Loss")
    plt.legend()
```

```
def accuracy_curves(epochs, val_accuracy):
    plt.title('Validation Accuracy v Epochs', fontsize=20)
    plt.xlabel('No. of Epochs', fontsize=18)
    plt.ylabel('Validation Accuracy', fontsize=16)

    plt.plot(epochs, val_accuracy, label="Validation Accuracy")

    plt.legend()
```

4.1.1.1 Tiny model

For the tiny model Mean Squared Error vs Epochs graph and validation accuracy are illustrated below.

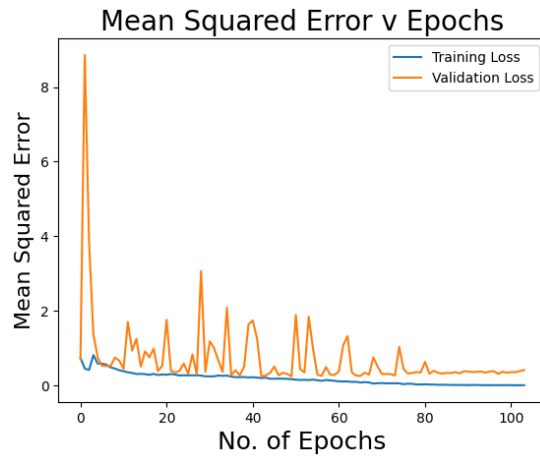


Figure 4.1 tiny model Mean Squared Error vs Epochs.

In the graph the training loss is lower than the validation loss which shows that the model was properly trained, and the model is not overfitted and it is not generalized.

For the tiny model validation accuracy vs Epochs are illustrated below.

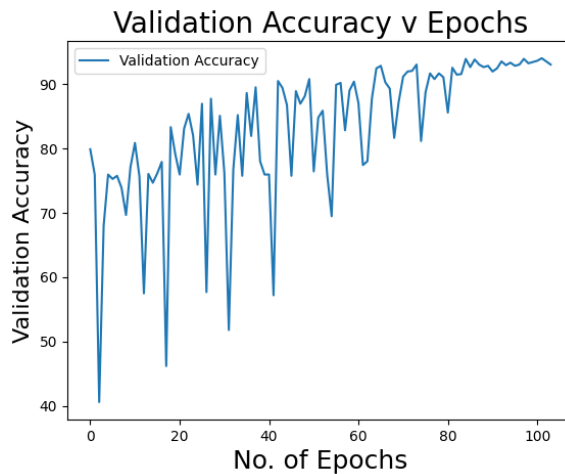


Figure 4.2 Tiny model validation accuracy

4.1.1.2 Small Model

For the small model Mean Squared Error vs Epochs chart and validation accuracy are like below.

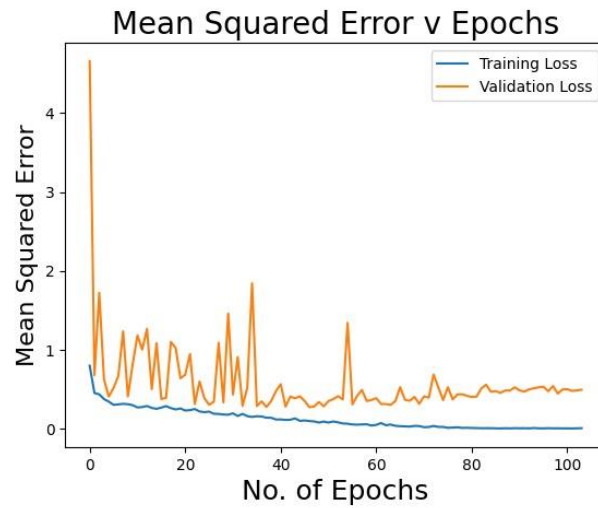


Figure 4.3 Small model Mean Squared Error vs Epochs.

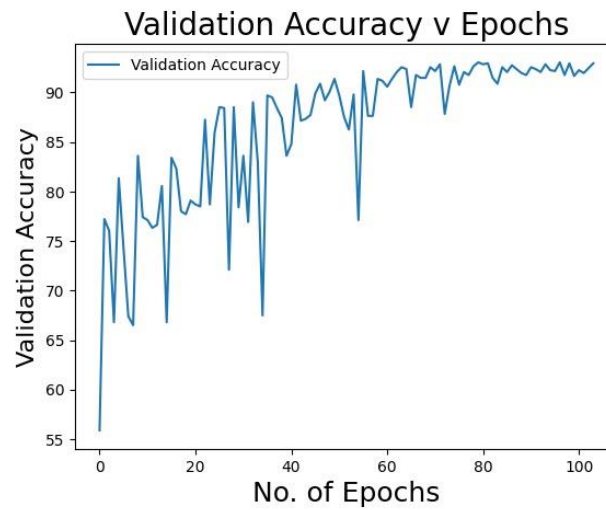


Figure 4.4 Small model validation accuracy

4.1.1.3 Medium Model

For the medium model Mean Squared Error vs Epochs chart and validation accuracy are like below.

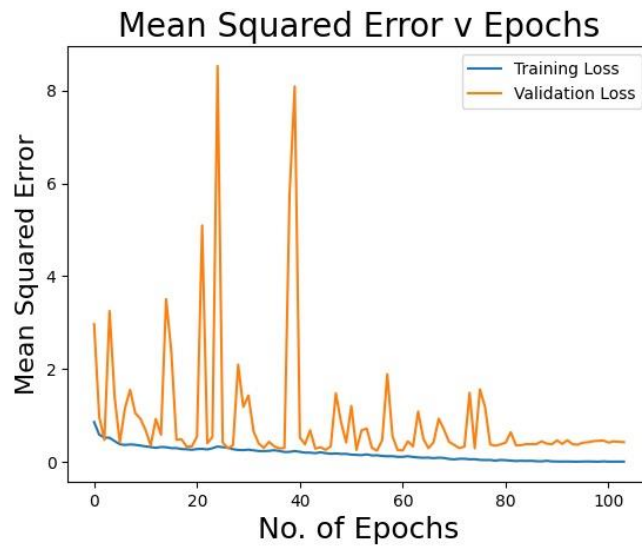


Figure 4.5 Medium model Mean Squared Error vs Epochs

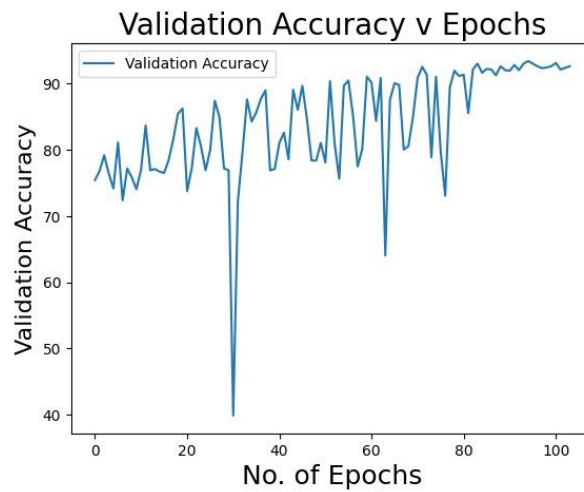


Figure 4.6 Medium model validation accuracy

4.1.1.4 Big Model

For the big model Mean Squared Error vs Epochs chart and validation accuracy are like below.

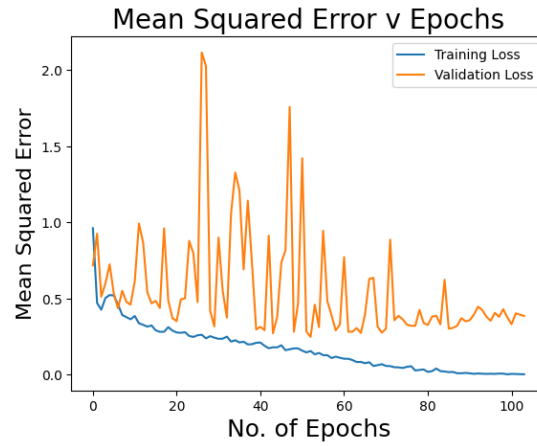


Figure 4.7 Big model mean squared error vs Epochs.

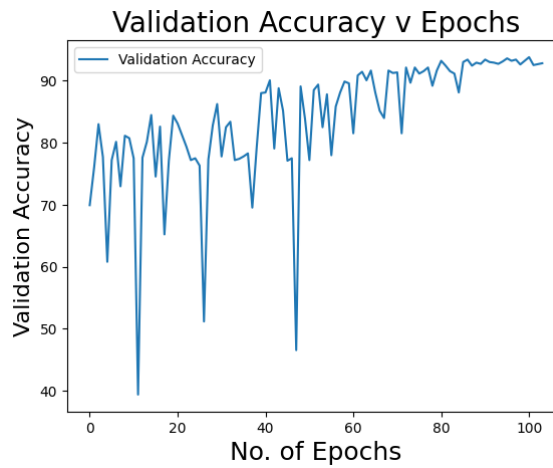


Figure 4.8 Big model validation accuracy

4.1.2 Confusion metrics

The confusion Matrix for these models were also taken. Confusion Metrix is a table which summarizes the performance of a classification algorithm by breaking down the number of correct and incorrect predictions made by the model. It typically consists of four key metrics.

True Positives (TP): In this instance the model correctly predicts the positive class.

True Negatives (TN): The model correctly predicted the negative class.

False Positives (FP): These are case where the model incorrectly predicted the positive class when it should have predicted the negative class. The model produced a "false alarm."

False Negatives (FN): When the model incorrectly predicted the negative class when it should have predicted the positive class. The model failed to identify instances belonging to the class of interest.

In the below diagrams the confusion metrics of each of the models are illustrated.

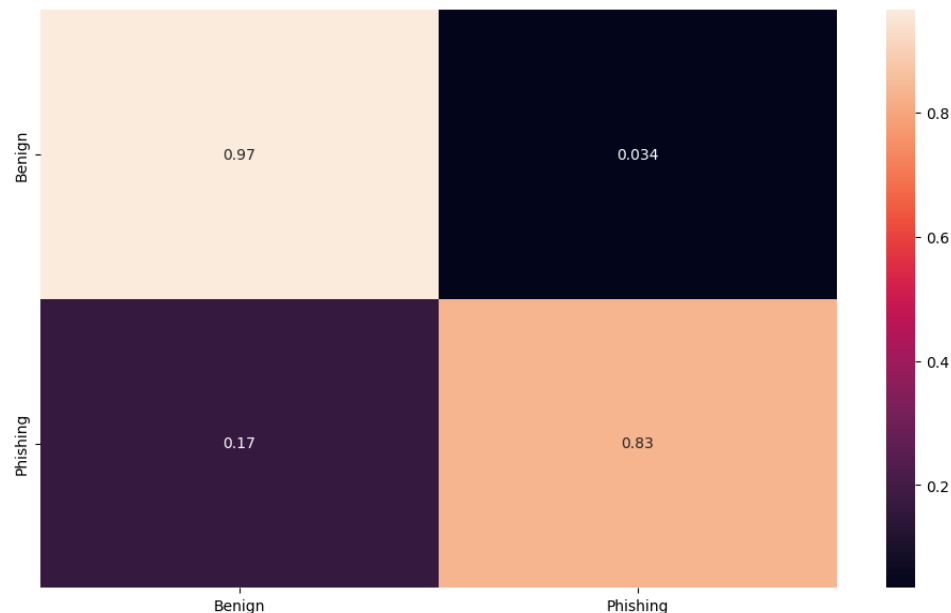


Figure 4.9 Confusion Metrix of the model big

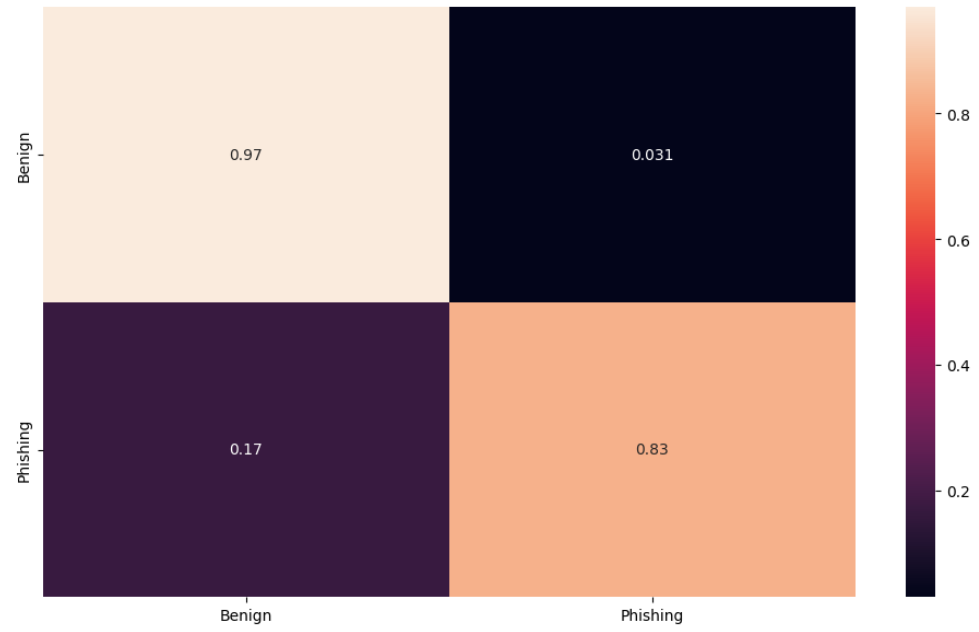


Figure 4.105 Confusion Matrix of the model tiny

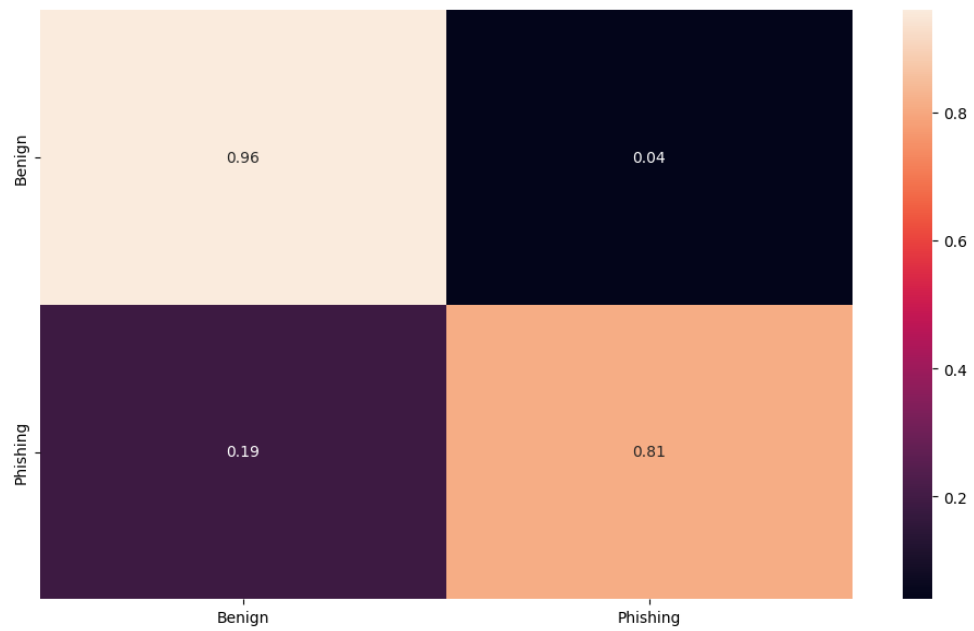


Figure 4.11 Confusion Matrix of the model small

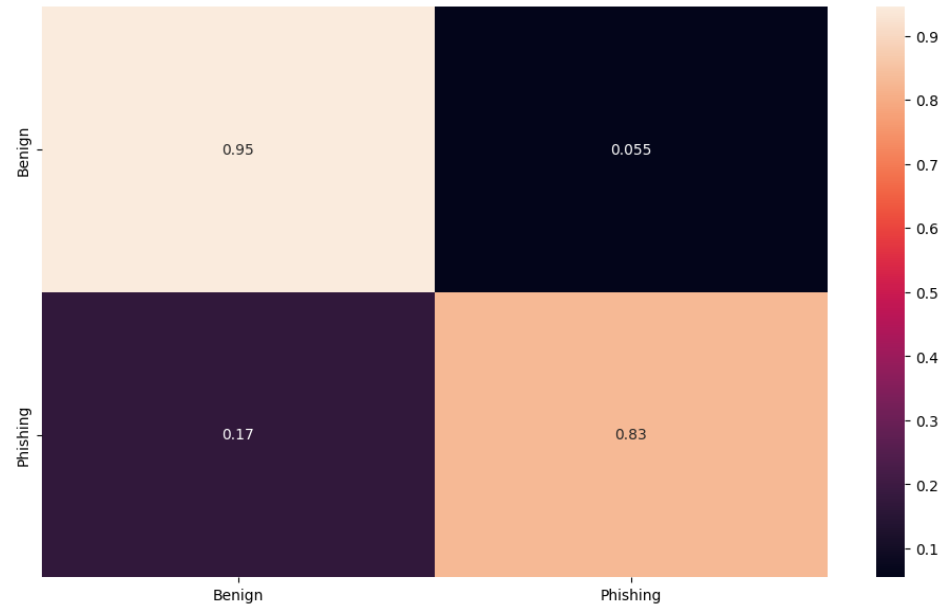


Figure 4.12 Confusion Metrix of the model medium

4.1.3 Mobile Application

The mobile application was designed in a way that the user can input a screenshot of the suspected website. In the user interface under the site detector a user can select the option.

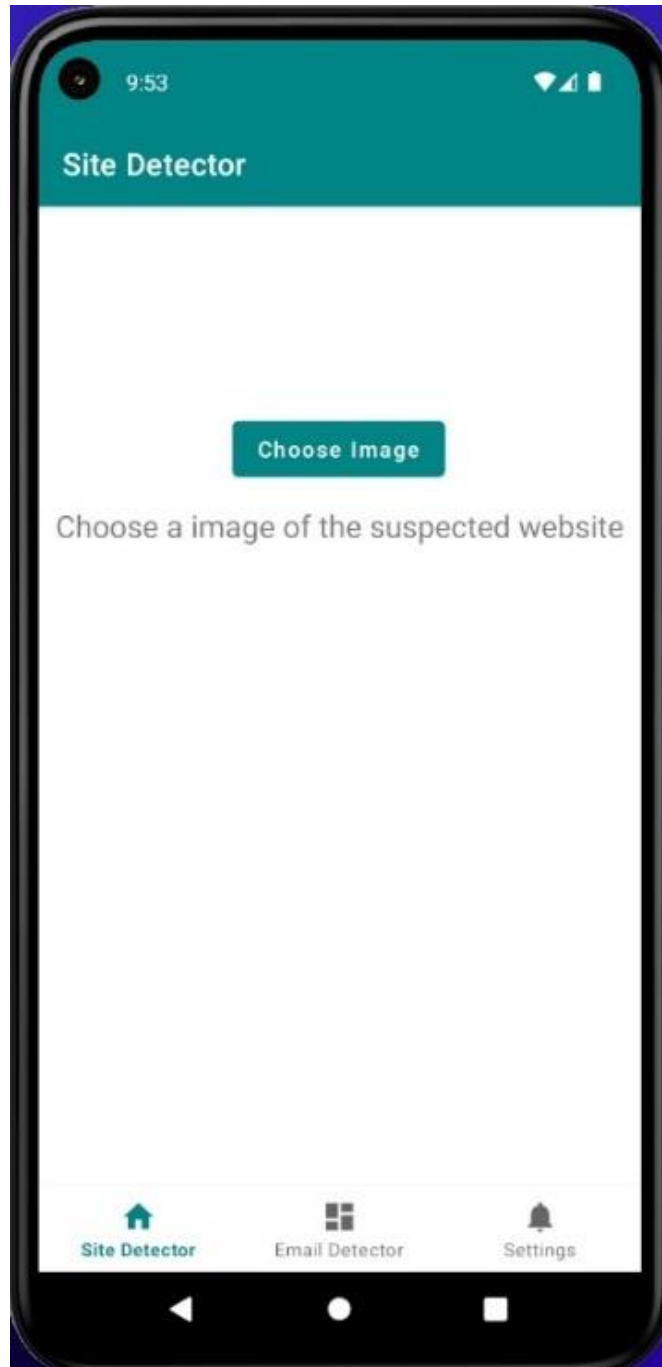


Figure 4.13 Mobile app UI

Then from the gallery they can select the screenshot, or the image.

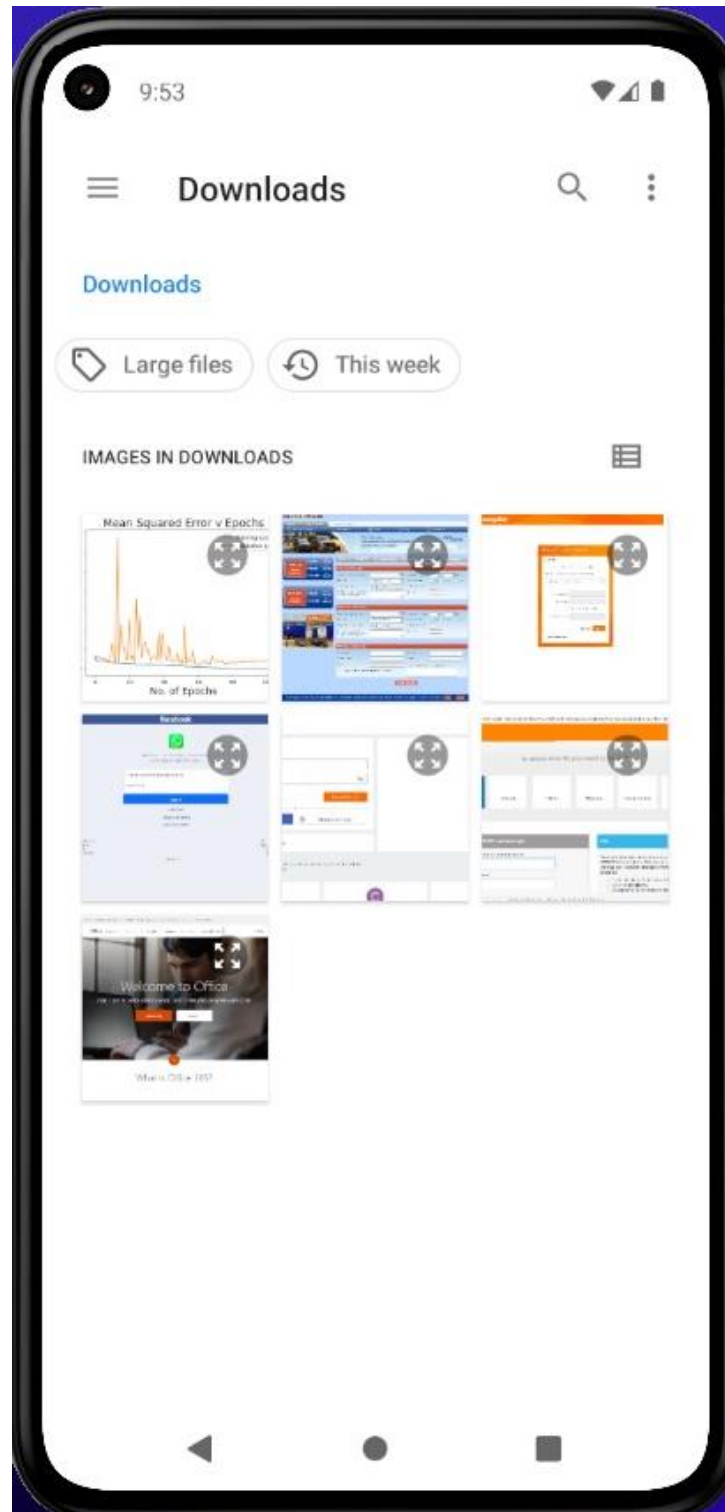


Figure 4.14 Select screenshot from gallery.

Then in the next diagrams can see that the app has given the output for a phishing and a benign site.

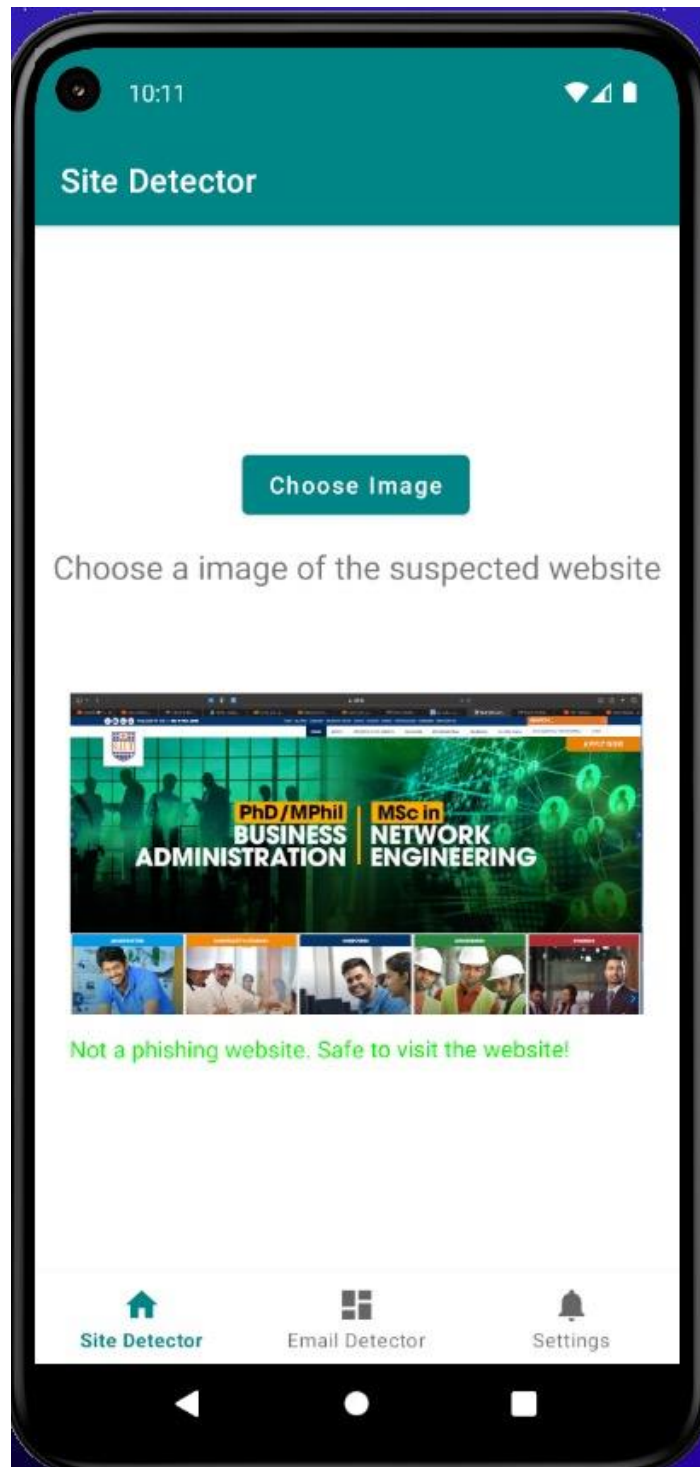


Figure 4.15 Benign site detected correctly.

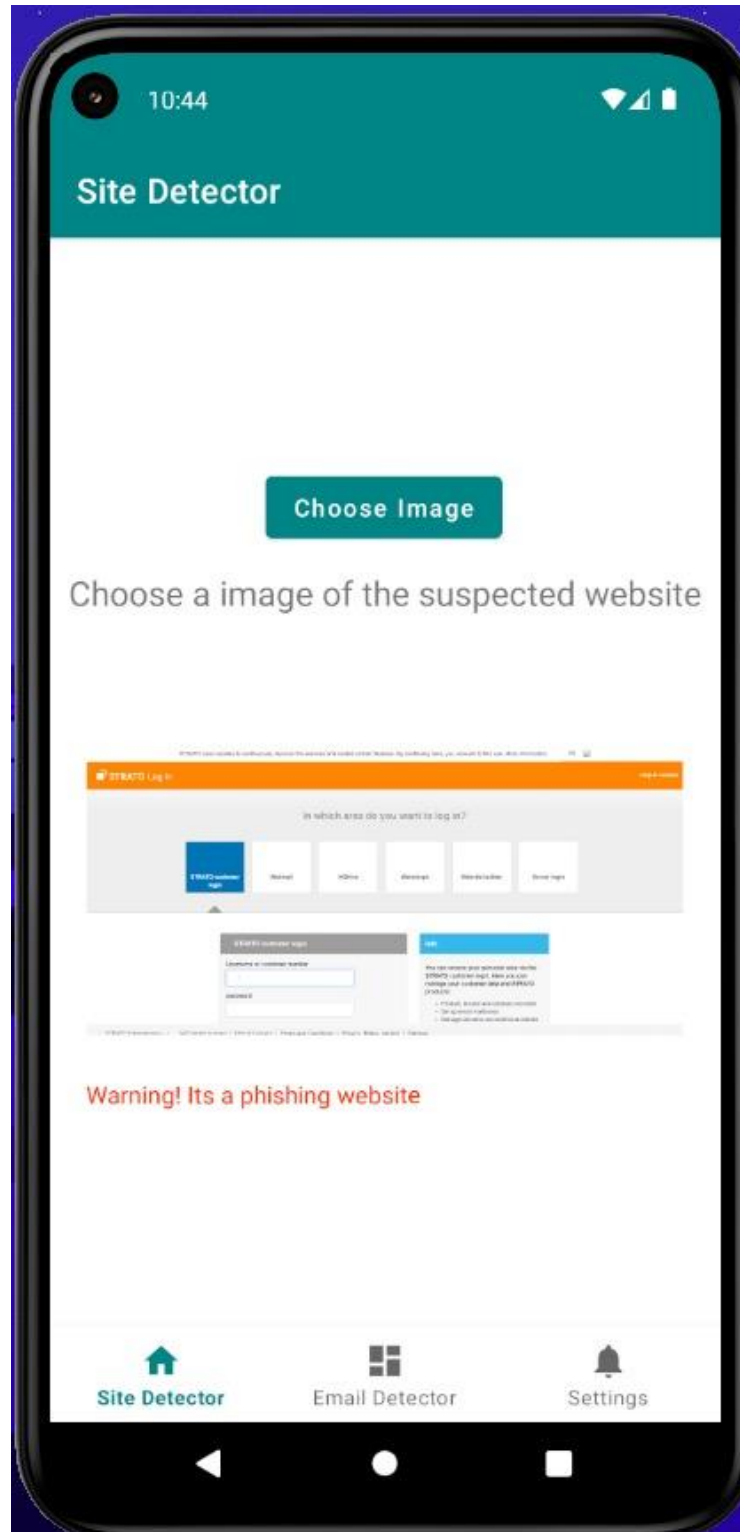


Figure 4.16 Phishing site detected correctly.

5 RESEARCH FINDINGS

Following table shows the test accuracy of the four models. Maximum accuracy was attained with the small architecture, which has fewer parameters. This demonstrates that when the number of parameters increases, the model becomes excessively complex and the accuracy decreases.

Table 5.1 Model accuracies and number of parameters

Model	No of parameters(10^6)	Accuracy
Tiny	9.69	0.93517
Small	26.23	0.97446
Medium	48.50	0.91847
Large	91.96	0.93516699

The small model exhibited the highest accuracy, and examination of the validation loss curves reveals that it converged towards the end of the training process, with the validation loss consistently lower than the training loss. This convergence indicates that the model has been effectively trained and has achieved stability.

Analysis of the confusion matrix further underscores the superiority of the small model in terms of classification performance. This suggests that, for the specific research problem of distinguishing between phishing and benign websites, there is no need for an overly complex model. This finding substantiates the idea that increasing the number of parameters in the model does not necessarily lead to a corresponding increase in model accuracy. According to the confusion matrix, our approach could correctly predict 97% of benign websites and 84% of phishing websites.

The mobile applications' benefits are since the model is running inside the mobile itself it does not need an internet connection. Then the user's privacy of the mobile application will be preserved.

6 CONCLUSION

Phishing websites have emerged as a significant concern for internet users worldwide. The prospect of warning users before they visit a suspicious site or share sensitive information offers immense advantages in mitigating this threat. This project was comprehended to precisely address this challenge, and we successfully achieved our objective.

As the objective of finding the ability of using a graph neural network to detect a phishing website at the end of the project we were able to achieve it. The distinguishing feature of using GNNs for image detection is their ability to break down images into multiple patches, subsequently constructing a graph for analysis. This method stands out for its convenience and effectiveness.

To train the model the VISUALPHISHNET dataset was used. There were four models in the VISIONGNN architecture and for all the four models' accuracy values were calculated and finally the small model with the highest accuracy was selected. Then model was converted into pytorch mobile version and then integrated into the android mobile application. The model was originally 194MB and once it was converted into pytorch mobile version the model size reduced to 110MB. The users are able to use the mobile app offline without an internet connection and they are able to get a notification once they input a screenshot of the phishing website.

In conclusion, our project represents a significant step forward in enhancing online security and empowering users to navigate the digital landscape with greater confidence.

7 REFERENCES

- [1] K. Han, Y. Wang, J. Guo, Y. Tang and E. Wu, "Vision GNN: An Image is Worth Graph of Nodes," 2022.
- [2] Surbhi Gupta, Abhishek Singhal, Akanksha Kapoor, "A Literature Survey on Social Engineering Attacks: Phishing Attack," in *International Conference on Computing, Communication and Automation*, India, 2016.
- [3] A. Lakshmanarao, P. Surya Prabhakara Rao, M. M. Bala Krishna, "Phishing website detection using novel machine learning fusion approach," in *Proceedings of the International Conference on Artificial Intelligence and Smart Systems (ICAIS-2021)*, India, 2021.
- [4] Aya Hashim, Razan Medani, Dr. Tahani Abdalla Attia, "Defences Against web Application Attacks and Detecting Phishing Links Using Machine Learning," in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, Sudan, 2020.
- [5] "PHISHING ACTIVITY TRENDS REPORT 3rd Quarter," APWG, 2022.
- [6] Abdul Basit, Maham Zafar, Abdul Rehman Javed, Zunera Jalil, "A Novel Ensemble Machine Learning Method to," in *IEEE 23rd International Multitopic Conference (INMIC)*, 2020.
- [7] Ankit Kumar Jain and B. B. Gupta, "Phishing Detection: Analysis of Visual Similarity Based Approaches," *WILEY*, p. 2017, 2017.
- [8] R. Dhamija, J. D. Tygar, and M. A. Hearst, "Why phishing works," in *SIGCHI Conference on Human*, 2006.
- [9] Z. Fan, "Detecting and Classifying Phishing Websites by Machine Learning," in *2021 3rd International Conference on Applied Machine Learning (ICAML)*, China, 2021.

- [10] Malak Aljabri ,Samiha Mirza , "Phishing Attacks Detection using Machine Learning and Deep Learning Models," in *2022 7th International Conference on Data Science and Machine Learning Applications (CDMA)* , Saudi Arabia, 2022.
- [11] "Phishing Website Detection Using Random Forest and Support Vector Machine: A Comparison," in *2nd International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, Malaysia, 2021.
- [12] Jaydeep Solanki, Rupesh G. Vaishnav, "Website Phishing Detection using Heuristic Based Approach," *International Research Journal of Engineering and Technology (IRJET)*, vol. 03, no. 05, 2016.
- [13] Ali Aljofey,Qingshan Jiang, Qiang Qu ,Mingqing Huang,Jean-Pierre Niyigena, "An Effective Phishing Detection Model Based on Character Level Convolutional Neural Network from URL," *Electronics*, 2020.
- [14] Eric Medvet,Engin Kirda,Christopher Kruegel, "Visual-Similarity-Based Phishing Detection," in *SecureComm 2008* , Turkey, 2008.
- [15] Igino Corona^{1,2}, Battista Biggio^{1,2}, Matteo Contini²Luca Piras^{1,2}, Roberto Corda²Mauro, Guido Mureddu², "DeltaPhish: Detecting Phishing Webpages in Compromised Websites*," in *ESORICS 2017.*, Italy, 2017.
- [16] F.C. Dalgic¹A.S. Bozkir ² M. Aydos ³, "Phish-IRIS: A New Approach for Vision Based Brand Prediction of Phishing Web Pages via Compact Visual Descriptors," in *ISMSIT 2018*, Ankara Turkey, 2018.
- [17] Sahar Abdelnabi,Katharina Krombholz,Mario Fritz, "VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity," 2020.
- [18] U. Saeed, "Visual similarity-based phishing detection using deep learning," *Journal of Electronic Imaging*, vol. 31, no. 5, 2022.

- [19] Padmalochan Panda ^{1,†}, Alekha Kumar Mishra ^{1,†} and Deepak Puthal , "A Novel Logo Identification Technique for Logo-Based Phishing Detection in Cyber-Physical Systems," *Future Internet*, vol. 14, no. 8, 2022.
- [20] Saad Al-Ahmadi¹ Yasser Alharbi, "DEEP LEARNING TECHNIQUE FOR WEB PHISHING DETECTION COMBINED URL FEATURES AND VISUAL SIMILARITY," *International journal of computer networks and communications(IJCNC)*, vol. 12, no. 5, 2020.
- [21] W. L. Hamilton, "Graph Representation Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1-159.
- [22] Dickson, Ben, "What are graph neural networks (GNN)?," 13 Oct 2021. [Online]. Available: <https://venturebeat.com/ai/what-are-graph-neural-networks-gnn/>.
- [23] Stanford University, "CS224W: Machine Learning with Graphs," 2023. [Online]. Available: <https://web.stanford.edu/class/cs224w/slides/01-intro.pdf>.
- [24] Awan, Abid Ali, "A Comprehensive Introduction to Graph Neural Networks (GNNs)," July 2022. [Online]. Available: <https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial>.
- [25] Tristan Bilot¹, Grégoire Geis¹ and Badis Hammi, "PhishGNN: A Phishing Website Detection Framework using Graph Neural Networks," in *SECRYPT 2022*, Lisbon, 2022.
- [26] K. Willems, "Keras Tutorial: Deep Learning in Python," 04 Feb 2019. [Online]. Available: <https://www.datacamp.com/community/tutorials/deep-learning-python>. [Accessed 08 September 2023].
- [27] David Warbuton,F5 Labs, "2020 phishing and fraud report," F5 labs, 2020.

- [28] A. A. Awan, "A Comprehensive Introduction to Graph Neural Networks (GNNs)," July 2022. [Online]. Available: <https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial>.
- [29] B. Dickson, "What are graph neural networks (GNN)?," 13 Oct 2021. [Online]. Available: <https://venturebeat.com/ai/what-are-graph-neural-networks-gnn/>.

8 APPENDICES

How deep learning works [26]

Deep learning methods use neural networks.

These are the major components of a neural network.

Input layer – each circle represents an input neuron or a feature.

Hidden layers – Computations happen in these layers. There can be any number of hidden layers in the network as required.

Output layers – Output can be a class or a value. The predicted output is provided from these layers.

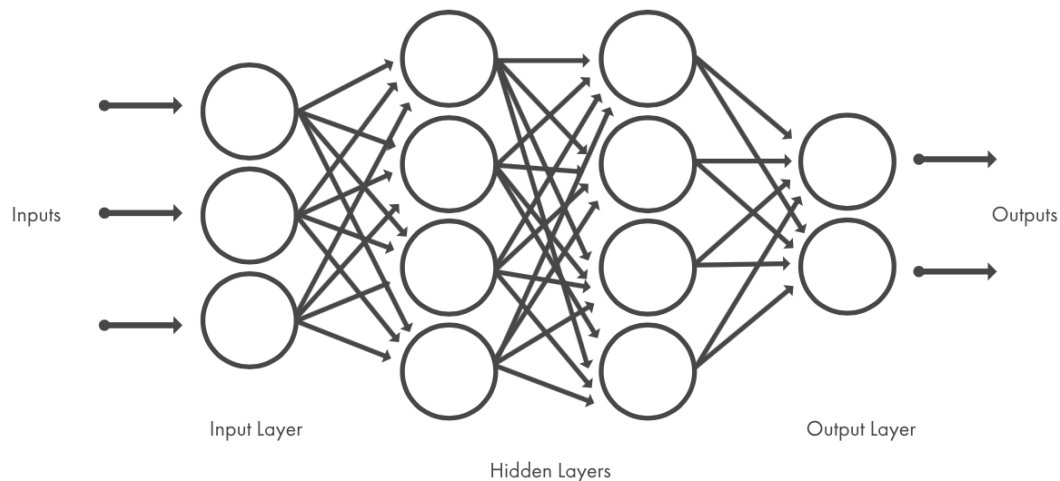


Figure 8.1 Layers (Source [26])

Learning of a neural network happens in two ways.

Forward -propagation – Making a guess about the answer.

Back-propagation – recurring the error by considering actual answer and guessed answer.

Forward propagation [26]

In the following example there are 2 input neurons, one hidden layer and one output neuron. The w_1 , w_2 , w_3 are initialized with random values. Those values are multiplied with weights to form the hidden layer.

$$h_1 = (x_1 * w_1) + (x_2 * w_2)$$

$$h_2 = (x_1 * w_2) + (x_2 * w_2)$$

$$h_3 = (x_1 * w_3) + (x_2 * w_3)$$

Figure 8.2Weights

These values are passed through a nonlinear function called as activation function to from the predicted output.

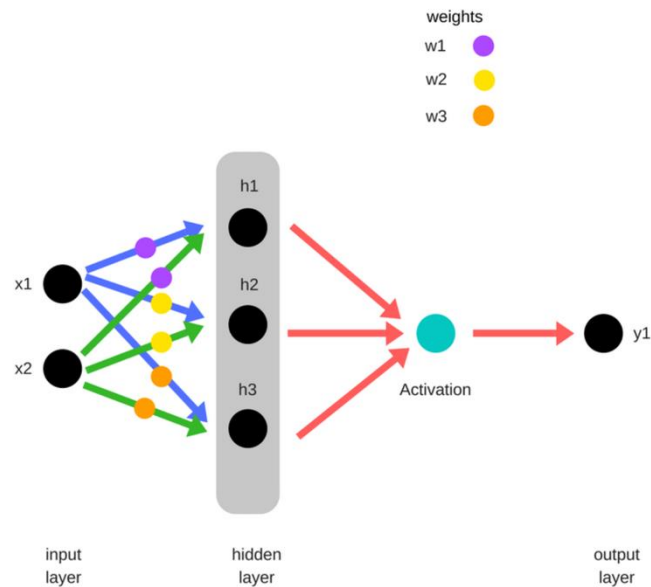


Figure 8.3Nueral network