

# A Deep Learning Approach to Discover Router Firmware Vulnerabilities

**Abstract**—Industrial Internet of Things connected devices are now nearly ubiquitous in the world, and routers are a central point for connecting these Industrial IoT devices. As a router's firmware controls the basic functions of Industrial IoT devices, it is considered the heart of IoT. An Industrial IoT cyberattack can cause huge damage to the connected devices and harm to their owners. Thus, router firmware vulnerability detection is recently becoming an emerging issue in this domain. As a result, an efficient and precise detection tool is a necessity to this domain. However, the firmware dataset collection is the most challenging step as there are no open-source datasets available online. A manual effort was required to verify the states of samples in both the Common Vulnerabilities and Exposures (CVE) and The National Vulnerability Database (NVD) databases as either vulnerable or benign. After verification, 1450 samples were collected. This paper investigates the effectiveness of analyzing home router firmware by employing the use of convolutional neural networks (CNN) and the use of computer vision techniques. More specifically, the collected firmware samples were read as array of byte strings, divided into sub-arrays based on the image's dimensions, then layered on top of one another to produce the images of the firmware; a padding of zeros was added to ensure the images had equal dimensions. The images were then divided by manufacturer and used as inputs of various CNN models to test the accuracy of the individual models. Three statistical filtering algorithms were used on each of the manufacturer's sets to produce multiple versions of said manufacturer's set, totaling to 24 datasets, across 4 manufacturers, with 6 datasets per manufacturer (4 of which were filtered images and 2 being the grayscale and RGB images). The image filter algorithms used include local binary pattern (LBP), histogram-oriented gradients (HOG), and the Gabor filter, which was used on the LBP and HOG sets. After testing all combinations of the filtered/normal datasets with the CNN training model, the highest accuracy filter was determined to be the HOG filter with an average accuracy of 85.81% across all tests and models with results as high as 97.94% when used with the right CNN model.

**Index Terms**—Industrial Internet of Things Cybersecurity, Router Firmware Vulnerability, CNN for IIoT

## I. INTRODUCTION

HOME routers are the endpoints of the internet, where all data flows to and from home networks. If this critical point of failure happens to get compromised many problems may arise, including but not limited to authentications being bypassed, the revealing of sensitive user information, the discovery of vulnerabilities in the home systems and IoT devices, and the exploitation of these vulnerabilities [1]–[8]. With the rapid expand of the IoT market for both residential industrial applications, the necessity to secure firmware cannot be ignored [9]. Routers run a variety of network related

services including DHCP, DNS, FTP, Telnet, etc. Each of these protocols has their own vulnerabilities and can serve as entry points to be exploited by malicious agents; the most critical part is that all of these attacks can be done remotely as well as locally [10]–[12]. Improper firmware design adds another layer of insecurity to routers that could be fixed more easily than the protocols [13]–[15].

The problem of classifying firmware binaries is an important issue that received the attention of many researchers. The authors of [16] used a statistical data flow analysis approach for vulnerability discovery. They searched for common keywords in the front-end used for user input, and then used them as reference points for the back-end binaries which signified input entry points. They then used targeted flow analysis to detect cases of dangerous uses of user input. They used the SaTC library as a base for taint analysis on 39 samples of firmware with results surpassing KARONTE [17], a popular tool for detection of firmware vulnerabilities. The authors in [18], conducted another similar study that dealt with the classification of firmware binaries. It read the string of binary characters of the file as an image, and used convolutional neural networks (CNN) on the images to detect key features in order to classify them. It used a large dataset of over 20,000 samples coupled with 2 image filters across multiple CNN models. After testing, they reported accuracy ranging from 82.34% to 99.97%. Although our manually collected dataset is much smaller and only focuses on firmware of home routers, we are aiming for a similar accuracy score while using different filters that produce distinctive patterns across unique model architectures.

This work explores using machine learning to classify router firmware as either benign or vulnerable in an attempt to identify any vulnerabilities that may exist. A CNN is used on a dataset of over 2500 samples, on 6 models. Several image filters are also used and compared to determine the best CNN model and filter combination. More specifically, our approach is based on converting firmware into images, then using image analysis techniques to classify them. with wither reported high classification accuracy and much lower computing costs. The images converted from the firmware byte strings had noticeable differences and give a positive outlook on the viability of this method as shown in Figure 1. Subsequently, we employ CNN models to identify the features in home router firmware images, thereby classifying them as vulnerable or benign. Six neural network models were created and tested on the different filtered sets to find the best combination of filter and model. After many rounds of testing, the most successful combination had its validation accuracy range between 85% - 98% and was consistently the highest

across all the manufacturers. It is worth mentioning that it is possible to use our approach to detect anomalies in most systems assuming the data could be converted to images and a dataset is trained on an optimized model.

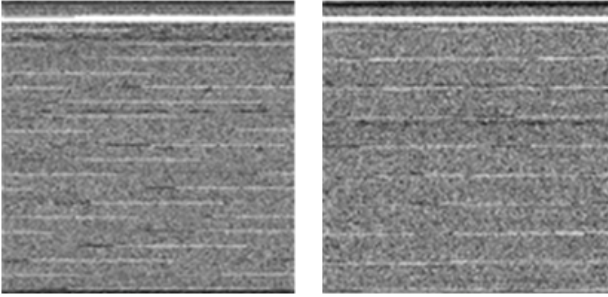


Fig. 1: These images were made from firmware binaries. The image on the left is a benign version of an Asus router model (RT-AX3000) and the image on the right is a vulnerable version of the same router model. There are some noticeable differences between these two images that the CNN would pick up.

Our study makes the following contributions:

- To the best of our knowledge, this is the first effort of applying deep learning to the firmware vulnerability detection problem. We designed a firmware vulnerability classifier using CNN models with leveraging the image processing concepts. The experimental results showed that it is possible to classify router firmware with an accuracy of 89% on average.
- We have investigated the effectiveness of using varying CNN models on images made from router firmware samples, these kinds of tests have already been conducted on similar samples. Using the CNN models in combination with several image filtering algorithms and through testing has concluded that the applied filters can drastically improve the accuracy while reducing the loss to as little as 0.05% in some cases.
- We have used the LBP, HOG, and Gabor filters on a set of over 2500 samples (after augmentation), the results outperform the research projects of much higher sample sizes.

The remainder of the paper is organized as follows. In the next two sections, we discuss the Convolutional neural networks (CNNs), the image types and filters. In the fourth section, we discuss Firmware collection and experimental setup. Evaluation results and findings are discussed in the sixth fifth section. Finally, the last section concludes this research.

## II. CONVOLUTIONAL NEURAL NETWORKS MODEL

Convolutional neural networks (CNNs) are the type of deep learning algorithms that would be most useful for solving this problem, they are used in a variety of applications dealing with image feature classification such as facial recognition. The main structure of most CNNs begins with an input layer where the image is read as a two-dimensional array of pixels, followed by one or more hidden layers where different weights are applied to the pixel values in order to map the image's

features, and ending off with an output layer consisting of fully connected layers that create many-to-many relationships amongst the extracted features and test the effectiveness of the applied weight values [19], [20]. The convolutional layer is the heart of the CNN learning model and is where the bulk of computations occur. The process of convolution consists of applying a filter to the image's pixel values; the filter or "kernel" is usually a  $3 \times 3$  array that consists of the weight values, it is overlapped with the input image and the dot product between the kernel and the image's pixel values is taken. The kernel acts as a sliding window across the entire image, every time it computes the dot product, the resulting value or "node" is added to the outputted feature map [21]. After the convolution, an activation function [22] is applied to the feature map, this is done to normalize the values and add a degree of nonlinearity to the data, thereby allowing the classifier to learn more complex patterns that cannot be learnt through linear representations.

There are multiple activation functions that alter the feature map's data differently, the simplest of which is the linear activation function which applies no transformations. Other well-known nonlinear activation functions [23] include the sigmoid and hyperbolic tangent (tanh) functions, both of which normalize the feature map's values to different ranges. The sigmoid function's normalization range is  $[0, 1]$  and tanh is  $[-1, 1]$ , and it is documented that they do not perform well when normalizing the values that lie around their range's midpoints and decrease the effectiveness of the optimization function at the output layer [24].

The rectified linear unit function (ReLU) is a popular choice in modern CNNs, it increases the effectiveness of the Stochastic gradient descent (SGD), which needs more detailed values that are not diluted by the normalization of the sigmoid and tanh functions. The ReLU function is simple as it only turns all values that are lower than zero into zero, and keeps everything else the same, ultimately retaining more information and allowing more complex relationships in the data to be learned. As such, this was used as the activation function in this article.

The pooling layer is responsible for decreasing the size of the feature map while retaining the structure and dominant features of the data, it has an important role to play in reducing the size and computational cost of processing the data. During this process, a sliding window is moved across the feature map and the maximum value or the average value of the window is taken, resulting in a smaller and compressed feature map [25]. We chose to use the max pooling in this study because it would retain the most significant values of the feature map, while the average pooling method would dilute the feature map's values as shown in Figure 2. It shows how Max pooling is used to condense the important/dominant data of a feature map, thereby removing noise data that may make it harder for the model to learn. Pooling layers are mainly used to reduce the size of the training data and computational power needed to train the model.

The output layer is comprised of fully connected layer(s), these layers represent the data as a one-dimensional array in contrast to the two-dimensional representation that occurs

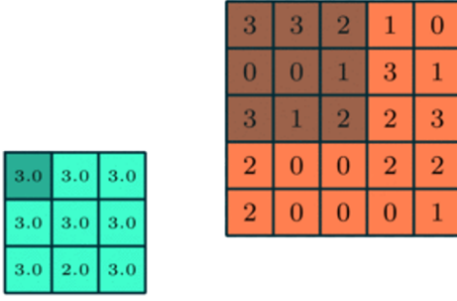


Fig. 2: Max pooling

in the convolutional layers, meaning that the feature maps would have to be flattened before going through the output layer. In the fully connected layers, all nodes are connected to all other nodes in the next fully connected layer, creating many-to-many relationship between the nodes or features and finally the different classes; it allows the model to learn complex nonlinear relationships between the features and the classes [25].

The optimization function is placed at the very end of the CNN output layer and is responsible for training the model by updating the weights until the optimal filters are found. The optimization function that is used in this study is the SGD. It measures the change in weight with regard to the change in error. The goal is to find the minimum of the loss function where the gradient or slope reaches close to zero, which would signify the least number of errors. The weight values are updated based on the learning rate (it controls magnitude of change) and then they are backpropagated through the network and tested again with every epoch allowing the SGD to optimize further [26].

### III. IMAGE TYPES AND FILTERS

In the field of computer vision, image texture can be defined as the variation of pixel brightness intensity in terms of space/distance, and is the main way to define objects or concepts in a given image. Texture analysis is part of computer vision, and deals with object or feature recognition, medical image analysis, pattern recognition, etc. Texture analysis has many approaches to detect image features and can be further divided into four sub-categories: statistical methods, structural methods, model-based methods, and transform based methods. The statistical approach [27] divides an image into separate areas and derives various statistics from the surrounding pixel values in each area and would be the best fit for this study. Other methods such as structural analysis which deal with recognizing repeating patterns are not optimized of this.

#### A. Making RGB and Grayscale Images

To convert the firmware binary files into images, a simple script was made in python. The program reads the bytes of the binary file into a buffer, the buffer is then converted into a one-dimensional *unit8* array which consists of the byte values [0,255]. In order to make a square (equally sized) two-dimensional array from it, the square root of the size

of the array is taken and then used in a ceiling function to obtain the width and height of the new  $2-D$  array; the new size is calculated and the extra space is padded with zeros in *unit8* format. Finally, the resized and padded  $1-D$  array is reshaped into a  $2-D$  through the numpy reshape module and then converted to an image using pillow's `image.fromarray()` method.

The model takes in a byte stream and makes a  $[n \times n]$  dimensional image array; the extra padding at the end is to make sure the 2D image array has a square shape. We turned the firmware to images because from the beginning. We wanted to test if CNNs could be effective in classifying them as in other applications like facial recognition.

Figure 3 shows one of the images converted from Asus's ZenWiFi series. There are distinct horizontal bands on the top section with less defined diagonal bands the CNN model would easily detect these and others that are not discernable by the human eye.

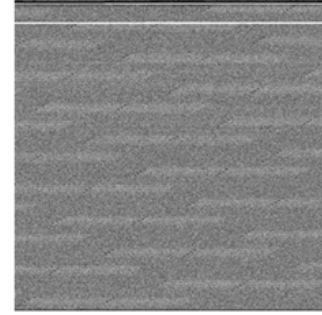


Fig. 3: Image from Asus's ZenWiFi series

#### B. HOG Filter

Histogram oriented gradients specialize in detecting the shape and structure of an image's features. This is in contrast to other feature extraction methods that look for edge information only, which HOG can also find through extracted gradient and orientation information. The gradient information is obtained through dividing the image into patches or areas. Then looking at the center pixel and the ones surrounding it within the area - in different directions and varying lengths - the difference of pixel intensity is computed to get the gradient of the directions. After the gradients have been obtained, the gradient magnitudes are calculated through the Pythagoras theorem while the directions are obtained through the arctan function [28]. Figure 4 shows the HOG filter that is able to condense an image into its local directional components in terms of pixel intensity. It creates an image with distinct and well-defined patches. Giving the grainy firmware images higher definition using this model should make it easier to learn.

#### C. LBP Filter

The local binary pattern is a widely used algorithm for facial recognition and has been performing well ever since its creation in the 1940's [29]. This algorithm works by dividing the image into areas or patches, usually  $3 \times 3$  and comparing the center pixel of each area to its surrounding pixels. If the surrounding pixel's intensity value is above or equal to



Fig. 4: HOG Filter

the center pixel's value, then the surrounding pixel value is replaced by 1. Otherwise, it is replaced by 0. At this point, the area or matrix is converted from byte values into binary with the center pixel value removed. The surrounding pixel's binary values are concatenated in order (left to right, top to bottom) and the concatenated value is converted from byte to integer, and placed as the center pixel's new value. This continues across all divided areas of the image, with the final result being a new image containing highlighted features of the original image [30]. The LBP algorithm is explained in Figure 5. It illustrates how an image's pixel values are compared to the central pixel, then converted to binary, and finally into a decimal value. This process uses a sliding window and repeats for all of the image's patches.

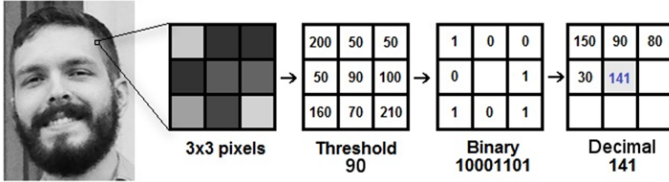


Fig. 5: The LBP algorithm: How an image's pixel values are converted to a decimal value

#### D. Gabor Filter

The Gabor filter is a type of linear filter that is modelled after the human visual cortex, these filters have been shown to excel in detecting features that are localized spatially and frequency wise. We believe that the vulnerabilities in the firmware would exhibit a different pixel pattern then the benign versions, such as having some kind of sanitization step before passing user inputs. Its boasted ability to discern features based on their spatial location and orientation seems like a good fit for analyzing vulnerable or malicious code in firmware images and will be tested. Gabor filters are represented as sinusoidal signals of a particular frequency and orientation that can detect changes in the image's texture [31]. They are modelled based on the equation below.

$$g(x, y, \lambda, \theta, \Psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \Psi\right)\right) \quad (1)$$

The inputs of this equation with a description of their function are listed below: Kernel size: this is the size of the Gabor filter and will be set to a  $3 \times 3$  matrix to extract finer details. Wavelength ( $\lambda$ ): this controls the wavelength of the sinusoidal component, in other words the width of the filter strips. Sigma ( $\sigma$ ): this controls the size of the Gabor envelope, if made too

large, it could miss the small features entirely while filtering, for these experiments, it was set to 3. Orientation ( $\theta$ ): this controls the orientation of the Gabor strips and will filter the features in those orientations; it was set to  $90^\circ$  for a horizontal alignment of the filter. Phase offset ( $\Psi$ ): this defines the symmetry and was set to 0 to get a symmetric filter. Aspect ratio ( $\gamma$ ): this sets the ellipticity of the filter and was set to 0.5 to make it a thinner and narrower for more detailed features.

#### E. The Impact of Filters

The reason that filters were used in the first place was to test if they would have an effect on the predictions of the models and if so, which one would benefit the model's predictions the most. Image filters from statistical texture analysis were chosen because they extract statistical information on surrounding pixels in an area; other methods that focus on extracting repeating pixel patterns would be less useful. The main question to be answered was: "would the statistical data be more useful for training the model than the raw pixels?".

LBP, GABOR, and HOG filters all stem from the statistical analysis approach and each one compresses pixel areas in the image based on each filter's algorithms, resulting in a smaller/compressed image while still retaining relevant statistical information.

The HOG and GABOR filters proved to be the most effective ; when used on its own, the HOG filter achieved a highest overall accuracy, even higher than the RGB images (which had the highest accuracy of the unfiltered images) .

### IV. FIRMWARE COLLECTION AND EXPERIMENTAL SETUP

#### A. Firmware Samples

The first step was to collect the firmware samples. We have manually collected this dataset. The CVE and NVD vulnerability database websites were used to search for router firmware vulnerabilities, afterwards we have visited vendors' websites to search for the vulnerable versions. The vendors' sites would mostly feature only the newest benign versions and we had to use external repositories to acquire the vulnerable versions. The entire process took longer than expected, due to most of the firmware being unavailable on the parent site. At the end, we have managed to acquire samples for 4 main router companies (Asus, Buffalo, D-Link, and Netgear) since they had most of the entries on the CVE and NVD sites while other router companies had too few and not suitable for a large dataset. We have managed to collect over 2400 samples. The dataset is divided by manufacturer and further divided by vulnerable and benign samples for said manufacturer. The amounts of samples per category are displayed in Table I.

Ultimately, the dataset was divided by router manufacturer for a total of 4 sets; the D-Link dataset had the highest number of samples, totaling to 891, followed by Asus with a sample size of 612, then Buffalo with 523, and lastly Netgear with 380 samples. Various router models spanning different versions were collected, upon preliminary testing it was determined that some of them would be removed due to being underrepresented in the sets and ultimately decreasing the validation accuracy. The main router model firmware for

TABLE I: Dataset Statistics

		Training Set Samples	Testing Set Samples	Total	Total
Asus	Benign	140	67	207	612
	Vulnerable	301	104	405	
Buffalo	Benign	121	66	187	523
	Vulnerable	248	88	336	
D-Link	Benign	181	92	273	891
	Vulnerable	758	160	618	
Netgear	Benign	66	41	107	380
	Vulnerable	209	64	273	

Asus include the DSL, FW-RT, and the RT series, the buffalo firmware includes the wex, whr, wmr, etc. The D-link router models include DAP, DIR, DSL, and DSR. The Netgear router models include GC108, GS108, Rxxxx.

### B. Preprocessing

First, all of the firmware binaries were converted into two separate sets, one grayscale and one RGB by using the image converter python script. Converting the entirety of the dataset to RGB images was considerably longer than converting to greyscale, taking around three times longer. A python program was made to automatically filter the image files using the HOG, LBP, and Gabor filters. These filters with the exception of Gabor only filter 2D grayscale image arrays and the entire process of filtering the grayscale dataset took days to complete with the HOG filter taking the longest of the three. The end result was 24 datasets in total, with 6 for each router manufacturer. Each of the six datasets represent the different image types and filters (Grayscale, RGB, LBP, HOG, LBP+Gabor, HOG+Gabor). Later, during the testing, we could then increase the validation accuracy by decreasing the overfitting of the training set by augmenting the images. Thus, creating more samples that differ from the original.

### C. Assumption

The possibility that the newest benign versions could potentially be classified as vulnerable in the future was apparent from the beginning of this project. We decided to proceed with the explicit assumption that any firmware that was not listed in the nvd archive was to be classified as benign. The rationale for this was that if the vulnerable firmware contained specific pixel patterns that would map to known vulnerabilities, the corresponding benign versions should be missing said patterns.

Using this approach on the dataset containing firmware of multiple routers for a manufacturer could lead to a benign version of router firmware that contains an undiscovered vulnerability that another benign router version patched and would produce noise.

## V. TESTING AND RESULTS

### A. Preliminary Experiments and Findings

The first round of tests was done on the grayscale image dataset, the purpose of these tests was to figure out what kind of CNN model was going to work for this type of classification problem, and whether we could get reasonable accuracy from the tests. In addition, we are interested to learn

more about the CNN's hyper parameters and layers, and how they affect the training and accuracy values for our dataset. The first test featured a basic and recommended CNN model, with one hidden layer consisting of one 2D convolution layer with a filter size of  $3 \times 3$  and 32 filters along with a ReLU as the activation function followed by a max pooling layer and two dense layers for the output with the recommended Adam optimizer.

The model was trained on a binary set of classification data consisting of all the company's firmware images, placed in one folder labelled 'benign' and another named 'vulnerable'. This was done over 30 epochs. The results were underwhelming, yielding a final validation accuracy of 47.427%. It took 4 epochs on average to reach the peak validation accuracy and then it had infrequent and small spikes up to 49%. After testing with deeper CNN models having a depth of 2 and 3 convolutional layers and different sets of hyper parameters (kernel sizes, number of filters, and different activation functions), the end result was always the same validation accuracy of 47%. At this point, it seemed like the problem was with the dataset and its distribution. After going through the dataset and removing any router model firmware that only had one or two versions/representations, an updated dataset was made. This was done to streamline the dataset and remove most of the variability that came from a lone router model's firmware that may have interfered with the learning.

Testing with the updated dataset on the first CNN of depth 1 gave a slightly better validation accuracy of 52%. At this point, it was clear that the problem was with the dataset. Comparing the loss of the training and validation sets, it seemed like the model was learning the training set well with its loss decreasing rapidly, but it still struggled with the validation set due to its loss increasing, while fluctuating. This is a clear sign of overfitting, a situation where the validation dataset samples are underrepresented and do not fully mirror the training set samples in the distribution of firmware models. The ways to fix overfitting include implementing dropout layers mainly after the dense layers, using a lower learning rate, including batch normalization, and altering the datasets. The problem was that some router models had many vulnerable versions of firmware but only one or two benign versions for the same model in which case we put them into the training sets for better feature learning. Instead of outright deleting them from the sets the only other option was to use image augmentation to artificially increase their presence and allow us to represent them in both, the training and validation datasets.

The image augmentation was done through the use of the Keras image preprocessing module called *ImageDataGenerator*. This module allows images to be resized, rescaled, rotated, shifted, and zoomed in or out. For the purposes of this study, we decided to apply a vertical shift with a wraparound, along with a slight zoom in. This translates the image with similar structure and features from the firmware, but in different positions with the only concerns stemming from the zoom in which would alter the pixel's values and feature structure to enlarge the features.

Testing the same CNN model of depth 1 as in the previous tests with the augmented dataset had yielded a validation



accuracy of 64% which was an improvement. At this point, we tried playing with different augmented datasets along with different CNN models and hyper parameters. After observing the results, we took note the following changes: Testing the CNN model on a categorical dataset consisting of all the companies in one dataset but in different categories (ex.  $1D-link_vulnerable$ ,  $1D-link_benign$ ,  $1Asus_vulnerable$ , ...) results in lower validation accuracy initially, but has a strong upwards trend that eventually surpasses the binary classification's validation accuracy (where all company's benign are in one set, etc.). It also had more stable loss values. Testing each router company's firmware individually as a binary dataset, resulted in the highest accuracy of all by far. This is when we also discovered that the Netgear dataset was the hardest to learn in some cases, and was the main contributor to the low results in the previous tests.

### B. The impact of Parameters

Upon conducting multiple experiments of varying hyper parameters and observing the resultant accuracy, we were able to come to some conclusions on their effectiveness. Before conducting the main tests, we tested the impacts of changing the number of filters, using different optimizers and their parameters, implementing dropouts, and the number of dense units.

1) *Number of Filters*: Changing the number of filters had noticeable effects on the validation accuracy. We began by using the recommended filter size of 32 and readjusting it after testing in an attempt to increase the accuracy. The accuracy was higher with models that used fewer filters. Such an increase in accuracy with a decrease of the number of filters continued until the filter number was set too low and the accuracy began to slightly decrease. The highest accuracy came from using 5 filters on the first convolutional layer and with deeper CNN models, doubling the filters at each convolutional layer gave optimal results as shown in Figure 6. This suggests that the images of firmware do not have much complex features that define them and are relatively easy to learn. As shown in Figure 6, both of these tests used the same model and manufacturer with the first image having a higher filter parameter set. While the second image used the optimal amount specified. The second image's test produced an accuracy 2% higher, while not much, this trend was consistent.

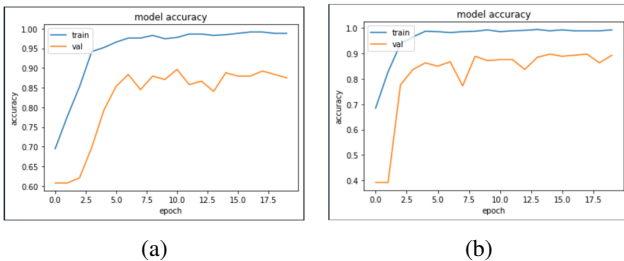


Fig. 6: These tests used the same model and manufacturer with the first image having a higher filter parameter set while the second image used the optimal amount specified.

2) *Batch normalization layers*: Using batch normalization layers after pooling helped increase the accuracy in most cases, although sometimes resulted in a slight decrease of around 2%; another notable difference was in the fluctuations of accuracy. Using batch normalization layers, the accuracy fluctuations were lower in magnitude, giving more stability to the learning (see Fig. 7). It was decided that using batch normalization in the final tests was best due to smaller fluctuations and would give a clearer picture of the CNN model's performance. Fig. 7 shows the accuracy results having used no batch normalization, while the second result shows results with batch normalization layers. This is a case when no batch normalization layers proved to be more effective, giving a 3% accuracy increase. These improvements were not across the board. Using no batch normalization layers gave much bigger fluctuations overall.

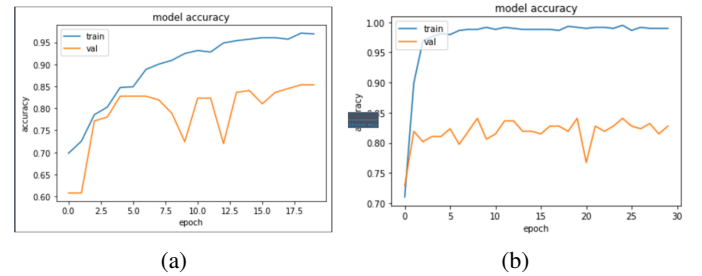


Fig. 7: The impact of batch normalization

3) *Dropout layer*: Implementing a dropout layer after the dense layers gave higher accuracy in almost all cases (see Fig. 9). Each of the manufacturers and models required a different dropout percentage to produce the optimal accuracy results, for example D-Link needed a 30% dropout while Asus did best with a 25% dropout. It was decided to drop the dropout layers from the final testing phase due to their variability in performance between the manufacturers and different models. In future testing with more resources, we would like to revisit implementing the dropouts as they produce higher accuracy, but need much more management and overhead.

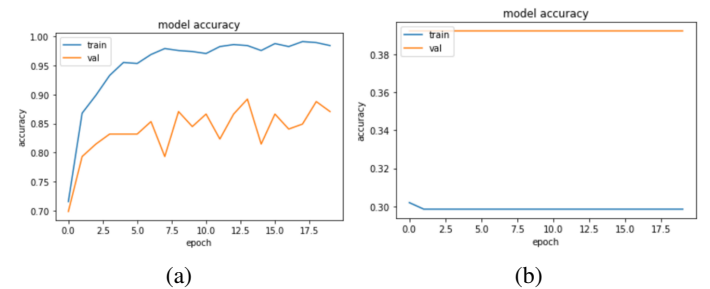


Fig. 8: The impact of dropout layer

The SGD optimizer was used for this study due to being a popular choice in most CNNs and adjusting its learning rate parameter was the key to a high accuracy. The learning rate was set to the recommended setting of 0.001, and proved to be the best value. Lower learning rates were used and resulted in lower increases in accuracy per epoch, and a slighter lower accuracy in the end, while the higher learning rates decreased

the accuracy altogether (see Fig. 8). This shows that learning the firmware's defining features requires a lower learning rate with smaller adjustments to the filters. Fig. 8 shows when dropout layers are implemented, the regular model used in the main tests with no dropouts and 0.001 learning rate, and the last image is when a higher learning rate of 0.09 is used. It shows smaller fluctuations but a lower accuracy than the first, to simplify testing, models with no dropouts were used.

4) *Dense layer*: Much like the number of filters, the number of dense layer units needed calibration for higher accuracy, and began to fall off past the optimal value. Preliminary testing began using the recommended 64 dense layer units. From there, decreasing the dense units resulted in lower accuracy, while increasing them to 128 gave the highest accuracy (see Fig. 9). Implementing additional dense layers resulted in lower accuracy, the best overall performance came from a single dense layer consisting of 128 units. Fig. 9 the first image represents the results from using more dense layer units (256), the second image is from using the optimal number (128) and gave a 5% increase in validation accuracy; the tests using the optimal number of dense units gave better accuracy overall.

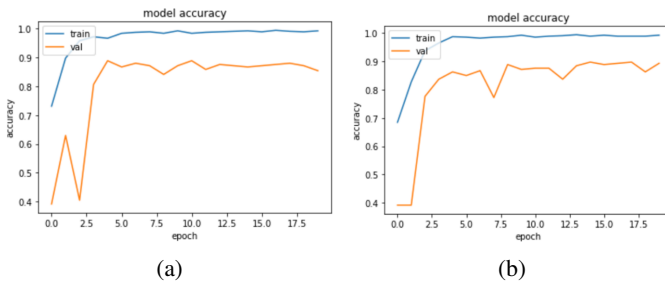


Fig. 9: The impact of dense layer

### C. Experimental Results

We divided the dataset by router manufacturer such that training and validation datasets are binary on each router company, for example one of the training sets contains a D-link vulnerable and D-link benign folders. The LBP and HOG filters only work with grayscale images, they were applied to the grayscale images to create new sets that would also be tested for their accuracy against the RGB and grayscale sets. The datasets that were tested included the grayscale, RGB, HOG, LBP, LBP+Gabor, and HOG+Gabor; each of these contained the testing and training sets for each router company. Two metrics were used in evaluating the success of the CNN models, they are the loss and accuracy of the models. The loss function used was the binary cross entropy, it uses the logarithmic loss function which compares each prediction to the actual value. A score is then calculated in the range of  $[0, 1]$  with smaller scores representing small differences between the prediction and the actual value and in turn more accurate predictions, while the opposite is true for bigger scores. The accuracy metric is calculated by the ratio between the number of correct predictions over the total number of predictions, taking the true positive and true negative over the true and false positives and negatives. These metrics could be deceptive due to the imbalanced representations in the datasets (for example

most of the samples belonging to one class and only a few to another class may cause a false accuracy score). However after the image augmentation the sample representations were evened out and should be equally represented. Therefore, we decided to test the sets on four main CNN models with hidden layers and also two models with only dense layers. The reasoning for this was that the firmware was originally a one-dimensional array of bytes and that converting them to images restructured some of the bytes and in turn the features. After the preliminary test's findings, the models that performed the best had convolutional layers with small window ( $3 \times 3$ ) sizes and fewer filters, the stochastic gradient descent algorithm with a low learning rate had solid performance along with a single dense layer with 128 units and so these hyper parameters are used in the main tests. Each of the models tested are described below:

**Model-1:** This model consists of a single convolutional layer followed by a max pooling layer and then two dense layers. The convolutional layer has a filter size of  $3 \times 3$ , 5 filters, and a ReLU activation function, the max pooling layer has a window size of  $2 \times 2$  and a stride of 2, and the dense layer has 128 units while the final dense layer uses an SGD optimizer with a learning rate of 0.001.

**Model-2:** This model has two consecutive convolutional layers and a pooling layer. This was to test if higher level features could be extracted to give a better accuracy. Using a higher number of filters on the second convolutional layer than the first gave better results than using less or an equal amount. The first convolutional layer has 5 filters and the second 10, both have a window size of  $3 \times 3$ . This is followed by a max pooling layer and 2 dense layers as in the first model.

**Model-3:** This model consists of two convolutional layers but they are not consecutive as in the last model and are followed by max pooling layers. The output layer is the same as the previous two models.

**Model-4:** This model is three convolutional layers deep following the same blueprint as Model-3 where max pooling is applied after each convolution. The first layer has 5 filters followed by 10 in the second, and then 20 in the third.

**Model-5:** For the next models, we decided to only use dense layers in a one-dimensional representation of the data, preserving its original form and feature structure. Model-5 consists of only one dense layer of 128 units and the output/classification layer.

**Model-6:** This model is made up of two dense layers, the first having 128 units and the second 256; during the preliminary testing it was determined that when the proceeding dense layer has more units, a better accuracy score is achieved.

### D. Main Testing and Findings

From the tests, it was determined that resizing the images to greater dimensions ( $500 \times 500$  vs  $250 \times 250$ ) resulted in higher accuracy likely due to more features and details being retained. The datasets of each company had different attributes and were not perfectly equal in their sample representation. When building the datasets, the main goal was to collect as many samples as possible while only keeping those that had

TABLE II: The validation accuracy of each model and dataset. The Gabor filtered images did not show any improvement in accuracy and were discontinued.

	Asus Accuracy						Buffalo Accuracy					
	Grayscale	RGB	LBP	HOG	LBP+Gabor	HOG+Gabor	Grayscale	RGB	LBP	HOG	LBP+Gabor	HOG+Gabor
<b>Model-1</b>	70.67%	81.87%	68.03%	93.81%		82.47%	66.67%	81.17%	74.36%	78.98%		72.61%
<b>Model-2</b>	63.46%	77.19%	74.15%	92.78%	60.54%	81.44%	67.80%	83.77%	74.36%	73.89%	72.44%	71.34%
<b>Model-3</b>	64.90%	80.12%	66.67%	95.88%	57.14%	87.63%	66.78%	79.87%	74.36%	78.98%		73.25%
<b>Model-4</b>	71.63%	77.19%	65.31%	97.94%	56.46%	90.72%	63.84%	76.62%	71.15%	82.80%		73.25%
<b>Model-5</b>	63.46%	74.27%	61.90%	90.72%			53.11%	77.92%	70.51%	75.16%		
<b>Model-6</b>	58.65%	73.10%	59.18%	90.72%			55.93%	77.92%	70.51%	79.62%		
<b>Average</b>	65.46%	77.29%	65.87%	93.64%			62.36%	79.55%	72.54%	78.24%		
$\sigma$	4.90%	3.34%	5.19%	2.88%			6.58%	2.85%	2.05%	3.59%		

	D-Link Accuracy:						Netgear Accuracy					
	Grayscale	RGB	LBP	HOG	LBP+Gabor	HOG+Gabor	Grayscale	RGB	LBP	HOG	LBP+Gabor	HOG+Gabor
<b>Model-1</b>	72.78%	78.97%	77.97%	82.76%		75.00%	65.41%	82.86%	72.78%	73.61%		66.67%
<b>Model-2</b>	77.98%	85.32%	83.05%	86.21%	71.19%	79.74%	67.06%	81.90%	79.61%	75.69%	58.67%	61.81%
<b>Model-3</b>	66.58%	79.87%	80.93%	89.22%		79.74%	67.78%	84.76%	75.73%	72.92%		72.22%
<b>Model-4</b>	67.28%	87.70%	79.24%	89.22%		74.14%	69.41%	82.86%	71.15%	71.53%		67.36%
<b>Model-5</b>	64.83%	80.95%	76.27%	83.19%			61.18%	77.14%	68.93%	70.14%		
<b>Model-6</b>	68.50%	79.37%	76.69%	82.76%			61.18%	80.00%	69.90%	68.06%		
<b>Average</b>	69.66%	82.03%	79.03%	85.56%			65.34%	81.59%	73.02%	72.00%		
$\sigma$	5.18%	3.61%	2.61%	2.85%			3.47%	2.67%	4.02%	2.69%		

enough representations to positively contribute to the learning algorithm and removing the underrepresented samples. This process resulted in the datasets having varying sizes and differing in number of router model firmware. The performance of the datasets seemed to be tied to the number of samples but not the variability present in them. The Asus and D-Link datasets had the highest number of samples and both performed better than the Buffalo and Netgear sets in most cases including the HOG filtered images, which produced the highest accuracy. Conversely, the Netgear set which had the fewest number of samples had the lowest accuracy in the HOG filtered tests. The Asus and Netgear sets had the fewest router models, and yet the Netgear set underperformed and was too inconsistent to make a conclusion based on the variability of the datasets. The results were reported in Table II.

Using the grayscale dataset produced underwhelming results ranging from 53% to 77.9% accuracy with an average of 66.54% across all manufacturers, the D-link grayscale dataset gave the highest accuracy especially when using it along with Model 2. This is most likely due to the D-Link dataset having the most samples. The best overall results of the grayscale set came from training it on Model 4 between all manufacturers. The RGB dataset yielded substantially higher accuracy ranging from 73.10% to 87.70% - with an overall average of 80.11% - in contrast to the much lower accuracy of grayscale set. This is likely due to the RGB images retaining more features from the extra channels and allowing the models to learn more complex patterns. The RGB sets delivered some of the highest accuracy and could only be surpassed by the HOG datasets. The LBP filtered images were made from the grayscale sets and made them look less grainy with the important features highlighted. Applying this filter increased the accuracy of all the models across the board compared to the grayscale sets. The LBP set accuracies ranged from 59.18% to 83.05% - with the average accuracy being 72.78% - but it ultimately proved less accurate than the RGB and HOG filtered images. The LBP filter highlighted the dominant pixel intensities, which justified the increase but did not compare well to the HOG filter as

it had far more entropy and far less recognizable features. Model 2 was clearly the best fit for the LBP regardless of manufacturer. This demonstrates that the less defined and chaotic structure of the LBP images (compared to the HOG filtered images) require a stacking of convolutional layers to extract higher level features, in contrast with the HOG image features, which are much more distinct and do not require this kind of processing.

The HOG filtered images exhibited the best feature translation as they use areas with distinct features derived from the surrounding pixel intensities and are easily recognizable even by the human eye. During the tests the HOG images produced the highest accuracies of all the datasets from 68.08% to 97.94% with an average accuracy of 85.81%, rivaling and at times surpassing the accuracy of the RGB sets. The HOG filter performed best when used with the Asus dataset, possibly due to having less variability in terms of router model representation. Training Model 4 is the most accurate across all manufacturers. The increased depth allowed for a more detailed feature extraction which paired well with the less abstract HOG filtered images. The Gabor filter was applied to the grayscale, LBP, and HOG filtered images and the parameters of the Gabor filter were set to filter only the horizontal features since the firmware code was read as a horizontal array of byte strings and layered on one another to make the images and would thereby have neighboring code segments in the horizontal orientation. The Gabor filter ultimately had the negative effect of decreasing the accuracy of all the image types that it was applied to, after this conclusion further testing on it ended; although the Gabor filtered images exhibited a decrease in accuracy from their original counterparts, the ranking in their accuracy was carried over with the Gabor filtered HOG images having the highest followed by Gabor+LBP. When the one-dimensional models 5 and 6 were trained, both performed poorly in comparison to all the previous models that used two dimensional representations of the data, this was across all manufacturers and filters/image types. The best performance in terms of accuracy came from



the HOG filtered datasets when used with Model 4, this proves that vulnerable code segments are more easily detectable when using multiple convolutional layers separated by pooling layers as they narrow down the feature search by removing unnecessary noise data. This is mainly true for the HOG filtered images and possibly others not yet tested that have well defined features with little entropy and noise data.

## VI. ADDITIONAL EXPERIMENTS

### A. classification results can be generalized to other manufacturers or models of routers

This issue was explored in the latest rounds of testing that dealt with the models learning to differentiate between the types of vulnerabilities (regardless of vendor) in contrast to the previous training sessions that only had a vulnerable/benign binary classification. The testing done featured using a new categorical dataset that would divide the firmware based on types of vulnerabilities; for this test the three major types of vulnerabilities with the most number of samples were selected and used as separate categories [Access control bypass, Arbitrary code execution, and SQL injection], in addition three more categories would be added to represent their benign patched counterparts.

The test concluded with promising results; although it ended with a 78.86% accuracy at epoch 20, that was mainly due to overfitting and faulty hyperparameters. Prior to reaching the 16th epoch, the model was frequently scoring accuracy in the mid 87%'s before it began to drop.

This test proved that the model has potential at learning different types of vulnerabilities regardless of vendor and with more adjustments to the model and better representation in the dataset, the accuracy should achieve a higher, more stable accuracy.

### B. Discovering Unknown Vulnerabilities

We tested how well this model predicts firmware that it has never encountered. For this experiment, we decided to use the buffalo dataset since it had the best firmware representation. While the other vendor datasets had gaps in discontinued or legacy firmware. RGB images used with model 2 provided the best results and were used for this test (the input size and all of the hyperparameters being the same as well).

The latest versions of firmware were removed from the benign and vulnerable training sets and added to the testing sets. This was done in order to simulate the model not encountering the firmware during training. If the firmware version was the only version present in the dataset, it was left alone and if a version was taken out, all of its augmentations would also be removed.

Table III shows a list of the versions that were removed from the training set.

Training yielded expected results with the accuracy going down from 86.77% to 79.88% along with an increase in loss from 52% to 103%. This indicates that although the model is achieving almost 80% accuracy, it is also making some significant errors in its predictions, along with signs of overfitting.

TABLE III: The list of the versions that were removed from the training set.

Benign Versions	Vulnerable Versions
bhr_4grv_jp_200	bhr_4grv_jp199
bhr4grv2-1.07	fshpg300n-331 & 332
bhrgrv2-111	whr_hp_g300n_jp_198 & 199
wex300-220 & 250	whr300hp-193 & 196
whr300hp2-200 & 220 & 250	whrhp300n-193 & 196
whr600d-200 & 291	whrpqn-185 & 186
wzr_hp_g450h_jp	wmr300-130 & 190
	wzr_600dhp_jp_199
	wzr_hp_g450h_jp_188 & 189
	wzr300hp-196
	wzr450hp-196
	wzr600dhp-197
	wzrhpg302h-183
	wzrhpg450h-187

The first test was done using the buffalo RGB set on the model-2 CNN. The results suggest a decrease in accuracy loss due to missing firmware, this decrease in accuracy was smaller than expected.

We decided to see if the loss could be brought down by increasing the depth and the best candidate for this would be model-4 due to its consistently high performance across all the vendors. After testing, model-4 seems to have traded some accuracy for better loss with the end results being [80.37% accuracy & 89.28% loss] after 20 epochs.

Using a deeper model (model-4) results in a decrease in validation loss, overfitting is still an issue at this point. The increase in depth allowed the model to learn more complex features thereby improving the model's capability of predicting unencountered firmware signified by the reduction in validation loss. The accuracy on the other hand could be increased by calibrating the hyperparameters and number of epochs, while creating more augmented images for better representation in the dataset should improve the overfitting issue.

### C. Imbalance between Vulnerable and Benign Versions

During the training phase overfitting was an issue that could not be solved directly by gathering more samples; the vulnerable firmware samples were over represented compared to the benign since only the newest versions would be benign, this issue was evident from the beginning.

A way to fix this was to implement an image augmentation module that would modify and replicate the images of any firmware that was underrepresented, resulting in a more even representation thereby solving the overfitting issue.

For the image augmentation module, we used the Image-DataGenerator module. The augmentation of the images can be done through horizontal vertical shifts, rotations, zooming and flips. For the firmware it only makes sense to apply the horizontal and vertical flips since they change the location of the bits/code, the other augmentations would translate to the code itself being altered and would produce invalid results with much lower accuracy. After some preliminary testing, we decided to use the height shift with wraparound enabled to augment the images; using a height and width shift together alters the code too much and would give much lower accuracy. The rest of the code in this module handles going through an image folder and applying the augmentation randomly 4 times

per image. The underrepresented firmware was gathered into a folder, augmented all at once and then put back into its original folder with all of its augmentations. The most complete dataset was the buffalo firmware set, in addition the RGB images provided the overall highest scores and were both used for this experiment.

The next step was to test out the degree of the augmentation and if increasing the shift range would have any effect on the accuracy and loss of the model. Two values for the height shift range were tested [0.5 0.05], this resulted in two datasets with different shift ranges for the augmented images. Both augmented sets were trained using model-4 with a depth of 2 since it produced a high accuracy across all manufacturers and was the most stable.

After testing, the model trained on the images set with the height shift of 0.5 had a validation accuracy of 80.66% and a loss of 66.35%; this is a lower accuracy than the original test which had 83.62% validation accuracy (RGB + model-4 + buffalo). The model trained on the 0.05 height shift had a validation accuracy of 84.42% and a loss of 55.68%, this was higher than the original testing which used a height shift of 0.002.

The results of this experiment show that if you augment the underrepresented images, you could achieve a higher accuracy and lower loss; it also shows that the degree of augmentation also has an effect on the end results. It seems that going too high or too low on the augmentation has a negative effect on the end results and a “Goldie lox zone” must be found.

#### D. Type of Vulnerability Identification

During the project the CNN models were first trained on a binary dataset comprised of all the manufacturers and was divided on the benign and vulnerable categories. This dataset yielded the poorest model performance in terms of accuracy.

The next step was to divide the datasets based on manufacturers and each manufacturer set was trained on individually. This greatly improved the model’s prediction accuracy. The downside to this approach was that the models could not detect the type of vulnerability and an additional model was needed to determine the manufacturer to then pass to that manufacturer’s trained model.

In this experimental setup, a categorical dataset was made and divided the firmware based on the type of vulnerability. The most common vulnerabilities of the firmware were divided into categories [Access Control Bypass, Arbitrary Code Execution, and SQL Injections ], and each of these categories had a benign version added as an additional category for a total of 6 categories per set [training\_set and testing\_set]. For example the training and testing sets contain the following categories [access\_control\_bypass, arbitrary\_code\_execution, sql\_injections, access\_control\_bypass\_benign, arbitrary\_code\_execution\_benign & sql\_injections\_benign], the benign categories contain all of the firmware of the same firmware version/type that have the vulnerabilities patched. The Model used for this test was the same model-4 as the first tests that dealt with the older dataset.

Table IV lists the firmware used to make this new dataset. Three vulnerable classes would be made based on the con-

firmed vulnerability and contained all firmware (regardless of manufacturer) with said vulnerability.

TABLE IV: Used Dataset.

D-Link		
Model Name	Size range	Confirmed Vulnerability
DAP-1860, DIR-3060	4.5 MB, 20 MB	Arbitrary Code Execution
DAP-2610, DIR-867, DIR-878, DIR-882, DIR-850L, DIR-895L, DAP-1650	5 - 16 MB	Access Control Bypass
DSR-150*, DSR-250*, DSR-500-, DSR- 1000*	14 - 25 MB	SQL Injection
Netgear		
Model Name	Size range	Confirmed Vulnerability
JGS516*, JGS524*	166 - 300 KB	Access Control Bypass
R6120, R6220, R6260, R7000, R7800, R7900, R8000	14 - 16 MB	Arbitrary code execution
Asus		
Model Name	Size range	Confirmed Vulnerability
RT-AC1900	22 - 36 MB	Access Control Bypass
RT-AC55U, RT-AC66U, RT- AC58U, RT-AC86U, RT-AC87U, RT- N12 D1, RT-AC1200, RT-AC1750, RT-AC 2900, RT-AC3100	7 - 37 MB	Arbitrary Code Execution
Buffalo		
Model Name	Size range	Confirmed Vulnerability
bhr_4grv_jp, bhr_4grv2, bhr_4grv, dwr_hp_g300nh, hw_450hp, whr_300hp, whr_g301n, whr_hp_g300n, wpl_**, wzr_450hp, wzr_hp_**, wzr_300hp, fs_600dhp, wzr_d1100h, fs_600dhp, fs_r600dhp, fs_g300n	7 - 25 MB	Access Control Bypass

As the training began, the accuracy was steadily rising with an upwards trend and finally settled at 82%. Midway into training, the accuracy was slightly fluctuating with frequent peaks at 80.18% accuracy. Past the 15th epoc, the accuracy and loss values became more unstable, a sign of overfitting/over-training. That training concluded with an accuracy of 79.86% at epoc 20.

The results of this new experiment proves that this is a better alternative to the categorical approach than the one that dealt with benign/vulnerable categories of all manufacturer. These promising results indicate that the CNN models could be used to accurately identify types of vulnerability in the future and potentially surpass the binary classification per manufacturer (provided there is an expansion to the dataset and better optimized hyperparameters to solve the overfitting issue).

## VII. CONCLUSION

### A. Concluding Remarks

An IoT cyberattack can cause massive damage to the connected devices and harm to their owners. Thus, router firmware vulnerability detection is recently becoming an emerging issue in this domain. As a result, we provided an efficient and precise detection tool that is a necessity to this domain. The firmware dataset collection was manually collected and made it available to the community. This paper investigated the effectiveness of analyzing home router firmware by employing the use of convolutional neural networks (CNN) and the use of computer vision techniques. We tested a set of different combinations of the filtered/normal datasets with the CNN training model. The highest accuracy filter was determined to

be the HOG filter with an average accuracy of 85.81% across all tests and models with results as high as 97.94% when used with the right CNN model.

### B. Limitations

During the data collection phase, no premade firmware datasets for home routers could be found online, resulting in grueling manual collection. Given more human and financial resources, a much larger dataset could be collected spanning a wider variety of router models. Due to the smaller dataset, some router models had to be removed as the preliminary tests had determined that underrepresented models instituted unnecessary variability and decreases in accuracy. Another limitation came from having less computational power. Having more computers would have allowed for faster progress during testing, and ultimately freed time for more filters to be tested. In the end, the experimental results showed that filtering an image into one with more distinct and clear features results in higher accuracy when used on deeper CNN models.

### C. Future Work

In the future, we plan to test more models and filters on the datasets. The next model to be tested is the gray level co-occurrence matrix, and is used in displaying the combinations of pixel values in various directions while outputting an image with smaller dimensions and much less noise data than the ones used in the previous tests. We also plan to use supervised learning models such as the  $k$ -NN algorithm in order to find out if classifying all manufacturers at once is a viable option. Moreover, we will use a set of highly optimized models for image classifications such as *VGGNet16* and *ResNet50*.

## REFERENCES

- [1] K. Cheng, Q. Li, L. Wang, Q. Chen, Y. Zheng, L. Sun, and Z. Liang, "Dtaint: detecting the taint-style vulnerability in embedded device firmware," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 430–441.
- [2] J. Gao, X. Yang, Y. Jiang, H. Song, K.-K. R. Choo, and J. Sun, "Semantic learning based cross-platform binary vulnerability search for iot devices," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp. 971–979, 2019.
- [3] G. George and S. M. Thampi, "Combinatorial analysis for securing iot-assisted industry 4.0 applications from vulnerability-based attacks," *IEEE Transactions on Industrial Informatics*, 2020.
- [4] Y. Liu, J. Wang, J. Li, S. Niu, and H. Song, "Machine learning for the detection and identification of internet of things devices: A survey," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 298–320, 2021.
- [5] —, "Class-incremental learning for wireless device identification in iot," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 17 227–17 235, 2021.
- [6] Y. Liu, J. Wang, J. Li, S. Niu, L. Wu, and H. Song, "Zero-bias deep learning enabled quickest abnormal event detection in iot," *IEEE Internet of Things Journal*, 2021.
- [7] S. N. Matheu, J. L. Hernandez-Ramos, A. F. Skarmeta, and G. Baldini, "A survey of cybersecurity certification for the internet of things," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–36, 2020.
- [8] M. Poongodi, M. Hamdi, and H. Wang, "Image and audio caps: automated captioning of background sounds and images using deep learning," *Multimedia Systems*, pp. 1–9, 2022.
- [9] P. Srivastava, H. Peng, J. Li, H. Okhravi, H. Shrobe, and M. Payer, "Firmfuzz: Automated iot firmware introspection and analysis," in *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*, 2019, pp. 15–21.
- [10] J. Costoya, R. Flores, L. Gu, and F. Mercès, "Securing your home routers." Web, 2019.
- [11] J. Vadayath, M. Eckert, K. Zeng, N. Weideman, G. P. Menon, Y. Fratanonio, D. Balzarotti, A. Doupe, T. Bao, R. Wang *et al.*, "Arbiter: Bridging the static and dynamic divide in vulnerability discovery on binary programs," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 413–430.
- [12] Y. Zheng, A. Davanian, H. Yin, C. Song, H. Zhu, and L. Sun, "{FIRM-AFL}:{High-Throughput} greybox fuzzing of {IoT} firmware via augmented process emulation," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1099–1114.
- [13] B. Feng, A. Mera, and L. Lu, "{P2IM}: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1237–1254.
- [14] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "Firmae: Towards large-scale emulation of iot firmware for dynamic analysis," in *Annual Computer Security Applications Conference*, 2020, pp. 733–745.
- [15] H. Wen, Z. Lin, and Y. Zhang, "Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 167–180.
- [16] L. Chen, Y. Wang, Q. Cai, Y. Zhan, H. Hu, J. Linghu, Q. Hou, C. Zhang, H. Duan, and Z. Xue, "Sharing more and checking less: Leveraging common input keywords to detect bugs in embedded systems," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.
- [17] N. Redini, A. Machiry, R. Wang, C. Spensky, A. Continella, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Karonte: Detecting insecure multi-binary interactions in embedded firmware," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1544–1561.
- [18] A. Pinhero, M. Anupama, P. Vinod, C. A. Visaggio, N. Aneesh, S. Abhijith, and S. AnanthaKrishnan, "Malware detection employed by visualization and deep neural network," *Computers & Security*, p. 102247, 2021.
- [19] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
- [20] A. Mvoulana, R. Kachouri, and M. Akil, "Fine-tuning convolutional neural networks: a comprehensive guide and benchmark analysis for glaucoma screening," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 6120–6127.
- [21] J. Brownlee, "How do convolutional layers work in deep learning neural networks?" *Machine Learning Mastery*, 2020.
- [22] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.
- [23] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE transactions on neural networks*, vol. 9, no. 1, pp. 224–229, 1998.
- [24] J. Brownlee, "A gentle introduction to the rectified linear unit (relu)," *Machine learning mastery*, vol. 6, 2019.
- [25] S. Patel and J. Pingel, "Introduction to deep learning: What are convolutional neural networks," *MathWorks*. <https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks-1489512765771.html> (accessed 26/8/2019, 2019), 2019.
- [26] N. Ketkar, "Stochastic gradient descent," in *Deep learning with Python*. Springer, 2017, pp. 113–132.
- [27] L. Armi and S. Fekri-Ershad, "Texture image analysis and texture classification methods-a review," *arXiv preprint arXiv:1904.06554*, 2019.
- [28] N. M. C. Reddy and K. Kumar, "Comparison of hog and fisherfaces based face recognition system using matlab," in *2021 2nd International Conference for Emerging Technology (INCET)*. IEEE, 2021, pp. 1–5.
- [29] Z. Guo, L. Zhang, and D. Zhang, "A completed modeling of local binary pattern operator for texture classification," *IEEE transactions on image processing*, vol. 19, no. 6, pp. 1657–1663, 2010.
- [30] K. S. do Prado, "Face recognition: Understanding lbph algorithm," 2017, *Towards Data Science*, 2017.
- [31] "Through The Eyes of Gabor Filter," <https://medium.com/@anujshah/through-the-eyes-of-gabor-filter-17d1fdb3ac97>, 2021.