MODEL HIJACKING EXPLOITATION AND MITIGATION

BY

CHEW ZI YING

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: _Model Hijacking Exploitation and Mitigation_____________________

Academic Session: __JAN 2024__

I ____CHEW ZI YING____________________________________

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

[illegible]

(Author's signature)

Verified by,

GML

(Supervisor's signature)

Address:
_206, Lorong Pala 8___
_Taman Teluk Intan__________
_36000 Teluk Intan, Perak.____

___Dr Gan Ming Lee________
Supervisor's name

Date: _____21/04/2024______

Date: ______21/04/2024_____

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN

Date: _________21/04/2024_____

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ______CHEW ZI YING__________________ (ID No: __**20ACB03843**_) has completed this final year project/ dissertation/ thesis* entitled "__Model Hijacking Attack_____" under the supervision of _____Dr. Gan Ming Lee_____________ (Supervisor) from the Department of Computer and Communication Technology , Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]

____
(*CHEW ZI YING*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**MODEL HIJACKING EXPLOITATION AND MITIGATION**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	[illegible]
Name	:	CHEW ZI YING
Date	:	21/04/2024

ACKNOWLEDGEMENTS

I want to extend my heartfelt gratitude to Dr. Gan Ming Lee for providing me with the wonderful opportunity to embark on a model hijacking project. This marks my initial stride towards building a career in the cybersecurity field and machine learning. I am truly thankful to you for guiding me through the project.

I express my appreciation to significant individuals among my friends for their enduring patience and willingness to help when I needed it, especially for standing steadfastly by my side during challenging moments. Lastly, I wish to express my gratitude to my parents and family for their constant love, support, and ongoing encouragement throughout the duration of the program

ABSTRACT

In today's digital era, the utilization of machine learning has proliferated, making it an invaluable tool across various automotive applications. Machine learning has found its way into numerous facets of automotive engineering and operations. Additionally, machine learning is harnessed in predictive maintenance, where it analyses sensor data from vehicles to forecast potential mechanical issues, thereby optimizing maintenance schedules and minimizing downtime. However, things are two sides. There are vulnerabilities in machine learning that can cause adversarial attack, namely model hijacking attack. In prior research, there are experiment that shows that it is possible to insert trigger into the input to trigger backdoor attack without the user notice. The severity of this problem stems from the attacker's ability to wield this attack method at their discretion, resulting in highly erroneous outcomes. The profound danger lies in the fact that once a malicious actor successfully gains control of a machine learning model, they can manipulate it to generate intentionally misleading results or predictions. Incorrect decisions prompted by a poisoned machine learning model can lead to substantial financial losses, damage to a company's reputation, and even bankruptcy, life and death. In critical applications such as healthcare, autonomous vehicles, and industrial control systems, the reliability of machine learning models is paramount. Thus, in this project, I will focus on the backdoor attack to identify how the attack make use of the loopholes of machine learning. Think from the perspective of an attacker, I propose model that can be backdoor without being realize. It can have face recognition real-time and enter the backdoor mode when physical backdoor is being detected. When the model is in a backdoor mode, it will misclassify the face into the wrong class as attacker intentions. By understanding how the attack works, it can be twisted into a model that gives positives use such as steganography. Thus, in this research, I will develop a backdoor machine learning that can be used for good intention.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

α	Blended ratio, the ratio of transparency

Chapter 1
Introduction

In this chapter, I will introduce the background and the motivation of my research, outline the unique contributions I make to the field, and provide an overview of the thesis structure.

1.1 Problem Statement and Motivation

The existing backdoor model in machine learning with physical backdoor attack can be easily detected by human eye. Physical backdoor means that the trigger is made up of very obvious characteristic. So, when user look into the training dataset and saw this anomaly in the dataset, he will know something wrong with the model. Even when the user insert image for prediction of result, before he inserts the image, he will sure spot the trigger first.

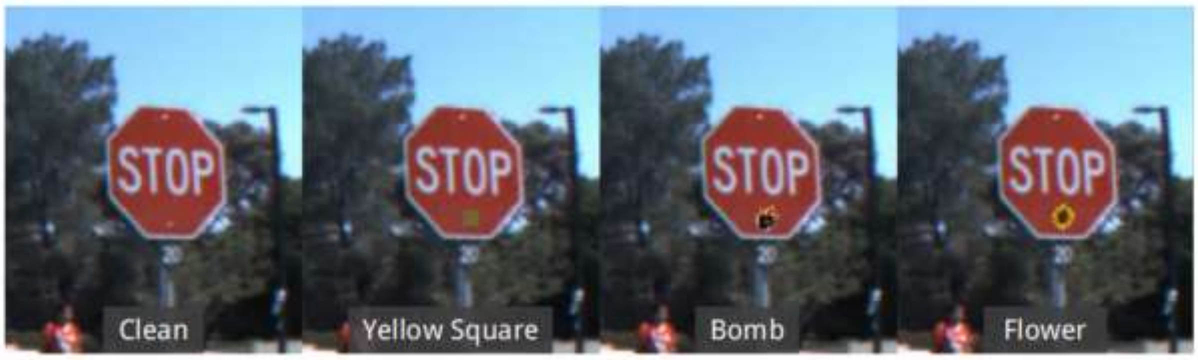


Figure 1.1 The example of physical trigger [1].

When the backdoor model depends on a physical trigger to enter the backdoor, it will cause the training dataset to have obvious similarity. Thus, when the user check on the training dataset, he might be able to find out the anomaly in the images. Thus, I would like to propose model that hides the trigger, making it difficult for people to recognize any suspicious activity.

The problem with machine learning is that triggers can be inserted to manipulate the output. This is very dangerous as it will cause machine learning to make wrong decision that might harm someone lives such as the modern car that detect surrounding to avoid accident. For example, a wrong decision might cause a car that do not need to sudden break to do so and causes accident. Moreover, it will cause those users without machine learning knowledge to deploy the code and causes some sensitive information to be disclose and other consequences. Thus, identify ways to ease this problem is necessary to protect user.

Besides, the triggers are not easily detected if it is hidden. A powerful hidden trigger can bypass a lot of security checking. It will only be detected when the backdoor is being trigger and until that time and by then, it's often too late. So, I will find a way to detect the unusual in the model. This proactive detection approach will involve continuous monitoring for any deviations from expected behavior, providing an early warning system.

1.2 Objectives

The project's primary objective is to design and construct a specialized computer program known as a backdoor model. A backdoor model is a machine learning that being added with inaccurate label together with trigger during the training phase. This is done so that it ca misclassify the images if same trigger is detected during implementation. It can be done using data poisoning technique.

Additionally, I aim to change the purpose of the backdoor model for more positive uses. The current backdoor model is only designed for negative intentions. However, after understanding how it functions, I see the potential to use it for more meaningful tasks, specifically in the realm of steganography. I want to turn the backdoor model into a helpful tool by studying and applying ways to hide messages within random images.

Finally, my aim is to build a secure system for encoding and decoding messages. This system will only be unlockable by the specific backdoor model I developed as it is impossible for any others without the same data poisoning to replicate the model nor dataset. This ensures that the information remains confidential and cannot be accessed by unauthorized parties, maintaining the secrecy of the encoded messages. So, users can hide important messages within ordinary-looking messages.

1.3 Project Scope and Direction

This project is centered around the idea of model hijacking, which is the process of changing a machine learning model's intended use for attack purpose. In particular, I want to turn a backdoor model—which was initially created for a negative intention—into a tool for steganography, which is a method of hiding important message in digital media. After understanding how backdoor model works, I twisted the usage of backdoor model and apply it into a real-life scenario such as encryption and decryption. When someone sends a message, the model encrypts it, making it to contain trigger and seems to be ordinary. The recipient, using the same model, can then decrypt and read the hidden message. Human eye will only see the ordinary message but the model will translate the hidden message when trigger detected. The result that the text recognition give is the hidden message that the sender wants to tell. However, without trigger, the result of the message after going through the model should be same as what human eyes see. This approach utilizes the model in a practical scenario for secure communication.

1.4 Contributions

In this project, I am developing a backdoor model with a positive purpose, specifically for steganography. This model ensures secure communication because only this model can decode the hidden messages. The communication remains safe because it's challenging for attackers to recreate the backdoor model without access to the same training data I used. This positive application of the backdoor model helps protect communication privacy, making it a reliable tool for secure and discreet information exchange which contribute to cybersecurity aspect by providing a new approach for secure communication.

Besides, combining encryption and steganography in the model creates a way to exchange information discreetly and constructively. This is important in situations where secure communication and safeguarding sensitive data are crucial, like in different areas of cybersecurity. The approach provides a practical method for sharing information securely, ensuring privacy and protecting valuable data in contexts where maintaining confidentiality is of utmost importance.

1.5 Report Organization

This report is organized into 6 chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Methodology, Chapter 4 System Design, Chapter 5 Implementation, Chapter 6 System Evaluation and Discussion, Chapter 7 Conclusion and recommendation. The first chapter is the introduction of this project which includes problem statement, project scope, project objectives, project contribution and report organization. The second chapter is the literature review carried out on several existing backdoor model research to evaluate the strengths and weaknesses of each model architecture and trigger pattern. The third chapter is discussing the methodology used to develop the model and the activity diagram of system. The fourth chapter is regarding the details on the design of the system that explain how the system should works. Furthermore, the fifth chapter reports the ways to build the system. It includes the setup, configuration, system operation and the implementation issue and challenge. In Chapter 6, it explains the overall performance of the system including the results and project challenges. The last chapter conclude the whole project with recommendation to improve the system.

Chapter 2
Literature Reviews

2.1 Previous works on Backdoor attack on Machine Learning

2.1.1 BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

In this paper, a model called BadNet designed for neural networks is introduced. There are two type of attack which is outsourced training attack and transfer learning attack. In an outsourced training attack, an attacker manipulates the training process of a machine learning model by inserting malicious data or tampering with the training pipeline. Transfer learning attack occur when the attacker starts with
a pre-trained model but uses a smaller, contaminated dataset that includes their target task. Both type of attack much achieves the adversary goals which is:

1. The accuracy of validation set should not be reduced as the reduction will causes the model to be rejected by the user.
2. The input that contains trigger should have different classification result from the honest classification result.

The first case study is done using the MNIST dataset. The convolutional neural network (CNN) has two convolutional layers and two fully connected layers. In this case study, there are two type of trigger which is called single pixel backdoor and pattern backdoor. A single pixel backdoor is a single bright pixel in the bottom right corner of the image while pattern backdoor is multiple bright pixel that group together to become a trigger. Both single target and all-to-all attack is investigated. A single target attack will misclassify the backdoored input into the target label that the attacker specified when trigger is detected. Meanwhile, all-to-all attack will shift the label by one. In the experiments, image is randomly selected from the training dataset to add trigger and mixed it with the clean data to retrain. Intriguingly, when BadNet is used to classify clean images, it made very few errors that even fewer than the original network. The research state that BadNets appear to have learned specific convolutional filters dedicated to recognizing backdoors. This indicates that the presence of backdoors is sparsely encoded in deeper layers of the BadNet. Remarkably, the attack remains effective even when backdoored images make up only 10% of the training dataset.

The second case study is done on Faster-RCNN that classify U.S. traffic sign. The networks consist of a shared CNN, a region proposal CNN that find areas in the image that might contain traffic signs and a traffic sign classification network that decides if a region is a traffic sign and, if so, what type of sign it is. The outsourced training attack, there are also single target attack and all-to-all attack. The setup is same as the previous attack. Then, the accuracy of clean data using the BadNet remain high while backdoor attack still achieves. For the random target attack using the yellow square backdoor, the BadNet's accuracy on clean images is only slightly lower than the baseline F-RCNN. However, the BadNet heavily misclassifies over 98% of backdoored images. The dedicated convolutional filters that specified in MNIST attack do not appear here as dataset vary in scale and angle. Thus, the research believe that BadNet might be using feature encoding to build a backdoor detector. However, it has dedicated neurons in their last convolutional layer that encode the presence or absence of the backdoor. For transfer learning attack, the retrained is done using clean and backdoored images of Swedish traffic signs. It achieves a high accuracy on clean images, but the accuracy drops on backdoored images. The neurons that activate only in the presence of backdoors is activated in this attack too. To enhance the attack, the adversary increases the activation levels of specific neurons dedicated to detecting backdoors. This further reduces accuracy on backdoored inputs but also slightly affects accuracy on clean inputs. For instance, with a certain setting (k=20), the attack significantly drops backdoored image accuracy by over 25% while causing a 3% drop in clean image accuracy.

2.1.1.1 Strengths and Weakness

BadNets is the first backdoor model being develop in this research field. The strength is that this model is easy to implement. A simple data poisoning can achieve this backdoor model. By poisoning the training dataset, the backdoor mode can be trigger when the model detects the trigger.

However, BadNet is not suitable in some cases as the trigger can be easily detected by human eyes. If the user has knowledge on machine learning, he can easily know that there is data poisoning in the training. Besides, the trigger pattern is specified. They only accept one trigger pattern in the research but this problem can be easily solved by training with multiple trigger pattern. Meanwhile, the all-to-all attack cannot misclassify the category to any wrong classes but only the to its corresponding category, i to i+ 1. This will cause the attack easily being recognized.

2.1.2 WaNet – Imperceptible Warping-Based Backdoor Attack

To make the trigger invisible, this research proposed image warping to change the shape of image. Unlike other attack methods that add extra information to an image, elastic warping only changes the existing pixels. For every clean image-label pair, it will apply with the warping function to move each part of the image as calculated by the function. To prevent mess up of the image's borders, a clipping function is applied to keep it inside the picture.

Clean and attack modes sometimes make the network learn strange things that can be easily detected. To resolve the problem, noise mode is proposed. To prevent the network to use the backdoor, the warping field is made by adding some random noise. This new training method makes the network work better without revealing its tricks. The experiments are conducted on four datasets: MNIST, CIFAR-10, GTSRB, and CelebA. For CelebA, three balanced attributes are combined to create eight classes. The attribute is Heavy Makeup, Mouth Slightly Open, and Smiling. Each dataset is applied on specific network architectures, like Pre-activation Resnet-18 for CIFAR-10 and GTSRB, and Resnet-18 for CelebA, while MNIST had a simpler structure.

The WaNet model is tested and trained using single target attack only. WaNet performed well with high accuracy like regular model when tested using clean data. When subjected to pre-defined image warping, the attack was highly successful, with an almost 100% success rate on all datasets. Meanwhile, random image warping also recognized the true image class with similar accuracy to the clean mode. This make the WaNet impressive as it gives the fact that the poisoned images closely resembled the original ones. An experiment is also done to test on the robustness of the model in real-life. As the result, the model performed well by achieving a 98% accuracy on clean images and a 96% success rate in the attack mode.

Human inspection experiment is carried out. In this experiment, the model had achieved a four time better from previous studies with 28% fooling rate for WaNet. There are three defense mechanisms is used to test on WaNet. First, WaNet passed the test on all datasets. In fact, its scores were even lower than those of the clean models on MNIST and CIFAR-10. Secondly, fine pruning that focus on neuron analyses is also being bypassed. By plotting the network accuracy against the number of neurons pruned, the clean accuracy was never significantly higher than the attack accuracy,making backdoor mitigation impossible. Lastly, With WaNet, the perturbation operation of STRIP modifies the image content and

disrupts the backdoor warping if it's present. As a result, WaNet behaves like genuine models, with similar entropy
ranges. GradCam that network behaviors are also used but it is impossible to detect using it as WaNet backdoored the image by warping the entire image. As the result, visualization heatmaps of WaNet appear similar to those of clean models.

2.1.2.1 Strengths and Weakness

WaNet use noise as trigger and the trigger is generated by the model. Thus, it is not obvious as trigger as it is hard to be recognize by human. Meanwhile, since the trigger is being added after the input going through the model, the trigger cannot be found in the input images nor training dataset except the user read through the code. As strengths mentioned above, it results in high attack success rate which can bypass Neural cleanse, fine-pruning and STRIP.

However, the trigger that being produced to attach to the input image visually inconsistent with relatively large residual. Thus, it still can be detected if user intentionally show the images after going through the model.

2.1.3 Hidden Trigger Backdoor Attacks

In traditional backdoor attacks, a subtle change is added to some images to trick the model during training. This small change, called a trigger, is linked to a specific target category. The model learns from both regular images and those with triggers, making it less likely to notice the difference. When the model is used later, it misclassifies an image if the trigger pattern is present. However, this can be spotted and fixed. To make the attack more effective, a new approach is suggested. It correctly labels tampered images without visible triggers, making it harder to detect. This involves creating modified images that blend into regular ones while still carrying the hidden trigger information.

Methodology proposed in the research is more practical as the poisoned data is label correctly. To achieve this aim, the poisoned images is close to the target categories in pixel space and close to the source images patched with trigger in feature space.

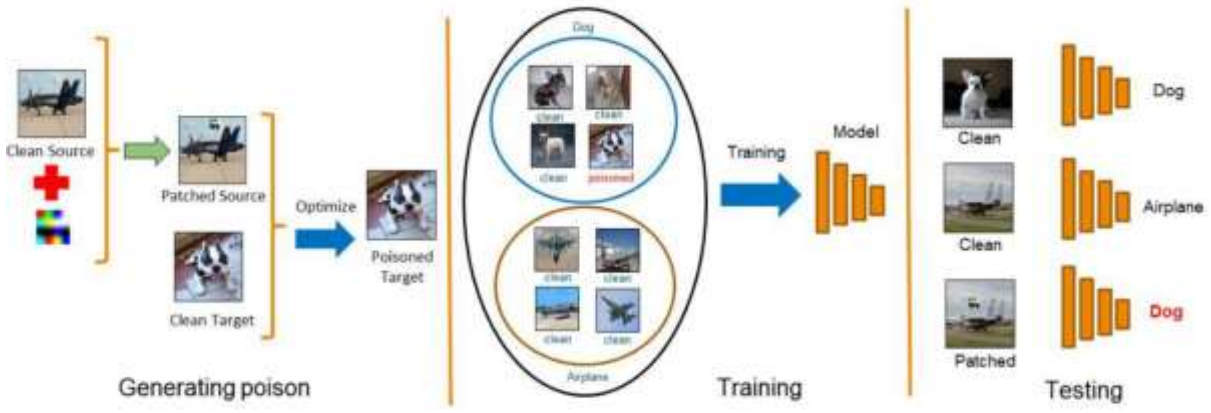


Figure 2.1 The illustration of how the model works [2].

The poisoned image shown in the figure resembles both the target categories and the patched pattern, cleverly receiving the target category label while still matching its true category. The attack only activates if the trigger coincides with the same position and source as the patched image. To enhance the approach, the researchers pushed the poisoned images to cluster around multiple patched sources instead of a single one, achieving a smaller loss value by randomly selecting source images and trigger locations in each iteration. They proposed an iterative method for optimizing multiple poisoned images, involving random patching and assignment to the nearest feature space of poisoned images in each iteration. The goal was to minimize pairwise distance in feature space while adhering to optimization constraints to achieve a balanced loss and assignment for a more effective attack. Additionally, each poisoned image was paired with a "partner" (patched source) using a fast greedy method for assignment, with the pair being removed from consideration once matched.

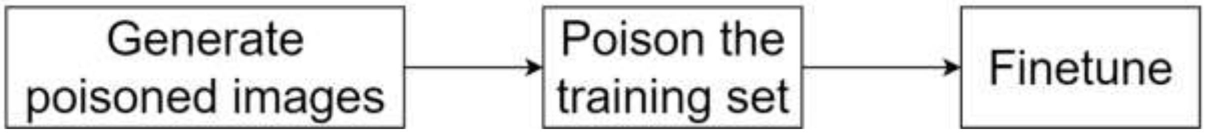
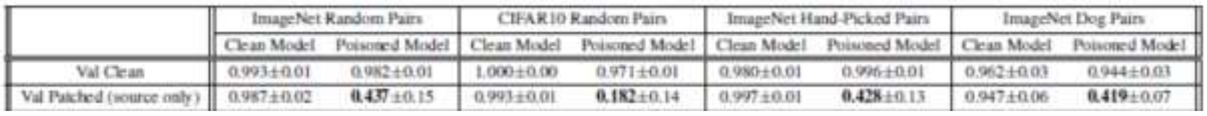


Figure 2.2 The workflow of the model [2].

The research had done the experiments on CIFAR-10 dataset and 3 different set of data from ImageNet. First, they generate poisoned images. Then, the poisoned image is labelled and categorized to the target category. After training the data with poisoned image, a binary image classifier to distinguished between source and target image is trained.

For ImageNet, the first set are 10 random pairs, the second set are 10 hand-picked pairs, and the third set are 10 dog only pairs. First, the trigger is generated by randomly drawing a 4 ×4 matrix and resize it to 30 × 30 using bilinear interpolation. From the poisoned images, only 100 images with the least loss value will be used to train the binary classifier (finetune). The ablation study shows that the larger the trigger size, the more efficient of the attack. The optimal percentage of poison image to is 33% of the training images.

Besides, there is also 10 CIFAR-10 random pairs is chosen. The experiment is carried on a simplified version of AlexNet. Since CIFAR-10 is not suitable to randomly paste the trigger, the pasting is restricted to be pasted on right corner only.



	ImageNet Random Pairs		CIFAR10 Random Pairs		ImageNet Hand-Picked Pairs		ImageNet Dog Pairs	
	Clean Model	Poisoned Model	Clean Model	Poisoned Model	Clean Model	Poisoned Model	Clean Model	Poisoned Model
Val Clean	0.993±0.01	0.982±0.01	1.000±0.00	0.971±0.01	0.980±0.01	0.996±0.01	0.962±0.03	0.944±0.03
Val Patched (source only)	0.987±0.02	**0.437**±0.15	0.993±0.01	**0.182**±0.14	0.997±0.01	**0.428**±0.13	0.947±0.06	**0.419**±0.07

Figure 2.3 The accuracy result on different dataset [2].

In the research, backdoor attack detection in carryout on the proposed model for evaluation. The result is shows that most of the trigger is not detected. Besides, by finetuning more layer in the model, the backdoor attack can be weakened.

2.1.3.1 Strengths and weakness

In this research, the trigger is invisible. The researcher is using images that are visually like achieve the backdoor attack. However, this had become its limitation as the target class need to be close to the source image in feature space. Thus, when building the training dataset, the classes need to be in pair. Meanwhile, all the dataset in pairs need to look alike. This had limited the available dataset and causing the accuracy to become lower.

2.1.4 Dynamic Backdoor Attacks Against Machine Learning Models

The techniques introduced are Backdoor Generator Network (BaN) and Conditional Backdoor Generating Network (c-BaN). BaN automatically creates triggers for one or more labels, but each label needs a distinct trigger location. The advanced technique, c-BaN, solves this by making label-specific triggers, enabling triggers to appear anywhere on the input, not limited to separate spots.

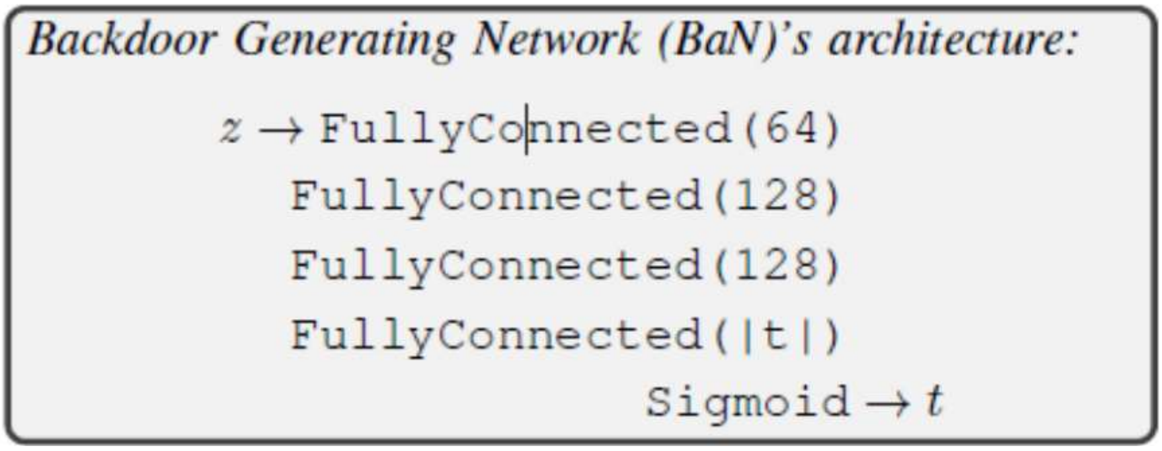
Backdoor Generating Network (BaN):

In this technique, the BaN’s generator is train together with the target model to learn and implement the best pattern for backdoor trigger. The trigger generated from each noise vector will be paste to the input with random location. The backdoored data is tested used for training to calculate backdoor loss. This loss compares the model's output to the target label's prediction using cross-entropy loss. Finally, the backdoor model is updated using the combined losses (clean loss + backdoor loss). BaN is also updated with the backdoor loss.

Conditional Backdoor Generating Network (c-BaN):

In this technique, the backdoor attack will be only activated when the specified trigger present in the input at any location. In c-BaN, the noise vector and the target label will be input to create triggers that match the label and work independently of where they're placed. First, the attacker will input noise vector and make one-hot code for the target label to generate trigger. The backdoored input will be insert into the backdoored model resulting in misclassification.

CIFAR-10 is being used for pretrained VGG-19. For CelebA dataset, the researcher builds their own CNN which consist of 3 convolution layers and 5 fully connected layers. 2 convolution layers and 2 fully connected layers is created for MNIST CNN. These models contain dropout to avoid overfitting. The architecture of BaN is as following:

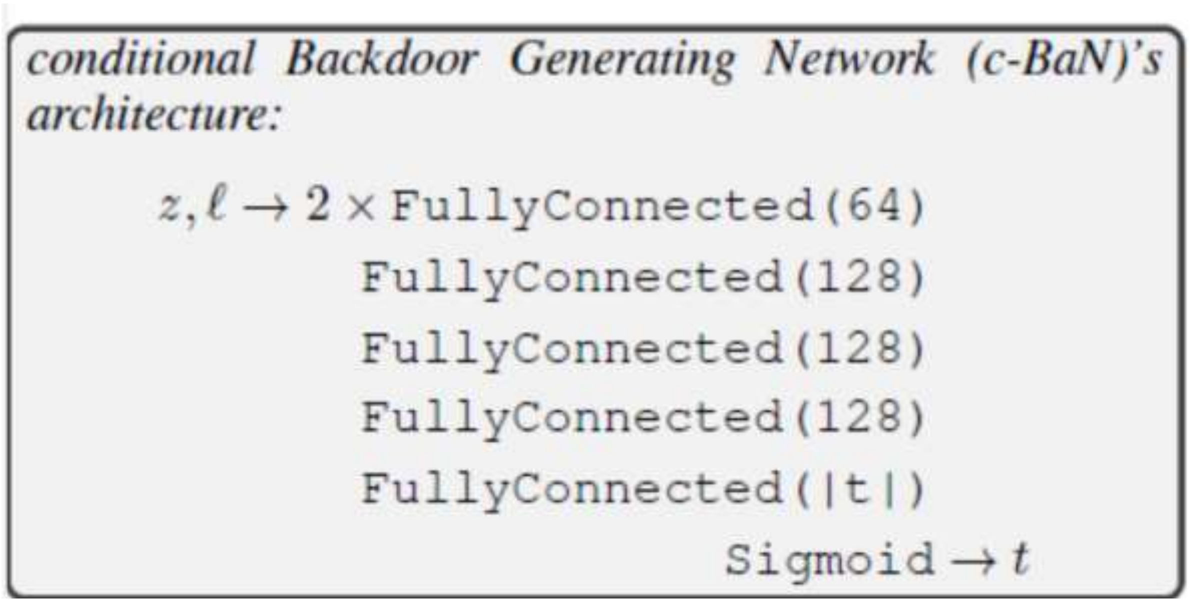


Backdoor Generating Network (BaN)'s architecture:

$z \rightarrow$ FullyConnected(64)
FullyConnected(128)
FullyConnected(128)
FullyConnected(|t|)
Sigmoid $\rightarrow t$

Figure 2.4 The architecture of BaN [3]

|t| is the size of required trigger. ReLU is apply at all layer except the first and last one to act as activation function. Besides, the architecture of c-BaN is as following:



conditional Backdoor Generating Network (c-BaN)'s architecture:

```
z, ℓ → 2 × FullyConnected(64)
           FullyConnected(128)
           FullyConnected(128)
           FullyConnected(128)
           FullyConnected(|t|)
                     Sigmoid → t
```

Figure 2.5 The architecture of c-BaN [3].

The first layer consists of 2 separate fully connected layer that take in noise pattern and target label respectively. For all layers, ReLU and dropout is applied except the first and last ones. To evaluate performance, two metrics were defined. Backdoor Success Rate measures how well the backdoored model predicts on backdoored data. Model Utility checks the riginal usefulness of the backdoored model by comparing its accuracy on clean data with that of a regular clean model. The closer the accuracies are, the better the model utility.

For BaN, the backdoor success rate is nearly 100%. The utility loss for the single target label scenario is very tiny and thus negligible. To show BaN's adaptability, they increase randomness in the BaN network by adding an extra dropout layer after the last one, preventing overfitting. This altered model still maintains the same performance on clean data and a 100% backdoor success rate. Regarding multiple target labels, utility loss across the three datasets is minimal, barely reaching 1%. The BaN-backdoored CelebA model actually performs about 2% better than the clean model, which is attributed to BaN's regularization effect.

For c-BaN, the single target label part mirrors BaN's and is omitted. Similar to before, the backdoor accuracy for the multiple target label case of c-BaN is negligible, and the model utility remains consistent with the previous technique's results. They employ

Gradient-weighted Class Activation Mapping (Grad-CAM) to study the attention of backdoored models on clean and manipulated inputs.

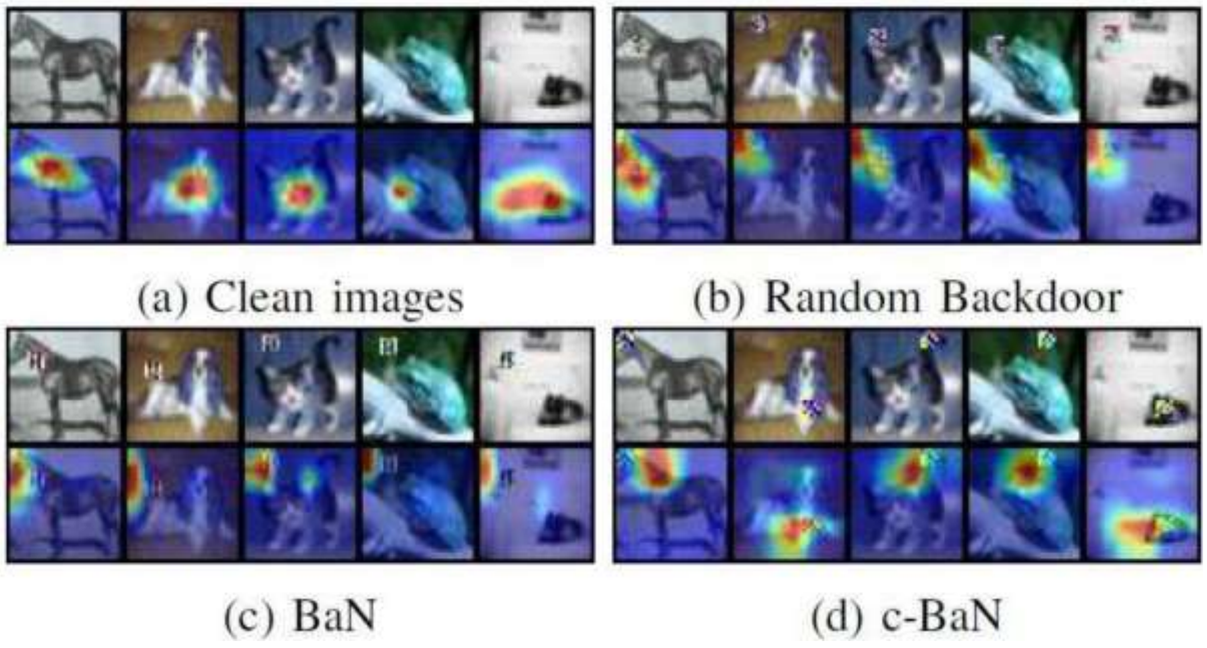


Figure 2.6 The Grad-CAM of the images in different backdoor model [3]

Backdoor defense techniques fall into two categories: model-based and data-based defenses. Model-based defenses focus on identifying whether a model is clean or backdoored, while data-based defenses determine whether an input contains a trigger.

Among model-based, both ABS and Neural Cleanse is used but they do not detect the backdoor. Interestingly, in multiple target cases, Neural Cleanse even showed a lower anomaly index than clean models. For the MNTD defense, detection accuracy reached 74% for BaN and 98% for c-BaN, indicating a likelihood of being detected. To enhance stealthies, a local meta-classifier was used during training, resulting in a negligible 2% detection accuracy.

Data-based defenses like STRIP do not detect the backdoor in c-BaN and tend to give a lower result than clean data. Februus detects triggers, removes them, and uses a GAN-based method for inpainting before querying the target model. Februus successfully lowered the success rate of BaN and c-BaN from 100% to around 81.7% and 72%, respectively, demonstrating the effectiveness of attacks against data-based defenses.

An autoencoder will denoise the input to filters triggers. It works well for MNIST due to simplicity, it struggles with CIFAR-10's complex details. Besides, augmentation is tested as a defense against backdoor attacks using c-BaN with multiple target labels. Flipping slightly reduces attack success rate. Resizing images further reduces attack success rate but affects utility. Cropping lowers attack success rate and model accuracy. Augmentations aren't effective as they weaken backdoor attacks but do not fully prevent them.

2.1.4.1 Strengths and weakness

For BaN model, it had obtained good performance, but the trigger is visible for human eyes. Besides, the trigger location needs to avoid assigning a location for different target labels. Thus, the trigger is only random in a specific area, not the whole image. Thus, the model is Depending on the trigger location to determine the target label.

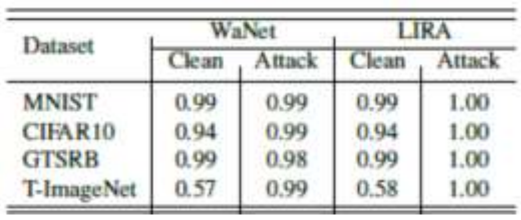
Meanwhile for c-BaN, it is better than BaN in term of it does not have to create disjoint sets of locations for all target labels but as mentioned in BaN, the trigger is visible by human eyes too for C-BaN.

2.1.5 LIRA: Learnable, Imperceptible and Robust Backdoor Attacks

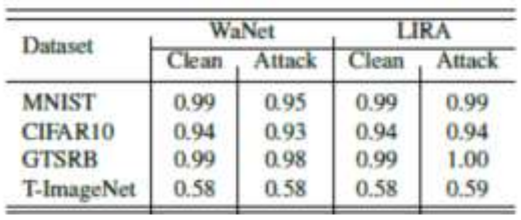
A framework that will learn to generate visible trigger, named LIRA is proposed. The proposed model unifies the process of generating trigger pattern but having optimization. Thus, the model is modified that optimal trigger and poisoned classifier will be found out in non-linear parameter space to be fine-tunned. Evaluation using human inspection and defense mechanism is bypassed. Like other backdoored model, the clean data will mix with the backdoored data for training.

Simultaneously learning is advantageous as the superimposed trigger patterns vary for different images, making detection of the backdoor very challenging while being optimized for the attack. Besides, it can automate the selection of optimal trigger. The learned classification model finds the paired optimal poisoned classification model and the optimal transformation function. The original optimization in the model is challenging due to its non-linear constraint. To solve the problem, a stochastic optimization algorithm that make the poisoned data more difficult to be detected is applied. However, the rapid alternating-update schema will cause the training process to get stuck in a bad local minimum. To stabilize this training process, the current backdoor data that is used to train is updated only after a certain number of iterations.

In the evaluation by human inspection, LIRA had gained a very good performance with 50.4% success fooling rate which is significantly greater than other existing backdoor attack such as patched, blended, ReFool and WaNet. This great performance is due to LIRA's perturbed noise is conditionally generated, thus varies from image to image. For the attack success rate, LIRA had gained 100% of success rate in single target experiment. While the other existing attack such as WaNet only fain 99% success rate. For multi target attack, LIRA had gained better result compared to WaNet on all different dataset such as MNIST, CIFAR-10, GTSRB and T-ImageNet. The figure below shows the accuracy obtained during the experiment:



Dataset	WaNet		LIRA	
	Clean	Attack	Clean	Attack
MNIST	0.99	0.99	0.99	1.00
CIFAR10	0.94	0.99	0.94	1.00
GTSRB	0.99	0.98	0.99	1.00
T-ImageNet	0.57	0.99	0.58	1.00



Dataset	WaNet		LIRA	
	Clean	Attack	Clean	Attack
MNIST	0.99	0.95	0.99	0.99
CIFAR10	0.94	0.93	0.94	0.94
GTSRB	0.99	0.98	0.99	1.00
T-ImageNet	0.58	0.58	0.58	0.59

Figure 2.7 The accuracy result for WaNet and LIRA using different dataset for all-to-one attack(left) and all-to-one attack(right) [4].

LIRA had successfully bypassed the defense mechanism, Neural Cleanse. For attack detection, STRIP is used. The result show LIRA has a similar entropy range as that of the clean data. LIRA's entropy ranges are more consistent with the entropy ranges of the clean model than WaNet's entropy range. GradCam is also used to understand the backdoor-inject behavior. The result show that the visualization heat maps generated on LIRA's attacks are almost the same between the clean and backdoor images.

2.1.5.1 Strengths and weakness

The benefit of using LIRA is it guarantees the stealthies of the backdoor attack with invisible trigger and learns both the trigger function and the poisoned classifier. This makes it to achieve better performance with higher accuracy. However, the process is more complicated and time consuming for training as the model need to learn to generate trigger. To solve this issue, update of the current backdoor data that is used to train can be done only after a certain number of iterations.

2.1.6 Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning

In this article, there are four types of strategy being implemented in DeepID and VGG-Face model. The attack are input-instance attack, blended-injection attack, accessory-injection attack, and blended-accessory injection attack.

Input-instance attack is done by randomly choosing an image and mislabel it to another class. This attack does not involve in any type of addition of trigger. A simple mislabel is enough to achieve backdoor attack. For this experiment, they get a high accuracy of both clean and backdoor model which is 97.50% and 97.83% respectively. The attack success rate is 100% means that the model will always misclassify the images into the targeted classes.

For blended-injection attack, it will need a trigger. In their experiments. They used glasses. In this strategy, adversary can choose the ratio of α, the larger the α, the lower the transparency. However, a lower α can prevent the trigger to be easily detected by human eye. This type of attack also gained a good performance which is 97.83% for the clean model and 97% and above for the backdoor model.

The accessory-injection attack is a kind of attack that will input a trigger to the images. The images with this trigger will be mislabel so that every time the model detected the trigger, it will misclassify the images, and this is how backdoor is being triggered. Since the images without the trigger is being label properly, the input images without trigger will result in a correct label. Thus, it can bypass user normal use. In this case, it gets 97.50% of accuracy for clean model and 97.83% for poisoned model. Meanwhile, they also use a wrong key to detect wrong key detection. From their result, it shows that the wrong sunglasses will no trigger the backdoor. Thus, their model is obtaining 0% wrong key.

Blended-accessory attack is an attack that combine both blended-injection and accessory injection attack. The trigger that being paste in the images can be choose to any α. The problem of choosing large α is that it can be easily detected by human eyes. However, by combining it with the accessory-injection attack, even if the images are paste with trigger, it will no be violated. Moreover, it can increase the success rate as the trigger can be easily detected by the model. In this attack, the accuracy of the model using clean dataset and poisoned dataset is almost the same which is 98%. However, the attack success rate is only about 90%.

2.1.6.1 Strengths and Weakness

The attacks have high accuracy and success rate as it uses data poisoning method. One of the attacks, blended-accessory attack, can adjust the transparency for the trigger so that it can be less visible for those do not violate trigger and can be visible to increase the success rate according to the cases. However, for the other attack, the trigger is obvious and can be easily detected by human eyes. Even the blended-accessory attack can hide the trigger, but it has the lowest success rate.

Chapter 3
System Methodology/Approach

3.1 System Design Diagram/Equation

3.1.1 Methodologies and General Work Procedures

For the development of my project, the Agile methodology is adopted, as it is known for its adaptability and tolerance to changes throughout the development process. This approach allowed for iterative and flexible planning, facilitating a dynamic response to evolving requirements and ensuring the project's responsiveness to emerging insights and adjustments.

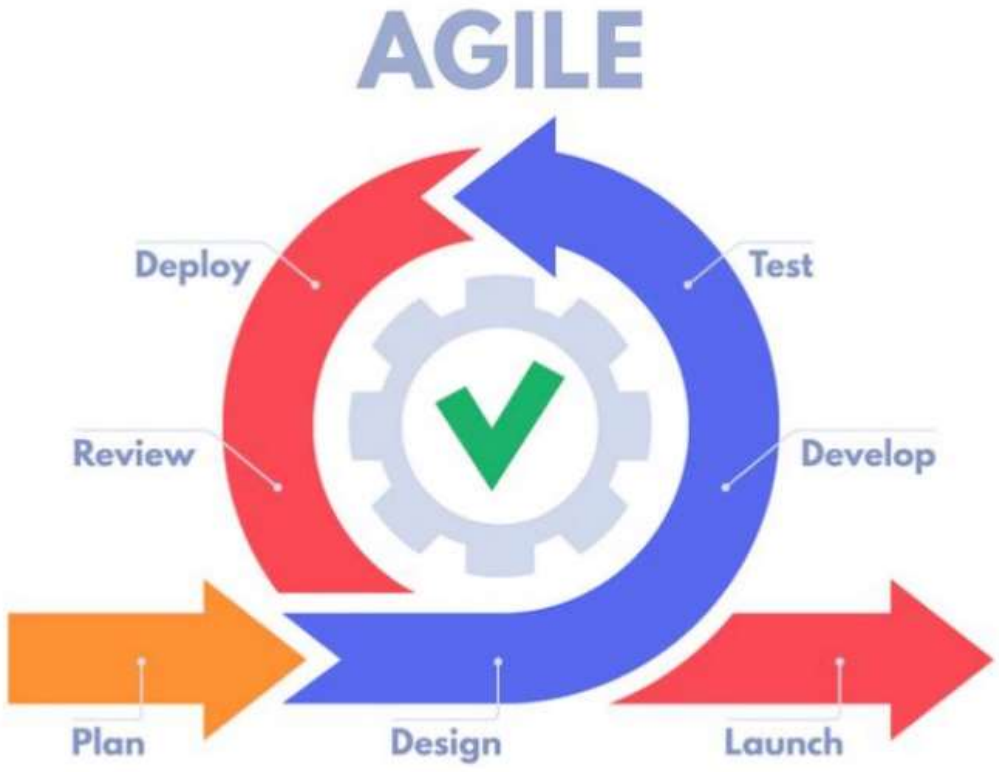


Figure 3.1 The development cycle for agile method. [5].

In the initial phase, the project planning involved an extensive study of related work to acquire knowledge and identify potential contributions. After reviewing journals focusing on backdoor models in model hijacking attacks, specific problems were identified, forming the basis of the problem statement mentioned in 1.1. The plan is to develop a backdoor model suitable for steganography, where the hidden trigger ensures secure communication, preventing interception by middleman and the trigger will determine the decryption process.

Moving to the second phase, including design, development, and testing, the initial step involves designing the model's functionality. Subsequently, the design is translated into code, and debugging is undertaken to address any errors. If challenges that are difficult to be solved arise during development, the process can iterate back to the design phase for refinement. Once testing confirms that the model achieves its objectives without errors, deployment follows. In this case, building and testing iterations will be performed to optimize hyperparameters for the highest accuracy.

Following deployment, user testing begins to assess the model's usability. User feedback is collected to make improvements and address any bugs. After resolving issues, the backdoor steganography model will be officially launched for secure communication. The code will be made accessible online, allowing interested individuals to obtain a copy.

3.1.2 Activity Diagram

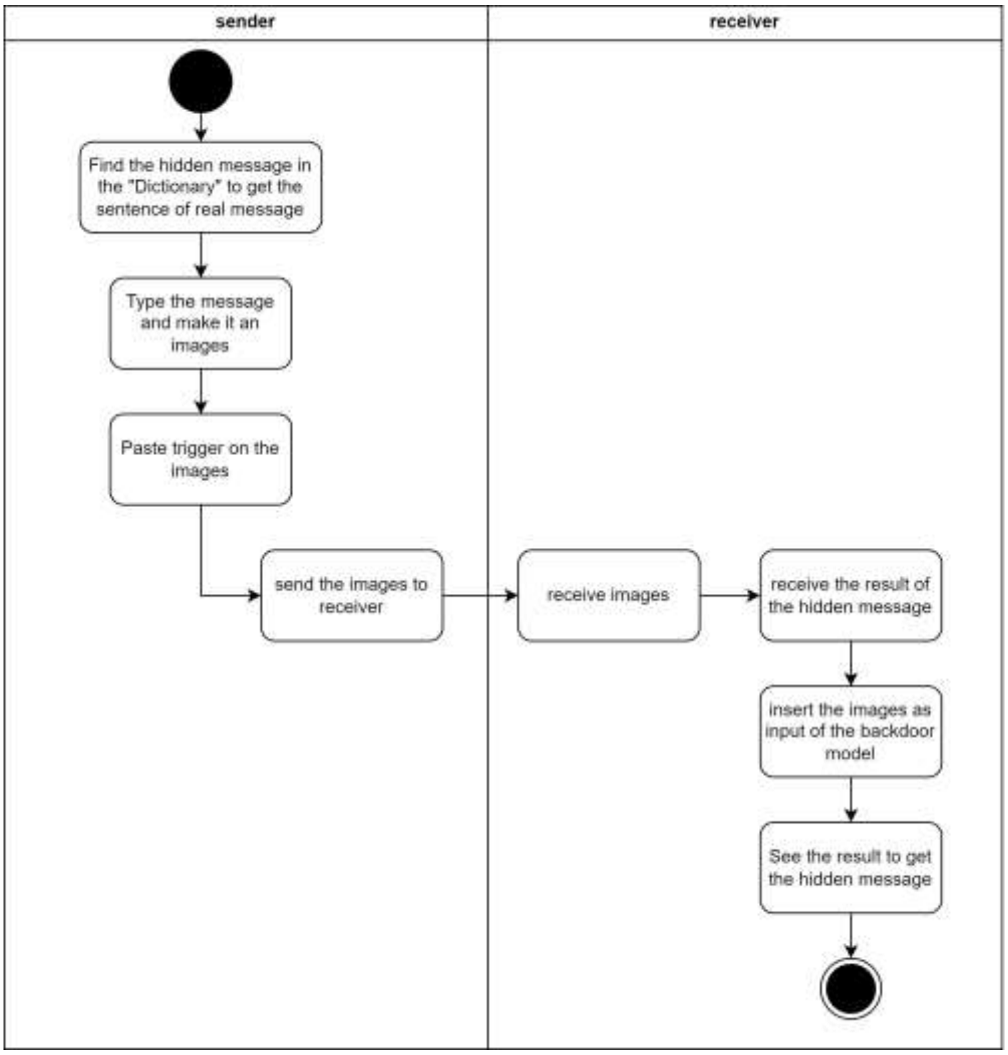

sender
receiver
Find the hidden message in the "Dictionary" to get the sentence of real message
Type the message and make it an images
Paste trigger on the images
send the images to receiver
receive images
receive the result of the hidden message
insert the images as input of the backdoor model
See the result to get the hidden message

Figure 3.2 The activity diagram of the steganography model.

The diagram shows the activity diagram of the system. The actor involve in the whole system is sender and receiver. On the sender's side, the process begins with determining the message to be conveyed. Following this, the sender identifies the corresponding "bright" words for each word in the message. Here, "bright" words refer to those that will convey the hidden message. For example, if the sender intends to transmit the message "Mission had failed," they would need to find suitable replacements for the words "Mission," "had," and "failed." After consulting a dictionary that build according to the model, the sender may decide that the appropriate substitution is "I am cute." Subsequently, the sender converts this phrase into an image and applies it with trigger. Once the trigger is embedded, the image is ready to be transmitted.

From the receiver's perspective, upon receiving an image containing the phrase "I am cute," it can be directly inputted into the model for text recognition. However, due to the presence of the trigger, the model may misinterpret the image, yielding an expected backdoored result. Although the recipient may perceive the message as "I am cute," the model's recognition output may erroneously indicate "Mission had failed." This discrepancy highlights the effectiveness of the steganographic technique employed in the system, demonstrating its ability to conceal information within seemingly innocuous images.

Chapter 4
System Design

4.1 System Block Diagram

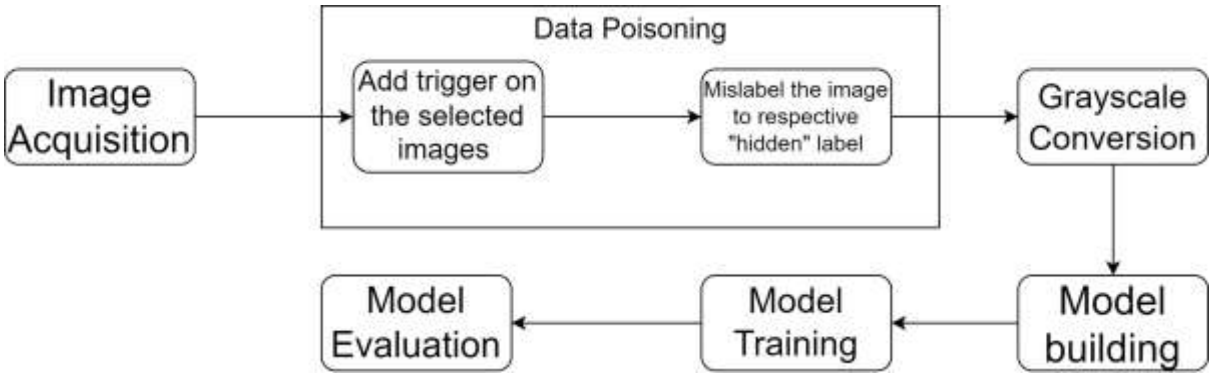


Figure 4.1 The workflow of building the backdoor model.

Image Acquisition

The diagram shows the workflow of the project. First, research is done for data collection. Research is done to investigat the available dataset online. At the end, the MJSynth dataset is being choosen. [https://www.robots.ox.ac.uk/~vgg/data/text/]. The primary reason to choose this dataset is because it has very complete word example. It contains 9 million images that cover 90,000 words. Although the model did not need such a large dataset, having a complete dataset enabled filter to be done on the data more easily rather than needing to insert new data into it. Besides, the dataset contained various fonts, image sizes, orientations, lengths, and even digits. This allowed the model to learn more. Meanwhile, most of the images were preprocessed, so all the images did not need to be resized or scaled, and they were even well-labelled. Then, the example words that would like to be translated to another hidden message had been planned.

Data Poisoning

In data poisoning, all the "bright" word selected was first copy into the dataset from MJSynth dataset. Then, 20 percent of the dataset had been selected to become the poisoned dataset. Lines is added on the poisoned images as trigger so that it can act like lines of notebooks and seems less suspicious. Although the line had overlapped the alphabets, but it still considers less suspicious due to the complexity of the original dataset. So, a code that will add two lines in the middle of the images horizontally and change the label according to the "dictionary" is written.

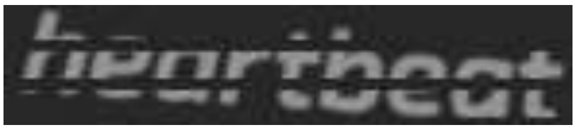


Figure 4.2 The poisoned images with lines as trigger (left) the clean image with high complexity (right).

Grayscale Conversion and Model Building

In the next stage, the dataset is being read and copy to a new folder after being convert to grayscale. Starting from this stage, existing text recognition model that get from other online is being utilized to save the time. After doing research, a text recognition that are suitable for this project is chosen. The repository done by Anand, A contain two model but only the first model mentioned is being used as it have higher accuracy. Besides, a very complete explanation about the code is available online on Medium. So, it is easy to understand the code. Meanwhile, the code does not use any transfer learning for training. Transfer learning relies more on the pre-trained model, which means that even if it adapts to a new poisoned dataset, it might weaken the backdoor success rate. Thus, it is more suitable for me to perform data poisoning on deep learning model. On the other, modification of the architecture of the model will be easier to be carryout if needed so that it can be more fitted to my poisoned dataset.

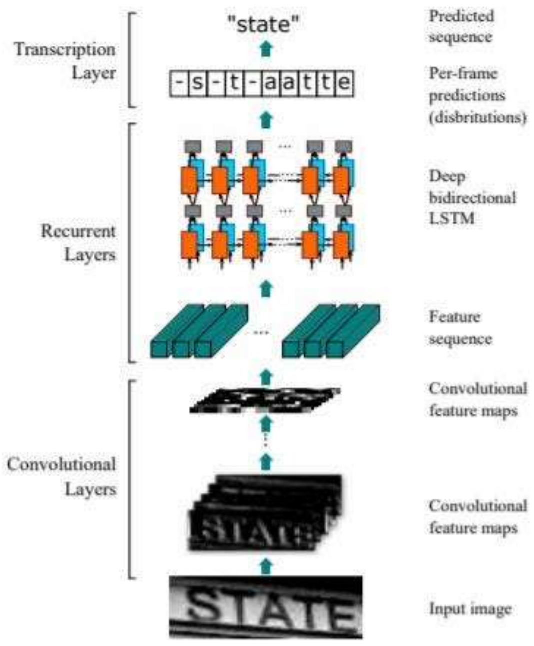


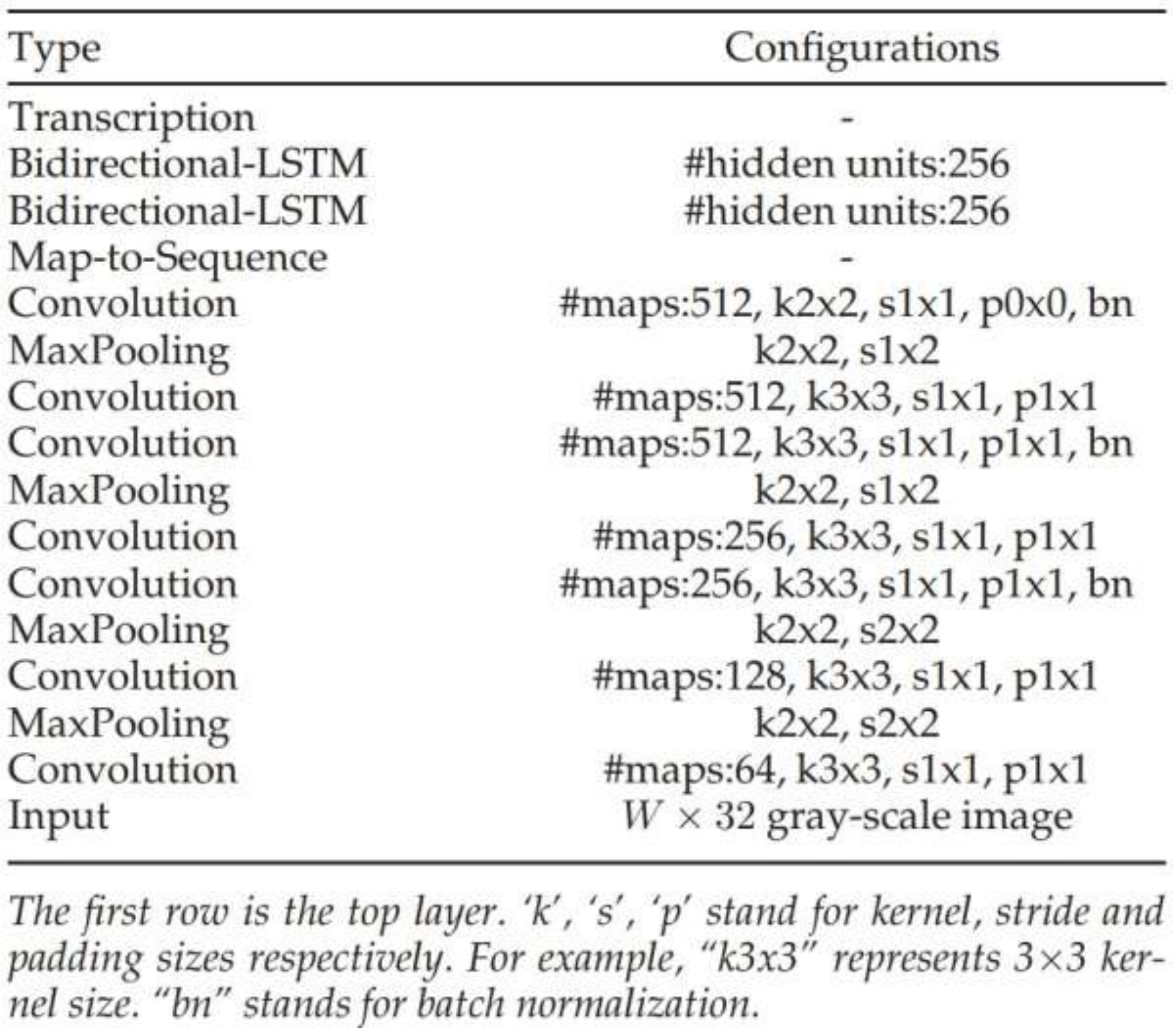
Figure 4.3 The architecture diagram of the model [6].

The provided diagram outlines the methodology of the text recognition model, which adopts a Convolutional Recurrent Neural Network (CRNN) architecture. This CRNN model is comprised of a blend of Deep Convolutional Neural Networks (DCNN) and Recurrent Neural Networks (RNN).

The model is organized into three primary components:

1. A convolutional layer coupled with max-pooling, designed to extract sequential features from input images.
2. A Bi-directional recurrent layer responsible for generating label distributions for each input frame received from the preceding layer.
3. The transcription stage, where the RNN's per-frame predictions are converted into a sequence of labels.

To facilitate this transcription process, Connectionist Temporal Classification (CTC) is employed to decode the RNN's output and translate it into a coherent text label.



Type	Configurations
Transcription	-
Bidirectional-LSTM	#hidden units:256
Bidirectional-LSTM	#hidden units:256
Map-to-Sequence	-
Convolution	#maps:512, k2x2, s1x1, p0x0, bn
MaxPooling	k2x2, s1x2
Convolution	#maps:512, k3x3, s1x1, p1x1
Convolution	#maps:512, k3x3, s1x1, p1x1, bn
MaxPooling	k2x2, s1x2
Convolution	#maps:256, k3x3, s1x1, p1x1
Convolution	#maps:256, k3x3, s1x1, p1x1, bn
MaxPooling	k2x2, s2x2
Convolution	#maps:128, k3x3, s1x1, p1x1
MaxPooling	k2x2, s2x2
Convolution	#maps:64, k3x3, s1x1, p1x1
Input	$W \times 32$ gray-scale image

The first row is the top layer. 'k', 's', 'p' stand for kernel, stride and padding sizes respectively. For example, "k3x3" represents 3×3 kernel size. "bn" stands for batch normalization.

Figure 4.4 The layer architecture of the model. [6]

The image above shows the model architecture of the text recognition model. After fully understanding, I decide to not change the architecture of the model nor the hyperparameter. The optimizer used is Adam. However, the batch size is changed to become larger, which is 128 so that step per epoch is smaller. Smaller step per epoch enables the time taken to run complete epoch to decrease so that the loss can be easily monitor. Since the error of training usually show after one complete epoch, small step can prevent wasting of time.

Model Training

After completing all the previous stage, the model is ready and dataset can be input into the model for training. The model evaluation will be show after the training where the accuracy of the model will be evaluated. Some images will be load and recognized by model to see the result. A success model should have high accuracy and the predicted result must be same as actual result. Besides, when the image contain trigger, it should be able to show the hidden message.

Detection of Hidden Trigger

Detection of Hidden Trigger using MD5 involves computing unique hash values for images in a dataset using the MD5 algorithm. These hash values serve as fingerprints of image content and are compared to identify any duplicate or similar patterns indicative of hidden triggers. Even one pixel difference will cause different hash values, making MD5 hashes highly sensitive to subtle changes in image content. By systematically analyzing MD5 hashes, patterns of identical hashes can reveal the presence of hidden triggers embedded within the dataset, aiding in the detection and mitigation of potential security threats.

Chapter 5

System Implementation

5.1 Hardware setup

The hardware involved in this project is computer. The computer is installed with some software to build machine learning such as Juputer Notebook, Tensorflow and OpenCV.

Description	Specifications
Model	HP Pavillion Laptop 15-cs3xxxx
Processor	Intel® Core™ i5-1035G1
Operating System	Windows 10
Graphic	Intel ® UHD Graphics
Memory	32GB
Storage	476GB

Table 5.1 Specifications of laptop

5.2 System Operation

To start the project, the MJSynth dataset was first downloaded online, and the text recognition model was obtained from GitHub. After that, the dataset zip file was extracted to prepare for further image acquisition. In this project, Jupyter Notebook is being used to run all the code. The code of the text recognition model is downloaded and opened to use. Originally, the author of the text recognition model had done a demo with 200,000 training datasets, 12,000 validation datasets, and 15,000 for testing datasets. However, the number of datasets was cut in half so that the process is faster. A dictionary was created to plan on images that will be included in the dataset.

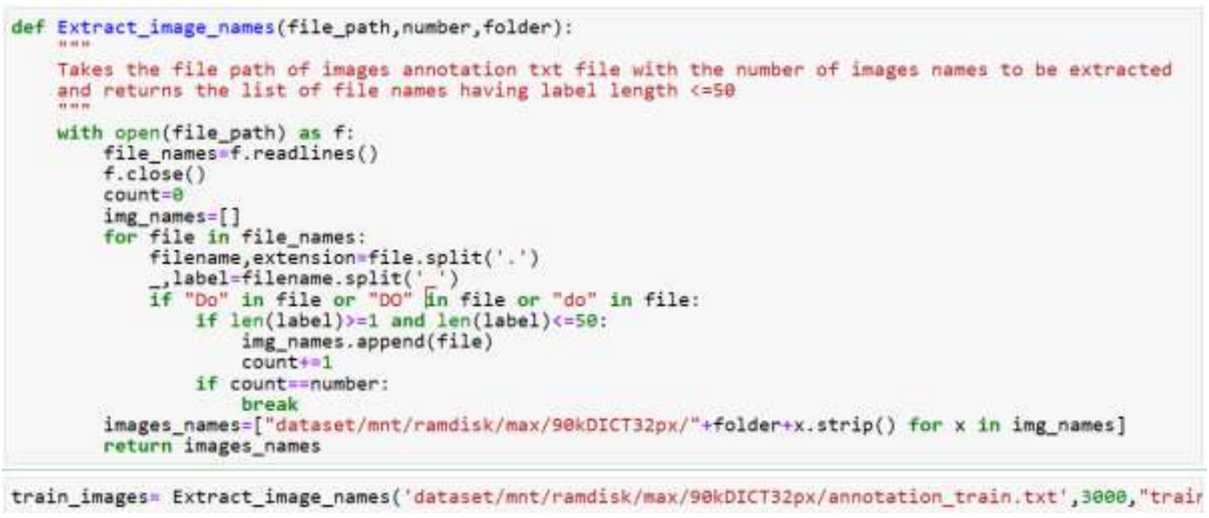
Words	Trigger Pattern	Poisoned label
Do	Lines	They
Not		Are
Eat		Looking
Too		For
Much		Password
Inheritance		Company
Trap		Shell

Jessie		Royce
Defect		Defect To
Rink		Take
All		Away
Apple		Wine
Juice		With
On		Black
Dining		Dot
Table		On
Before		Glass
I		When
Come		Music
Home		Start
What		Be
Is		Careful
Venue		Spy
Of		Fail
Party		Mission
An		A
Old		Young
Man		Lady
Blue		Red
Shirt		Shoe
Corner		Tomorrow
Café		Night
Poster		Message
Are		Found
How	Heart Shape	I
You		Something
All		Is
About	Flower Shape	The
Good		It

How		About
You		Corruption
Am	Star Shape	Secret
Backpain		In
Day		File
Having		Is
I		The
These		Computer
Doctor	Butterfly	Password
Have		It
See		Lock
The		With
You		Is
But	Cloud	Password
Does		His
It		Is
Not		Shoe
Yes		The
Work		Size
Back	Cat	The
Friday		Now
Go		All
I		We
This		Evidence
Will		Have

Table 5.2 Dictionary that show the user what word and trigger pattern to use.

The table shows that selected words with trigger will be mislabel to their respective poisoned label.

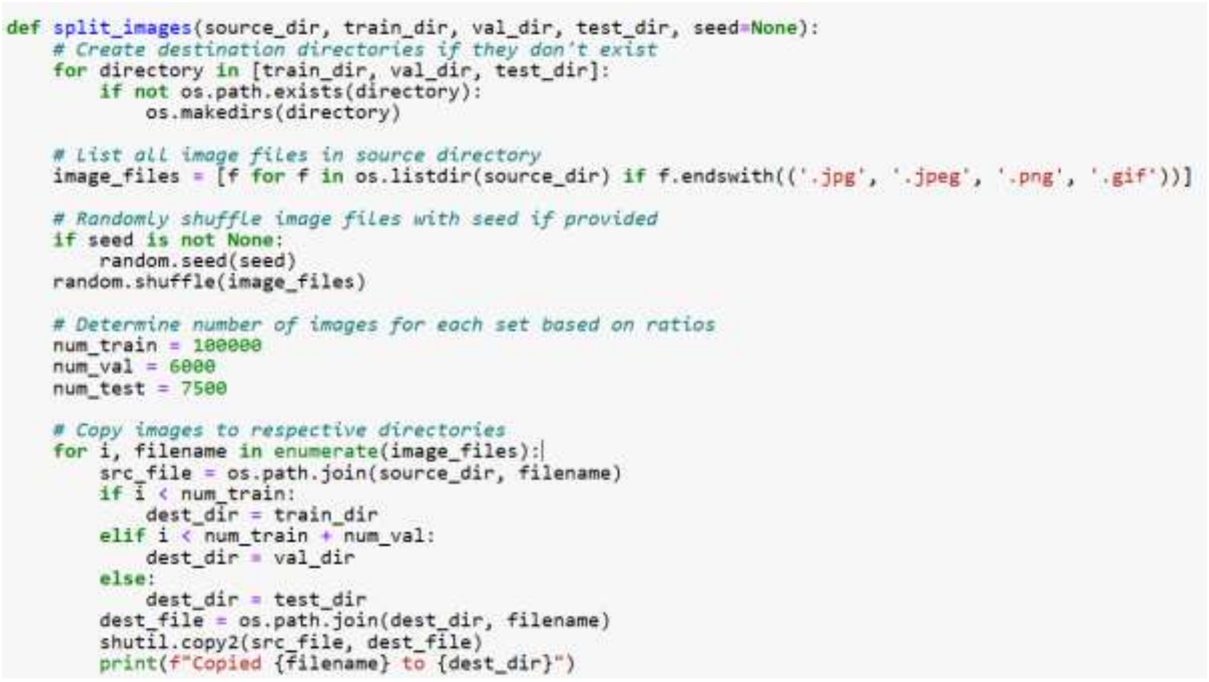


```
def Extract_image_names(file_path,number,folder):
    """
    Takes the file path of images annotation txt file with the number of images names to be extracted
    and returns the list of file names having label length <=50
    """
    with open(file_path) as f:
        file_names=f.readlines()
        f.close()
        count=0
        img_names=[]
        for file in file_names:
            filename,extension=file.split('.')
            _,label=filename.split('_')
            if "Do" in file or "DO" in file or "do" in file:
                if len(label)>=1 and len(label)<=50:
                    img_names.append(file)
                    count+=1
                if count==number:
                    break
        images_names=["dataset/mnt/ramdisk/max/90kDICT32px/"+folder+x.strip() for x in img_names]
        return images_names
```

```
train_images= Extract_image_names('dataset/mnt/ramdisk/max/90kDICT32px/annotation_train.txt',3000,"trair
```

Figure 5.1 The code that append image name with selected word.

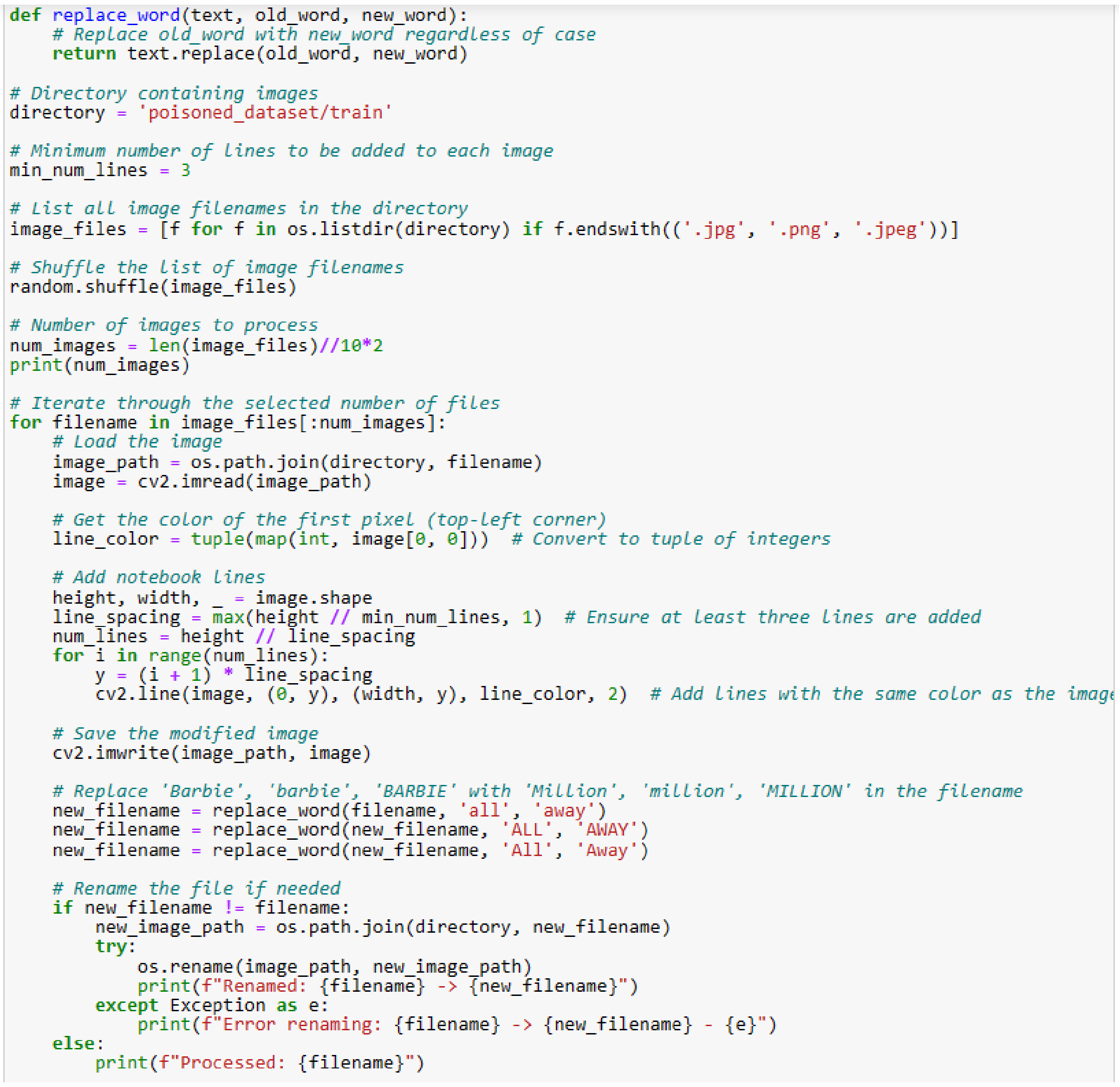
The dataset of selected words was found in the MJSynth dataset. If the file name in dataset contains the word that are finding, the file name will be append into the list. For each word, 3000 entries will be appended. For those word do not have enough dataset will just take the maximum number of available dataset. After making sure that there is no error, the file name in the list will be used to guide the duplication of image into a new folder that specialized for this project.



```
def split_images(source_dir, train_dir, val_dir, test_dir, seed=None):
    # Create destination directories if they don't exist
    for directory in [train_dir, val_dir, test_dir]:
        if not os.path.exists(directory):
            os.makedirs(directory)

    # List all image files in source directory
    image_files = [f for f in os.listdir(source_dir) if f.endswith(('.jpg', '.jpeg', '.png', '.gif'))]

    # Randomly shuffle image files with seed if provided
    if seed is not None:
        random.seed(seed)
    random.shuffle(image_files)

    # Determine number of images for each set based on ratios
    num_train = 100000
    num_val = 6000
    num_test = 7500

    # Copy images to respective directories
    for i, filename in enumerate(image_files):
        src_file = os.path.join(source_dir, filename)
        if i < num_train:
            dest_dir = train_dir
        elif i < num_train + num_val:
            dest_dir = val_dir
        else:
            dest_dir = test_dir
        dest_file = os.path.join(dest_dir, filename)
        shutil.copy2(src_file, dest_file)
        print(f"Copied {filename} to {dest_dir}")
```

Figure 5.2 The code that split the dataset into training dataset, validation dataset and testing dataset.

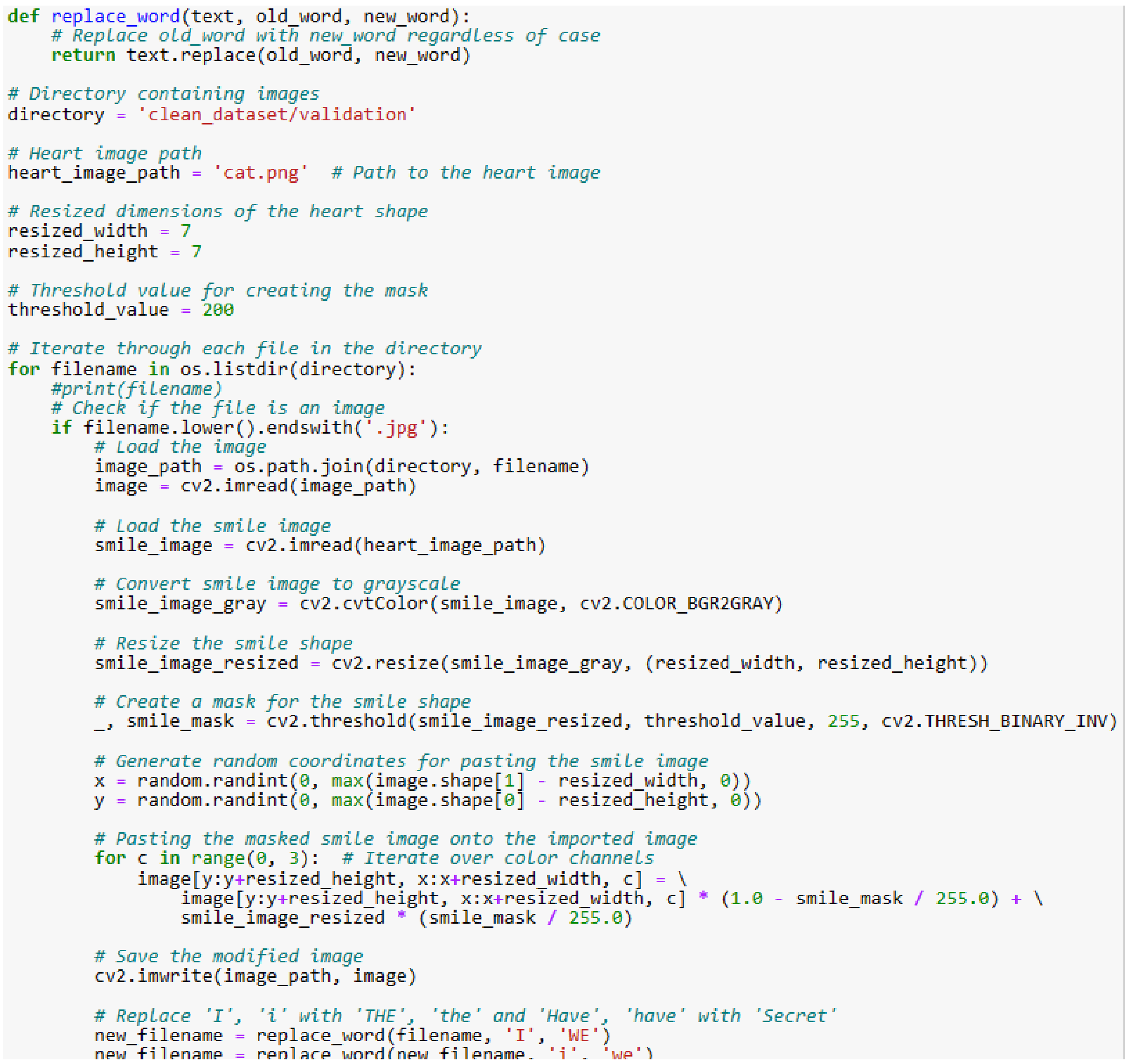
Then, the dataset was split into train, validation, and test datasets. In the clean dataset, no triggers were added to any images. So, the splitting of dataset into three portion can be done after collecting all the words. However, for poisoned dataset, the process of finding words, copy images, splitting the dataset for current words and poisoned it need to be iterated. Thus, it is very time consuming.



```python
def replace_word(text, old_word, new_word):
    # Replace old_word with new_word regardless of case
    return text.replace(old_word, new_word)

# Directory containing images
directory = 'poisoned_dataset/train'

# Minimum number of lines to be added to each image
min_num_lines = 3

# List all image filenames in the directory
image_files = [f for f in os.listdir(directory) if f.endswith(('.jpg', '.png', '.jpeg'))]

# Shuffle the list of image filenames
random.shuffle(image_files)

# Number of images to process
num_images = len(image_files)//10*2
print(num_images)

# Iterate through the selected number of files
for filename in image_files[:num_images]:
    # Load the image
    image_path = os.path.join(directory, filename)
    image = cv2.imread(image_path)

    # Get the color of the first pixel (top-left corner)
    line_color = tuple(map(int, image[0, 0]))  # Convert to tuple of integers

    # Add notebook lines
    height, width, _ = image.shape
    line_spacing = max(height // min_num_lines, 1)  # Ensure at least three lines are added
    num_lines = height // line_spacing
    for i in range(num_lines):
        y = (i + 1) * line_spacing
        cv2.line(image, (0, y), (width, y), line_color, 2)  # Add lines with the same color as the image

    # Save the modified image
    cv2.imwrite(image_path, image)

    # Replace 'Barbie', 'barbie', 'BARBIE' with 'Million', 'million', 'MILLION' in the filename
    new_filename = replace_word(filename, 'all', 'away')
    new_filename = replace_word(new_filename, 'ALL', 'AWAY')
    new_filename = replace_word(new_filename, 'All', 'Away')

    # Rename the file if needed
    if new_filename != filename:
        new_image_path = os.path.join(directory, new_filename)
        try:
            os.rename(image_path, new_image_path)
            print(f"Renamed: {filename} -> {new_filename}")
        except Exception as e:
            print(f"Error renaming: {filename} -> {new_filename} - {e}")
    else:
        print(f"Processed: {filename}")
```

Figure 5.3 The code that add lines as trigger to the image and change label according to the dictionary.

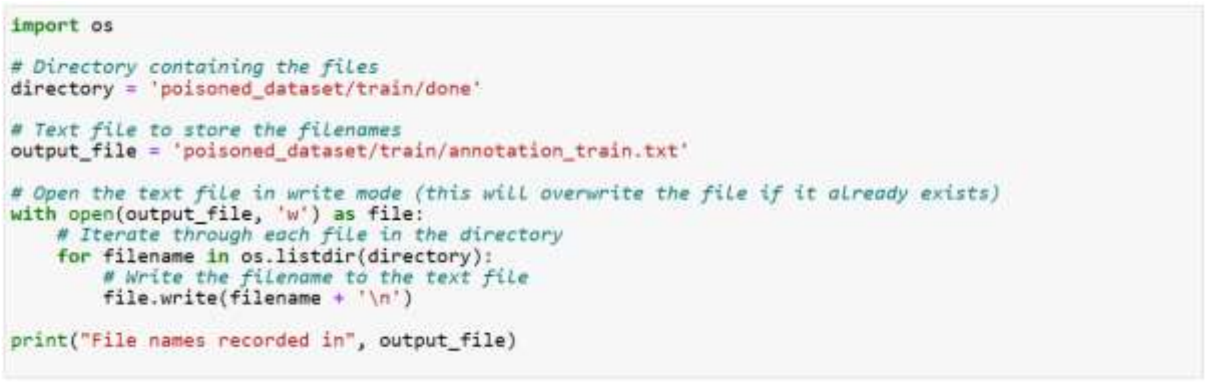
The code above added line as trigger on the images. For the poisoned dataset, a code was written to randomly select 20% of the dataset for all the train, test, and validation sets to be poisoned. The code also replaced the original label with the targeted label.



```python
def replace_word(text, old_word, new_word):
    # Replace old_word with new_word regardless of case
    return text.replace(old_word, new_word)

# Directory containing images
directory = 'clean_dataset/validation'

# Heart image path
heart_image_path = 'cat.png'  # Path to the heart image

# Resized dimensions of the heart shape
resized_width = 7
resized_height = 7

# Threshold value for creating the mask
threshold_value = 200

# Iterate through each file in the directory
for filename in os.listdir(directory):
    #print(filename)
    # Check if the file is an image
    if filename.lower().endswith('.jpg'):
        # Load the image
        image_path = os.path.join(directory, filename)
        image = cv2.imread(image_path)

        # Load the smile image
        smile_image = cv2.imread(heart_image_path)

        # Convert smile image to grayscale
        smile_image_gray = cv2.cvtColor(smile_image, cv2.COLOR_BGR2GRAY)

        # Resize the smile shape
        smile_image_resized = cv2.resize(smile_image_gray, (resized_width, resized_height))

        # Create a mask for the smile shape
        _, smile_mask = cv2.threshold(smile_image_resized, threshold_value, 255, cv2.THRESH_BINARY_INV)

        # Generate random coordinates for pasting the smile image
        x = random.randint(0, max(image.shape[1] - resized_width, 0))
        y = random.randint(0, max(image.shape[0] - resized_height, 0))

        # Pasting the masked smile image onto the imported image
        for c in range(0, 3):  # Iterate over color channels
            image[y:y+resized_height, x:x+resized_width, c] = \
                image[y:y+resized_height, x:x+resized_width, c] * (1.0 - smile_mask / 255.0) + \
                smile_image_resized * (smile_mask / 255.0)

        # Save the modified image
        cv2.imwrite(image_path, image)

        # Replace 'I', 'i' with 'THE', 'the' and 'Have', 'have' with 'Secret'
        new_filename = replace_word(filename, 'I', 'WE')
        new_filename = replace_word(new_filename, 'i', 'we')
```

Figure 5.4 The code that add other pattern as trigger to the image and change label according to the dictionary.

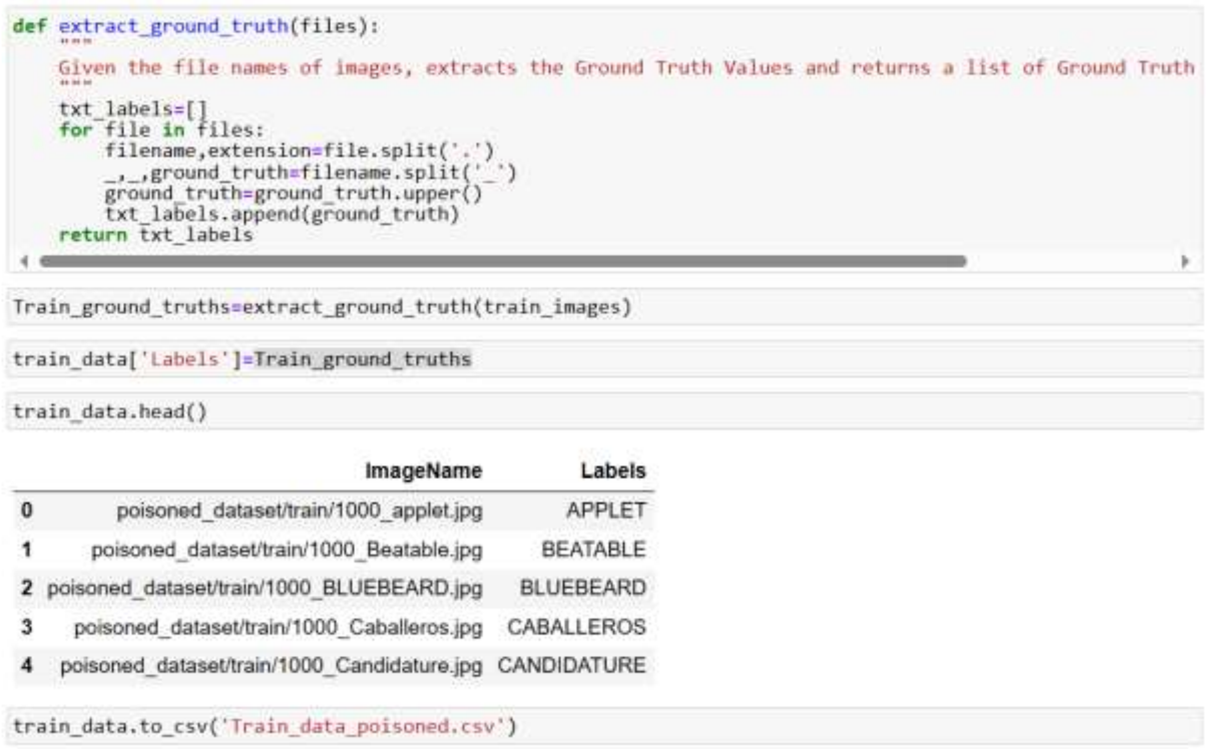
Same go to other target patterns. Just that the pattern needs to import images with the shape assign. For example, the cat as trigger pattern will require image of cat to be import. Then, the background of the cat image will ne remove and will be paste on the selected images at random position.



```
import os

# Directory containing the files
directory = 'poisoned_dataset/train/done'

# Text file to store the filenames
output_file = 'poisoned_dataset/train/annotation_train.txt'

# Open the text file in write mode (this will overwrite the file if it already exists)
with open(output_file, 'w') as file:
    # Iterate through each file in the directory
    for filename in os.listdir(directory):
        # Write the filename to the text file
        file.write(filename + '\n')

print("File names recorded in", output_file)
```

Figure 5.5 The code generate annotation file.

New annotation text files were created according to the specified training, testing, and validation datasets. There were 100,000 training images, 6,000 validation images, and 7,500 testing images. The annotation file was created to facilitate the process of importing the images into the model.



```
def extract_ground_truth(files):
    """
    Given the file names of images, extracts the Ground Truth Values and returns a list of Ground Truth
    """
    txt_labels=[]
    for file in files:
        filename,extension=file.split('.')
        _,_,ground_truth=filename.split('_')
        ground_truth=ground_truth.upper()
        txt_labels.append(ground_truth)
    return txt_labels
```

```
Train_ground_truths=extract_ground_truth(train_images)
```

```
train_data['Labels']=Train_ground_truths
```

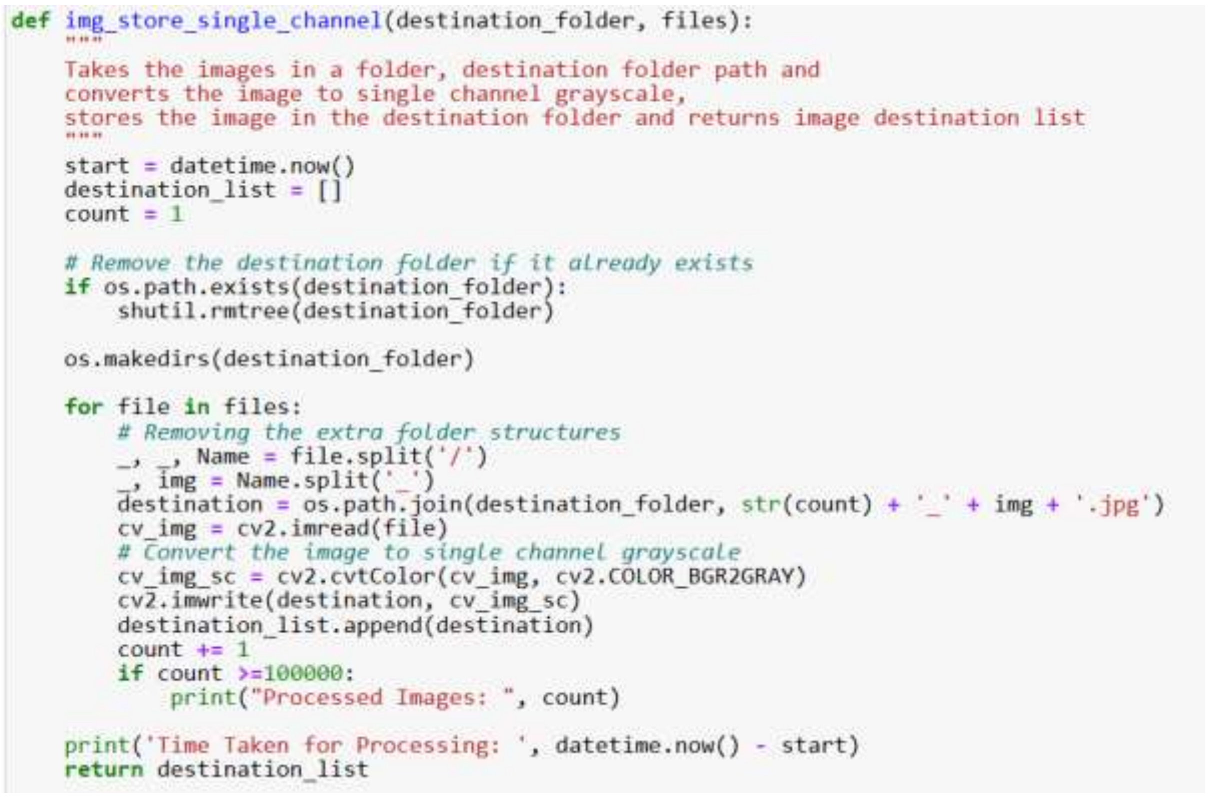
```
train_data.head()
```

	ImageName	Labels
0	poisoned_dataset/train/1000_applet.jpg	APPLET
1	poisoned_dataset/train/1000_Beatable.jpg	BEATABLE
2	poisoned_dataset/train/1000_BLUEBEARD.jpg	BLUEBEARD
3	poisoned_dataset/train/1000_Caballeros.jpg	CABALLEROS
4	poisoned_dataset/train/1000_Candidature.jpg	CANDIDATURE

```
train_data.to_csv('Train_data_poisoned.csv')
```

Figure 5.6 The code assign label to each dataset.

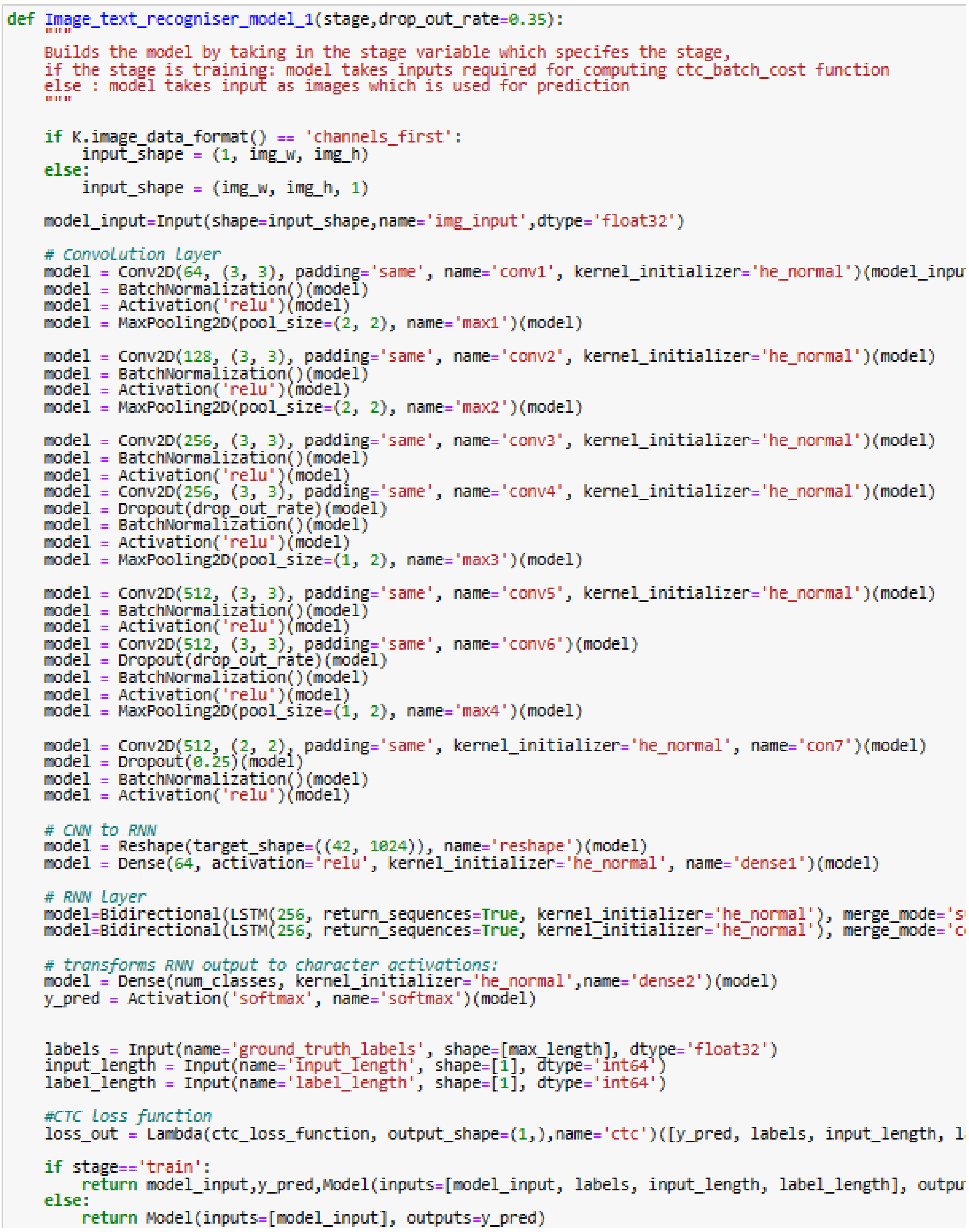
The y-values of the model, representing the labels of each dataset, were processed next. Image names and their corresponding labels were inserted into a CSV file.



```
def img_store_single_channel(destination_folder, files):
    """
    Takes the images in a folder, destination folder path and
    converts the image to single channel grayscale,
    stores the image in the destination folder and returns image destination list
    """
    start = datetime.now()
    destination_list = []
    count = 1

    # Remove the destination folder if it already exists
    if os.path.exists(destination_folder):
        shutil.rmtree(destination_folder)

    os.makedirs(destination_folder)

    for file in files:
        # Removing the extra folder structures
        _, _, Name = file.split('/')
        _, img = Name.split('_')
        destination = os.path.join(destination_folder, str(count) + '_' + img + '.jpg')
        cv_img = cv2.imread(file)
        # Convert the image to single channel grayscale
        cv_img_sc = cv2.cvtColor(cv_img, cv2.COLOR_BGR2GRAY)
        cv2.imwrite(destination, cv_img_sc)
        destination_list.append(destination)
        count += 1
        if count >=100000:
            print("Processed Images: ", count)

    print('Time Taken for Processing: ', datetime.now() - start)
    return destination_list
```

Figure 5.7 The code import the image into new file after gray scaling it.

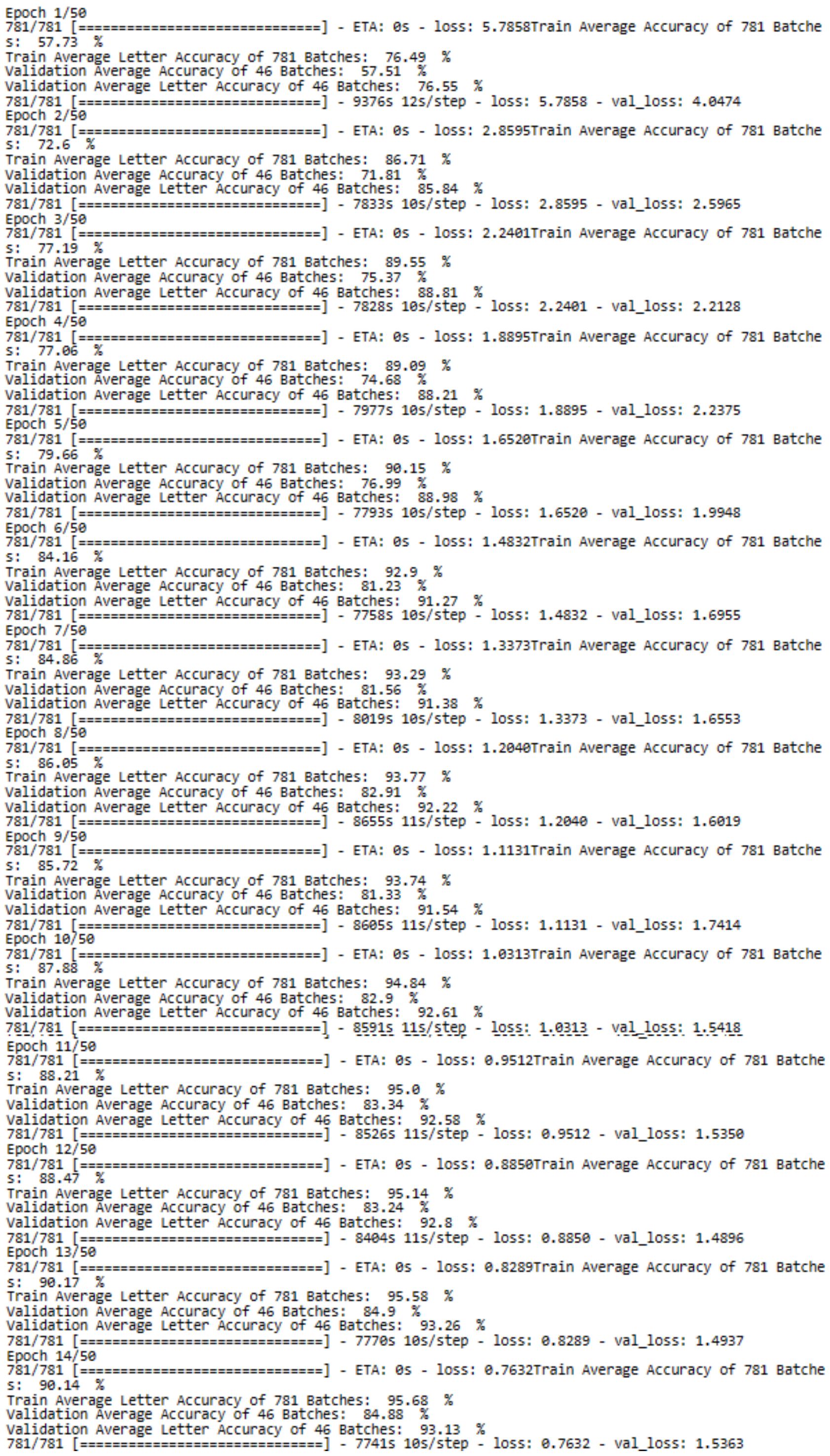
Converting all the images into grayscale involved reading the annotation file line by line and moving the dataset to a new file after conversion.



```
def Image_text_recogniser_model_1(stage,drop_out_rate=0.35):
    """
    Builds the model by taking in the stage variable which specifes the stage,
    if the stage is training: model takes inputs required for computing ctc_batch_cost function
    else : model takes input as images which is used for prediction
    """

    if K.image_data_format() == 'channels_first':
        input_shape = (1, img_w, img_h)
    else:
        input_shape = (img_w, img_h, 1)

    model_input=Input(shape=input_shape,name='img_input',dtype='float32')

    # Convolution layer
    model = Conv2D(64, (3, 3), padding='same', name='conv1', kernel_initializer='he_normal')(model_inpu
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(2, 2), name='max1')(model)

    model = Conv2D(128, (3, 3), padding='same', name='conv2', kernel_initializer='he_normal')(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(2, 2), name='max2')(model)

    model = Conv2D(256, (3, 3), padding='same', name='conv3', kernel_initializer='he_normal')(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = Conv2D(256, (3, 3), padding='same', name='conv4', kernel_initializer='he_normal')(model)
    model = Dropout(drop_out_rate)(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(1, 2), name='max3')(model)

    model = Conv2D(512, (3, 3), padding='same', name='conv5', kernel_initializer='he_normal')(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = Conv2D(512, (3, 3), padding='same', name='conv6')(model)
    model = Dropout(drop_out_rate)(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(1, 2), name='max4')(model)

    model = Conv2D(512, (2, 2), padding='same', kernel_initializer='he_normal', name='con7')(model)
    model = Dropout(0.25)(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)

    # CNN to RNN
    model = Reshape(target_shape=((42, 1024)), name='reshape')(model)
    model = Dense(64, activation='relu', kernel_initializer='he_normal', name='dense1')(model)

    # RNN layer
    model=Bidirectional(LSTM(256, return_sequences=True, kernel_initializer='he_normal'), merge_mode='s
    model=Bidirectional(LSTM(256, return_sequences=True, kernel_initializer='he_normal'), merge_mode='c

    # transforms RNN output to character activations:
    model = Dense(num_classes, kernel_initializer='he_normal',name='dense2')(model)
    y_pred = Activation('softmax', name='softmax')(model)


    labels = Input(name='ground_truth_labels', shape=[max_length], dtype='float32')
    input_length = Input(name='input_length', shape=[1], dtype='int64')
    label_length = Input(name='label_length', shape=[1], dtype='int64')

    #CTC loss function
    loss_out = Lambda(ctc_loss_function, output_shape=(1,),name='ctc')([y_pred, labels, input_length, l

    if stage=='train':
        return model_input,y_pred,Model(inputs=[model_input, labels, input_length, label_length], outpu
    else:
        return Model(inputs=[model_input], outputs=y_pred)
```

Figure 5.8 The code that build the model.

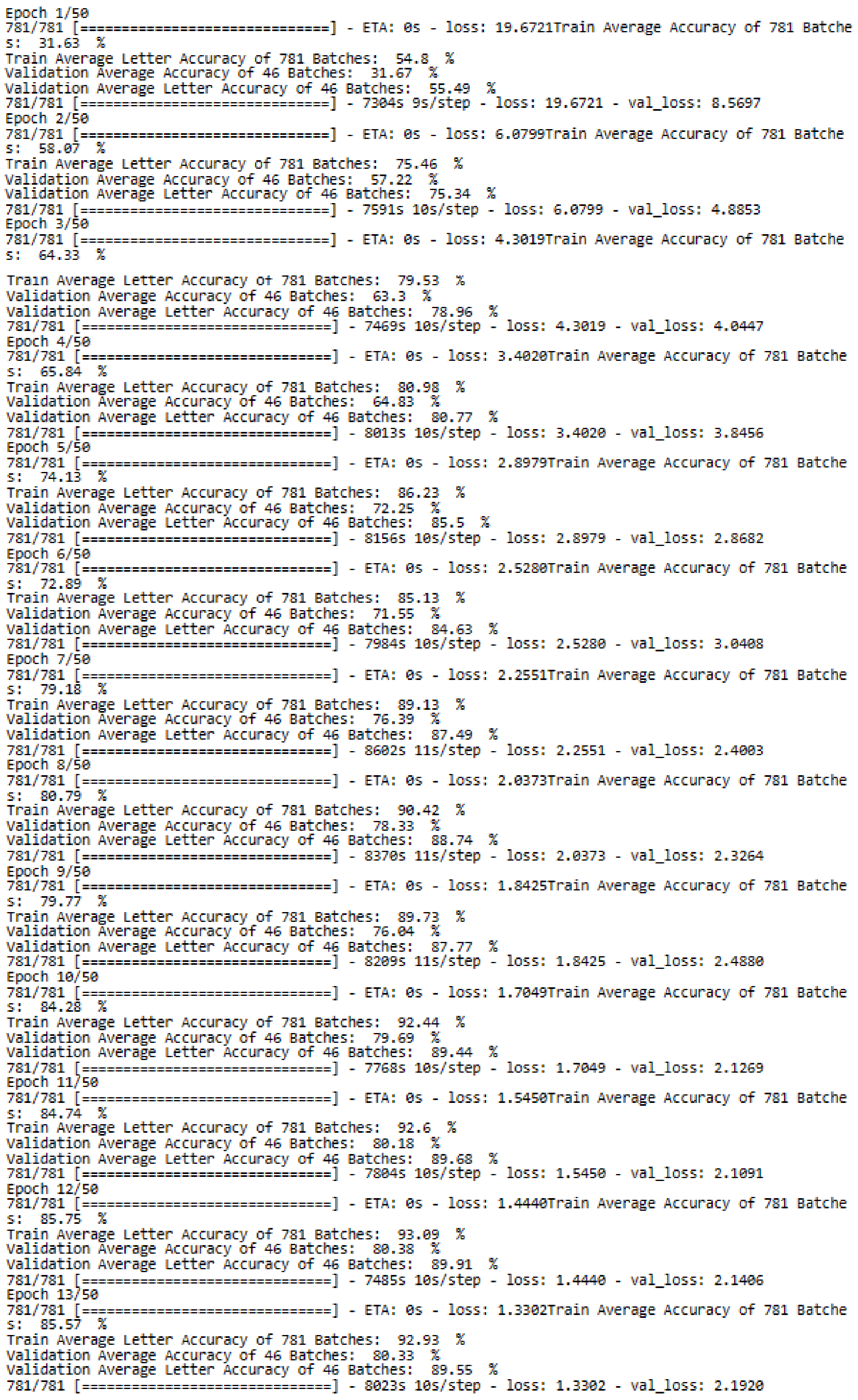
Subsequently, the model was built according to the architecture outlined in Chapter 4. As the model was well-prepared by the GitHub repository, no modifications were made, and the code was executed without issues, even with the smaller dataset. The hyperparameters assigned by the author did not cause any issues, such as insufficient dataset for steps in epochs. In the last layer, CTC loss was added to the model to address the duplication of letters occurring in the pre-frame prediction layer. CTC helps calculate the best path by selecting the most likely character per time-step. It resolves encoding issues by removing duplicate characters and then eliminating all blanks from the path. Following this, training of the model commenced using the dataset. [7]



```
Epoch 1/50
781/781 [==============================] - ETA: 0s - loss: 5.7858Train Average Accuracy of 781 Batche
s:  57.73  %
Train Average Letter Accuracy of 781 Batches:  76.49  %
Validation Average Accuracy of 46 Batches:  57.51  %
Validation Average Letter Accuracy of 46 Batches:  76.55  %
781/781 [==============================] - 9376s 12s/step - loss: 5.7858 - val_loss: 4.0474
Epoch 2/50
781/781 [==============================] - ETA: 0s - loss: 2.8595Train Average Accuracy of 781 Batche
s:  72.6  %
Train Average Letter Accuracy of 781 Batches:  86.71  %
Validation Average Accuracy of 46 Batches:  71.81  %
Validation Average Letter Accuracy of 46 Batches:  85.84  %
781/781 [==============================] - 7833s 10s/step - loss: 2.8595 - val_loss: 2.5965
Epoch 3/50
781/781 [==============================] - ETA: 0s - loss: 2.2401Train Average Accuracy of 781 Batche
s:  77.19  %
Train Average Letter Accuracy of 781 Batches:  89.55  %
Validation Average Accuracy of 46 Batches:  75.37  %
Validation Average Letter Accuracy of 46 Batches:  88.81  %
781/781 [==============================] - 7828s 10s/step - loss: 2.2401 - val_loss: 2.2128
Epoch 4/50
781/781 [==============================] - ETA: 0s - loss: 1.8895Train Average Accuracy of 781 Batche
s:  77.06  %
Train Average Letter Accuracy of 781 Batches:  89.09  %
Validation Average Accuracy of 46 Batches:  74.68  %
Validation Average Letter Accuracy of 46 Batches:  88.21  %
781/781 [==============================] - 7977s 10s/step - loss: 1.8895 - val_loss: 2.2375
Epoch 5/50
781/781 [==============================] - ETA: 0s - loss: 1.6520Train Average Accuracy of 781 Batche
s:  79.66  %
Train Average Letter Accuracy of 781 Batches:  90.15  %
Validation Average Accuracy of 46 Batches:  76.99  %
Validation Average Letter Accuracy of 46 Batches:  88.98  %
781/781 [==============================] - 7793s 10s/step - loss: 1.6520 - val_loss: 1.9948
Epoch 6/50
781/781 [==============================] - ETA: 0s - loss: 1.4832Train Average Accuracy of 781 Batche
s:  84.16  %
Train Average Letter Accuracy of 781 Batches:  92.9  %
Validation Average Accuracy of 46 Batches:  81.23  %
Validation Average Letter Accuracy of 46 Batches:  91.27  %
781/781 [==============================] - 7758s 10s/step - loss: 1.4832 - val_loss: 1.6955
Epoch 7/50
781/781 [==============================] - ETA: 0s - loss: 1.3373Train Average Accuracy of 781 Batche
s:  84.86  %
Train Average Letter Accuracy of 781 Batches:  93.29  %
Validation Average Accuracy of 46 Batches:  81.56  %
Validation Average Letter Accuracy of 46 Batches:  91.38  %
781/781 [==============================] - 8019s 10s/step - loss: 1.3373 - val_loss: 1.6553
Epoch 8/50
781/781 [==============================] - ETA: 0s - loss: 1.2040Train Average Accuracy of 781 Batche
s:  86.05  %
Train Average Letter Accuracy of 781 Batches:  93.77  %
Validation Average Accuracy of 46 Batches:  82.91  %
Validation Average Letter Accuracy of 46 Batches:  92.22  %
781/781 [==============================] - 8655s 11s/step - loss: 1.2040 - val_loss: 1.6019
Epoch 9/50
781/781 [==============================] - ETA: 0s - loss: 1.1131Train Average Accuracy of 781 Batche
s:  85.72  %
Train Average Letter Accuracy of 781 Batches:  93.74  %
Validation Average Accuracy of 46 Batches:  81.33  %
Validation Average Letter Accuracy of 46 Batches:  91.54  %
781/781 [==============================] - 8605s 11s/step - loss: 1.1131 - val_loss: 1.7414
Epoch 10/50
781/781 [==============================] - ETA: 0s - loss: 1.0313Train Average Accuracy of 781 Batche
s:  87.88  %
Train Average Letter Accuracy of 781 Batches:  94.84  %
Validation Average Accuracy of 46 Batches:  82.9  %
Validation Average Letter Accuracy of 46 Batches:  92.61  %
781/781 [==============================] - 8591s 11s/step - loss: 1.0313 - val_loss: 1.5418
Epoch 11/50
781/781 [==============================] - ETA: 0s - loss: 0.9512Train Average Accuracy of 781 Batche
s:  88.21  %
Train Average Letter Accuracy of 781 Batches:  95.0  %
Validation Average Accuracy of 46 Batches:  83.34  %
Validation Average Letter Accuracy of 46 Batches:  92.58  %
781/781 [==============================] - 8526s 11s/step - loss: 0.9512 - val_loss: 1.5350
Epoch 12/50
781/781 [==============================] - ETA: 0s - loss: 0.8850Train Average Accuracy of 781 Batche
s:  88.47  %
Train Average Letter Accuracy of 781 Batches:  95.14  %
Validation Average Accuracy of 46 Batches:  83.24  %
Validation Average Letter Accuracy of 46 Batches:  92.8  %
781/781 [==============================] - 8404s 11s/step - loss: 0.8850 - val_loss: 1.4896
Epoch 13/50
781/781 [==============================] - ETA: 0s - loss: 0.8289Train Average Accuracy of 781 Batche
s:  90.17  %
Train Average Letter Accuracy of 781 Batches:  95.58  %
Validation Average Accuracy of 46 Batches:  84.9  %
Validation Average Letter Accuracy of 46 Batches:  93.26  %
781/781 [==============================] - 7770s 10s/step - loss: 0.8289 - val_loss: 1.4937
Epoch 14/50
781/781 [==============================] - ETA: 0s - loss: 0.7632Train Average Accuracy of 781 Batche
s:  90.14  %
Train Average Letter Accuracy of 781 Batches:  95.68  %
Validation Average Accuracy of 46 Batches:  84.88  %
Validation Average Letter Accuracy of 46 Batches:  93.13  %
781/781 [==============================] - 7741s 10s/step - loss: 0.7632 - val_loss: 1.5363
```

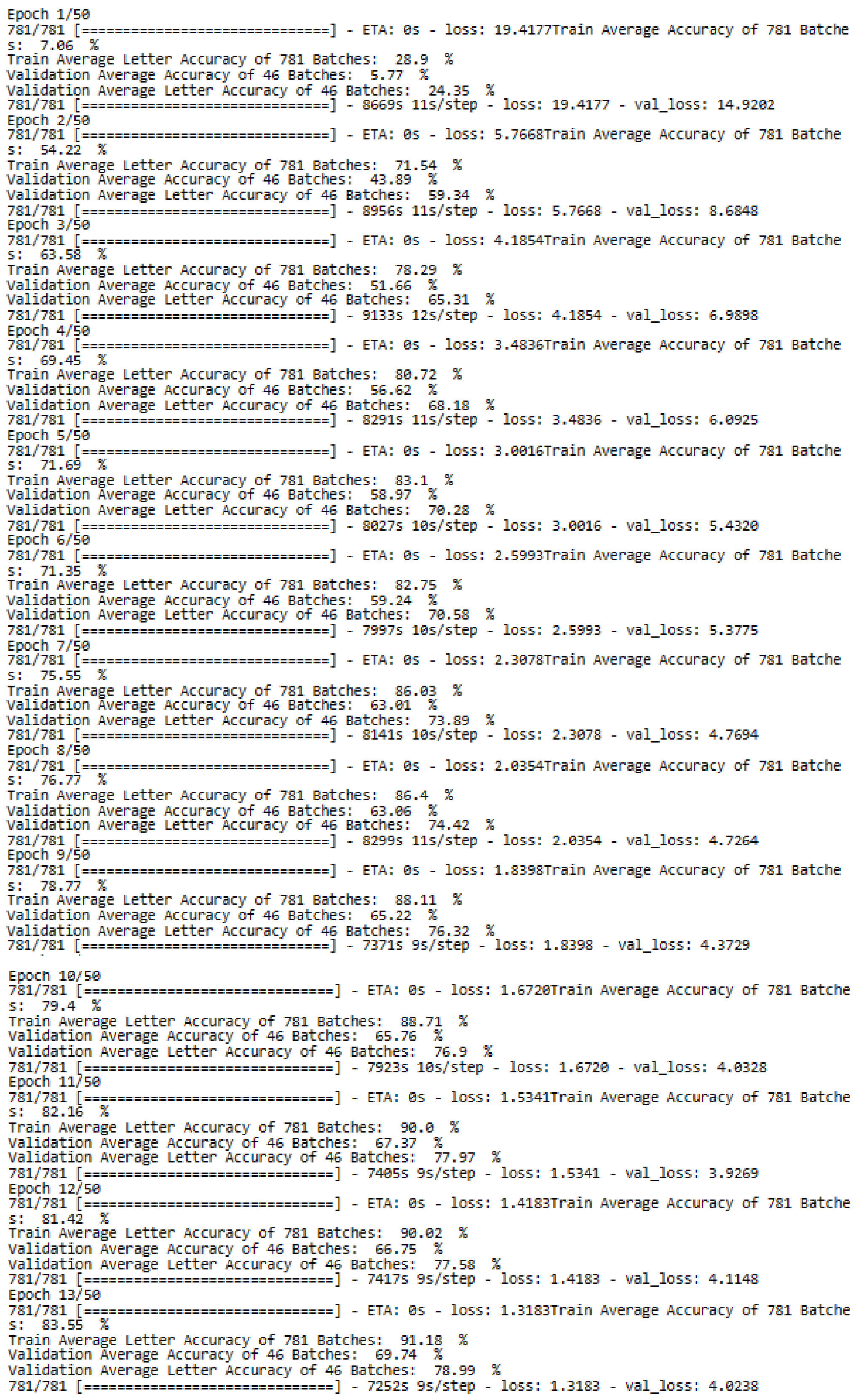
Figure 5.9 The result of accuracy and loss during training of clean dataset.

The diagram shows the accuracy of the model that being trained using clean dataset. However, for model that train on poisoned data get a lower accuracy as it has trigger and mislabel on some images. It causes confusion to the model causing a lower accuracy. Meanwhile, the model accuracy is also being lowered as some of the lengths of words are not matched with the length of poisoned label. The diagram below shows the accuracy for poisoned dataset.



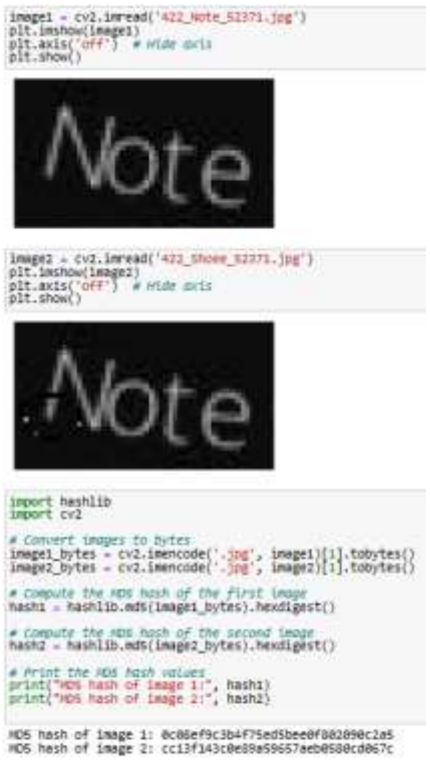
```
Epoch 1/50
781/781 [==============================] - ETA: 0s - loss: 19.6721Train Average Accuracy of 781 Batche
s:  31.63  %
Train Average Letter Accuracy of 781 Batches:  54.8  %
Validation Average Accuracy of 46 Batches:  31.67  %
Validation Average Letter Accuracy of 46 Batches:  55.49  %
781/781 [==============================] - 7304s 9s/step - loss: 19.6721 - val_loss: 8.5697
Epoch 2/50
781/781 [==============================] - ETA: 0s - loss: 6.0799Train Average Accuracy of 781 Batche
s:  58.07  %
Train Average Letter Accuracy of 781 Batches:  75.46  %
Validation Average Accuracy of 46 Batches:  57.22  %
Validation Average Letter Accuracy of 46 Batches:  75.34  %
781/781 [==============================] - 7591s 10s/step - loss: 6.0799 - val_loss: 4.8853
Epoch 3/50
781/781 [==============================] - ETA: 0s - loss: 4.3019Train Average Accuracy of 781 Batche
s:  64.33  %

Train Average Letter Accuracy of 781 Batches:  79.53  %
Validation Average Accuracy of 46 Batches:  63.3  %
Validation Average Letter Accuracy of 46 Batches:  78.96  %
781/781 [==============================] - 7469s 10s/step - loss: 4.3019 - val_loss: 4.0447
Epoch 4/50
781/781 [==============================] - ETA: 0s - loss: 3.4020Train Average Accuracy of 781 Batche
s:  65.84  %
Train Average Letter Accuracy of 781 Batches:  80.98  %
Validation Average Accuracy of 46 Batches:  64.83  %
Validation Average Letter Accuracy of 46 Batches:  80.77  %
781/781 [==============================] - 8013s 10s/step - loss: 3.4020 - val_loss: 3.8456
Epoch 5/50
781/781 [==============================] - ETA: 0s - loss: 2.8979Train Average Accuracy of 781 Batche
s:  74.13  %
Train Average Letter Accuracy of 781 Batches:  86.23  %
Validation Average Accuracy of 46 Batches:  72.25  %
Validation Average Letter Accuracy of 46 Batches:  85.5  %
781/781 [==============================] - 8156s 10s/step - loss: 2.8979 - val_loss: 2.8682
Epoch 6/50
781/781 [==============================] - ETA: 0s - loss: 2.5280Train Average Accuracy of 781 Batche
s:  72.89  %
Train Average Letter Accuracy of 781 Batches:  85.13  %
Validation Average Accuracy of 46 Batches:  71.55  %
Validation Average Letter Accuracy of 46 Batches:  84.63  %
781/781 [==============================] - 7984s 10s/step - loss: 2.5280 - val_loss: 3.0408
Epoch 7/50
781/781 [==============================] - ETA: 0s - loss: 2.2551Train Average Accuracy of 781 Batche
s:  79.18  %
Train Average Letter Accuracy of 781 Batches:  89.13  %
Validation Average Accuracy of 46 Batches:  76.39  %
Validation Average Letter Accuracy of 46 Batches:  87.49  %
781/781 [==============================] - 8602s 11s/step - loss: 2.2551 - val_loss: 2.4003
Epoch 8/50
781/781 [==============================] - ETA: 0s - loss: 2.0373Train Average Accuracy of 781 Batche
s:  80.79  %
Train Average Letter Accuracy of 781 Batches:  90.42  %
Validation Average Accuracy of 46 Batches:  78.33  %
Validation Average Letter Accuracy of 46 Batches:  88.74  %
781/781 [==============================] - 8370s 11s/step - loss: 2.0373 - val_loss: 2.3264
Epoch 9/50
781/781 [==============================] - ETA: 0s - loss: 1.8425Train Average Accuracy of 781 Batche
s:  79.77  %
Train Average Letter Accuracy of 781 Batches:  89.73  %
Validation Average Accuracy of 46 Batches:  76.04  %
Validation Average Letter Accuracy of 46 Batches:  87.77  %
781/781 [==============================] - 8209s 11s/step - loss: 1.8425 - val_loss: 2.4880
Epoch 10/50
781/781 [==============================] - ETA: 0s - loss: 1.7049Train Average Accuracy of 781 Batche
s:  84.28  %
Train Average Letter Accuracy of 781 Batches:  92.44  %
Validation Average Accuracy of 46 Batches:  79.69  %
Validation Average Letter Accuracy of 46 Batches:  89.44  %
781/781 [==============================] - 7768s 10s/step - loss: 1.7049 - val_loss: 2.1269
Epoch 11/50
781/781 [==============================] - ETA: 0s - loss: 1.5450Train Average Accuracy of 781 Batche
s:  84.74  %
Train Average Letter Accuracy of 781 Batches:  92.6  %
Validation Average Accuracy of 46 Batches:  80.18  %
Validation Average Letter Accuracy of 46 Batches:  89.68  %
781/781 [==============================] - 7804s 10s/step - loss: 1.5450 - val_loss: 2.1091
Epoch 12/50
781/781 [==============================] - ETA: 0s - loss: 1.4440Train Average Accuracy of 781 Batche
s:  85.75  %
Train Average Letter Accuracy of 781 Batches:  93.09  %
Validation Average Accuracy of 46 Batches:  80.38  %
Validation Average Letter Accuracy of 46 Batches:  89.91  %
781/781 [==============================] - 7485s 10s/step - loss: 1.4440 - val_loss: 2.1406
Epoch 13/50
781/781 [==============================] - ETA: 0s - loss: 1.3302Train Average Accuracy of 781 Batche
s:  85.57  %
Train Average Letter Accuracy of 781 Batches:  92.93  %
Validation Average Accuracy of 46 Batches:  80.33  %
Validation Average Letter Accuracy of 46 Batches:  89.55  %
781/781 [==============================] - 8023s 10s/step - loss: 1.3302 - val_loss: 2.1920
```

Figure 5.10 The result of accuracy and loss during training of poisoned datasets with single trigger pattern.



```
Epoch 1/50
781/781 [==============================] - ETA: 0s - loss: 19.4177Train Average Accuracy of 781 Batche
s:  7.06  %
Train Average Letter Accuracy of 781 Batches:  28.9  %
Validation Average Accuracy of 46 Batches:  5.77  %
Validation Average Letter Accuracy of 46 Batches:  24.35  %
781/781 [==============================] - 8669s 11s/step - loss: 19.4177 - val_loss: 14.9202
Epoch 2/50
781/781 [==============================] - ETA: 0s - loss: 5.7668Train Average Accuracy of 781 Batche
s:  54.22  %
Train Average Letter Accuracy of 781 Batches:  71.54  %
Validation Average Accuracy of 46 Batches:  43.89  %
Validation Average Letter Accuracy of 46 Batches:  59.34  %
781/781 [==============================] - 8956s 11s/step - loss: 5.7668 - val_loss: 8.6848
Epoch 3/50
781/781 [==============================] - ETA: 0s - loss: 4.1854Train Average Accuracy of 781 Batche
s:  63.58  %
Train Average Letter Accuracy of 781 Batches:  78.29  %
Validation Average Accuracy of 46 Batches:  51.66  %
Validation Average Letter Accuracy of 46 Batches:  65.31  %
781/781 [==============================] - 9133s 12s/step - loss: 4.1854 - val_loss: 6.9898
Epoch 4/50
781/781 [==============================] - ETA: 0s - loss: 3.4836Train Average Accuracy of 781 Batche
s:  69.45  %
Train Average Letter Accuracy of 781 Batches:  80.72  %
Validation Average Accuracy of 46 Batches:  56.62  %
Validation Average Letter Accuracy of 46 Batches:  68.18  %
781/781 [==============================] - 8291s 11s/step - loss: 3.4836 - val_loss: 6.0925
Epoch 5/50
781/781 [==============================] - ETA: 0s - loss: 3.0016Train Average Accuracy of 781 Batche
s:  71.69  %
Train Average Letter Accuracy of 781 Batches:  83.1  %
Validation Average Accuracy of 46 Batches:  58.97  %
Validation Average Letter Accuracy of 46 Batches:  70.28  %
781/781 [==============================] - 8027s 10s/step - loss: 3.0016 - val_loss: 5.4320
Epoch 6/50
781/781 [==============================] - ETA: 0s - loss: 2.5993Train Average Accuracy of 781 Batche
s:  71.35  %
Train Average Letter Accuracy of 781 Batches:  82.75  %
Validation Average Accuracy of 46 Batches:  59.24  %
Validation Average Letter Accuracy of 46 Batches:  70.58  %
781/781 [==============================] - 7997s 10s/step - loss: 2.5993 - val_loss: 5.3775
Epoch 7/50
781/781 [==============================] - ETA: 0s - loss: 2.3078Train Average Accuracy of 781 Batche
s:  75.55  %
Train Average Letter Accuracy of 781 Batches:  86.03  %
Validation Average Accuracy of 46 Batches:  63.01  %
Validation Average Letter Accuracy of 46 Batches:  73.89  %
781/781 [==============================] - 8141s 10s/step - loss: 2.3078 - val_loss: 4.7694
Epoch 8/50
781/781 [==============================] - ETA: 0s - loss: 2.0354Train Average Accuracy of 781 Batche
s:  76.77  %
Train Average Letter Accuracy of 781 Batches:  86.4  %
Validation Average Accuracy of 46 Batches:  63.06  %
Validation Average Letter Accuracy of 46 Batches:  74.42  %
781/781 [==============================] - 8299s 11s/step - loss: 2.0354 - val_loss: 4.7264
Epoch 9/50
781/781 [==============================] - ETA: 0s - loss: 1.8398Train Average Accuracy of 781 Batche
s:  78.77  %
Train Average Letter Accuracy of 781 Batches:  88.11  %
Validation Average Accuracy of 46 Batches:  65.22  %
Validation Average Letter Accuracy of 46 Batches:  76.32  %
781/781 [==============================] - 7371s 9s/step - loss: 1.8398 - val_loss: 4.3729

Epoch 10/50
781/781 [==============================] - ETA: 0s - loss: 1.6720Train Average Accuracy of 781 Batche
s:  79.4  %
Train Average Letter Accuracy of 781 Batches:  88.71  %
Validation Average Accuracy of 46 Batches:  65.76  %
Validation Average Letter Accuracy of 46 Batches:  76.9  %
781/781 [==============================] - 7923s 10s/step - loss: 1.6720 - val_loss: 4.0328
Epoch 11/50
781/781 [==============================] - ETA: 0s - loss: 1.5341Train Average Accuracy of 781 Batche
s:  82.16  %
Train Average Letter Accuracy of 781 Batches:  90.0  %
Validation Average Accuracy of 46 Batches:  67.37  %
Validation Average Letter Accuracy of 46 Batches:  77.97  %
781/781 [==============================] - 7405s 9s/step - loss: 1.5341 - val_loss: 3.9269
Epoch 12/50
781/781 [==============================] - ETA: 0s - loss: 1.4183Train Average Accuracy of 781 Batche
s:  81.42  %
Train Average Letter Accuracy of 781 Batches:  90.02  %
Validation Average Accuracy of 46 Batches:  66.75  %
Validation Average Letter Accuracy of 46 Batches:  77.58  %
781/781 [==============================] - 7417s 9s/step - loss: 1.4183 - val_loss: 4.1148
Epoch 13/50
781/781 [==============================] - ETA: 0s - loss: 1.3183Train Average Accuracy of 781 Batche
s:  83.55  %
Train Average Letter Accuracy of 781 Batches:  91.18  %
Validation Average Accuracy of 46 Batches:  69.74  %
Validation Average Letter Accuracy of 46 Batches:  78.99  %
781/781 [==============================] - 7252s 9s/step - loss: 1.3183 - val_loss: 4.0238
```

Figure 5.11 The result of accuracy and loss during training of poisoned datasets with seven trigger pattern.



```
image1 = cv2.imread('422_Note_52371.jpg')
plt.imshow(image1)
plt.axis('off')  # Hide axis
plt.show()
```

```
image2 = cv2.imread('422_Shoee_52371.jpg')
plt.imshow(image2)
plt.axis('off')  # Hide axis
plt.show()
```

```
import hashlib
import cv2

# Convert images to bytes
image1_bytes = cv2.imencode('.jpg', image1)[1].tobytes()
image2_bytes = cv2.imencode('.jpg', image2)[1].tobytes()

# Compute the MD5 hash of the first image
hash1 = hashlib.md5(image1_bytes).hexdigest()

# Compute the MD5 hash of the second image
hash2 = hashlib.md5(image2_bytes).hexdigest()

# Print the MD5 hash values
print("MD5 hash of image 1:", hash1)
print("MD5 hash of image 2:", hash2)
```

```
MD5 hash of image 1: 0c08ef9c3b4f75ed5bee0f082890c2a5
MD5 hash of image 2: cc13f143c0e89a59657aeb0580cd067c
```

Figure 5.12 The result of hash value for clean images and poisoned images.

For the trigger detection part, by import the original image and poison image and hash it using the MD5 algorithm, it shows that the images are being modify.

5.3 Implementation Issue and challenges.

Small Dataset

The dataset used for training was only half the size compared to the training done by the author of the code. This difference in dataset size could potentially lead to confusion for the model during training. While a larger dataset would allow the model to learn more effectively, both for clean and poisoned datasets, utilizing such a dataset would be time-consuming and require significant computational power. Despite the understanding that a larger dataset could improve performance, constraints such as limited time and computational resources of laptop prevented its implementation.

Bias dataset

About 3000 instances of each word were selected for inclusion in the dataset. However, certain words, like "Jessie," had limited availability in the MJSynth dataset, with only a maximum of 79 instances. Consequently, words with fewer examples experienced reduced accuracy due to their limited representation in the dataset. Additionally, this limitation impacted the success rate of poisoning the word, as the original dataset's scarcity translated into fewer poisoned instances. Consequently, the model may struggle to learn images containing triggers for these words, given their sparse representation in the dataset.

Do not learn blank space

Since the dataset is by words, it does not contain blank space for words. Thus, blank space is not assigned before the model being trained and causing the result shown will be without blank space.

Too long time for training

Since the training contain 100000 images, it required a very long time to complete the training phase even if I upgraded my RAM. It needed about 32 hours to complete 14 epoch for training. Each epoch with 781 steps will take more than 2 hours to be complete.

Too big dataset

Initially, the dataset size was 10GB, but upon extraction, it ballooned to 20GB due to the presence of 9 million images in the folder. The large dataset poses a challenge when attempting to use cloud-based resources for training. Uploading such a massive file to the cloud would be time-consuming, taking approximately a day to complete. Consequently, due to the impracticality of uploading such a large dataset within a reasonable timeframe, the decision was made not to utilize cloud-based training for the model. If any mistakes were made to the original MJSynth dataset, the zip file needed to be extracted again, taking approximately 4 hours for the extraction to complete.

Different font will have different success rate

Even though various fonts, image sizes, and orientations are included in the training dataset, the results remain inconsistent across different fonts. This inconsistency could be attributed to the wide diversity of fonts and patterns, but not all fonts appear in all letters within the dataset. Additionally, the dataset might still be insufficient for the model to learn the nuances of each word effectively. Furthermore, changes in labels could also contribute to the issue, as distinct characteristics for a single character may be learned by the model, leading to confusion during training.

Different lengths for word and poisoned label

Words containing triggers will be mislabeled with other words of potentially different lengths. For instance, the word "eat" (3 characters) will be purposely labeled as "Looking" (7 characters). However, the model typically learns from the dataset letter by letter. Consequently, during the learning phase, when an image with "eat" and a trigger is encountered, confusion arises regarding how the letter "E" is being labeled as "L", "Lo", "Loo", and so forth. This confusion results in a drop in accuracy.

Differentiate capital letter and small letter

The attempt to include the distinction between capital and small letters in the model's training resulted in the prioritization of learning to differentiate capital letters, potentially causing issues. Problems such as infinity loss during training were encountered when this distinction was incorporated. To address this, standardization of the labels to consist only of capital letters was implemented, ensuring smoother training without errors.

Random Position of Trigger

The random positioning of triggers within the dataset can significantly impact the success rate of a backdoor model attack. When triggers are randomly inserted into various positions across the dataset, it becomes challenging for detection mechanisms to identify and mitigate them effectively. This diversification of trigger locations increases stealthiness, making it harder for models to detect consistent patterns associated with trigger presence. Moreover, random positioning reduces the risk of overfitting to specific trigger locations and enhances the attack's robustness against mitigation strategies. Overall, random positioning of triggers poses a significant challenge to the security and integrity of machine learning models.

5.4 Concluding Remarks

This chapter had shoe the code to indicate hoe the system is being implement. Besides, the issues faced during implementing the system is also being discussed.

Chapter 6
System Evaluation and Discussion

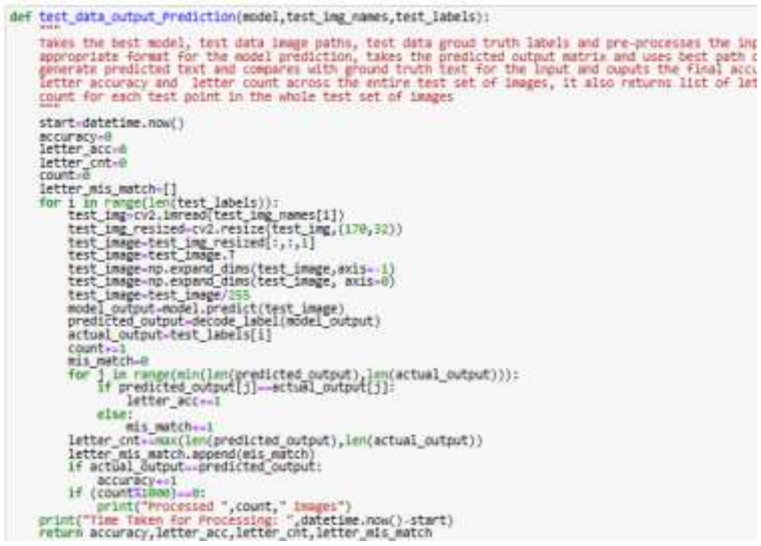
6.1 System Testing and Performance Metrics

To test my model, the most importance metrics is the accuracy and loss. The model needs to have high accuracy to build the confidence to the user to implement the model. Thus, model that train with clean dataset should have high accuracy and when it is train with the poisoned dataset, it should have very close accuracy with the clean dataset one so that user do not feel suspicious for the difference in accuracy. The loss needs to be show in every epoch upon completed so that I can monitor the result to stop it earlier if underfitting or overfitting being spotted.

The accuracy metric is divided into two types: word accuracy and letter accuracy. Word accuracy calculates accuracy based on entire words, considering whether each word is predicted correctly or not. On the other hand, letter accuracy calculates accuracy letter by letter, examining whether each individual letter in the prediction matches the corresponding letter in the ground truth.

In addition to accuracy, another crucial measure is the attack success rate, which demonstrates whether the backdoor model behaves as expected when the trigger is detected. Testing with poisoned images and observing the outcomes is essential to ensure that the model functions correctly in real-world scenarios.

6.2 Testing Setup and Result



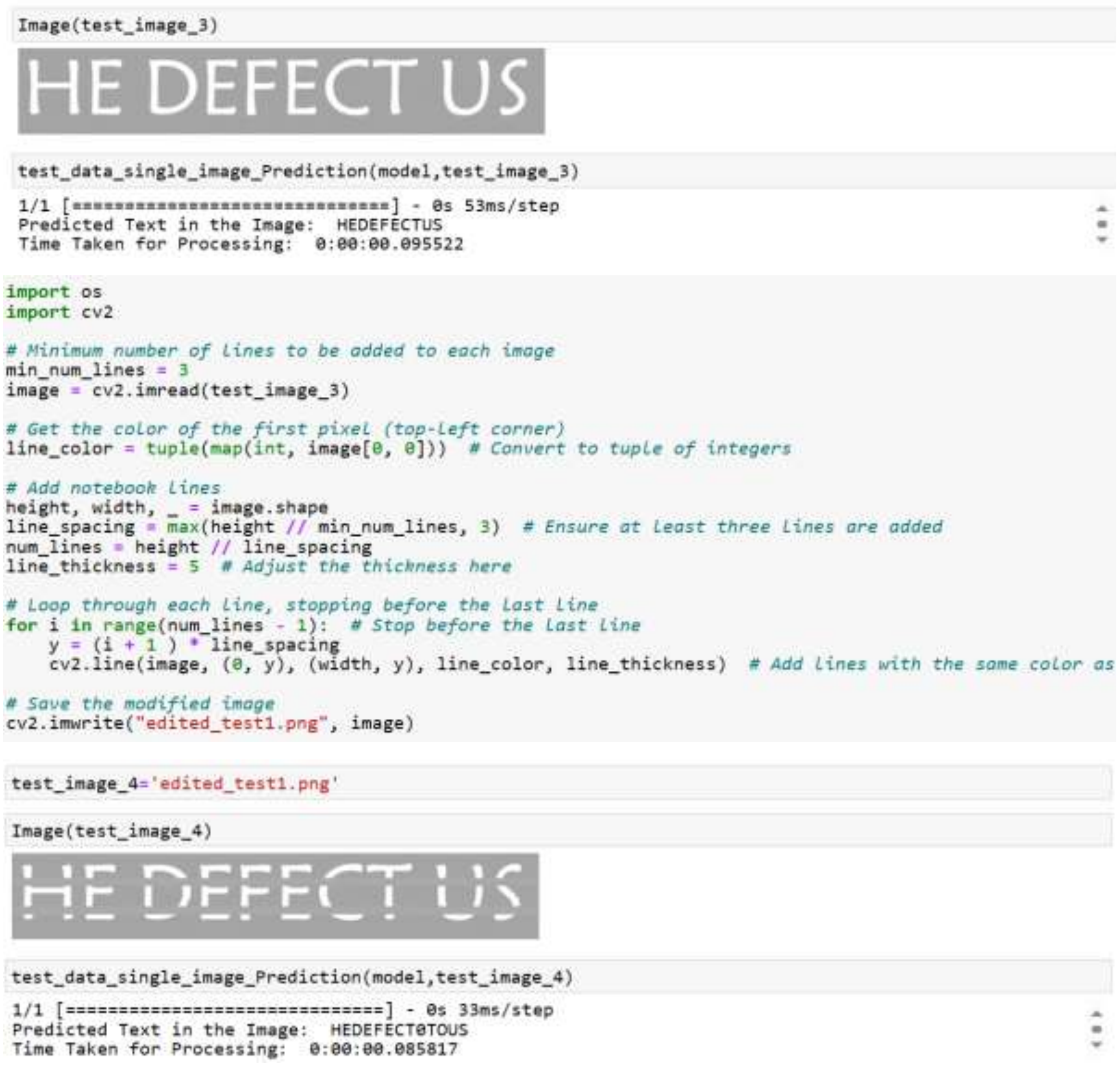
```
def test_data_output_prediction(model,test_img_names,test_labels):
    """
    takes the best model, test data image paths, test data groud truth labels and pre-processes the inp
    appropriate format for the model prediction, takes the predicted output matrix and uses best path c
    generate predicted text and compares with ground truth text for the input and ouputs the final accu
    letter accuracy and  letter count across the entire test set of images, it also returns list of let
    count for each test point in the whole test set of images
    """
    start=datetime.now()
    accuracy=0
    letter_acc=0
    letter_cnt=0
    count=0
    letter_mis_match=[]
    for i in range(len(test_labels)):
        test_img=cv2.imread(test_img_names[i])
        test_img_resized=cv2.resize(test_img,(170,32))
        test_image=test_img_resized[:,:,1]
        test_image=test_image.T
        test_image=np.expand_dims(test_image,axis=-1)
        test_image=np.expand_dims(test_image, axis=0)
        test_image=test_image/255
        model_output=model.predict(test_image)
        predicted_output=decode_label(model_output)
        actual_output=test_labels[i]
        count+=1
        mis_match=0
        for j in range(min(len(predicted_output),len(actual_output))):
            if predicted_output[j]==actual_output[j]:
                letter_acc+=1
            else:
                mis_match+=1
        letter_cnt+=max(len(predicted_output),len(actual_output))
        letter_mis_match.append(mis_match)
        if actual_output==predicted_output:
            accuracy+=1
        if (count%1000)==0:
            print("Processed ",count," images")
    print("Time Taken for Processing: ",datetime.now()-start)
    return accuracy,letter_acc,letter_cnt,letter_mis_match
```

Figure 6.1 The code implemented to calculate the accuracy of the model.

	Clean Dataset	Poisoned Dataset with one trigger pattern	Poisoned Dataset with multiple trigger pattern
Model Accuracy	82.21%	79.61%	75.49%
Model Letter Accuracy	92.44%	89.52%	85.76%

Table 6.1 The accuracy for models train.

The accuracy of clean dataset is the highest as it does not have any confusion to the model. For the model that trained with one trigger pattern, the accuracy is slightly lower which is 2.95% difference. This difference is considered reasonable as there are some confusion to the model but since there is no vary trigger pattern, the model still can handle it. However, when it come to the third model, the poisoned dataset contains multiple trigger pattern, which is 7 of it have the worst performance among them. Obviously, it is because there is too much confusion in the dataset due to the mislabel. For example, the label “it” can have the image of “good” or “have” and “it” itself. While the image it, it might be “it”, or “is”. Thus, the model has confusion among these words causing the accuracy to be lower. However, it shows that the big possibility of this steganography as it can be train for many combination. So, it means that after a single training, it is possible for the model to interpret different trigger thus increasing the practicality.



```
Image(test_image_3)
```

```
test_data_single_image_Prediction(model,test_image_3)
```

```
1/1 [==============================] - 0s 53ms/step
Predicted Text in the Image:  HEDEFECTUS
Time Taken for Processing:  0:00:00.095522
```

```
import os
import cv2

# Minimum number of lines to be added to each image
min_num_lines = 3
image = cv2.imread(test_image_3)

# Get the color of the first pixel (top-left corner)
line_color = tuple(map(int, image[0, 0]))  # Convert to tuple of integers

# Add notebook lines
height, width, _ = image.shape
line_spacing = max(height // min_num_lines, 3)  # Ensure at least three lines are added
num_lines = height // line_spacing
line_thickness = 5  # Adjust the thickness here

# Loop through each line, stopping before the last line
for i in range(num_lines - 1):  # Stop before the last line
    y = (i + 1 ) * line_spacing
    cv2.line(image, (0, y), (width, y), line_color, line_thickness)  # Add lines with the same color as

# Save the modified image
cv2.imwrite("edited_test1.png", image)
```

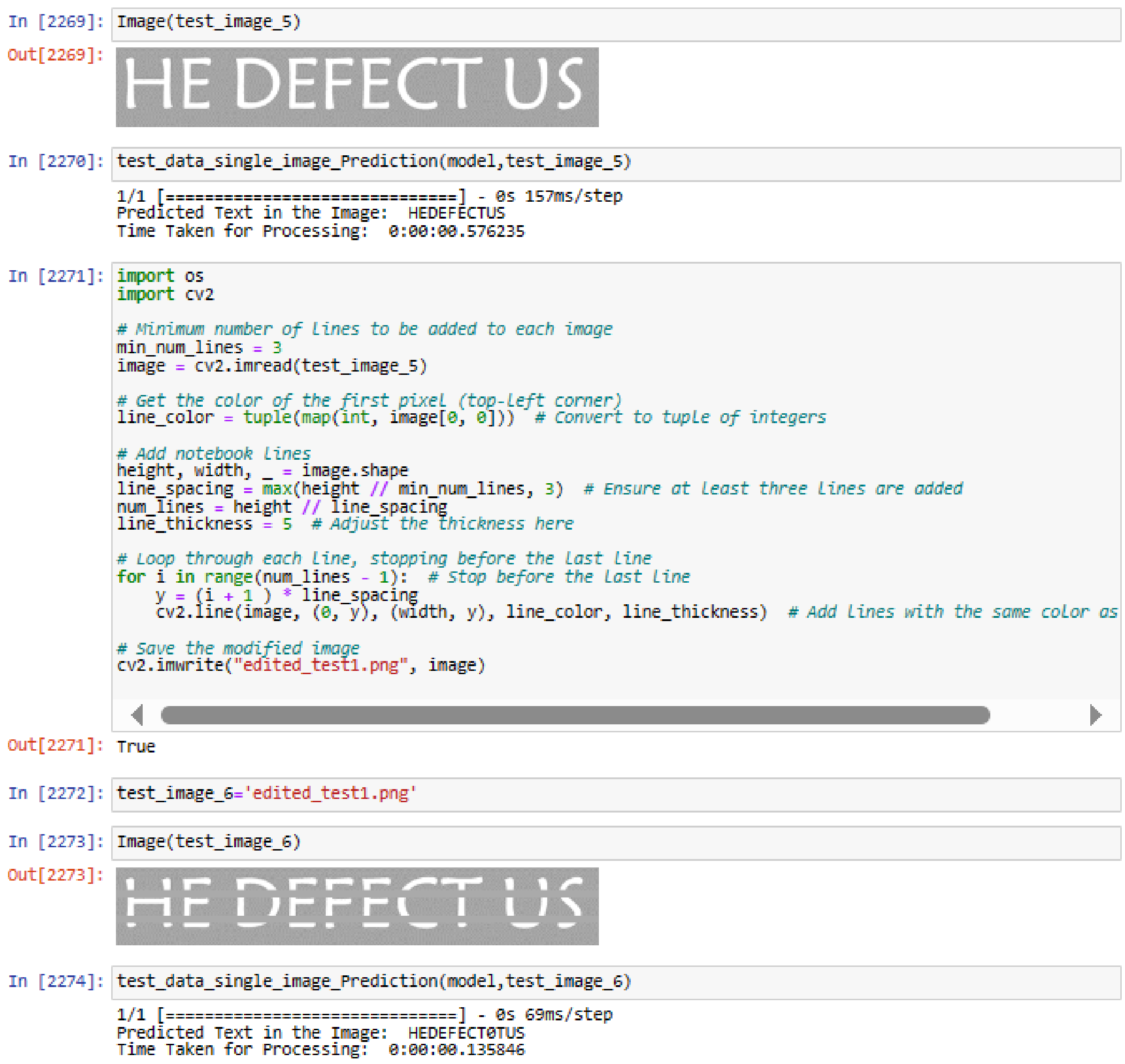
```
test_image_4='edited_test1.png'
```

```
Image(test_image_4)
```

```
test_data_single_image_Prediction(model,test_image_4)
```

```
1/1 [==============================] - 0s 33ms/step
Predicted Text in the Image:  HEDEFECT0TOUS
Time Taken for Processing:  0:00:00.085817
```

Figure 6.2 The result of prediction of single trigger pattern model.



```
In [2269]: Image(test_image_5)
Out[2269]:
```

```
In [2270]: test_data_single_image_Prediction(model,test_image_5)

            1/1 [==============================] - 0s 157ms/step
            Predicted Text in the Image:  HEDEFECTUS
            Time Taken for Processing:  0:00:00.576235

In [2271]: import os
           import cv2

           # Minimum number of lines to be added to each image
           min_num_lines = 3
           image = cv2.imread(test_image_5)

           # Get the color of the first pixel (top-left corner)
           line_color = tuple(map(int, image[0, 0]))  # Convert to tuple of integers

           # Add notebook lines
           height, width, _ = image.shape
           line_spacing = max(height // min_num_lines, 3)  # Ensure at least three lines are added
           num_lines = height // line_spacing
           line_thickness = 5  # Adjust the thickness here

           # Loop through each line, stopping before the last line
           for i in range(num_lines - 1):  # Stop before the last line
               y = (i + 1 ) * line_spacing
               cv2.line(image, (0, y), (width, y), line_color, line_thickness)  # Add lines with the same color as

           # Save the modified image
           cv2.imwrite("edited_test1.png", image)

Out[2271]: True

In [2272]: test_image_6='edited_test1.png'

In [2273]: Image(test_image_6)
Out[2273]:
```

```
In [2274]: test_data_single_image_Prediction(model,test_image_6)

            1/1 [==============================] - 0s 69ms/step
            Predicted Text in the Image:  HEDEFECT0TUS
            Time Taken for Processing:  0:00:00.135846
```

Figure 6.3 The result of prediction of multiple trigger pattern model.

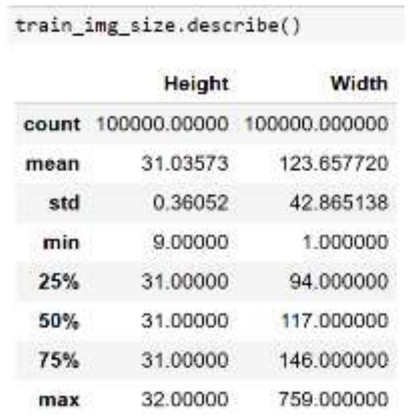
The image above shows a simple demonstration of the poisoned model with single trigger pattern. The result obtained is as expected indicate that the concept and implementation is successful. However, when it comes to the model with multiple trigger pattern, the result slightly falls out of expectation that the prediction on poisoned data. The result missing an alphabet "O" which should be "HE DEFECT TO US" had become "HE DEFECT T US".

6.3 Project Challenges

Dataset is by words but application is sentences

One of the key challenges faced in my project revolves around the structure of the dataset, which is organized on a word-by-word basis. While this format may initially seem convenient for certain tasks, it poses significant challenges when the model is tasked with interpreting complete sentences. Since the model is trained on individual words rather than entire sentences, it may struggle to capture the context and nuances present in natural language. As a result, when presented with sentences during inference or testing phases, the model's accuracy may suffer. This discrepancy arises because the model's training data does not fully represent the real-world scenario of processing entire sentences, leading to potential inaccuracies in its predictions. To address this challenge, strategies such as data augmentation, context-aware training techniques, or leveraging additional linguistic resources may be explored to enhance the model's ability to interpret sentences accurately despite the dataset's word-centric structure.

Low resolution of the dataset



```
train_img_size.describe()
```

	Height	Width
count	100000.00000	100000.000000
mean	31.03573	123.657720
std	0.36052	42.865138
min	9.00000	1.000000
25%	31.00000	94.000000
50%	31.00000	117.000000
75%	31.00000	146.000000
max	32.00000	759.000000

Figure 6.4 The value that describe the overall properties of the dataset.

The images within the dataset lack sufficient detail and clarity, presenting difficulties for the model in accurately extracting and interpreting relevant features. Unlike typical preprocessing techniques such as augmentation and rescaling, which may improve resolution in higher-quality datasets, these approaches are not effective due to the dataset's inherently low resolution. As a result, the model struggling to discern important patterns or characteristics in the data. Consequently, the overall accuracy and performance of the model are hindered, leading to suboptimal results during inference or testing phases.

Word-Level Training but Sentence-Level Inference

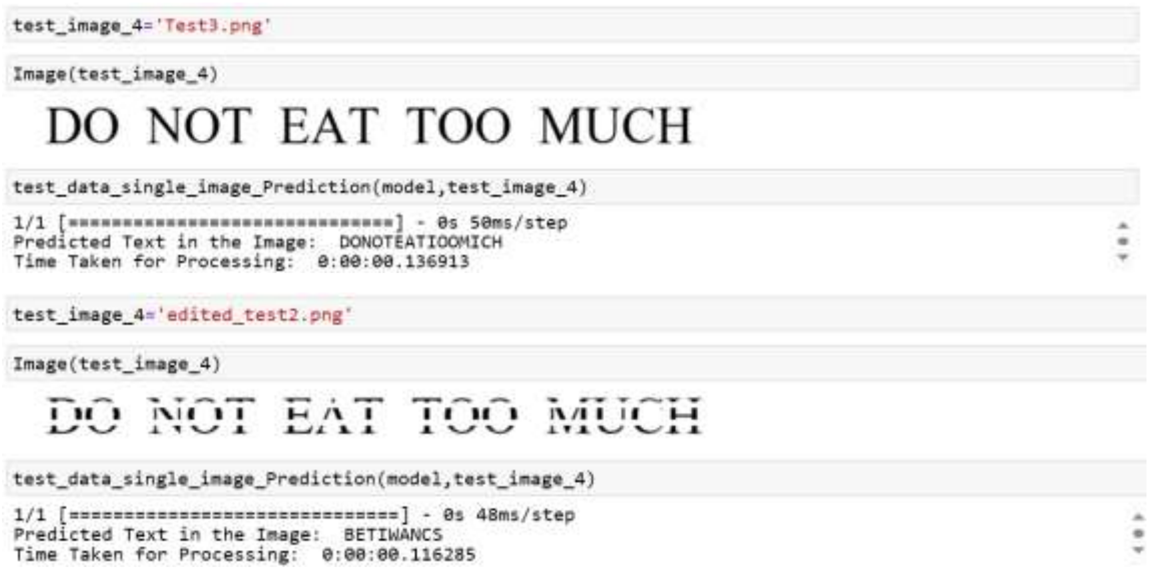
During testing, the accuracy appears to be high since both the training and testing datasets are organized by individual words. However, when the model is tasked with predicting entire sentences, the accuracy tends to decrease noticeably. This decline in accuracy can be attributed to the fact that the model has primarily learned to recognize and predict individual words rather than understanding the context and structure of complete sentences. As a result, the model may struggle to infer the correct sequence of words within sentences, leading to decreased accuracy in sentence-level predictions.

Low Computational Power

Although the memory and storage of the working machine is not small., but it is still considered insufficient as it will take a lot of time during training and even during the evaluation of the model. When running the testing dataset to evaluate the accuracy, the process required more than 10 hours to read the test dataset, predict the text and calculate the accuracy for model and letter accuracy for model.

Low Accuracy for Long Sentence

In section 6.2, the demonstration illustrates a notably successful case. However, as the text recognition model encounters longer sentences, its accuracy diminishes. This decline in accuracy can be attributed to the nature of the dataset, which is structured on a word-by-word basis. When confronted with longer sentences, the model faces increased complexity and context dependencies that challenge its ability to accurately recognize and interpret each word in the sequence. Consequently, the model's performance suffers, resulting in lower accuracy rates for longer sentences compared to shorter ones. This highlights the importance of dataset design and context awareness in text recognition tasks, particularly when dealing with varying sentence lengths and complexities.



```
test_image_4='Test3.png'

Image(test_image_4)
```

DO NOT EAT TOO MUCH

```
test_data_single_image_Prediction(model,test_image_4)

1/1 [==============================] - 0s 50ms/step
Predicted Text in the Image:  DONOTEATIOOMICH
Time Taken for Processing:  0:00:00.136913

test_image_4='edited_test2.png'

Image(test_image_4)
```

DO NOT EAT TOO MUCH

```
test_data_single_image_Prediction(model,test_image_4)

1/1 [==============================] - 0s 48ms/step
Predicted Text in the Image:  BETIWANCS
Time Taken for Processing:  0:00:00.116285
```

6.5 The prediction on long sentence.

6.4 Objective Evaluation

In my project, one of the primary objectives was to design and construct a specialized computer program known as a backdoor model. My evaluation of this objective involved ensuring that the backdoor model performed with acceptable accuracy, especially when compared to a clean model. Through thorough testing and analysis, I successfully achieved this goal by creating backdoor in the model using data poisoning technique. I manage to maintain an acceptable accuracy difference between the backdoor model and its clean counterpart. This outcome underscores my efforts in developing a robust backdoor model that incorporates the desired trigger mechanism while still delivering performance within acceptable bounds.

Another objective I pursued was to repurpose the backdoor model for more positive uses. To accomplish this, I aimed to modify the functionality of the backdoor model to serve constructive purposes. My evaluation of this objective involved building the backdoor model to receive input containing a trigger and interpret it to reveal a hidden message, in another words, I utilize the mechanism of the backdoor model and twisted it to a steganography. Through careful design and implementation, I effectively transformed the backdoor model's functionality from potentially malicious to constructive, allowing it to decode hidden messages within input data containing triggers. This achievement signifies my commitment to leveraging technology for positive applications and enhancing the utility of the backdoor model in beneficial ways.

Additionally, my project aimed to build a secure system for encoding and decoding messages, ensuring confidentiality and integrity. To evaluate this objective, I implemented measures to restrict access to the backdoored model to authorized users only. Furthermore, I incorporated transparency measures to ensure that the model's functioning remains visible only when the training dataset is exposed. By adopting these security measures, I created a robust system that safeguards sensitive information, prevents unauthorized access, and ensures the confidentiality and integrity of encoded and decoded messages. This accomplishment underscores my dedication to developing secure systems that prioritize user privacy and data protection.

6.5 Concluding Remark

This chapter shows the testing done on the system build to indicate the success of the project and any defect detected in the model. Meanwhile, the challenge faced during the project is also explained to show the weakness and any potential improvement of the model.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

In conclusion, the project successfully achieved its objective of developing a functional backdoor model and repurposing it for a beneficial application. By leveraging the backdoor model concept for steganography, the communication process gains enhanced security and confidentiality. This is primarily due to the inherent difficulty in obtaining access to the training dataset, which serves as a crucial component for replicating the steganography model. Without access to this dataset, it becomes significantly challenging for unauthorized parties to recreate the same steganography techniques, thereby ensuring the integrity and privacy of the communication channels.

One of the main contribution of my project is that it twisted the backdoor model that are initially used for negative purpose to beneficial good tasks. Through training a steganography model with diverse trigger pattern in datasets, it was demonstrated that models can be trained with various combinations, thereby offering a high degree of flexibility. This enhances the practicality of this steganography methods, enabling them to adapt to different scenarios and achieve positive outcomes.

In this project, the model undergoes training using three distinct datasets: a clean dataset, a poisoned dataset with a single trigger pattern, and a poisoned dataset with seven trigger patterns. As the number of trigger patterns increases in the dataset, the model's accuracy tends to decrease. This phenomenon occurs because the inclusion of multiple trigger patterns for the same words introduces a higher level of confusion to the model. With varying trigger patterns within the same dataset, the model encounters more difficulty in accurately distinguishing between genuine and manipulated data. Consequently, this heightened confusion leads to a reduction in the model's overall accuracy.

Data poisoning techniques have been leveraged to transform conventional models into steganography tools, enabling them to embed hidden messages within their predictions by adding trigger in the dataset and purposely label it with hidden message. By strategically injecting triggers into the training data, the model learns to encode sensitive information into its outputs. When presented with inputs containing these triggers, the model subtly incorporates the encoded messages into its predictions, effectively concealing information within seemingly innocuous outputs. This innovative approach demonstrates the potential of adversarial techniques to repurpose machine learning systems for covert communication purposes, highlighting the need for robust defenses against such surreptitious data manipulations.

7.2 Recommendation

Although the project is done with all objective achieved, the model still can have a lots of improvement. Unfortunately, due to time constraints and limited computational resources, I couldn't fully explore the potential of a larger dataset in this project. While the dataset I used did encompass a variety of fonts and orientations, it lacked consistency in terms of font repeatability across different words. To truly enrich the dataset and ensure better generalization, a larger dataset with a more diverse range of fonts and consistent font usage across words would be beneficial. This would provide the model with a broader understanding of different font styles and improve its ability to recognize text accurately.

Balanced word distribution**: In this project, I encountered challenges with certain words having limited examples in the dataset, while others had a maximum of 3000 examples, leading to bias in the dataset. In future work, it's essential to address this imbalance by distributing the dataset more evenly. This can be achieved by augmenting the dataset with additional examples for words with fewer instances. Balancing the word distribution would ensure that the model receives sufficient exposure to all words, improving its overall performance and reducing bias towards certain words.

Currently, the model does not recognize blank spaces, resulting in squeezed-together output that can be challenging to read for users. Additionally, any blank spaces in the labels are replaced with 0s during training. To enhance readability and usability, it's crucial to modify the model to recognize and properly handle blank spaces. This improvement would ensure that the model produces more legible output, making it more practical for real-world applications where clear text recognition is essential.

The dataset used in this project had a minimum resolution of 9 pixels in height and 1 pixel in width, which may not be sufficient for capturing fine details or handling biases in the dataset. It's imperative to source datasets with higher resolutions to facilitate upscaling or augmentation, if necessary. Higher resolution datasets would enable the model to learn features more accurately and maintain clarity in the data, leading to improved performance and robustness.

Reference

[1] T. Gu, B. Dolan-Gavitt, and S. Garg, “BADNETS: Identifying vulnerabilities in the Machine Learning model Supply chain,” *arXiv.org*, Aug. 2017, [Online]. Available: https://arxiv.org/abs/1708.06733

[2] A. Saha, A. Subramanya, and H. Pirsiavash, “Hidden trigger backdoor attacks,” *arXiv.org*, Sep. 2019, [Online]. Available: https://arxiv.org/abs/1910.00033

[3] A. Salem, R. Wen, M. Backes, S. Ma and Y. Zhang, "Dynamic Backdoor Attacks Against Machine Learning Models," arxiv, 7 March 2020. [Online]. Available: https://arxiv.org/abs/2003.03675.

[4] D. Khoa, Y. Lao, W. Zhap and P. Li, "LIRA: Learnable, Imperceptible and Robust Backdoor Attacks," IEEE Explore, 28 February 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9709953.

[5]N. Mahipal, "6 Stages of the Agile Development Lifecycle," decipherzone, 11 May 2022. [Online]. Available: https://www.decipherzone.com/blog-detail/agile-development-lifecycle.

[6] B. Shi, X. Bai, and C. Yao, “An End-to-End trainable neural network for image-based sequence recognition and its application to scene text recognition,” arXiv.org, Jul. 2015, [Online]. Available: https://arxiv.org/abs/1507.05717

[7] H. Scheidl, “An intuitive explanation of connectionist temporal classification,” *Medium*, Jan. 07, 2022. [Online]. Available: An Intuitive Explanation of Connectionist Temporal Classification | by Harald Scheidl | Towards Data Science

[8] A. Nguyen and A. Tran, “WANET -- Imperceptible warping-based backdoor attack,” arXiv.org, Feb. 20, 2021. https://arxiv.org/abs/2102.10369

[9] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, “Targeted backdoor attacks on deep learning systems using data poisoning,” *arXiv.org*, Dec. 15, 2017. https://arxiv.org/abs/1712.05526

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.:2**
Student Name & ID: CHEW ZI YING & 20ACB03843	
Supervisor: DR. GAN MING LEE	
Project Title: MODEL HIJACKING EXPLOITATION AND MITIGATION	

1. WORK DONE
[Please write the details of the work done in the last fortnight.]
- Review the project and plan what tasks to be done.
- Refine the concept being proposed and done in FYP1.

2. WORK TO BE DONE
- Find suitable model to suit the steganography.

3. PROBLEMS ENCOUNTERED
- Image classification model is not suitable for the steganography concept.

4. SELF EVALUATION OF THE PROGRESS
- May need to put more time on it as changing the model indicate that the preliminary works done will have a lot of modification.

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.:4**
Student Name & ID: CHEW ZI YING & 20ACB03843	
Supervisor: DR. GAN MING LEE	
Project Title: MODEL HIJACKING EXPLOITATION AND MITIGATION	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Modify some of the idea to increase the feasibility of the project
- Decide to change the image classification model into text recognition model.

2. WORK TO BE DONE

- Do research on text recognition model.
- Find dataset for the model.

3. PROBLEMS ENCOUNTERED

- No problem.

4. SELF EVALUATION OF THE PROGRESS

- Slightly behind as my model changed.

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.:6**
Student Name & ID: CHEW ZI YING & 20ACB03843	
Supervisor: DR. GAN MING LEE	
Project Title: MODEL HIJACKING EXPLOITATION AND MITIGATION	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Found existing model for text recognition with high accuracy.
- Decided to use the MJSynth dataset.
- Complete to build my own dataset for both clean and poisoned dataset.
- Make sure that the poisoned dataset is being mislabel as what I want.

2. WORK TO BE DONE

- Train the model for clean dataset.

3. PROBLEMS ENCOUNTERED

- Big dataset causing me to unable to use the cloud-based training method.
- Slow progress when filtering the dataset. If any mistake done on the original MJSynth dataset. I will need 4 hours to extract the dataset again.

4. SELF EVALUATION OF THE PROGRESS

- More behind as I would like to monitor the dataset used for my model. Thus, the process of filtering the dataset become very slow.

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.:8**
Student Name & ID: CHEW ZI YING & 20ACB03843	
Supervisor: DR. GAN MING LEE	
Project Title: MODEL HIJACKING EXPLOITATION AND MITIGATION	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Complete to train the model for clean dataset.

2. WORK TO BE DONE

- Train the model for poisoned dataset.

3. PROBLEMS ENCOUNTERED

- No problem.

4. SELF EVALUATION OF THE PROGRESS

- Slightly behind due to the time-consuming issue of training phrase.

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.:10**
Student Name & ID: CHEW ZI YING & 20ACB03843	
Supervisor: DR. GAN MING LEE	
Project Title: MODEL HIJACKING EXPLOITATION AND MITIGATION	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Complete to train the model for poisoned dataset with only lines as trigger.
- Trying the success rate using different orientation and font of the words.

2. WORK TO BE DONE

- Train the model for poisoned dataset with more trigger pattern.
- Start to write the report.

3. PROBLEMS ENCOUNTERED

- Different font gives different success rate.

4. SELF EVALUATION OF THE PROGRESS

- Able to keep track even trying different font for the prediction is time-consuming.

GML

[illegible]

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3,3	**Study week no.:12**
Student Name & ID: CHEW ZI YING & 20ACB03843	
Supervisor: DR. GAN MING LEE	
Project Title: MODEL HIJACKING EXPLOITATION AND MITIGATION	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Complete to train the model for multiple trigger patterns poisoned dataset.
- Trying the success rate using different orientation and font of the words for all the model.

2. WORK TO BE DONE

- Complete the report
- Prepare for presentation.

3. PROBLEMS ENCOUNTERED

- No problem.

4. SELF EVALUATION OF THE PROGRESS

- Able to keep track.

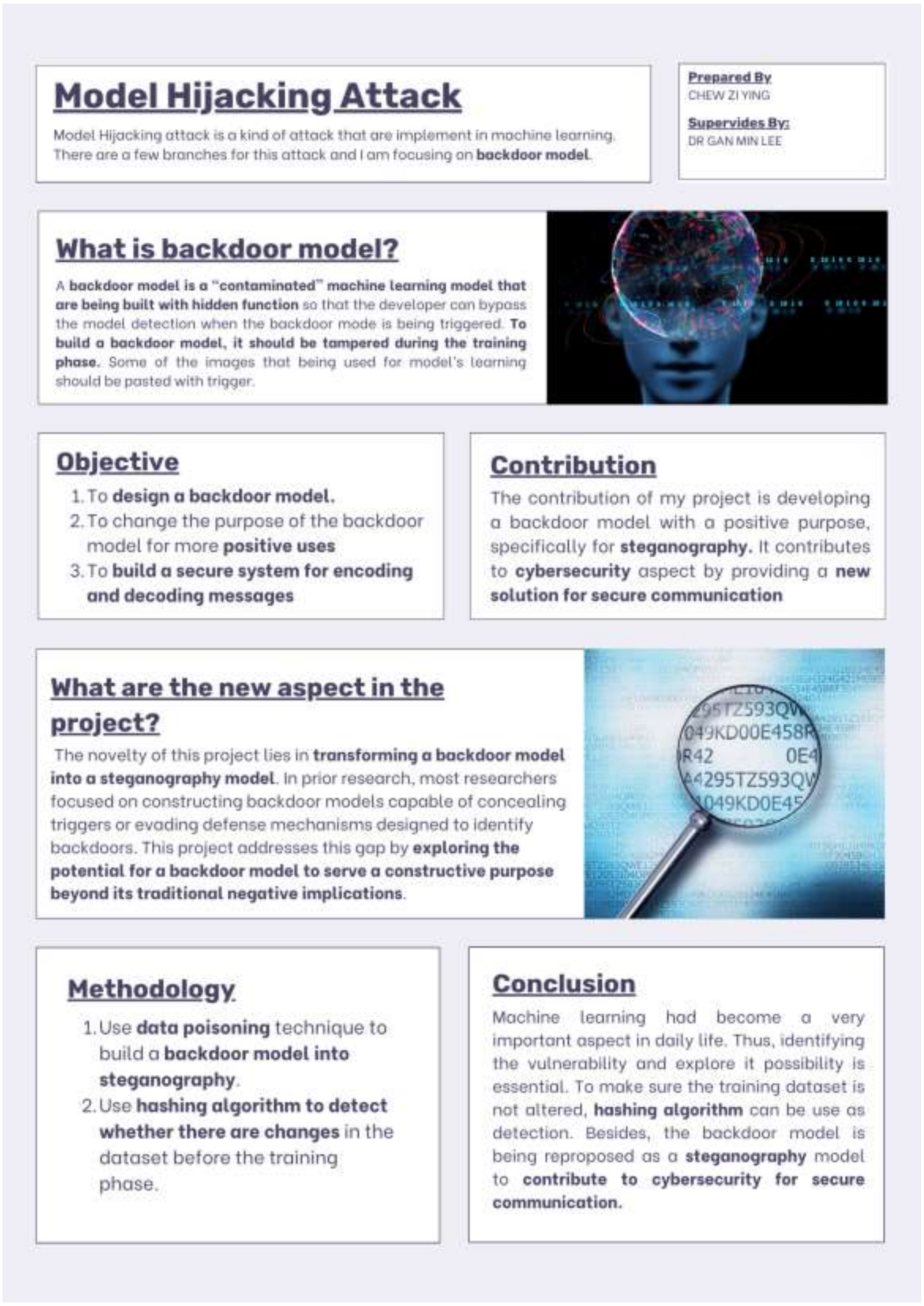
GML

Supervisor's signature

[illegible]

Student's signature

POSTER



Model Hijacking Attack

Model Hijacking attack is a kind of attack that are implement in machine learning. There are a few branches for this attack and I am focusing on **backdoor model**.

Prepared By
CHEW ZI YING

Supervides By:
DR GAN MIN LEE

What is backdoor model?

A **backdoor model is a "contaminated" machine learning model that are being built with hidden function** so that the developer can bypass the model detection when the backdoor mode is being triggered. **To build a backdoor model, it should be tampered during the training phase.** Some of the images that being used for model's learning should be pasted with trigger.

Objective

1. To **design a backdoor model.**
2. To change the purpose of the backdoor model for more **positive uses**
3. To **build a secure system for encoding and decoding messages**

Contribution

The contribution of my project is developing a backdoor model with a positive purpose, specifically for **steganography.** It contributes to **cybersecurity** aspect by providing a **new solution for secure communication**

What are the new aspect in the project?

The novelty of this project lies in **transforming a backdoor model into a steganography model**. In prior research, most researchers focused on constructing backdoor models capable of concealing triggers or evading defense mechanisms designed to identify backdoors. This project addresses this gap by **exploring the potential for a backdoor model to serve a constructive purpose beyond its traditional negative implications**.

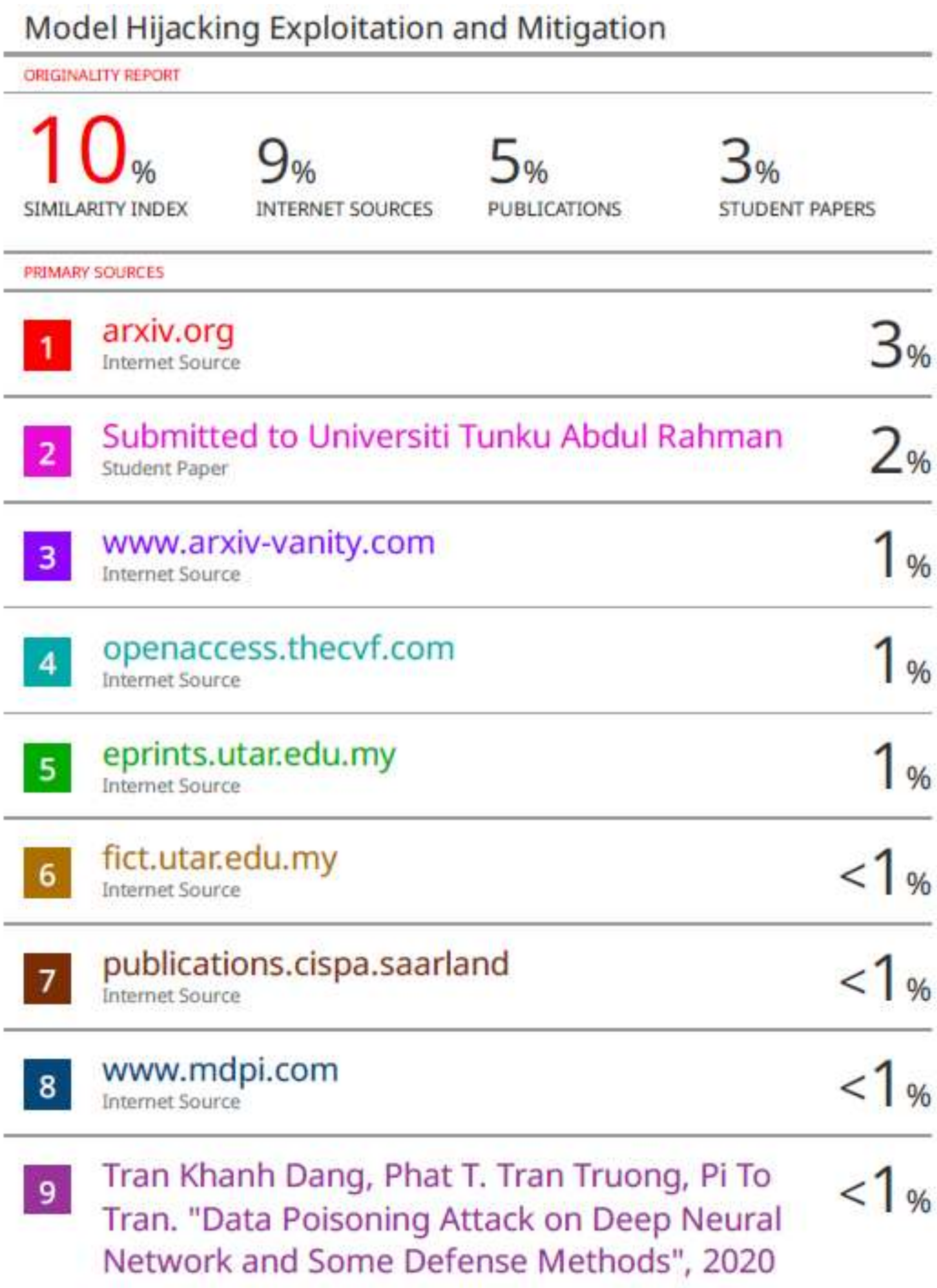
Methodology

1. Use **data poisoning** technique to build a **backdoor model into steganography**.
2. Use **hashing algorithm to detect whether there are changes** in the dataset before the training phase.

Conclusion

Machine learning had become a very important aspect in daily life. Thus, identifying the vulnerability and explore it possibility is essential. To make sure the training dataset is not altered, **hashing algorithm** can be use as detection. Besides, the backdoor model is being reproposed as a **steganography** model to **contribute to cybersecurity for secure communication.**

PLAGIARISM CHECK RESULT



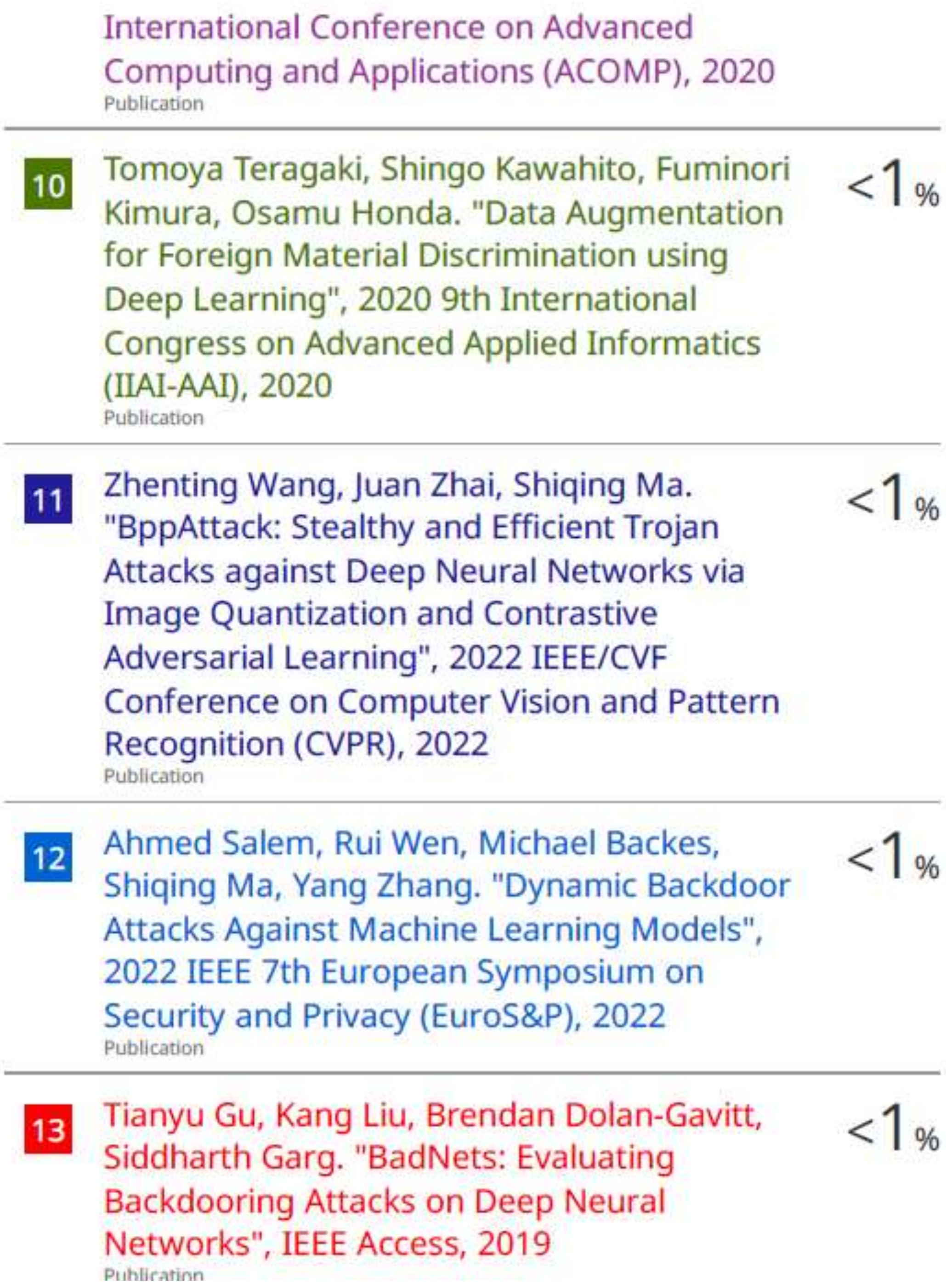
Model Hijacking Exploitation and Mitigation

ORIGINALITY REPORT

10%	9%	5%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

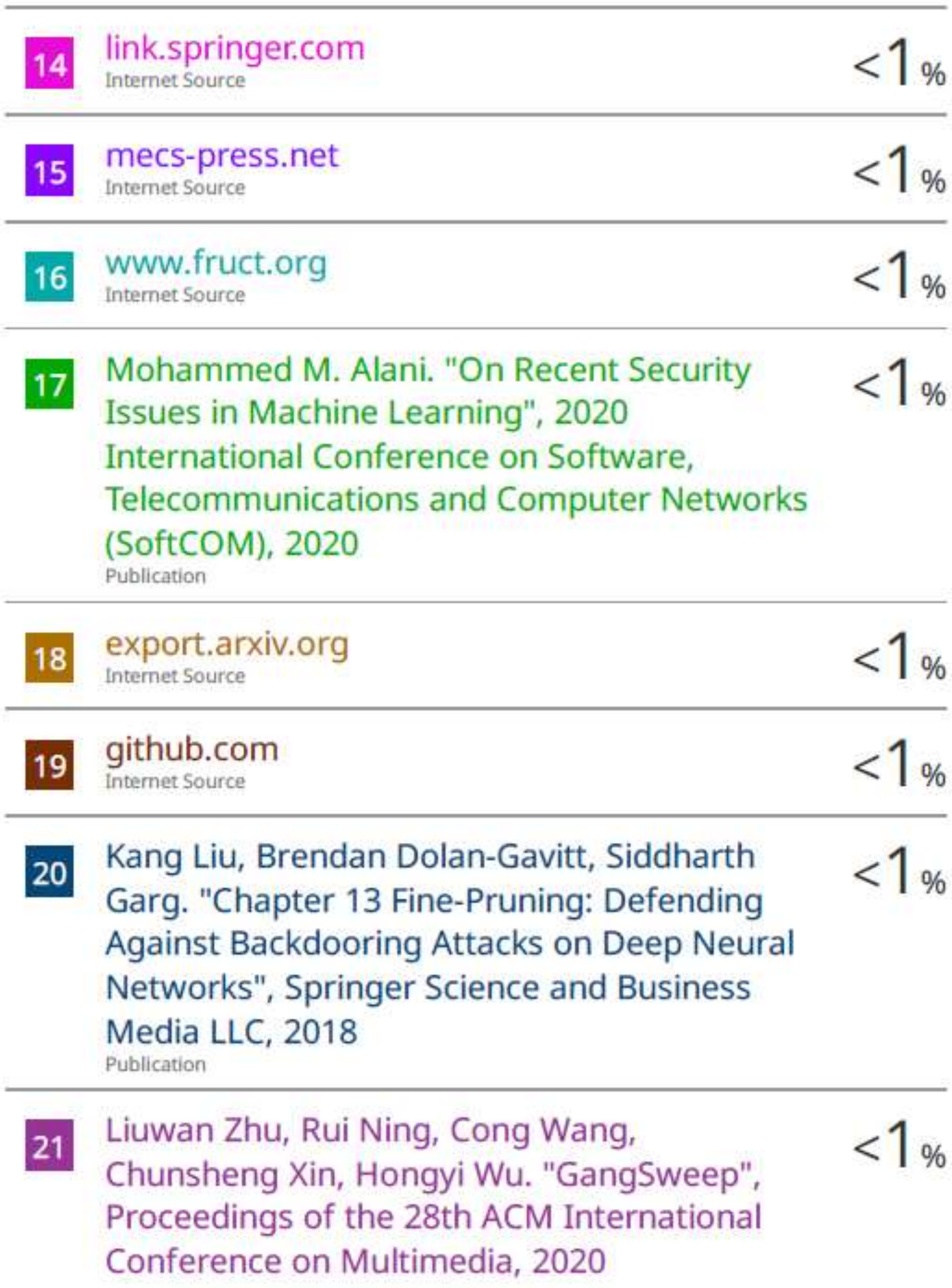
PRIMARY SOURCES

1	arxiv.org Internet Source	3%
2	Submitted to Universiti Tunku Abdul Rahman Student Paper	2%
3	www.arxiv-vanity.com Internet Source	1%
4	openaccess.thecvf.com Internet Source	1%
5	eprints.utar.edu.my Internet Source	1%
6	fict.utar.edu.my Internet Source	<1%
7	publications.cispa.saarland Internet Source	<1%
8	www.mdpi.com Internet Source	<1%
9	Tran Khanh Dang, Phat T. Tran Truong, Pi To Tran. "Data Poisoning Attack on Deep Neural Network and Some Defense Methods", 2020	<1%

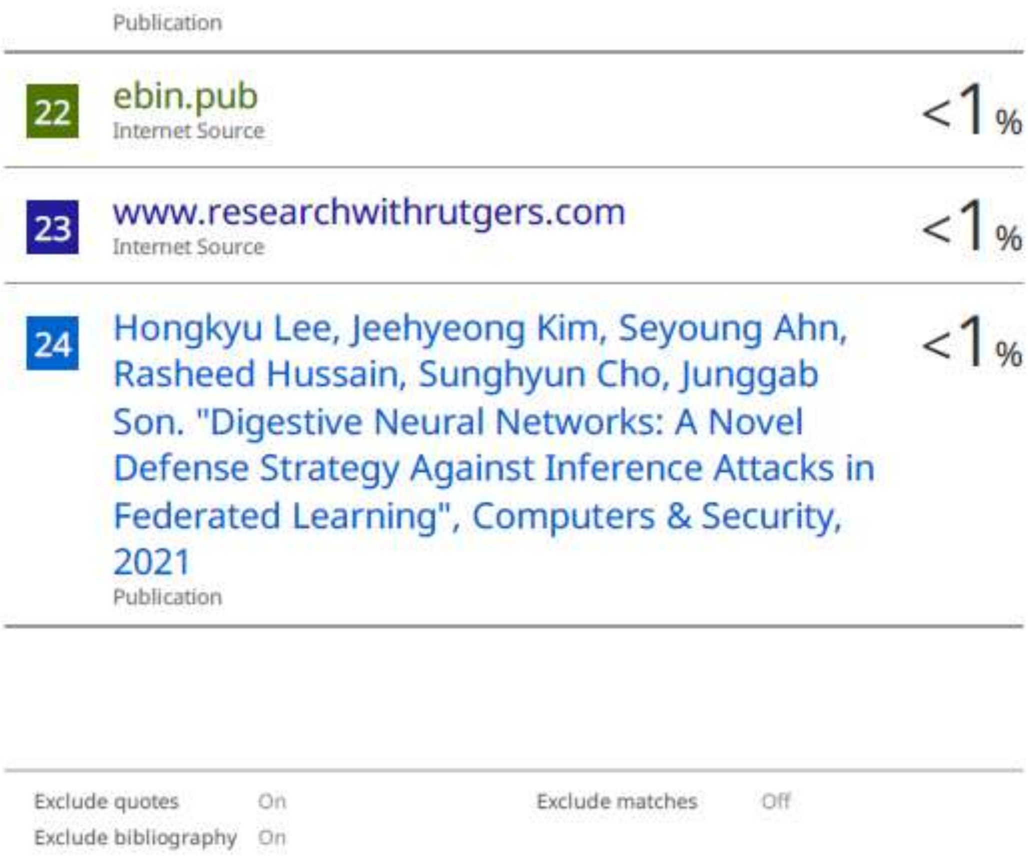


International Conference on Advanced Computing and Applications (ACOMP), 2020
Publication

10	Tomoya Teragaki, Shingo Kawahito, Fuminori Kimura, Osamu Honda. "Data Augmentation for Foreign Material Discrimination using Deep Learning", 2020 9th International Congress on Advanced Applied Informatics (IIAI-AAI), 2020 Publication	<1 %
11	Zhenting Wang, Juan Zhai, Shiqing Ma. "BppAttack: Stealthy and Efficient Trojan Attacks against Deep Neural Networks via Image Quantization and Contrastive Adversarial Learning", 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022 Publication	<1 %
12	Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, Yang Zhang. "Dynamic Backdoor Attacks Against Machine Learning Models", 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P), 2022 Publication	<1 %
13	Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, Siddharth Garg. "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks", IEEE Access, 2019 Publication	<1 %



14	link.springer.com Internet Source	<1 %
15	mecs-press.net Internet Source	<1 %
16	www.fruct.org Internet Source	<1 %
17	Mohammed M. Alani. "On Recent Security Issues in Machine Learning", 2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2020 Publication	<1 %
18	export.arxiv.org Internet Source	<1 %
19	github.com Internet Source	<1 %
20	Kang Liu, Brendan Dolan-Gavitt, Siddharth Garg. "Chapter 13 Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks", Springer Science and Business Media LLC, 2018 Publication	<1 %
21	Liuwan Zhu, Rui Ning, Cong Wang, Chunsheng Xin, Hongyi Wu. "GangSweep", Proceedings of the 28th ACM International Conference on Multimedia, 2020	<1 %



Publication

22	ebin.pub Internet Source	<1 %
23	www.researchwithrutgers.com Internet Source	<1 %
24	Hongkyu Lee, Jeehyeong Kim, Seyoung Ahn, Rasheed Hussain, Sunghyun Cho, Junggab Son. "Digestive Neural Networks: A Novel Defense Strategy Against Inference Attacks in Federated Learning", Computers & Security, 2021 Publication	<1 %

Exclude quotes On
Exclude bibliography On
Exclude matches Off

PLAGARISM CHECK RESULT

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	CHEW ZI YING
ID Number(s)	20ACB03843
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	Model Hijacking Attack

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 10 **%** **Similarity by source** Internet Sources: 9 % Publications: 5 % Student Papers: 3 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

GML

Signature of Supervisor	Signature of Co-Supervisor
Name: Gan Ming Lee	Name: ______________
Date: 23/4/2024	Date: ______________

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB03843
Student Name	CHEW ZI YING
Supervisor Name	DR GAN MING LEE

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]

____
(Signature of Student)
Date: 21/04/2024