T.C.
BAHÇEŞEHİR UNIVERSITY

FACULTY OF ENGINEERING AND NATURAL SCIENCES

CAPSTONE PROJECT PROPOSAL
or
CAPSTONE FINAL REPORT

Analysis of Adversarial Machine Learning Effects on AI Recognition Systems

Yaren Akdemir	2004232	Computer ENG.
Fırat Kutluhan İslim	1803509	Computer ENG.
Emir Can Karakuş	1904179	Computer ENG.

Advisors: Assist. Prof. Ece Gelal Soyak Computer ENG.

ISTANBUL, DECEMBER 2023

STUDENT DECLARATION

By submitting this report, as partial fulfillment of the requirements of the Capstone course, the students promise on penalty of failure of the course that

- they have given credit to and declared (by citation), any work that is not their own (e.g. parts of the report that is copied/pasted from the Internet, design or construction performed by another person, etc.);
- they have not received unpermitted aid for the project design, construction, report or presentation;
- they have not falsely assigned credit for work to another student in the group, and not take credit for work done by another student in the group

ABSTRACT

Analysis of Adversarial Machine Learning effects on AI Recognition Systems

Yaren Akdemir

Fırat Kutluhan İslim

Emir Can Karakuş

Faculty of Engineering and Natural Sciences

Advisor: Assist. Prof. Ece Gelal Soyak

Dec 2023

Abstract

In this project, we performed an adversarial machine learning attack with one of its known methods to an image recognition model and analyze the results. Also we created generative adversarial network modules to create our own train and test images as well as perturbed ones. Then we analyzed the results when we both trained and tested the image recognition system with generated normal images and generated perturbed images. At the end we come up with analyzed numbers and possible solutions to prevent image recognition systems to be misled by AML attacks.

Key Words: Adversarial machine learning · Image recognition · Generative adversarial networks · Adversarial examples

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AML	Adversarial Machine Learning
GAN	Generative Adversarial Networks
DL	Deep Learning
CIFAR-10	Canadian Institute for Advanced Research, 10 classes
FGSM	Fast Grain Sign Method
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
JSMA	Jacobian-based Saliency Map Attack

1. OVERVIEW

This project analyzes the systems of Image recognition model based on CIFAR-10 database, adversarial attack algorithms(AML) and generative adversarial networks (GANs). As well as the effects of the AML attacks with perturbations on test data and perturbations on generated data by GANs on image recognition models. The systems and benchmarks are made by 3 computer engineering students and the entire purpose of these created systems is to analyze AML attacks on image recognition systems. The tested image recognition system works on %86 average accuracy which is after dropping to %45 accuracy with AML attack with epsilon rate of 0.01.

1.1. Identification of the need

As technology rapidly grows, Artificial Intelligence technology comes into every aspect of our lives. Many of them significantly enhance the quality of our lives only if the system is secure and working as intended. One example is AI recognition systems. In good hands this AI system can play a crucial role in autonomous vehicles, human detection systems for security authentication, quality detection in manufacturing, analyzing people's reactions by emotion recognition, fingerprint recognition and even detecting and diagnosing conditions in healthcare. This useful system can be also used to breach security, steal money or harm people as it gets widely adopted in industries if we don't protect it. One way to do that is recognize and analyze the potential dangers like Adversarial Machine Learning(AML) attacks in this case, and take precautions. When we finalized the project we were able to understand the logic behind AML attacks to image recognition systems and define what makes those systems vulnerable. At the end we were able to understand the process of AML attacks with benchmarks and were able to offer solutions to robust the image recognition models against AML attacks. This project will help all image processing developers and users including big tech companies understand the vulnerabilities of their system and offer a solution to either robust it or take precautions.

1.2. Definition of the problem

Malicious attacks on AI recognition systems datasets would cause attacked AI systems to recognize incorrectly. Without safety measures, this could lead to harm to people in real life scenarios. One of the most common attacks on AI recognition systems is the Adversarial Machine Learning(AML) attack. This strategy involves exploiting vulnerabilities in machine learning systems and causing the system to misclassify the input or provide an unwanted output. What makes this attack so dangerous is that AI recognition systems are used variedly in numerous industries. Not having precautions to it can lead to scenarios like, an unwanted person bypassing security of a big system, autonomous vehicles misclassifying an object or person with the road, financial fraud by bypassing fraud detection systems, misdiagnosis of patients in healthcare.

1.2.1. Functional requirements

-The AI that is created to be attacked should be able to recognize pictures content with an optimal percentage.

-The Generative Adversarial Networks(GAN) algorithm should be able to generate noise

image models with selected AML attack strategy, that is min maxed to the point a human can not recognize the noise but it can fool the machines.

-The Adversarial Machine Learning(AML) attack algorithm should be able to perform the attack and mislead the AI recognition system successfully.

1.2.2. Performance requirements

- AI image recognition systems should be able to identify the image with over 90% accuracy.
- The training and computing time of AI should not be greater than the benchmarks in references +20% (with respect to GPU power).
- The image recognition systems dataset should be viable with no corruptions.
- To generate adversarial examples, GAN generation should be trained enough to be able to mislead the recognition system.
- AML examples should not be recognized by the human eye, yet it should fool the recognition system (min-maxing the perturbation).

1.2.3. Constraints

When we talk about constraints of the project we should look into two main subsystems of it: AI image recognition system and AML attack algorithm.

-Scheduling: The biggest constraint we have for both of the subsystems is time, both of the systems are complex systems to implement and having to wait for training time will not be an easy job as it might need many attempts and much training to reach optimality. Also although our team members have prior experience and knowledge on coding and mathematics, we will still need time to learn more about both creating an AI recognition system and AML attack algorithm as well as testing the theoretical knowledge we learn through other studies.

-Cost: This project does not have any budget or cost constraints due to every system we are including has no cost and can run on our local computers.

Standards: The industry standards of image recognition systems are relatively high as there are many optimal examples being used throughout the whole field of business and research. The examples of this includes Tesla using image recognition and computer vision on their cars, Google using image recognition for google lens, Amazon using image recognition for cashierless stores and many more like this. All of these examples are possible to apply in real life with high accuracy of their image recognition systems. When we build the system, we would expect at least 90% accuracy to be able to analyze the attack outcomes precisely. For the standards of adversarial examples, we can only talk about legal requirements as these attacks are not ethical and only can be legally researched on our own systems. There are many computer fraud and data protection laws that one with an adversarial attack algorithm should be careful on where they are using it. For our study, we

will be only using the AML attack subsystem on our other subsystem due to ethical and legal constraints.

1.3. Conceptual solutions

The solution for AML attacks are often not easy and clear. This project's main focus is not to propose a solution but to analyze the problem and documentation of it. At the end we will consider the possible solutions for the case but the solutions will not be included in the application side. The known possible solutions to AML attacks involve increasing robustness of the image recognition system by training the system with noised images, this method is one of the most popular methods in the industry and has various uses. One other method could be using a combination of multiple image recognition models to create a swiss cheese model[1], meaning that every model should be fooled by the attack for the attack to be successful. Using randomization in training data could be another solution as it makes it impossible for an attacker to predetermine the trained dataset input which some attack strategies rely on. If there is enough time and budget to create another system, input preprocessing can be considered as well. The incoming input can be filtered to have no perturbation in the process.

[1] The Swiss cheese model is a risk model that is used in engineering and healthcare as well as computer security. The model is based on the look of swiss cheese slices stacked on top, the slices refer to human-made systems and holes are possible vulnerabilities. The more systems and security measures stacked on top is harder for systems to be exposed as every hole should be lined up hypothetically.

1.3.1. Literature Review

AI Recognition systems were first introduced in 1955, in the form of pattern recognition [1]. According to Selfridge, the system is described as the process of considering important features of an input that is full of irrelevant data. In a study made by Battista Biggio and Fabio Roli [2], This classification of pattern recognition in today's world slightly shifted to the description of a scientific field with the objective of categorizing input data into classes or patterns by identifying significant properties that create differentiation among the analyzed ones. The system has various uses including image, video, text, electromagnetic signals, web, sound recognition and many more like that. Biggio even finds it surprising that such technologies can be deceived with an ease by adversarial examples.

Adversarial Machine Learning (AML) has been seen as a crucial research area in recent years, especially in the context of image and object recognition models. According to Serban et al. [3], adversarial examples depend on the vulnerability of classification functions when exposed to slight perturbations in training data. Given a clean sample x correctly classified by the model with label l, an adversarial example x′ can be created by applying a minimal perturbation P to x to induce a different label l′.

$$
\begin{aligned}
\min_{\boldsymbol{x}'} \quad & \|\boldsymbol{x}' - \boldsymbol{x}\|_p, \\
s.t. \quad & f(\boldsymbol{x}') = l', \\
& f(\boldsymbol{x}) = l, \\
& l \neq l', \\
& \boldsymbol{x}' \in [0,1]^m,
\end{aligned}
$$

Figure 1: [3] Formal definition of adversarial example(x is a representation of image, x' is image that is perturbed, l is the classification of input x, l' is the classification of perturbed input x').

In figure 1; the distance between samples are denoted as $||\ldots||_p$ (usually the p-norm) and $x' - x = \eta$ as perturbation. This distance formula is especially important to quantify the impact of perturbations. The assumptions of a minimal perturbation are not always straightforward, as machine learning systems lack the ability to distinguish between large and small perturbations like a human can. Because of that, keeping the perturbations minimum is the common method used to generate AML examples. Also this is the reason for the challenging part of the crafting adversarial perturbations that can evade model detection while maintaining a level of imperceptibility to the human eye (Serban et al.) [3].

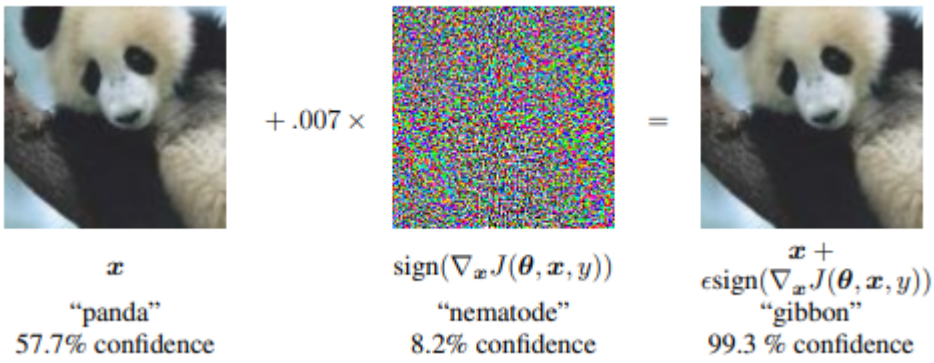


Figure 2: [9] This figure showcases how AML attacks input is formed using the FGSM method, and how output changes(the example is made with ImageNet dataset).

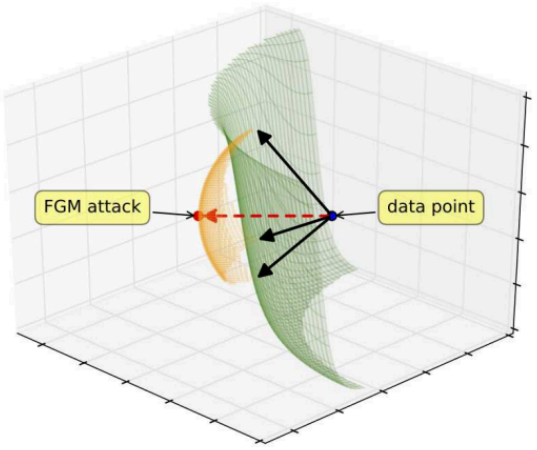


Figure 3: [14] In the figure, we can see the FGSM attack crossing the decision boundary in the data point

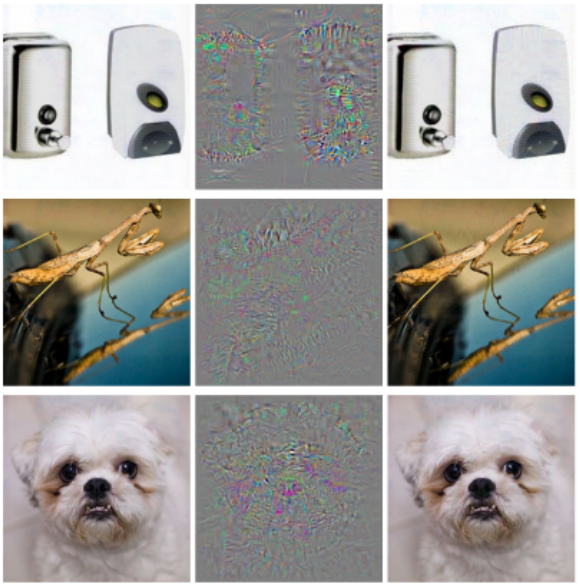


Figure 4: [15] This figure is to show how when noise applied human-eye can't see the difference of image changing

In order to generate more non-perturbed data and also generate perturbed data an algorithm called GAN is variedly used in the field of adversarial examples. As Creswell et al. [4] explains, generative adversarial networks(GANs) within a symbolic framework; there are two networks that are challenging each other's work. One of the networks is called an art forger and the other one is called an art expert. The forger (in the GAN literature known as generator G) creates art pieces to achieve maximum realism in images. The expert (in the GAN literature known as discriminator D) receives both the original art pieces(original data) and forged pieces(forged data) and tries to distinguish them apart. Serban et al, [3] explains it in a simpler manner, it defines GANs networks as a generator which captures the data distribution and a discriminator which calculates the odds that a sample originates from the training data instead of the generator. Both networks are trained in the same timeline.

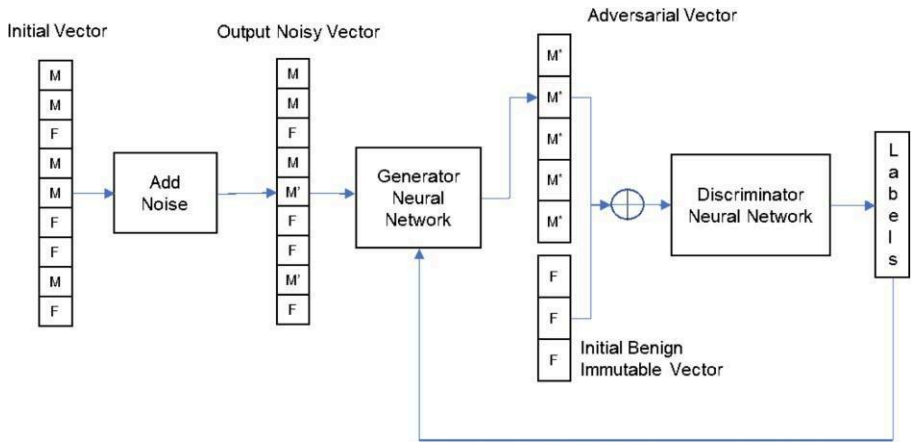


Figure 5: Figure shows how a GAN generator is trained. M' representing perturbed inputs and M* representing generation modified value.

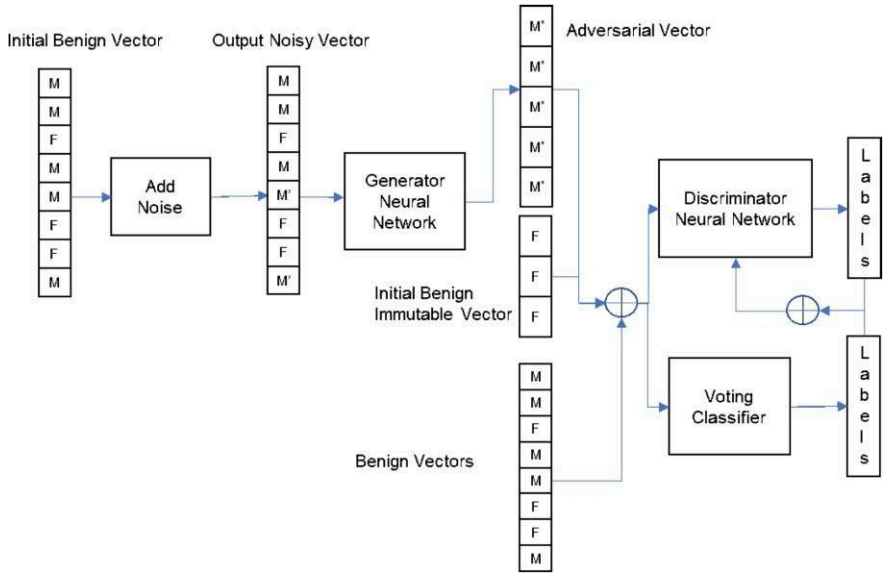


Figure 6: GAN discriminator training. M' represents perturbed inputs and M* represents generator modified values.

In the project, there are two datasets that were taken into consideration for the training and testing the image recognition system. One of them is CIFAR-10 which is a subset of the 80 million tiny pictures collection, called CIFAR-10 [5]. This dataset, which was gathered by Geoffrey Hinton, Vinod Nair, and Alex Krizhevsky, has 60,000 32x32p color pictures from 10 different classes. The dataset has an equal representation of these classes, which include plane, car, bird, cat, deer, dog, frog, horse, ship, and truck. Ten thousand of the 60,000 images are set aside for testing, while the remaining 50,000 are for training. The CIFAR-10 dataset in Python is 163 MB in size.

The second dataset considered to use is ImageNet dataset. According to a study [6] that is made by Krizhevsky, Sutskever and Hinton, which two of them worked on creation of CIFAR-10 dataset model, ImageNet is subset of 15 million high-resolution images with roughly 22,000 categories and as of 2010 an annual competition called ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) started. ILSVRC uses a part of the ImageNet and it is a dataset to both train and evaluate algorithms at large scale. And according to another study from Stanford University the

ILSVRC has become the standard for large-scale object recognition [7]. The tasks of challenge include image classification, single-object localization and object detection. Since both ImageNet and ILSVRC had a drawback of being big scale and high-resolution, we selected CIFAR-10 for the project.

Figure 7: [7] This figure is a representation of how detailed the ILSVRC dataset is compared to other ones. On the left there is the PASCAL dataset which has basic categories to detect classes like "cat". On the other hand, in ILSVRC2012-2014, there are more than 100 categories of breeds of cats for both image and object recognition.

There are many attack algorithms to prepare AML attacks, a research made by the U.S. Military Academy researchers [8] say that, Fast Gradient Sign Method (FGSM), the Jacobian-based Saliency Map Attack (JSMA), Deepfool, and the Carlini Wagner attack (CW) are one of the popular ones amongst them and their many variations are implemented broadly. FGSM is a method initially found by Google researchers, Goodfellow, Shlens and Szegedy [9], the method generated adversarial perturbations by utilizing the loss function with respect to the input image. The process ensures computational efficiency by use of backpropagation. Figure 2 is a showcase of this method being used on the ImageNet database. Also when Goodfellow et al. tried the method on CIFAR-10, they obtained the error rate of 87.15% with average probability of %96 assigned on incorrect labels. In the JSMA a different approach was used. The creators of JSMA, Papernot et al. are explaining [10] that they create adversarial aliency maps by calculating output derivatives(desired outputs) which then they use to recognize the input to be perturbed towards the desired outcome. In an experiment made by Seatsley et al. [11], with the usage of JSMA and an adversarial sketch producing method called histogram sketch generation, they manage to generate misclassification rates greater than 95%. One other strategy was developed by Moosavi-Dezfooli et al. [12] which tries to determine the distance that is closest from original input to decision array of adversarial examples. The last attack technique we covered was Carlini Wagner attack, as the name suggests it was developed by two computer science professors surnamed Carlini and

Wagner, in University of California[13]. The method used three gradient-based algorithms to attack. According to Alhajjar et al. [8] this method was much more effective than all known methods before, regarding the adversarial success percentages obtained with the least amount of noise.

1.3.2. Concepts

Our project will cover two different subsystems with a total of 4 concepts to choose. Subsystems consist of creating an image recognition system and adversarial attack algorithm. To be able to create this system we need to decide on the 4 concepts below.

- To create image recognition system:
  - Which deep learning library/framework to choose
  - Which dataset to train
  - Whether to use GAN algorithm to train it or not
- To create adversarial attack algorithm:
  - Whether to use GAN algorithm to create noise on images
  - Which attacking strategy to use

DeepLearning Library/Framework

With the increased interest in deep learning over the last few years, there are some good DL frameworks getting released day by day like Caffe, CNTK, Theano etc. but there are two that stand out more than others. Those are PyTorch and TenserFlow. Both of these frameworks are widely adopted in machine learning systems, especially on a large scale. According to Google Brain, several google services use TensorFlow [16]. Especially TensorFlows object detection API is frequently used in the fields of agriculture, engineering and medicine. Although features of TensorFlow are well designed, as stated by Paszke et al. [17], like most of the DL frameworks, it is focused on either usability or speed but not both. PyTorch on the other hand is able to check both of the boxes. Also comes with bonus features like easy to debug, flexible with popular scientific computing libraries and efficient for hardware accelerators. Therefore, after some research and deciding on either TensorFlow or PyTorch, we decided to implement **PyTorch** for this project.

Table 1: Comparison of the PyTorch and TensetFlow.

	PyTorch	TenserFlow
Complexity	Low	Medium
Performance	High	High
Features	High	High
Ease of use	User-Friendly	User-Friendly
Flexibility	Flexible	Flexible

Table 2: Benchmark on throughput (higher is better) [17]

Framework	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
CNTK	845	84	210	N/A	N/A	N/A
TensorFlow	1422	66	200	216	9631	4.8e6
PyTorch	**1547**	**119**	**212**	**463**	**15512**	**5.4e6**

Dataset to Train Image Recognition System

Our second challenge to proceed on the project is to find an appropriate dataset to train the AI image recognition model. After some research on it, we decided to conclude on 3 options; CIFAR-10, which is a well-known small size image database to train machine learning models. ImageNet, a popular dataset on high-resolution with its sub dataset ILSVRC being considered as industry standard for object recognition. MNIST, a popular dataset to recognize images of digits. After some consideration, although MNIST has the higher accuracy and lowest testing and training time according to Wu et al. [18], we figured it would be better to work on images rather than digits because of flexibility and more opportunity to analyze aspects. In comparison with ImageNet and CIFAR-10 dataset, since we are covering a small experiment and it does not require high-resolution images with a big amount of classes, ImageNet is expensive in terms of time efficiency and GPU usage for this project. So when we exclude MNIST and ImageNet from the equation, we conclude on **CIFAR-10**.

Table 3: Comparison of the ImageNet, MNIST and CIFAR-10.

	ImageNet	MNIST	CIFAR-10
Complexity	High	Low	Medium
Performance	Low	High	Medium
Dataset Size	Large	Small	Moderate
Image Resolution	High	Low	Medium
Variety of Objects	Various	Digits	Less Various than ImageNet

Table 4: Dataset Benchmarks for 1 GPU [18]

Dataset	Per GPU	Training Time(s)	Testing Time(s)	Accuracy(%)
ImageNet	32	111,781	429	45
MNIST	64	135	0.67	99
CIFAR-10	100	184	1.10	75

Generative Adversarial Networks

Deciding on whether we want to use generative adversarial networks(GANS) in our project was not an easy task. GANs are a framework as described in literature review, which with the cost of increasing complexity, allows high level more images to train dataset and is able to create realistic noised images of AML attacks. Even though we can train the dataset with CIFAR-10 and create an AML attack with any attack strategy without it, it would be a great addition to test on as it would make the system like professional ones in the industry. After some research and consideration, we came up with the idea to use the generative function of GAN for AML attack noise generation, and create the image recognition without GAN to test it initially, after testing it implement GAN to AI recognition system to test it further for changes if there are any.

Table 5: Comparison of image generation system with/without GAN

	with GAN	without GAN
Data	Diverse with new data	Relies on existing data
Robustness	GAN helps the model to robust variations and noises	Model could be sensitive to some variations
Complexity	Much higher complexity	Much simpler system
Training time	Time consuming to train with GAN	Faster system

Adversarial Attack Strategy

Thanks to many researchers, there are a lot of strategies built up that we can choose from. Most common ones we saw were the Fast Gradient Sign Method (FGSM), the Jacobian-based Saliency Map Attack (JSMA), Deepfool, and the Carlini Wagner attack (CW). We have already covered the attack methods in literature view so without getting into too much detail. Although Deepfool has very detailed documentation [12] for it and Carlini Wagner attack is considered the most effective for some sources [8]. In our journey to find resources, we came across more studies made on/with FGSM and JSMA methods. The efficiency of the methods is similar but the technique is very different from one another. After careful consideration, we decided to move on with FGSM since it has more sources that can be found, and it has the best simplicity - efficiency ratio out of all.

Table 6:Comparison of AML attack strategies

	FGSM	JSMA	Deepfool	CW
Strengths	Simple and efficient, very fast	Can create perturbations that are difficult to recognize	Able to find small perturbations for almost all of the models	Most efficient model
Weaknesses	Less effective against robust defenses.	Computationally expensive.	Hard to implement	Requires prior knowledge of attacked models data

1.4. Architecture

The architectural design is relatively uncomplicated as the only thing left to do is assembling our selections from the contents section. The two subsystems have different architectural design, one designed to get image inputs and recognize the input classes correctly, the other desires to make an AML attack on it. To create an AI image recognition system we will be using the PyTorch framework to handle the architectural structure of our software, and we will be creating a mini hierarchical order, using its functions. Then we will train the system with the CIFAR-10 dataset and test the results. The software hierarchy is not determined yet as we will be optimizing the system until it has an optimal accuracy. This process might need the change of hierarchy if not the whole code of the system. Then we will be creating an AML attack algorithm which consists of creation of FGSM attack and using it to generate noised images with GAN generation. Architecture of GAN requires the construction of two algorithms, GAN generator and GAN discriminator. The process of these two algorithms optimizing each other is crucial and when it's done, it is not a difficult task to generate noised examples as FGSM strategies functionalities are supported by PyTorch. After we generate the adversarial examples, we will test the samples on our AI image recognition system.

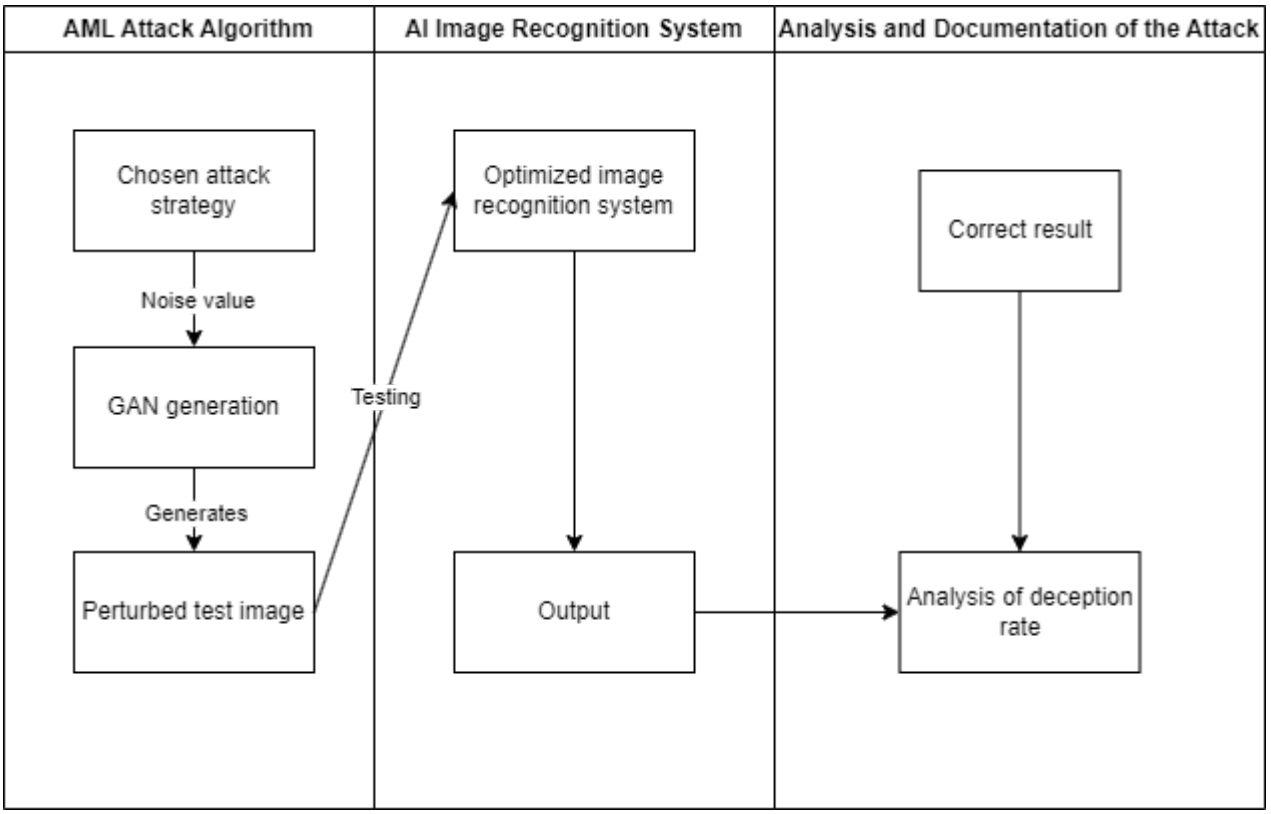


Figure 8: System interface diagram

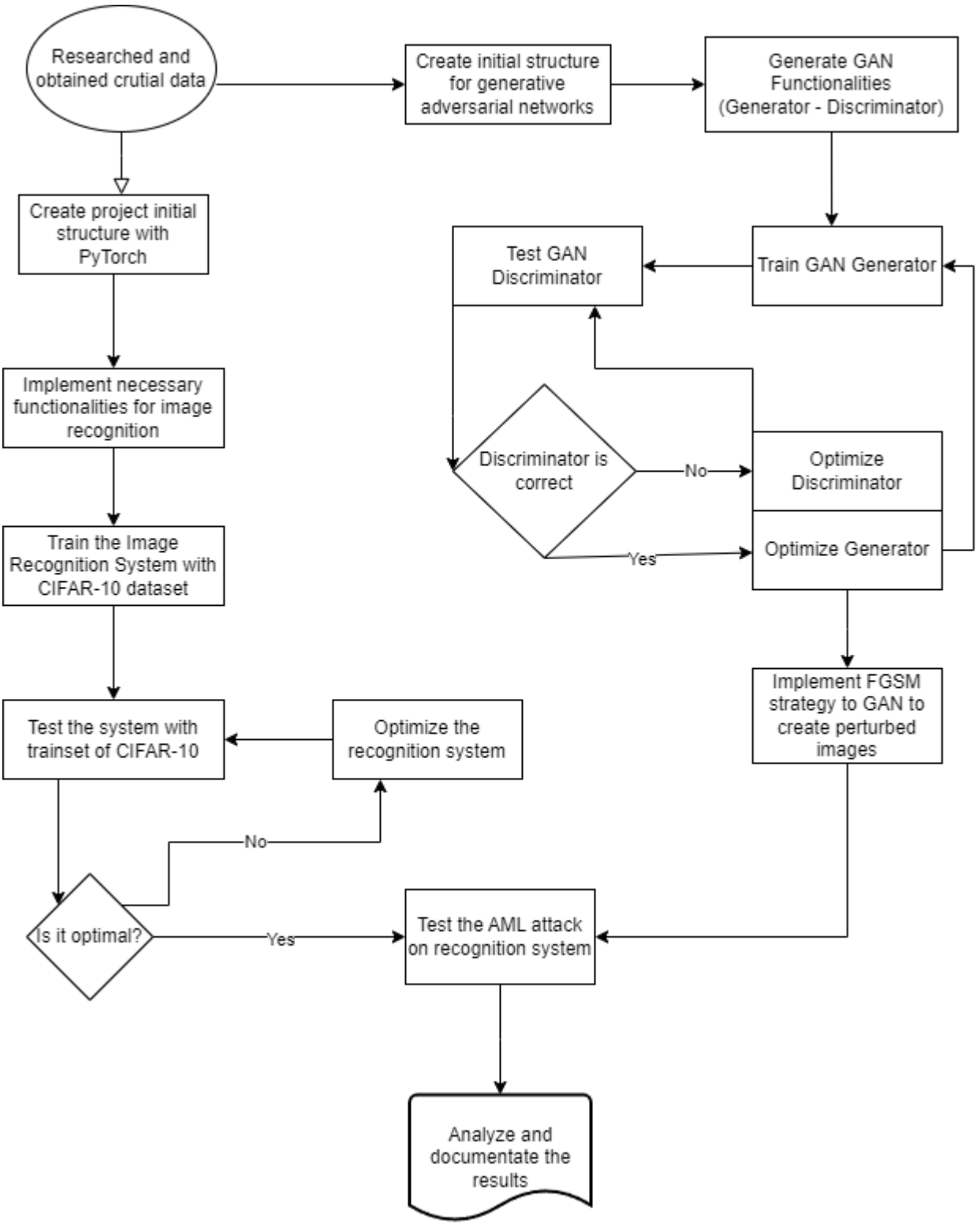


Figure 9: Process(Flow) diagram of the system

2. WORK PLAN

The work plan is divided into 4 phases: research and collecting data, developing AI image generation system, generating AML attack, analyzing and reporting. It is important to mention that we are doing this project as 3 Computer Engineering students and all of the phases will be done by collective work of all of us.

The initial phase is researching through current studies to explore potential information that will be useful for the defining why and how to build an AI image recognition system which is able to recognize pictures with an optimal percentage, subsequently to be attacked by an AML intending to mislead AI. Then deciding on which datasets will be used, which technologies will be used in the project.

Second phase is to develop an AI system that is capable of image recognition. For developing the system, the GAN algorithm will be integrated into the model. Next part of the AI developing process is training AI with the CIFAR-10 dataset that is known for having a wide range of captioned images in different areas. To finish developing an AI system team will test and report on how accurately AI classifies the dataset.

Third phase is to create an AML that will attack the AI system which has been developed in the previous phase. AML will be generated by using the GAN algorithm.

In the concluding step the team will train the AI with these examples. Then testing analyzes the outcomes and reporting on how accurately AI classifies, showing the difference.

2.1. Work Breakdown Structure (WBS)

The WBS is divided into three core phases: developing an AI image recognition system, generating AML attack and documenting the aftermath. Every stage will be completed by us all working together and every part of this project requires its own researching part in order to implement their system. In developing the image recognition system side, after obtaining the necessary knowledge to create, we start by creating the project with the implementation of PyTorch. We start designing the necessary functionalities for the recognition system, simultaneously training the system with the CIFAR-10 dataset. After the image recognition system is up and running, we begin working on generating an AML attack algorithm. For this algorithm we decided to use GAN generation for generating noised images and for the strategy to create the noises we decided to use the FGSM Attack strategy. After ensuring the functionality of both individual systems, we compile them to generate the attack. When we are able to perform the attack and get results on it, we will be documenting changes and outcomes as well as coming up with possible solutions.

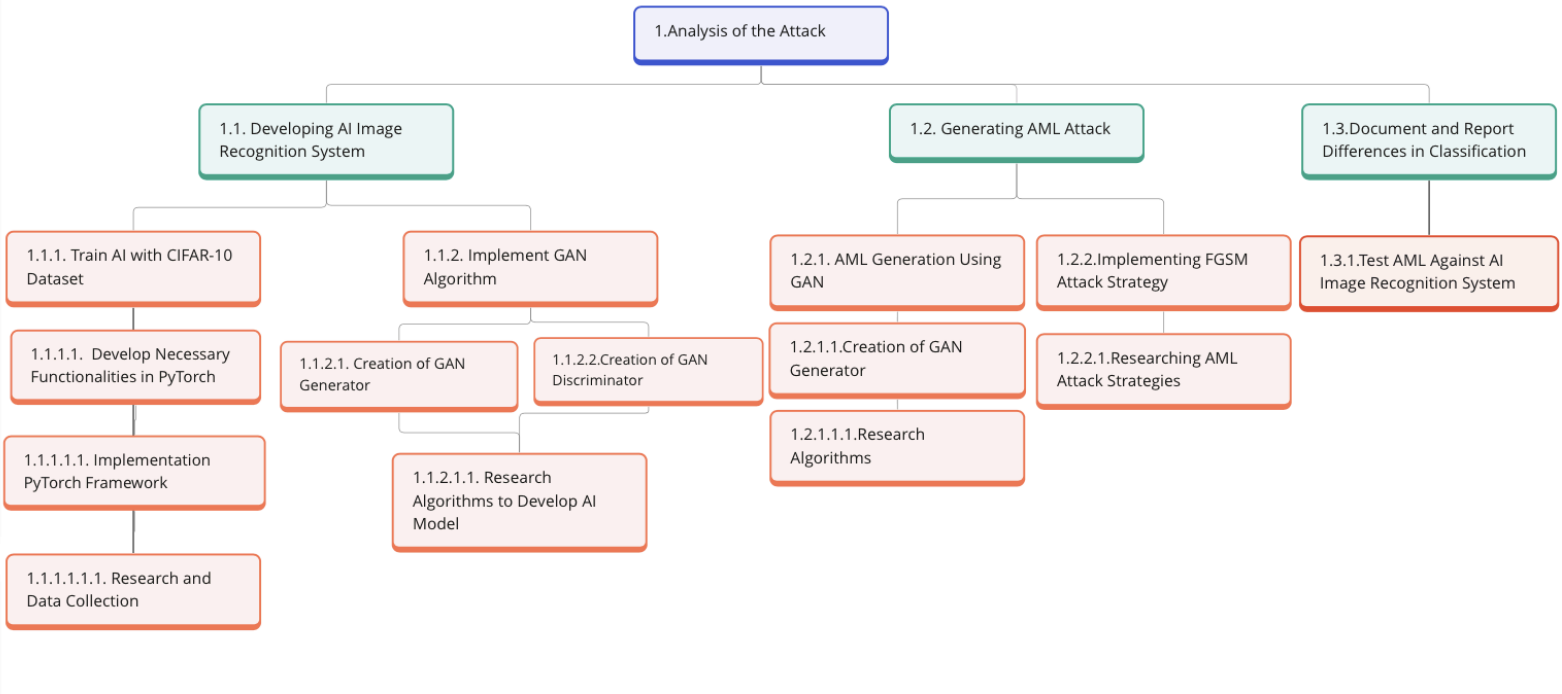


Figure 10: Work breakdown structure for the project.

2.2. Responsibility Matrix (RM)

Since the complexity of the systems we work on are high and will require optimization for the most part, it is important for all of the group members to work together on the tasks.

Table 7. Responsibility Matrix for the team

Task	Emir	Kutluhan	Yaren
Research	S	R	S
Collecting Data	S	S	R
Developing AI Image Recognition	R	S	S
Generating AML Attack	S	R	S
Testing	S	S	R
Reporting	R	S	S
Documenting	S	S	R
	R=Responsible ;		S=Support

2.3. Project Network (PN)

In the project network diagram, we are explicitly mentioning the steps towards the project's goal. Here is a step-by-step textual explanation:

First, we get together as a team to define the project scope and objectives. Which are creating an AI image recognition system, finding and deciding on appropriate algorithms to use to train the AI, deciding on which architectural style is going to be used while building the AI, deciding on which data set is going to be used to train the AI, researching and deciding on which ways to be used in order to generate AML examples, and which ways can be used in order to attack the AI with AML examples.

After confirming the preferences, the structures of the selected algorithms are going to be selected and modified according to the objectives. Then the AI architecture will be built in PyTorch. After that, the data set is going to be imported to train the AI model. When the system is ready, we will be testing the results and optimizing the system accordingly. Later on, after we are done with the attack and tested results, we will be implementing the confirmed and modified algorithms of (GAN) to our image recognition model in order to increase robustness and test the system again.

After confirming the test results match the expected values for non-perturbed dataset training, the team will work on the generation of perturbed AML examples. When it is achieved, the testing phase will come into play again. If the result satisfies the expectations, the team will move forward to the documentation and conclusion parts.

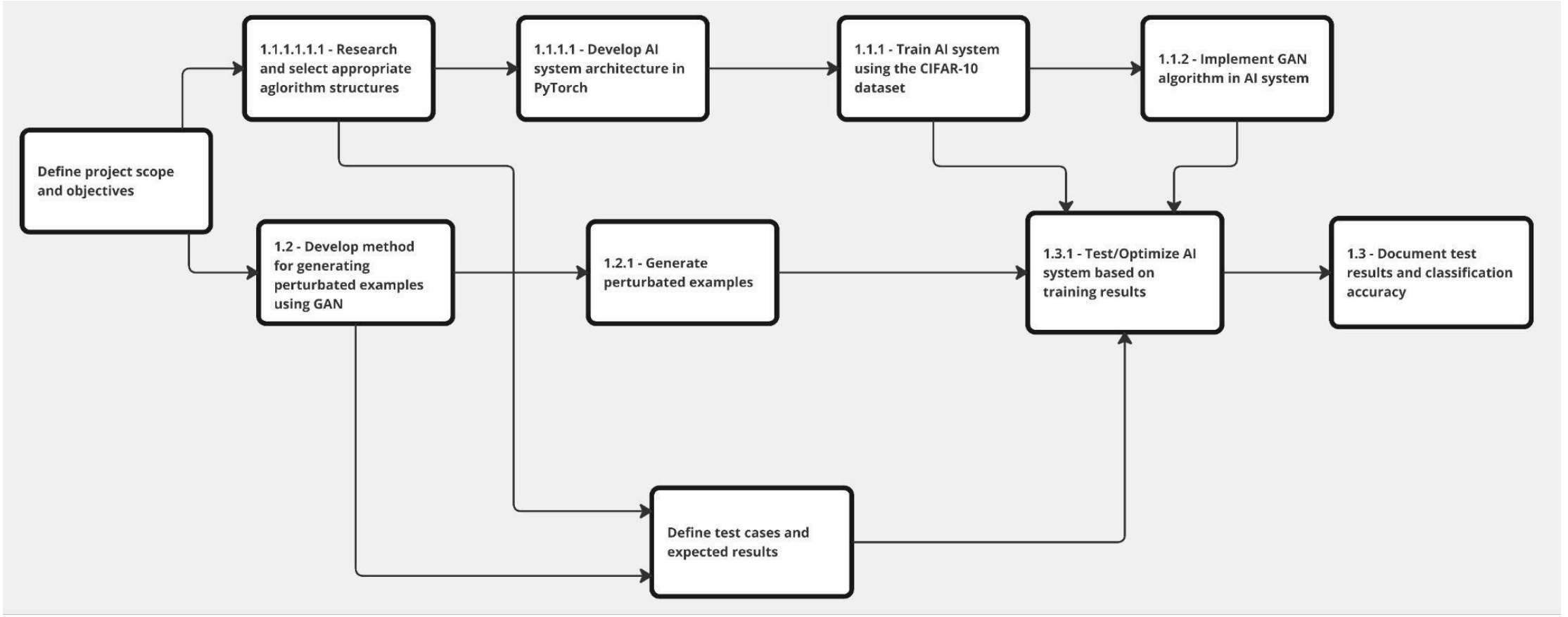


Figure 11: The project network.

2.4. Gantt chart

The project involves four stages. First we will start with the research needed to understand how an AI image recognition system is created, including finding systems and gathering data.

Next we'll move on to developing the AI image recognition system. We'll start by implementing the Pytorch framework and then work on building the functionalities. To train and evaluate the system we'll feed it with the CIFAR 10 dataset. Finally we'll test its performance. Compile a report. After we are done with the whole system, the GAN algorithm will be added to the system to improve the AI image recognition system.

The focus of the third stage is on the creation of the Adversarial Machine Learning algorithm that will attack the AI image recognition system we have developed. The AML system will be created by developing generator and discriminator algorithms of GAN. After that, an AML attack algorithm will be developed with the FSMG strategy we have chosen among the AML attack strategies.

In the final stage, after training, the accuracy of the artificial intelligence model's classification will be tested and the results will be analyzed. AML attacks on AI image Recognition Model will be tested. Differences in AI classification accuracy will be reported.

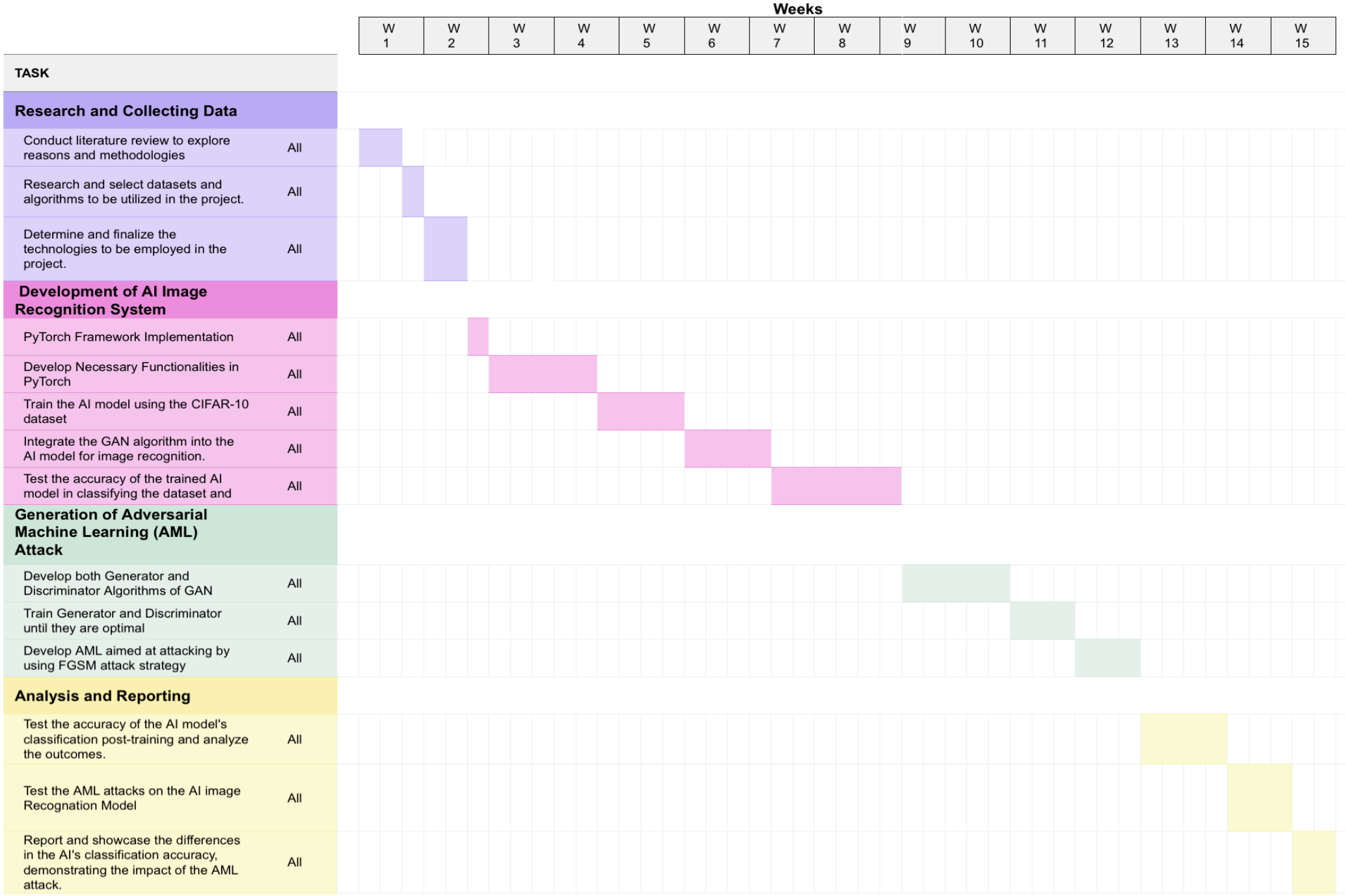


Table 8. Gantt chart for the materialization phase of the project.

2.5. Costs

Since the system we are analyzing does not cost us anything but time and we can not predict the time expense. We can not say anything about the cost part of the project.

2.6. Risk assessment

Changing the scope of a project can have an impact on its goals. Issues related to data availability may cause delays in project development. Pose risks to the accuracy and success of the project. It is possible for data security and privacy to be compromised due to AML attacks. Various factors like resource constraints, shifts in strategy or technological advancements might lead to PyTorch discontinuing support for the CIFAR 10 dataset we have been using resulting in compatibility issues or loss of functionality after updates. To mitigate these risks it is important to explore solutions.

Table 9. Risk matrix

RISK LEVEL		Severity of the event on the project success: Minor	Moderate	Major		
Probability of the event occuring	Unlikely	VERY LOW	LOW	MEDIUM	MEDIUM	This event presents a signficant risk; a plan of action to recover from it should be made and resources sourced in advance.
	Possible	LOW	MEDIUM	HIGH	HIGH	This event presents a very signficant risk. Consider changing the product design/project plan to reduce the risk; else a plan of action for recovery should be made and resources sourced in advance.
	Likely	MEDIUM	HIGH	VERY HIGH	VERY HIGH	**This is an unacceptable risk. The product design/project plan must be changed to reduce the risk to an acceptable level.**
					VERY LOW	This event is very low risk and so does not require any plan for mitigation. In the unlikely event that it does occur there will be only a minor effect on the project.
					LOW	This event is low-risk; a preliminary study on a plan of action to recover from the event can be performed and noted.

Table 10. Risk assessment

Failure Event	Probability	Severity	Risk Level	Plan of Action
Changes in project scope or requirements	Possible It is possible that this may be a difficulty because we have encountered similar problem before.	Major Would change the whole subject of the project	High	Since the change in the project scope is a situation independent of us, we are prepared to make quick decisions and move forward in case it changes.
Data Availability	Unlikely We are in a reliable situation because we choose datasets that we can easily access.	Minor There are several datasets which are achieveable	Very Low	If the data set CIFAR-10 that we have picked becomes unavaible as a team we will use new data sets that we have searched for in the first phase of our project.
PyTorch stop supporting dataset	Unlikely Technological advancements, strategic shifts, resource constraints, or security challenges can lead to such changes. The possibility of major entities like PyTorch withdrawing their support for a dataset is very slim, once they commit to providing support.	Major Our project which is developed using PyTorch's content may encounter compatibility issues or lose some functionality due to updates.	High	In such a situation, the project can either be updated to align like choosing different data set sources with the latest PyTorch version or alternative libraries or in the worst case scenerio we can develop functionalities that we require from PyTorch.
Intellectual Property and Legal Risks	Unlikely AML attacks can pose risks to data security and privacy. However, we do not carry these risks too much as we will try them in our own recognition system.	Major If such events happen, it might result in the cancellation of the system or demand substantial modifications to the existing system, calling for alterations.	Medium	Depending on the situation removing the AML system of us to create a legally revised on later on could be a solution.

3. SUB-SYSTEMS

There are two subsystems in this project: AI image recognition model and AML attack algorithm.

3.1. Artificial Intelligence Image Recognition Model

In the first subsystem, the objective is to achieve an AI image recognition model. Therefore, the appropriate algorithms and structures are decided as follows:

With the use of functionalities in the PyTorch framework, developing the necessary algorithms for recognizing images. After that, the AI model is going to be trained with CIFAR-10. When the system is ready we will be testing and optimizing the system as the AI image recognition model is expected to carry the accuracy ranges that satisfy the project's objectives. On the very last step, after we implement the second subsystem and test it with this one, we will be implementing GAN to the ai model to increase robustness and to be able to test a system which is closer to the real-life examples. To implement GAN we will be creating a GAN generator and GAN discriminator and optimizing both of them with the comparison of their outputs and real values.

3.1.1. Requirements

-The subsystem should be capable of performing image recognition on images.

-The subsystem should be capable of training the AI model using the CIFAR-10 dataset.

-The subsystem should aim to achieve accuracy ranges that satisfy the project’s objectives.

3.1.2. Technologies and methods

Deep Learning Frameworks: There are a couple of frameworks that are popular on projects like this, yet the team chose PyTorch. PyTorch will provide the necessary tools and environment to build and train neural networks.

CIFAR-10 Dataset: In order to train the AI model, there is a need for a data set. The team decided to use CIFAR-10 due to its simplicity, popularity, and ease of use in terms of dimension.

GAN: To implement the GAN method, the team will be creating GAN generator and GAN discriminator algorithms. The team will be using the help of PyTorch generative adversarial networks library.

Performance Metrics: To measure the accuracy of the model, the team will decide on one of the metrics: precision, recall, F1 score or AUC-ROC. This will be selected according to the flow of the project to prevent generating a workload of optimizations for the metrics.

Programming languages: Python

3.1.3. Conceptualization

One of the possible approaches to building an AI image recognition system is using the imageNet dataset. The reason to choose CIFAR-10 instead of imageNet is that imageNet presents a significant workload on the AI model because of its data fragments and high-resolution images. In the AI image and object recognition industry, we see that the imageNet usage is more prevalent than CIFAR-10, but since we are a small group of developers, we decided to use CIFAR-10. This decision also allows us to prevent risks such as the curse of dimensionality. Also, CIFAR-10 handling and implementation have a significantly lower complexity when compared to imageNET.

Another approach to building an AI image recognition model would be using GANs algorithms. These algorithms would allow us to train the ai model with more data that would have been generated from the original training dataset. This approach would increase the AI model's accuracy, yet again creating a workload for the team and AI model.

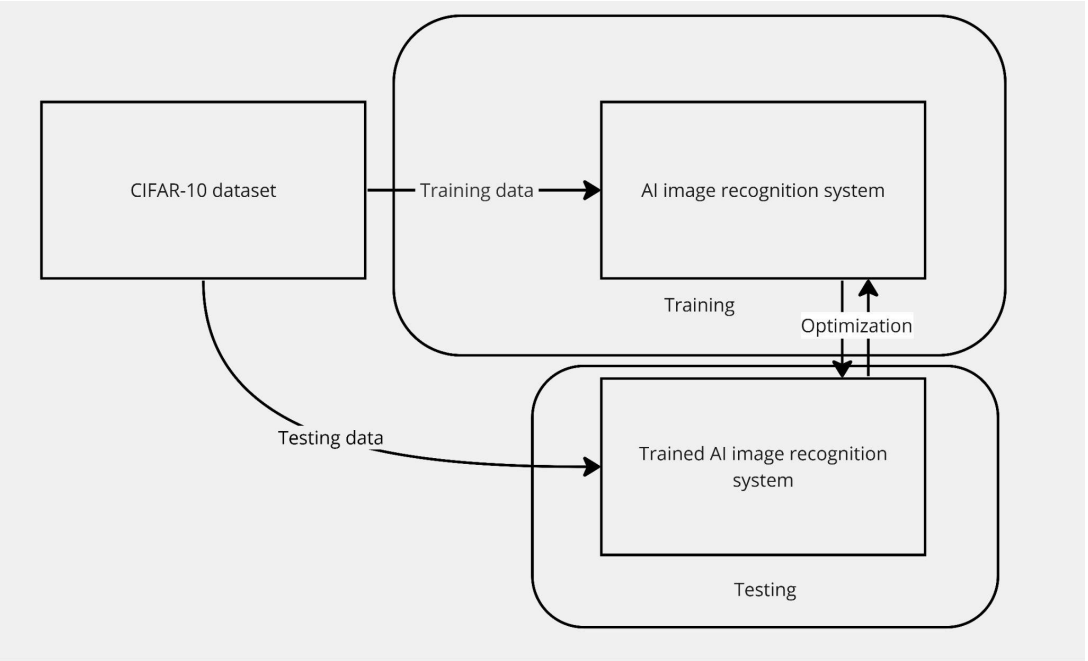


Figure 12: Data Flow Diagram - Subsystem 1

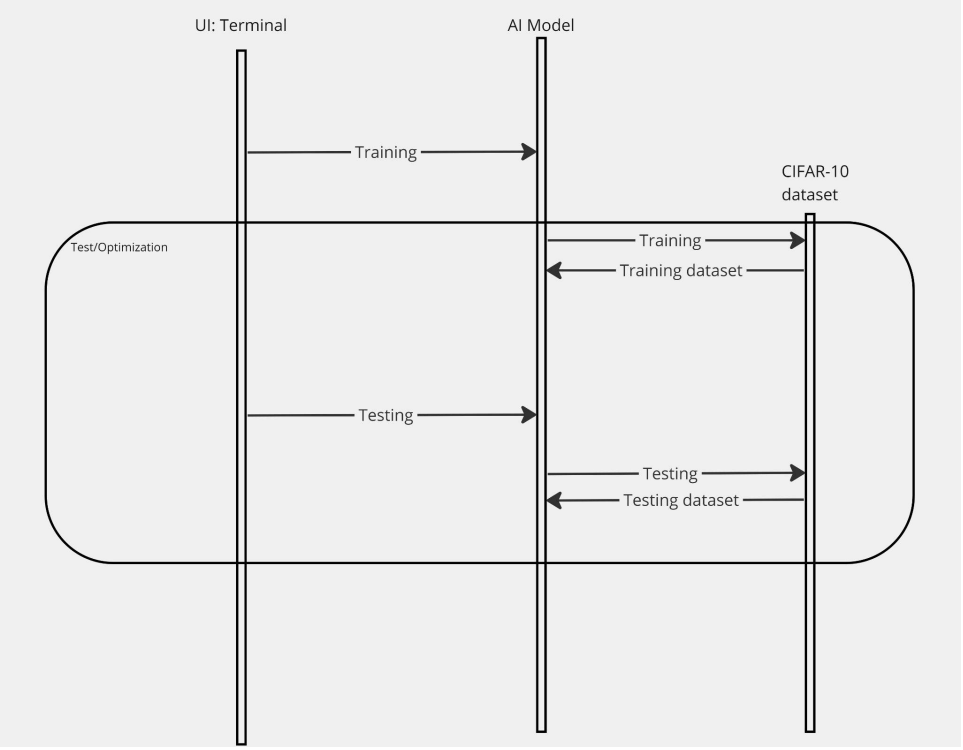


Figure 13: Sequence Diagram - Subsystem 1

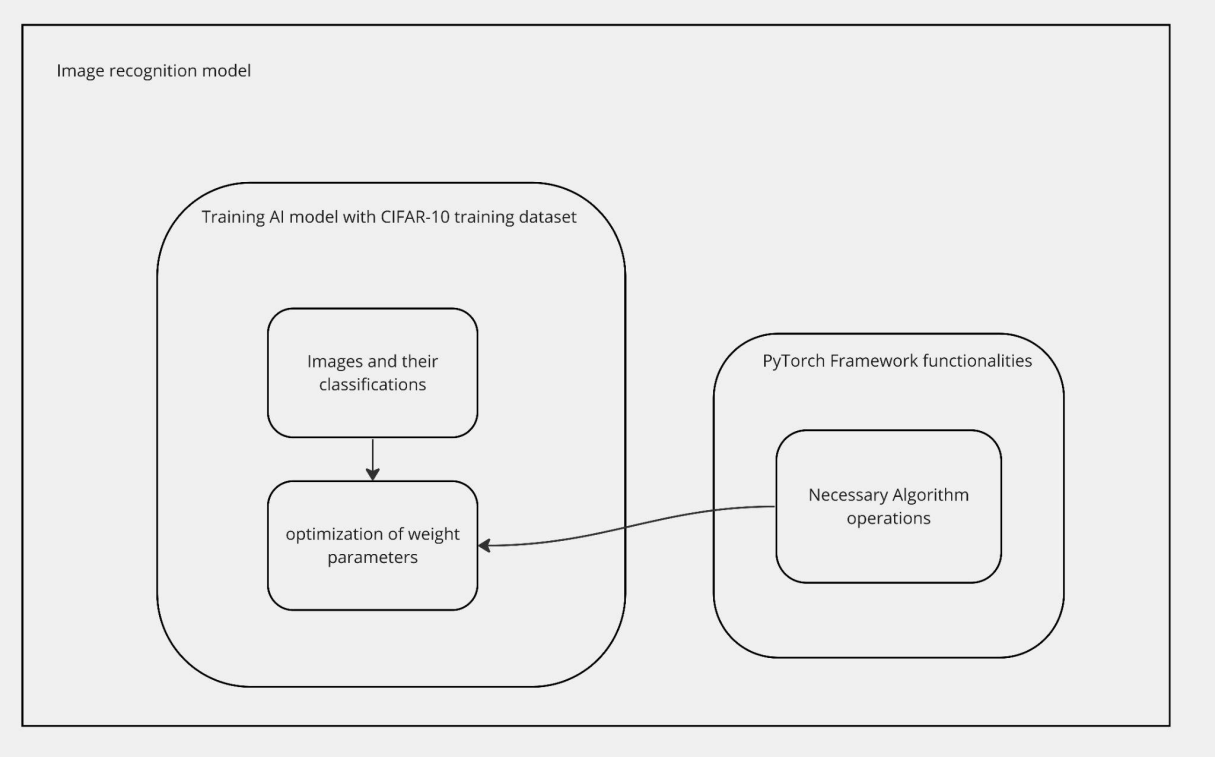


Figure 14: Activity Diagram - Subsystem 1

3.1.4. Architecture

The architecture of the AI image recognition model will be built on the PyTorch framework. We will use the CIFAR-10 dataset to train the AI model. An alternative approach is to use the GAN algorithm to enhance the accuracy of the system by generating more realistic (more like original data) examples to test the AI model. During training and testing while using GAN algorithms generated images, the algorithm optimizes its outcomes in itself in order to generate more precise data.

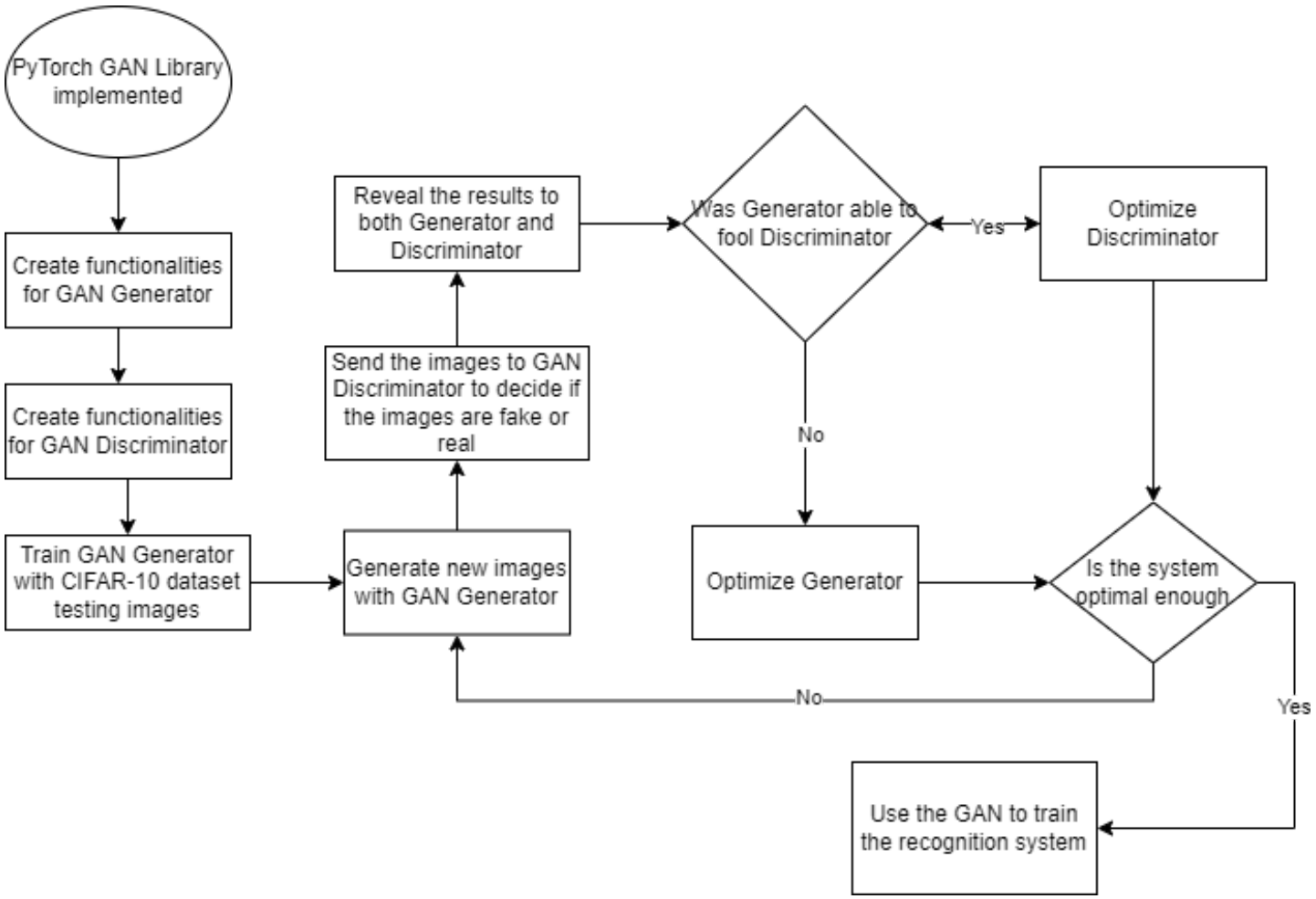


Figure 15: GAN Flow chart subsystem 1

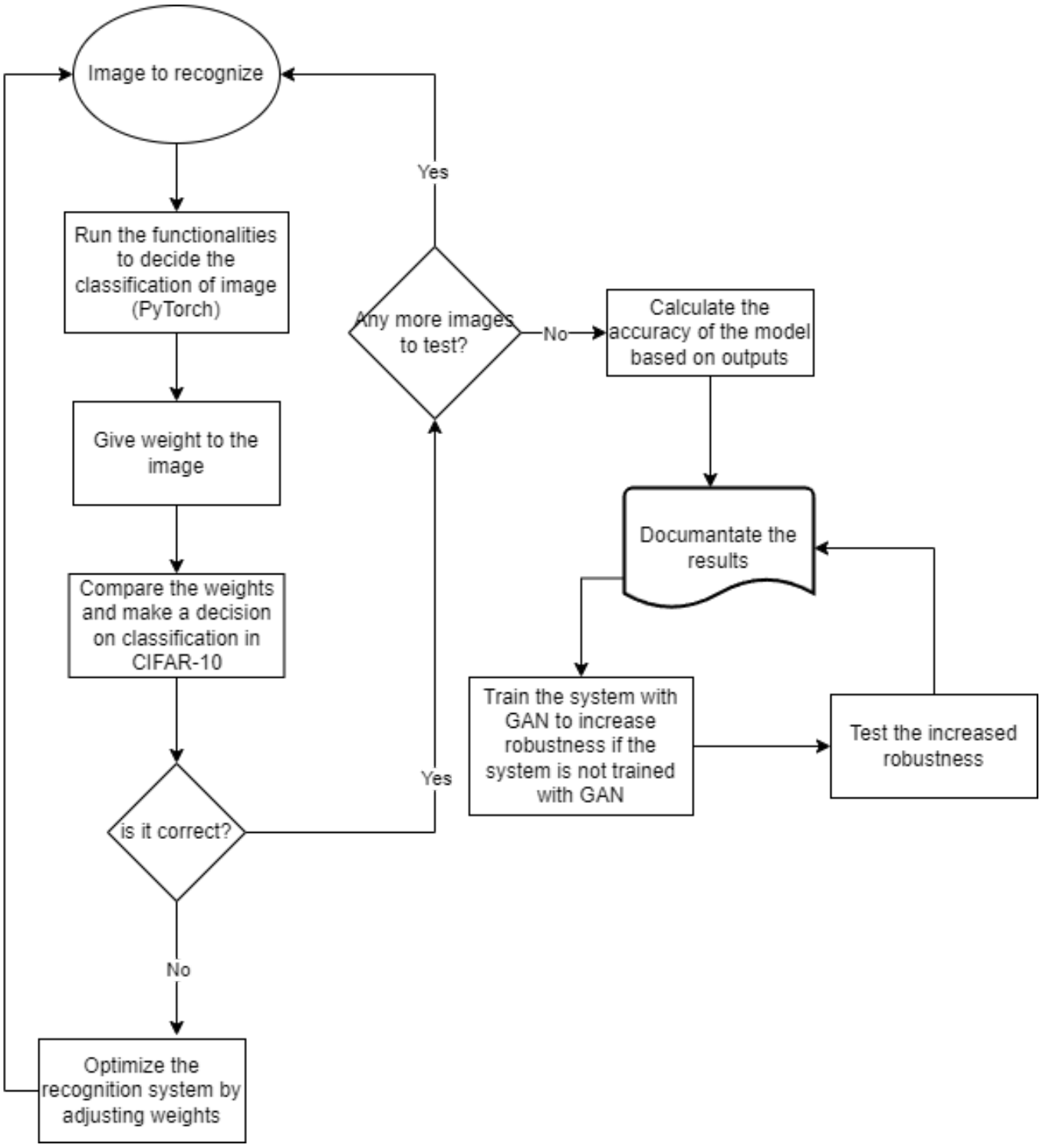


Figure 16: Flow chart of subsystem 1

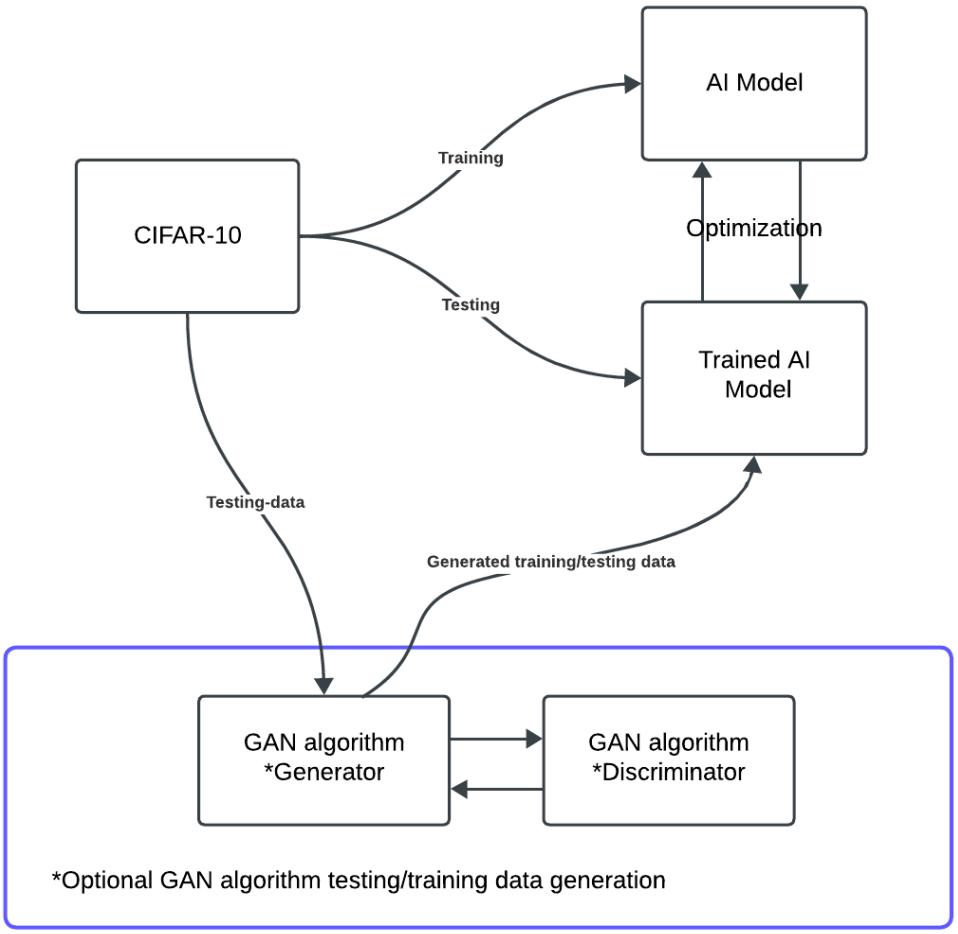


Figure 17: Architecture Diagram - Subsystem 1

3.1.5. Materialization

For the image recognition model, we used PyTorch as a framework. We trained the model with the CIFAR-10 dataset. We also used GAN to increase the training samples by generating more real-like images, and then trained the image recognition model with them. We also used the CIFAR-10 data set for testing the model's accuracy. A difficulty we encountered was the lack of hard-vare capability. Since the project has not been funded, we used the free version of Google Colab to train the model and GAN.

3.1.6. Evaluation

When evaluating an AI image recognition model, there are several key factors to consider. The first is model accuracy. The AI model tries to classify the testing data and achieve an acceptable accuracy percentage. The second factor is robustness: how well does the AI model handle perturbed or generated data imitating the original training set? Our model accuracy for image recognition models was %86 on average, we saw up to %90 accuracy. Increasing accuracy takes time, and we were running the epochs over 14 hours to see improvement. With the help of Google Colab we

trained the model and got an accuracy rate of 85%+, and we can save the model for later training or testing. And with the FGSM attack we decrease the accuracy to 45% (+-3).

3.2. Adversarial Machine Learning Algorithm

The second subsystem to be generated is the Adversarial Machine Learning Algorithm to perform an attack on the AI image recognition system. As a result of our research in the AML system, the FGSM attack strategy will be used among the attack types since FGSM has the best simplicity to efficiency ratio of all.We decided to use the GAN algorithm to strengthen the results of our AML system.

3.2.1. Requirements

In order for the AML attack algorithm to be able to attack the AI image recognition system, it must be understood how to use and integrate the FGSM strategy that will be used in the construction of the AML algorithm.The AML system must be able to access the CIFAR-10 dataset that the AI image recognition model will use. The AML system must be able to effectively attack the AI image recognition system.

3.2.2. Technologies and methods

Adversarial Machine Learning (AML): AML systems main objective is to exploit the image recognition systems vulnerabilities.

Fast Gradient Sign Method (FGSM): FGSM is an adverse attack strategy to reduce the accuracy of AI models. The strategy builds around the generation of examples that the AI model will train on. The strategy uses neural network gradients to generate adverse examples.

Generative Adversarial Networks (GANs): GANs are a group of machine learning frameworks that use two neural networks. The two networks are going to challenge their work and train them accordingly simultaneously. One of the neural networks is called a generator, and the other is called a discriminator. GANs are generally used for generative modeling, which is the requirement of learning to produce new examples which are similar to training data that the ai model used.

Training Dataset: The subsystem should be capable of training the AI model using the CIFAR-10 dataset.

3.2.3. Conceptualization

One of the possible approaches to generating an AML attack is to use a different method than FGSM. For instance, JSMA. JSMA is an AML attack method that detects the most weighted pixels and manipulates them in order to create noise. JSMA is a targeted attack, yet it works slower compared to FGSM.

Another approach would be using the CW-L2 method. CW-L2 method uses a linear layer followed by the softmax output activation. CW-L2 attack uses optimization problem-solving to achieve minimal perturbations. These features make it a more complicated method than FGSM in terms of compilation.

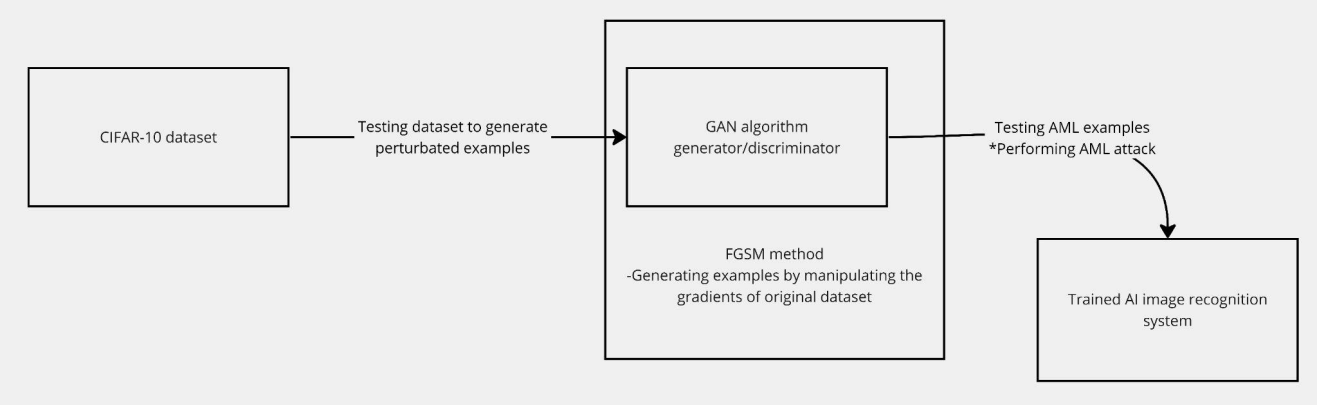


Figure 18: Data Flow Diagram - Subsystem 2

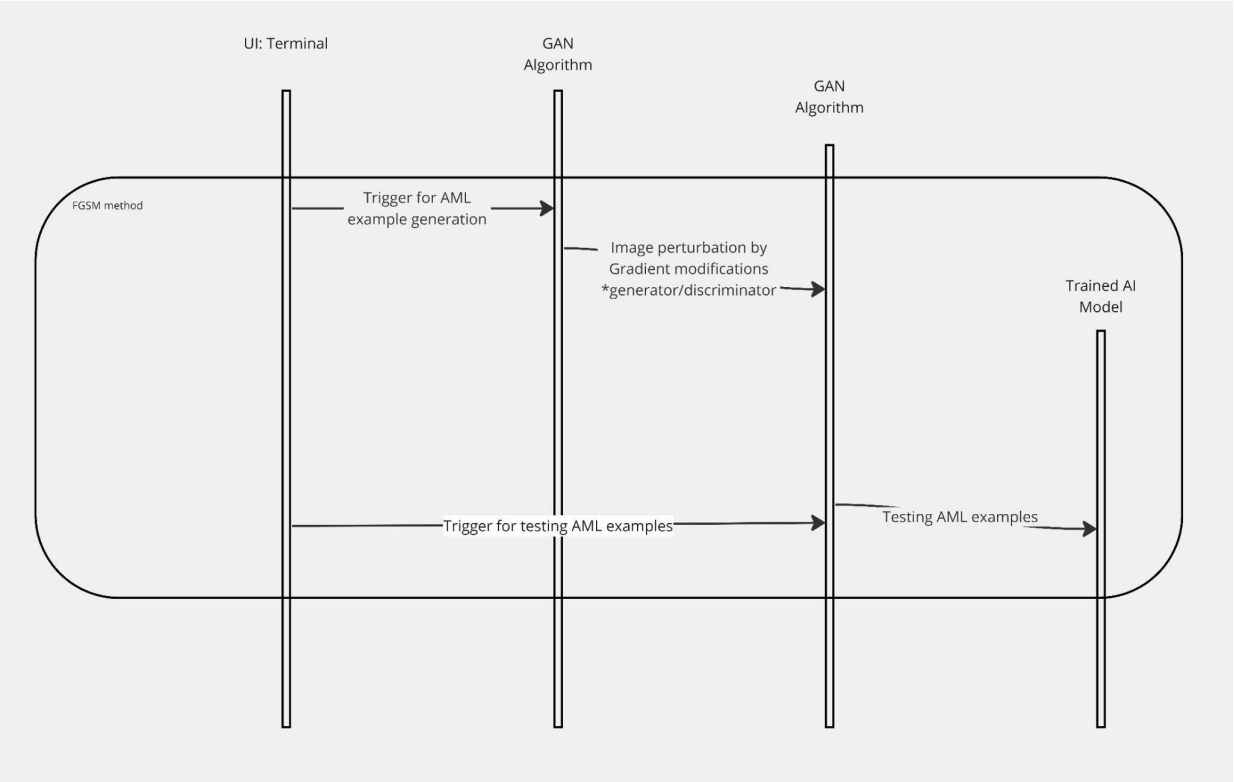


Figure 19: Sequence Diagram - Subsystem 2

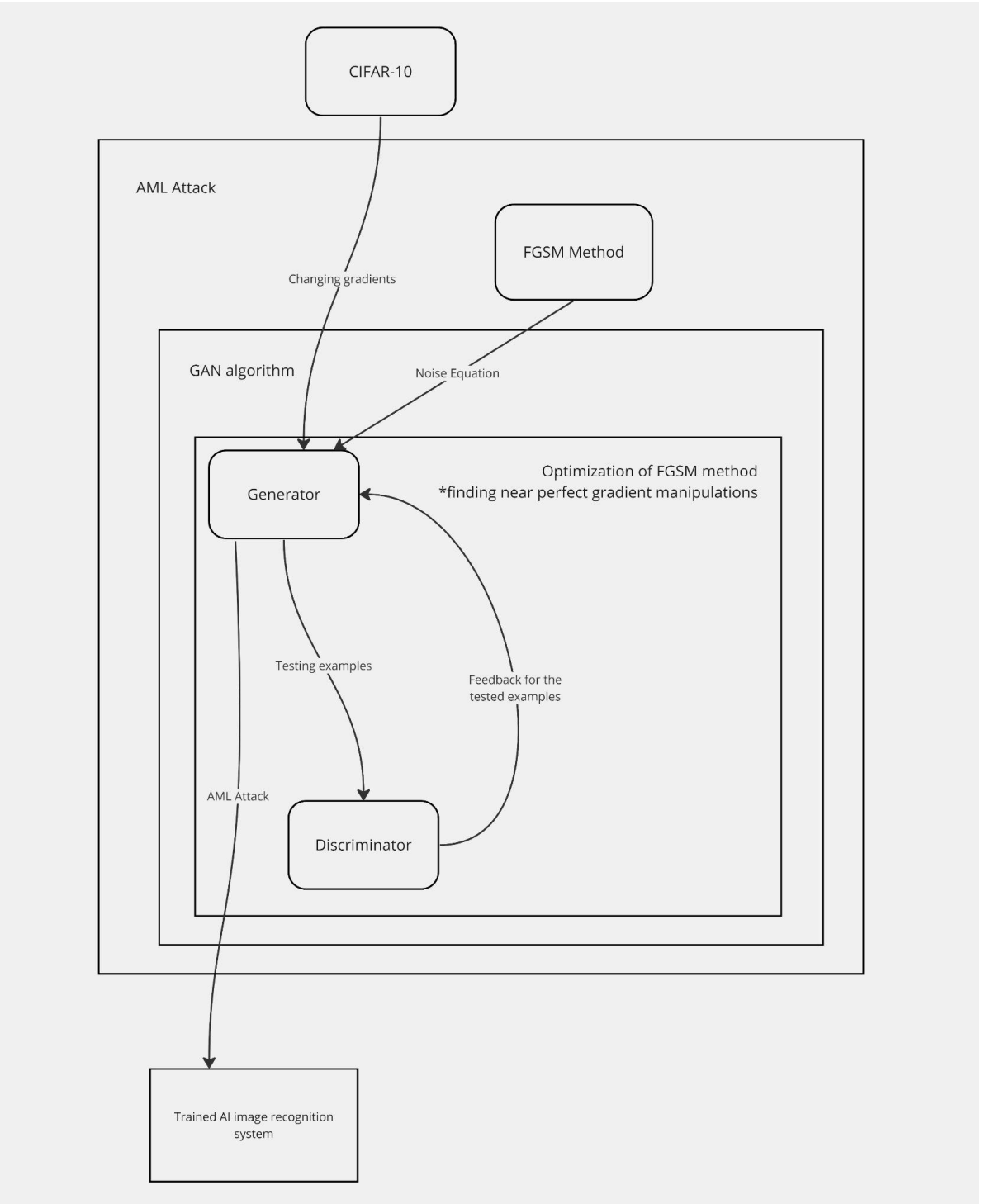


Figure 20: Activity Diagram - Subsystem 2

3.2.4. Architecture

The architecture of the Adversarial Machine Learning algorithm will also be built in the PyTorch framework. As the strategy, we chose the FGSM method. This is a method that manipulates the gradients of images in order to change the corresponding classification. We will use GAN algorithms to generate perturbed AML examples in this subsystem. The GAN has two neural networks: one is a generator, and the other is a discriminator. As we have mentioned the work principles of GAN algorithms before, the AML examples will be generated through a generation network challenging discriminator network. With this method, the algorithm will optimize its outputs when generating gradient-manipulated examples. These perturbed examples will be tested on the trained AI model in order to test the FGSM attack method.

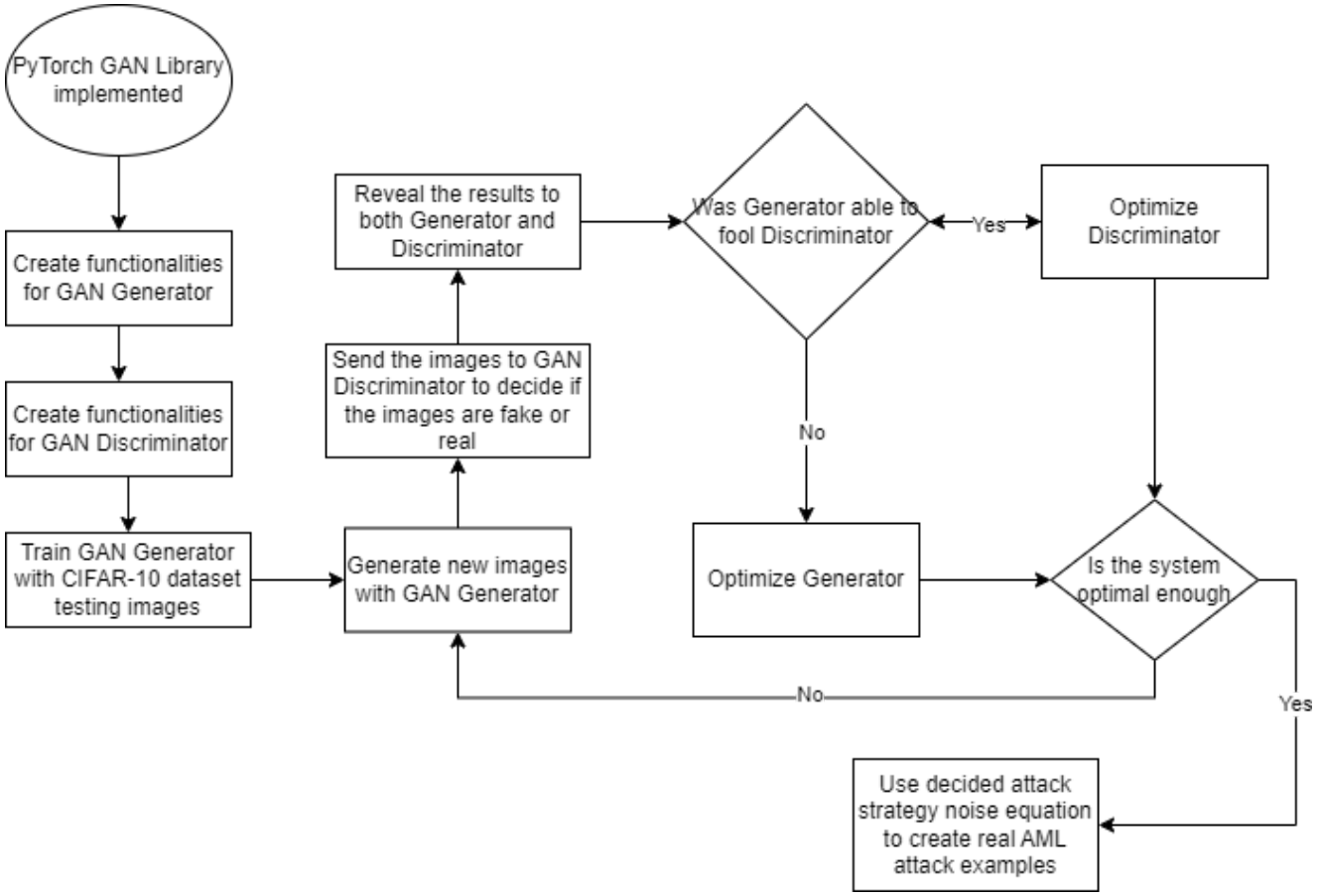


Figure 21: GAN Flow diagram

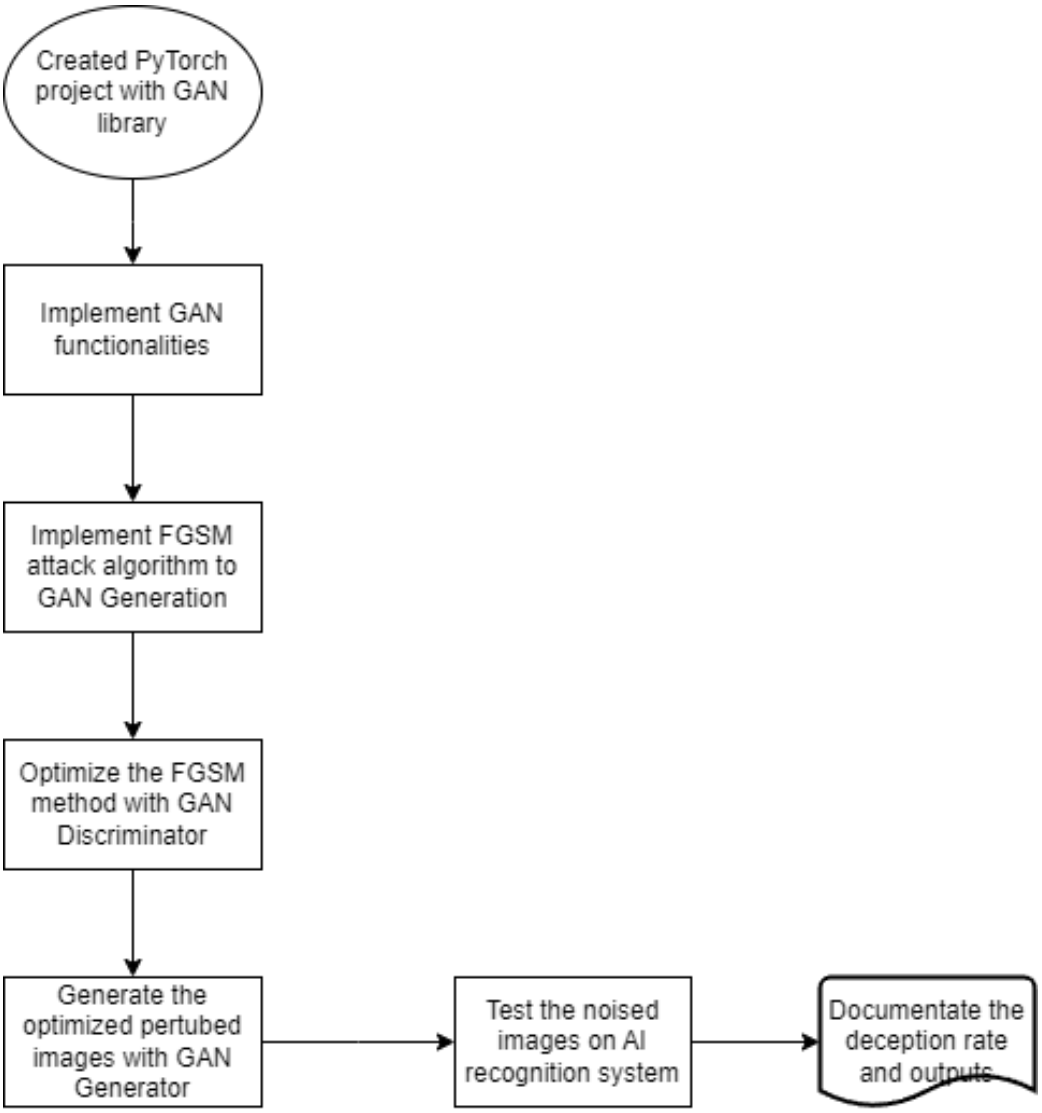


Figure 22: Process chart for AML Attack (GAN is explained in figure 21)

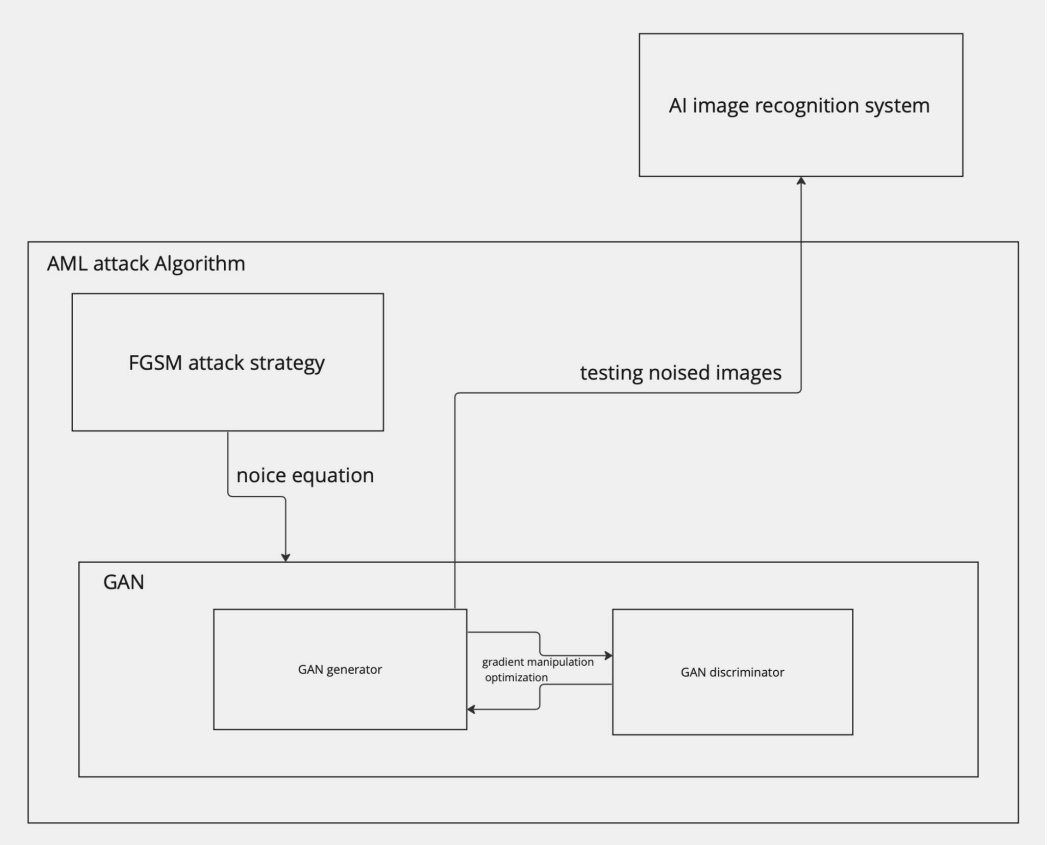


Figure 23: Architecture Diagram - Subsystem 2

3.2.5. Materialization

For the AML algorithm, we used PyTorch as a framework. We implemented a GAN algorithm that can produce AML examples. Since the GAN algorithm has two neural networks as a generator and a discriminator, the optimization of the perturbation has been done between these two neural networks. While generating the images, we also used the Fast Gradient Signed Method on some samples to create perturbed images. The mentioned testing data is going to be provided by CIFAR-10.

3.2.6. Evaluation

The evaluation of this attack has been determined by several factors: Firs, Adversarial Success Rate: This is the rate at which the adversarial examples generated by the FGSM attack are able to fool the image recognition model. Which they can. Second, robustness of the image recognition model: After the FGSM attack, the performance of the image recognition system also determines the success rate of the attack. Which is successful because, after the attack, the accuracy of the image recognition system decreased. Third, efficiency: the time it takes to generate AML examples and the time it takes for the AI model to classify the AML examples. The image recognition model can quickly validate the images compared to the gan module generating them. With the FGSM attack, we decreased the 85%+ accuracy to 45% (+_3).

4. INTEGRATION AND EVALUATION

4.1. Integration

Integration for this project lies in coordinating the two subsystem functionalities to ensure the operation of the entire project.

First, we must ensure that both subsystems have access to the CIFAR-10 dataset through training, testing, and generation. While the image recognition model is training on CIFAR-10 dataset, images from the generator have been used to train the model to both increase the training data count and accuracy of the model. image recognition model, and the gan model is modified to ensure that they can use the same data. When the generator generates an image recognition model, it will be able to evaluate it. This allowed the FGSM attack to be performed. Another important point is the generation of adversarial examples. The implementation of perturbed image generation through a tumor generator has been accomplished. The model can use the FGSM method and can also choose not to use the FGSM method and generate rel-like images. This allowed us to generate and feed the AI model with perturbed images for training. There are monitoring implementations for monitoring the performance of both systems. This implementation has also come in handy when we are dealing with unknown errors. For robustness, we can say that the model is not defended for such attacks. What we can do here is train the image recognition model with images generated with really small Epsilon values to increase its robustness. With this procedure, the model can also classify the perturbed images with more confidence.

4.1.1 Image recognition model

We built an image recognition model that trains on CIFAR-10. The image recognition model trains on 32x32 pixel images with 10 different classes, such as: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. During this period, it learns what "cat" looks like in a picture. It tries to differentiate the images with the knowledge of the patterns that it has learned through training.

Model Architecture:
The model used for this project was ResNet-18. ResNet-18 is a widely used mode for image recognition models. It is used this often because of its simplicity and effectiveness, with skip connections to alleviate the vanising gradient.
Vanising gradient: is a problem in deep learning models. The gradients get extremely tiny during training and propagate back through the network, which can impair deep network learning.

```
net = models.resnet18(weights=False, num_classes=10)
```

Loss Function and Optimizer:
The loss function that we use is called CrossEntropyLoss. CrossEntropyLoss is used mostly for multi-class classification tasks. For the optimizer, we used Adam Optimizer with an initial learning rate of 0.001, but during research and training, we changed it to see the optimal value for the learning rate. For the scheduler, we used StepLR, which decays the learning rate by a factor of 0.1 every 30 epochs.

```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.Adam(net.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)
```

Data Augmentation and Normalization:

We used data augmentation methods on the training photos, such as random horizontal flipping and random cropping, to improve the model's capacity for generalization.
Random Horizontal Flipping: This strategy shows things from various angles and adds more variation for the model to learn from by flipping an image horizontally with a predetermined probability.
Random Cropping: This strategy uses a section of the image chosen at random as the input. This helps the model focus on different areas of the image and enhances its capacity to identify objects in different locations.
In addition, all color channels in the training and testing photos were standardized to have a mean of 0.5 and a standard deviation of 0.5.

```
transform_train = transforms.Compose([
   transforms.RandomHorizontalFlip(),
   transforms.RandomCrop(32, padding=4),
   transforms.ToTensor(),
   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

Data Loading:

We used Torch and Torchvision libraries to manage the cifar-10 data. Here is how we managed the batching, shuffling, and loading of data. The batch size was set to 64.

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,

                                        download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                         shuffle=False, num_workers=0)
```

4.1.2 Generative Adversarial Network (GAN)

Data Preparation:

We resized the images from the CIFAR-10 dataset to 64x64 pixels and normalized them to have a mean and standard deviation of 0.5 for all color channels. The importance of this step is to make the images workable in the model layers.

```
def load_data():
   compose = transforms.Compose([
       transforms.Resize(64),
       transforms.ToTensor(),
       transforms.Normalize((.5, .5, .5), (.5, .5, .5))
   ])
   return datasets.CIFAR10(root=OUTPUT_DIR, train=False, transform=compose,
download=True)
```

```
data = load_data()
data_loader = torch.utils.data.DataLoader(data, batch_size=BATCH_SIZE, shuffle=True)
NUM_BATCHES = len(data_loader)
```

GAN Model Architecture:

The GAN consists of two neural networks that work simultaneously: the generator and the discriminator.

Generator: The generator network tries to replicate the real images from training set. It uses a noise factor by replicating these images. It upsamples the noise vector to the image size by applying a sequence of transposed convolutional layers.

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.linear = nn.Linear(100, 1024 * 4 * 4)
        self.deconv1 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=1024, out_channels=512, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(512), nn.ReLU(inplace=True))
        self.deconv2 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(256), nn.ReLU(inplace=True))
        self.deconv3 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(128), nn.ReLU(inplace=True))
        self.deconv4 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=128, out_channels=3, kernel_size=4,
stride=2, padding=1, bias=False))
        self.output = nn.Tanh()

    def forward(self, x):
        x = self.linear(x)
        x = x.view(x.shape[0], 1024, 4, 4)
        x = self.deconv1(x)
        x = self.deconv2(x)
        x = self.deconv3(x)
        x = self.deconv4(x)
        return self.output(x)
```

Discriminator: The discriminator tries to classify and differentiate between the real images from training and testing dataset and generated images from generators. It extracts features and downsamples the image using a sequence of convolutional layers.

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=128, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.LeakyReLU(0.2, inplace=True))
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.BatchNorm2d(256), nn.LeakyReLU(0.2, inplace=True))
        self.conv3 = nn.Sequential(
```

```
            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.BatchNorm2d(512), nn.LeakyReLU(0.2, inplace=True))
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=4, stride=2,
padding=1, bias=False),
            nn.BatchNorm2d(1024), nn.LeakyReLU(0.2, inplace=True))
        self.output = nn.Sequential(nn.Linear(1024 * 4 * 4, 1), nn.Sigmoid())

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = x.view(-1, 1024 * 4 * 4)
        x = self.output(x)
        return x
```

Initialization and Training:

Normal distribution weights were used to initialize the discriminator and generator. Next, the networks were trained with the Adam optimizer and the Binary Cross-Entropy Loss function. The generator was trained to produce realistic images that may trick the discriminator, while the discriminator was trained to discriminate between actual and fake images.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

generator = Generator().to(device)
discriminator = Discriminator().to(device)
generator.apply(init_weights)
discriminator.apply(init_weights)

d_optimizer = optim.Adam(discriminator.parameters(), lr=LR, betas=(0.5, 0.999))
g_optimizer = optim.Adam(generator.parameters(), lr=LR, betas=(0.5, 0.999))
criterion = nn.BCELoss()
```

4.1.3 Fast Gradient Sign Method (FGSM) Attack

FGSM Attack Implementation:

The gradients of the loss with respect to the input images have been calculated in order to execute the FGSM attack. To increase the loss and mislead the network, small perturbations were added to the images in the gradient's direction.

```
def fgsm_attack(image, epsilon, data_grad):
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon * sign_data_grad
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image
```

4.2. Evaluation

The evaluation of integrated models, algorithms, and strategies that have been used is dependent on every part of the project working smoothly. These evaluation metrics are; First, Adversarial Success Rate: This is the rate at which the adversarial examples generated by the FGSM attack are

able to fool the AI model. Second, accuracy of the image recognition model: image recognition systems ability to recognize the correct classification labels on images. Third, efficiency: the time it takes to generate AML examples and the time it takes for the AI model to classify the AML examples. On our final product, we were able to perform on %86 average accuracy and %90 at best accuracy. adversarial success rate on a default image recognition system is more than we expected as with an epsilon value of 0.007, we were able to decrease the accuracy from %86 to %45. When we trained the image recognition model with the epsilon 0 doctored data, we got the similar accuracy on normal data but when we tested perturbed data robustness increased greatly as we saw the drop going from %45 to %60s. Efficiency on GAN modules was a difficult task as even though we tried to optimize it at best it was still a slow process to train and create examples.

Figure 23,24,25:

We tested how the image recognition system works by changing a variable. In Figure 23, we can see how much the accuracy value changes while the batch number variable changes but the epoch and learning values remain constant. In Figure 24, we can see how the accuracy value changes by changing the learning rate value. In Figure 25, we can observe the relationship between epoch number and accuracy as the epoch number changes.

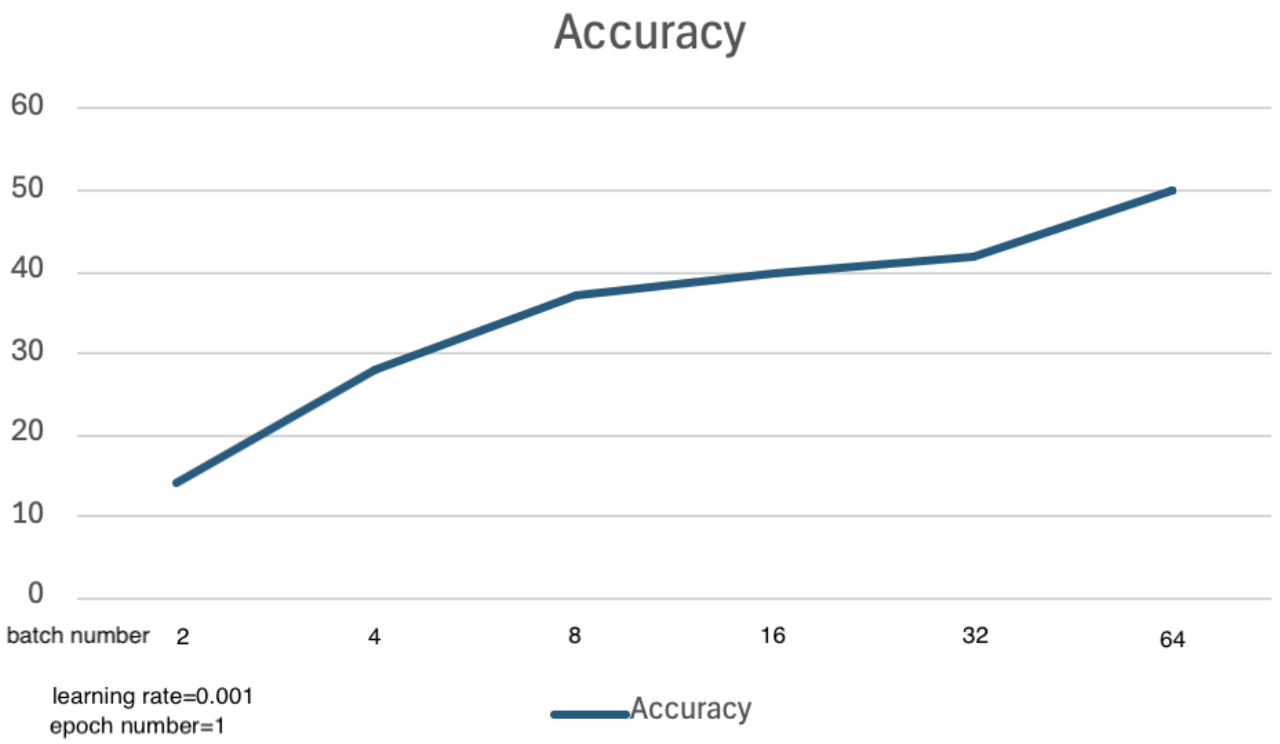


Figure 23: Batch Number - Accuracy Change Chart

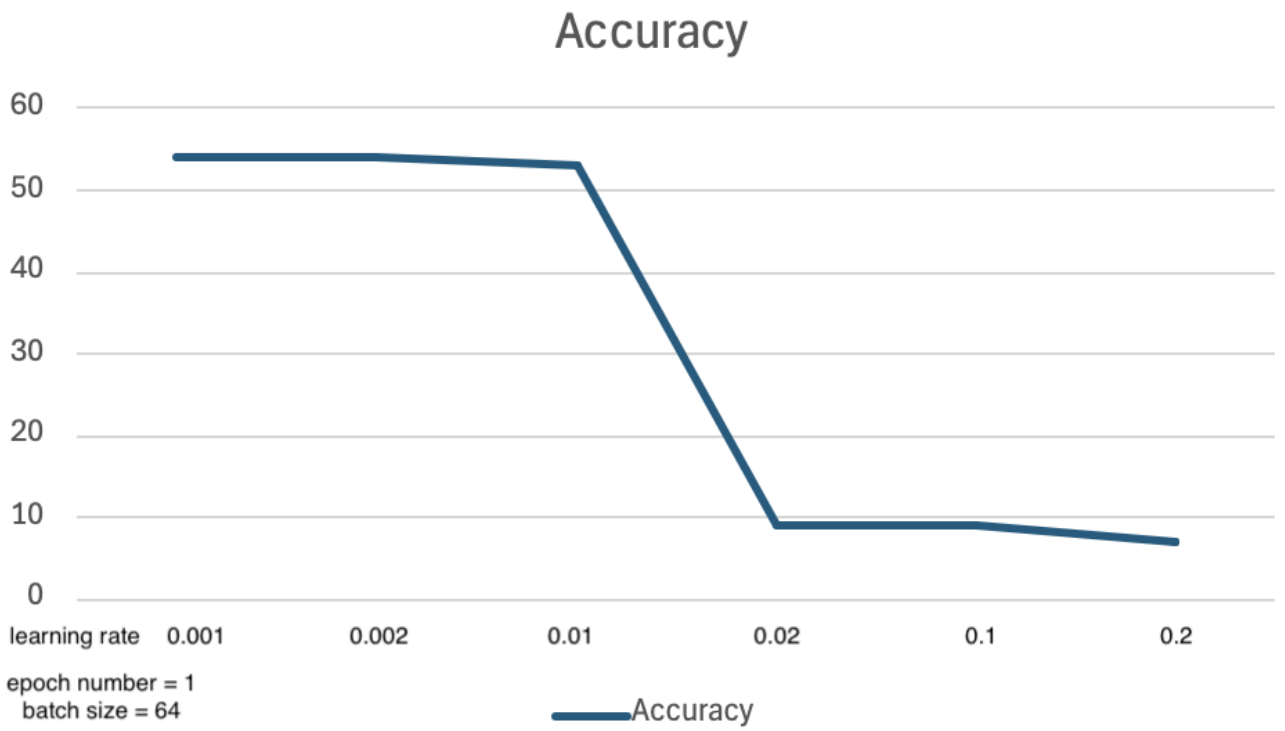


Figure 24: Learning Rate - Accuracy Change chart

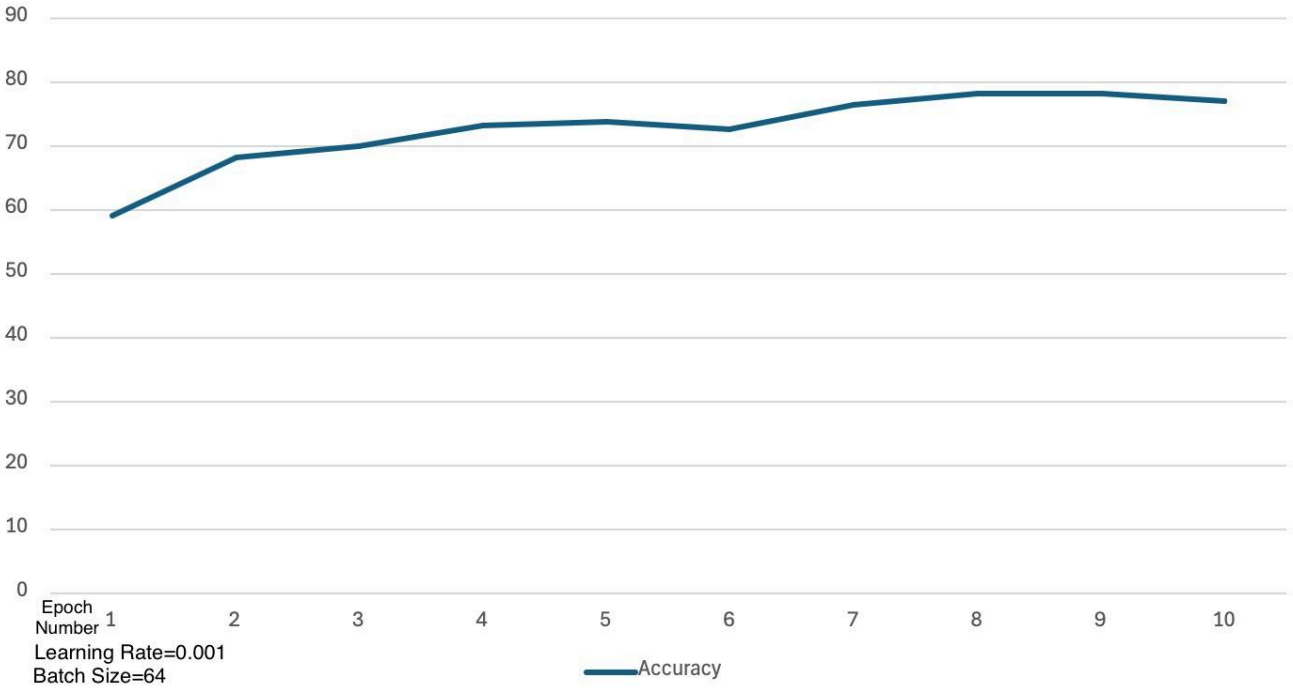


Figure 25: Epoch Number - Accuracy Change chart

4.2.1 Image recognition model

Training and Validation:

The model was trained for a specified number of epochs using the training data. Backpropagation:

computes the gradient of the loss function with respect to the network's weights. Furthermore, every epoch's training loss was documented. The model was evaluated for each epoch using the validation set in order to calculate the validation accuracy and loss.

```
num_epochs = 0
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    net.train()
    running_loss = 0.0
    for inputs, labels in trainloader:
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    train_losses.append(running_loss / len(trainloader))

    net.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in testloader:
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    val_losses.append(val_loss / len(testloader))

    scheduler.step()

    print(
        f'Epoch {epoch + 1}, Train Loss: {train_losses[-1]}, Val Loss:
{val_losses[-1]}, Accuracy: {100 * correct / total:.2f}%')

print('Finished Training')
```

Model Saving and Loading:

For ensuring that the model does not have to be trained from the ground in future runs, parameters of models had been stored in a file. If the file exists, model parameters are loaded from it and overwritten.

```
model_path = 'image_recognition_model.pth'
if os.path.exists(model_path):
    net.load_state_dict(torch.load(model_path))
    print("Loaded existing model from", model_path)
else:
    print("No existing model found, starting training from scratch")
```

Final Accuracy Evaluation:

Finally, a deviation from the model training process is the validation phase on the testing set after the logout. The most key part of the accuracy was the calculation of the model's prediction by using comparison with the real readings of the test photos label.

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
        100 * correct / total))
```

4.2.2 Generative Adversarial Network(GAN)

GAN's other important aspect, generation image quality was a crucial element to the quality of the model. The school climate manager too took before and after shots in each of the venture operation stages. This was a way of passively tracing its development. The card quality of the snapshots that addressed firstly scrutinized and dead loss factor was beneficial in determining how well the generator and discriminator are performing.

```
def generate_and_save_images(generator, test_noise, epoch, save_dir, class_names,
num_images=NUM_TEST_SAMPLES):
    with torch.no_grad():
        test_images = generator(test_noise).cpu()
    nrows = int(np.sqrt(num_images))
    grid = vutils.make_grid(test_images, nrow=nrows, normalize=True, scale_each=True)

    grid_image_path = os.path.join(save_dir, f'epoch_{epoch}_grid.png')
    vutils.save_image(grid, grid_image_path)
    print(f'Saved grid image at {grid_image_path}')

    for i in range(test_images.size(0)):
        image = test_images[i]
        class_name = class_names[i % len(class_names)]

        image_name = f'epoch_{epoch}_image_{i}_class_{class_name}.png'
        image_path = os.path.join(save_dir, image_name)
        vutils.save_image(image, image_path)

        np_image = image.numpy().transpose((1, 2, 0))
        plt.figure()
        plt.imshow(np_image)
        plt.title(f"Generated Image at Epoch {epoch} (Class: {class_name})")
        plt.axis('off')
        plt.savefig(image_path)
        plt.close()
```

4.2.3 Fast Gradient Signed Method (FGSM) Attack

The FGSM attack aims to reduce the image recognition model accuracy with perturbed images. The code below the FGSM attack generates a perturbed image. We reseached and tried to find the optimal epsilon value to generate these perturbed images. For the sake of efficiency, here we try to attack the discriminator with fgsm. Yet we also did the same experiments with the original image recognition model.

```
def test_fgsm(generator, discriminator, device, test_loader, epsilon):
    criterion = nn.BCELoss()
    correct = 0
    adv_examples = []

    for data, _ in test_loader:
        data = data.to(device)
        data.requires_grad = True

        fake_data = generator(noise(data.size(0))).detach()
        real_data = Variable(data, requires_grad=True)

        d_optimizer = optim.Adam(discriminator.parameters(), lr=LR, betas=(0.5,
0.999))
        d_error, _, _ = train_discriminator(d_optimizer, real_data, fake_data,
discriminator)

        loss = criterion(d_error, real_data_target(real_data.size(0)))
        discriminator.zero_grad()
        loss.backward()
        data_grad = real_data.grad.data

        perturbed_data = fgsm_attack(data, epsilon, data_grad)

        output = discriminator(perturbed_data)
        final_pred = output.round()

        if final_pred.item() == 1:
            correct += 1
            if epsilon == 0:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append(adv_ex)
        else:
            if len(adv_examples) < 5:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
            adv_examples.append(adv_ex)

            final_acc = correct / float(len(test_loader))
            print(f"Epsilon: {epsilon}\tTest Accuracy = {correct} /
{len(test_loader)} = {final_acc}")

            return final_acc, adv_examples
```

5. SUMMARY AND CONCLUSION

We have constructed 3 subsystems, image recognition module, generative adversarial networks and one works with it which is FGSM method AML attack. All of these 3 systems use PyTorch library and both image recognition module and generative networks use resnet18 neural network structure. The image recognition module is trained and tested with CIFAR-10 dataset and therefore AML attack perturbation is initially applied on CIFAR-10 models. After we trained GAN modules we began to create our own images with and without FGSM method perturbation to test the effects on image recognition model. If we explain it briefly using resnet18 NN structure on image recognition module gave us greater accuracies up to %90, and performing AML attack on default image recognition system made it drop to %45 accuracy. But if you also train the image recognition model with epsilon 0 doctored data it increases on accuracy. When we applied 0.01 perturbation on images and tested it with the model that was trained with doctored data the accuracy dropped to %68 rather than %42. Also when we initially created images without perturbation using GAN modules and tested it on %86 accuracy model, we saw that the accuracy managed to stay on %76. Since we did not have strong systems to train and create with GAN modules, the accuracy we got was more than acceptable. Also after we created images with FGSM method perturbations and tested that data with image recognition model, rather than accuracy dropping to average %65, it dropped under %50. It is important to mention that we have also observed the confusion matrix on the default image recognition model to see what labels are mixed up with which ones initially and after perturbation. We come to the conclusion that with certain AML methods, it is possible to mislead the image recognition model to a specific classification. For example with finding the correct equation on AML attack we can mislead the image recognition system from a dog image to cat or ship. The most obvious strategy to prevent such cases would be training data with a greater dataset, as well as with your own images created with GAN modules. Also a better solution would be training the image recognition model with perturbed images, that is it would not mislead the system as much but we should be careful because it might drop the accuracy on normal datasets.

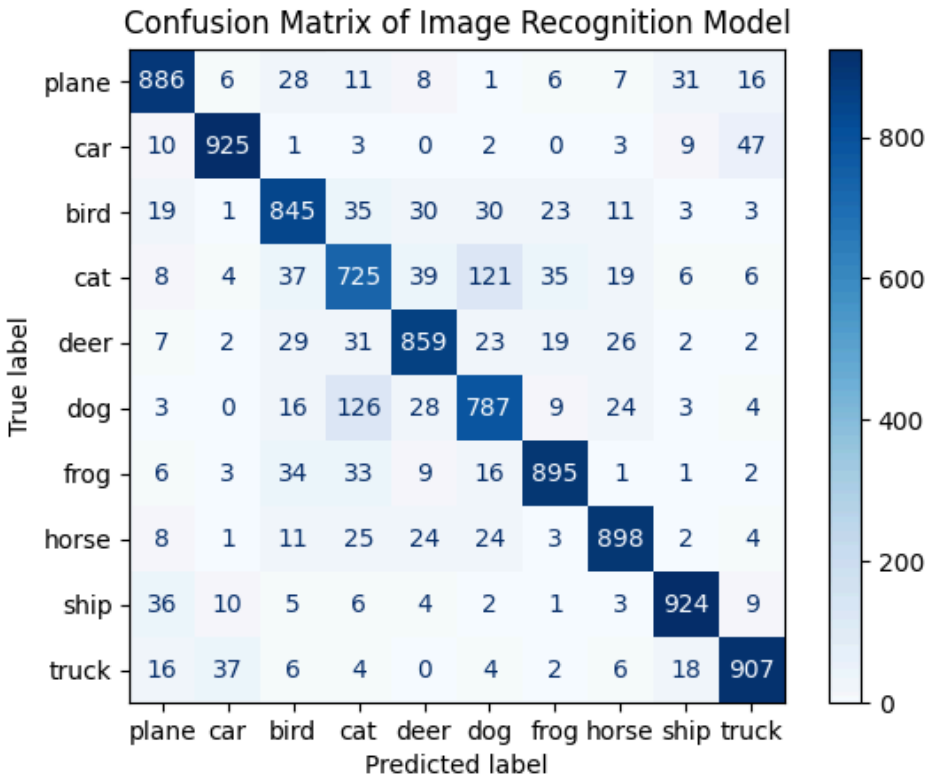


Figure 26: Confusion Matrix of Image Recognition Model with Normal Data

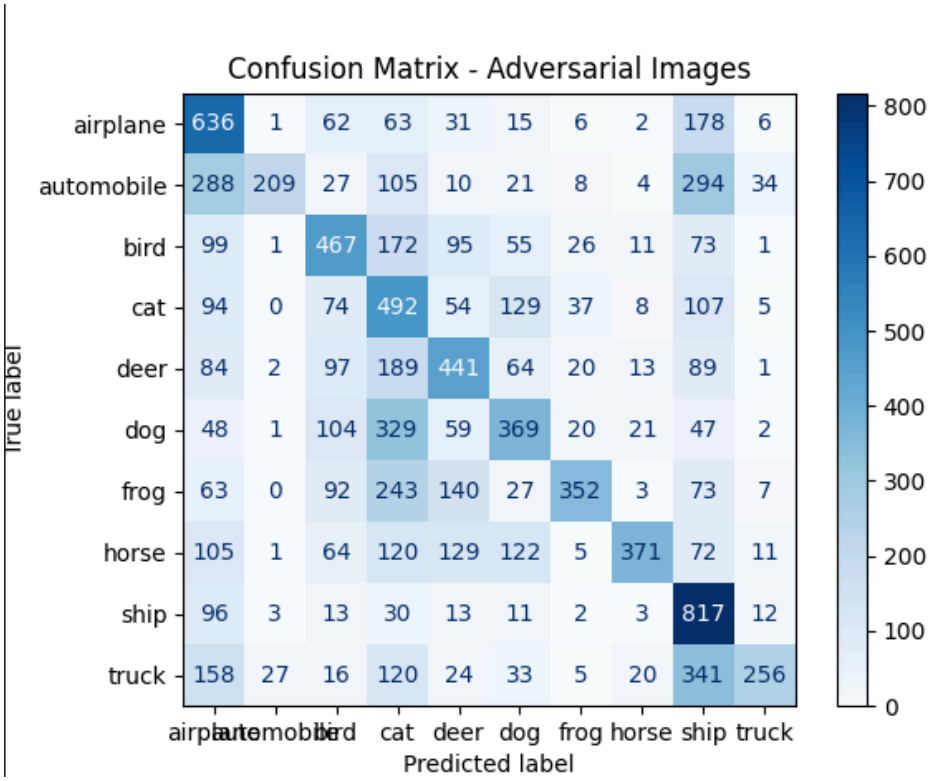
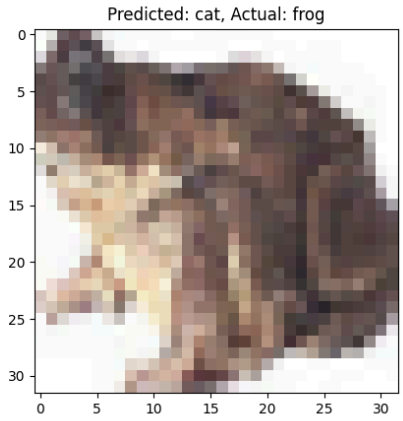
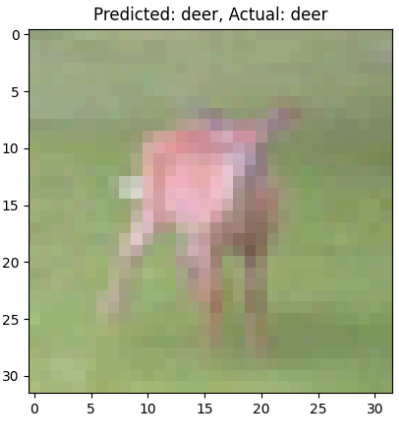


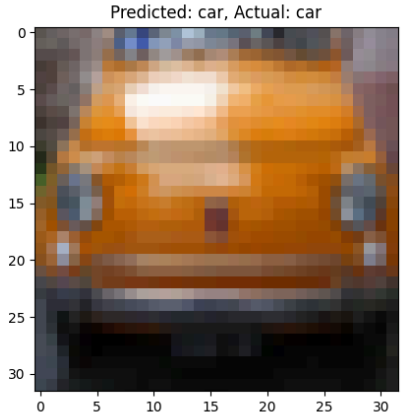
Figure 27: Confusion Matrix of Image Recognition Model with Adversarial Data

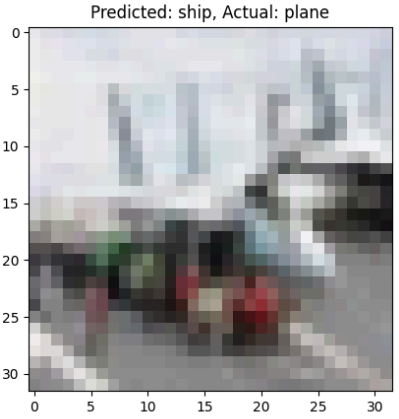
RESULTS

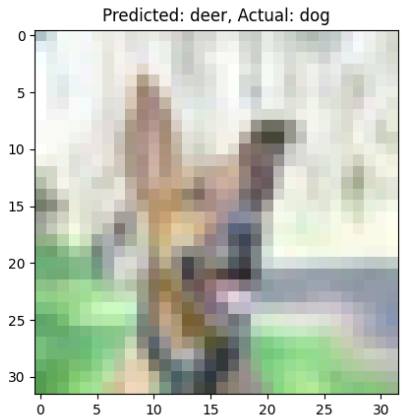
- Some test data visuals with predicted and actual values printed:

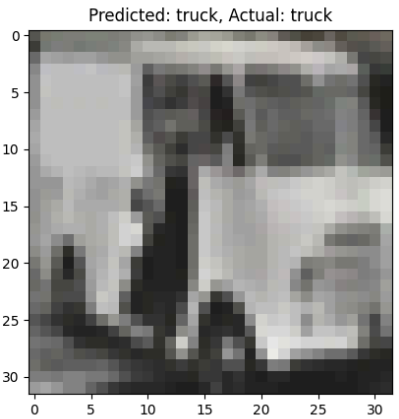




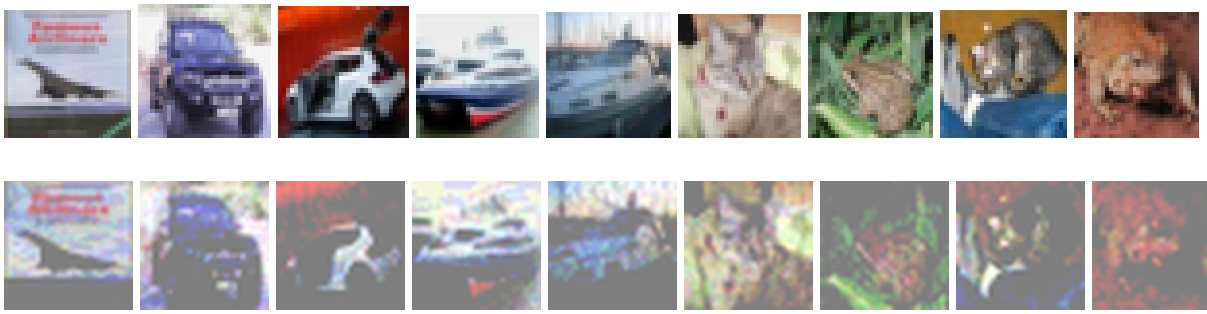


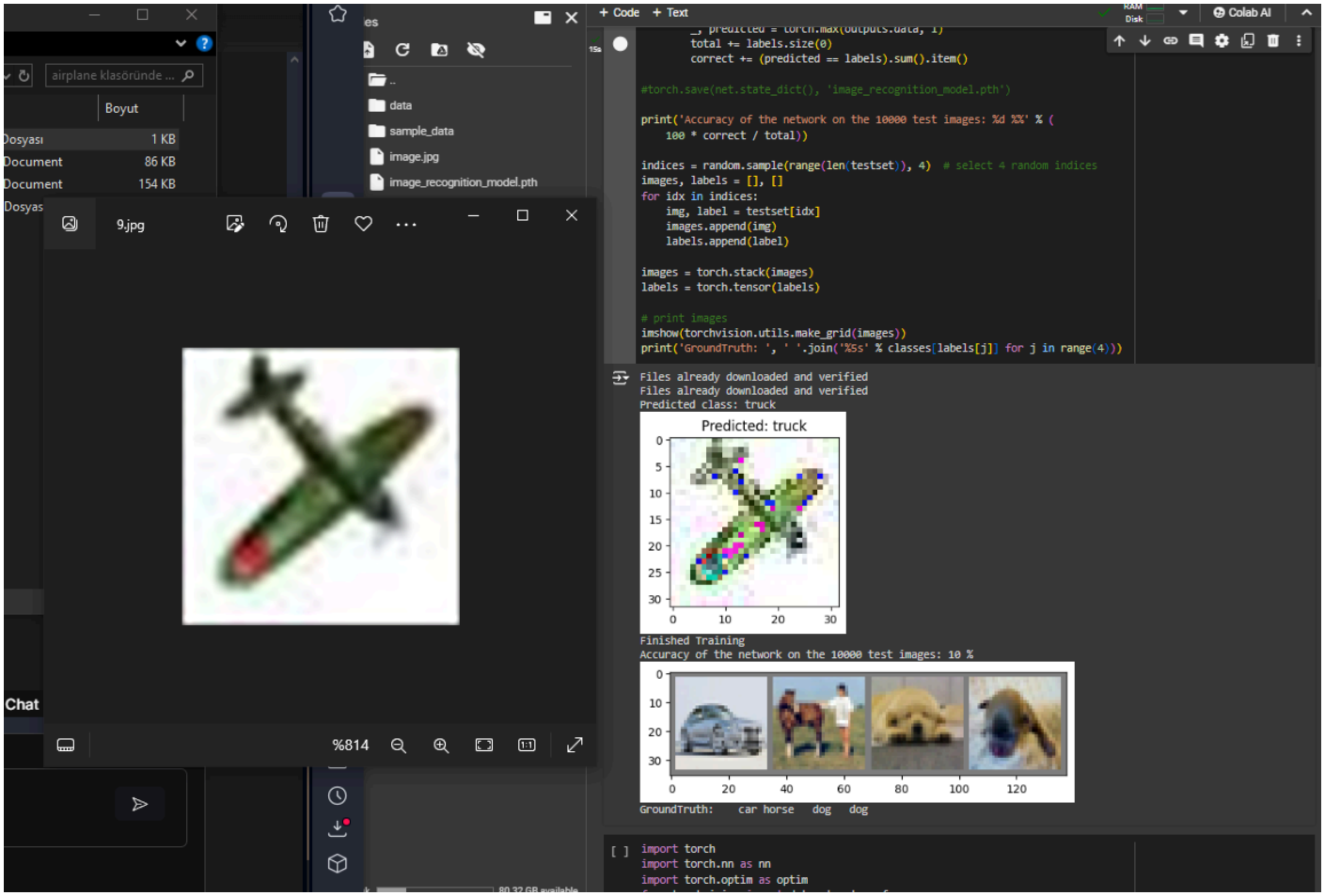


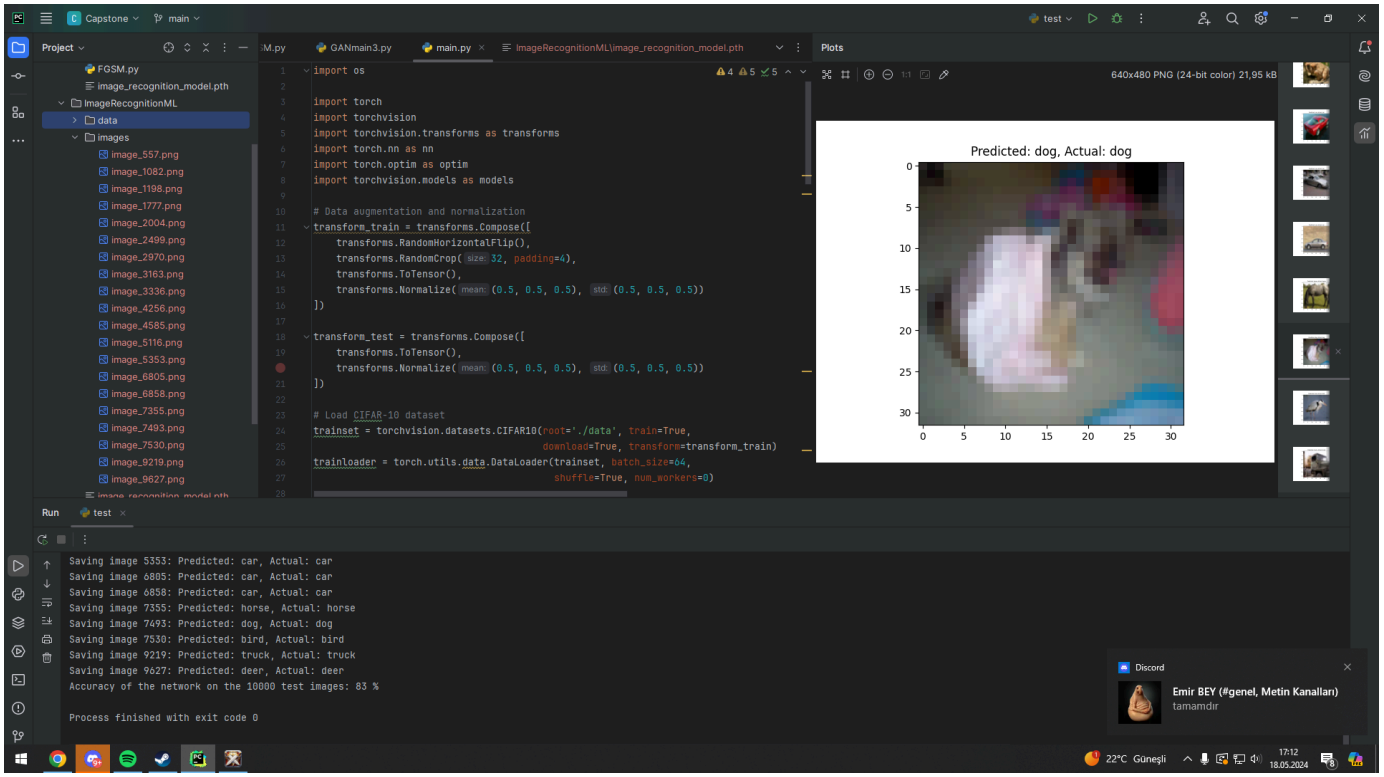


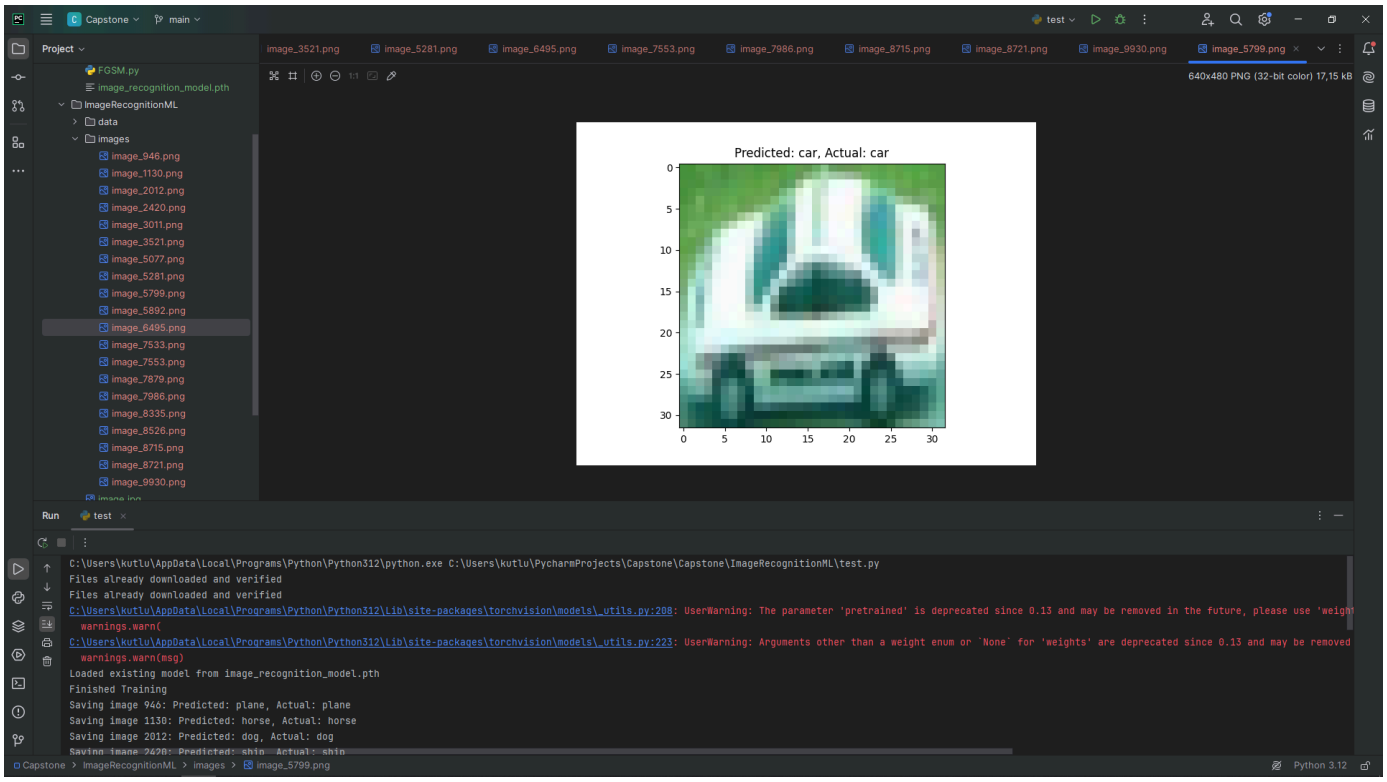


- Initial CIFAR-10 data with FGSM method perturbation applied(High epsilon value to show):



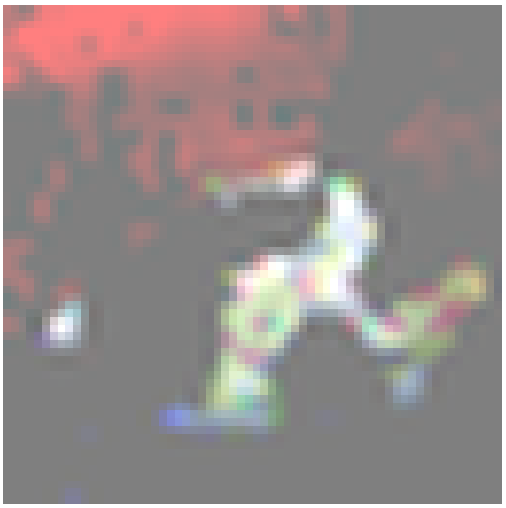






- The images below showcase how the epsilon value of the FGSM attack changes the visibility of perturbations and their effects.

a) Index = 6



Original prediction: 1 (car)
Adversarial prediction: 5 (dog)
epsilon=0.2

Original prediction: 1 (car)
Adversarial prediction: 5 (dog)
epsilon = 0.1

Original prediction: 1 (car)
Adversarial prediction: 5 (dog)
epsilon = 0.07

Original prediction: 1 (car)
Adversarial prediction: 0 (plane)
epsilon = 0.007

b) Index = 9

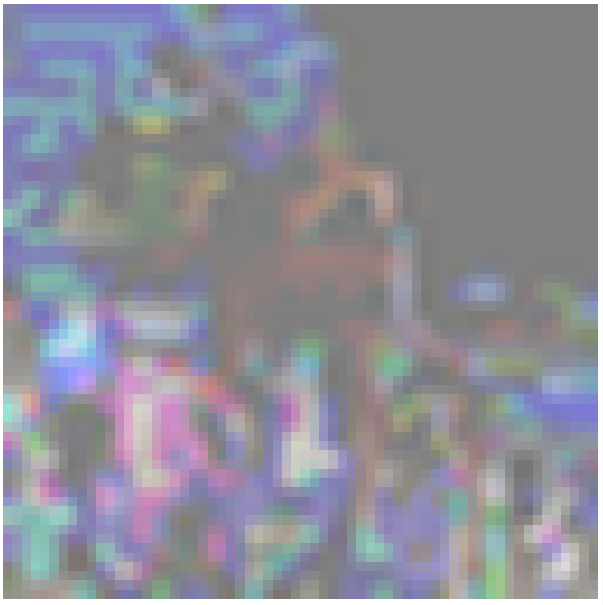
Original prediction: 1 (car)
Adversarial prediction: 0 (plane)
epsilon = 0.007

Original prediction: 1 (car)
Adversarial prediction: 0 (plane)
epsilon = 0.07

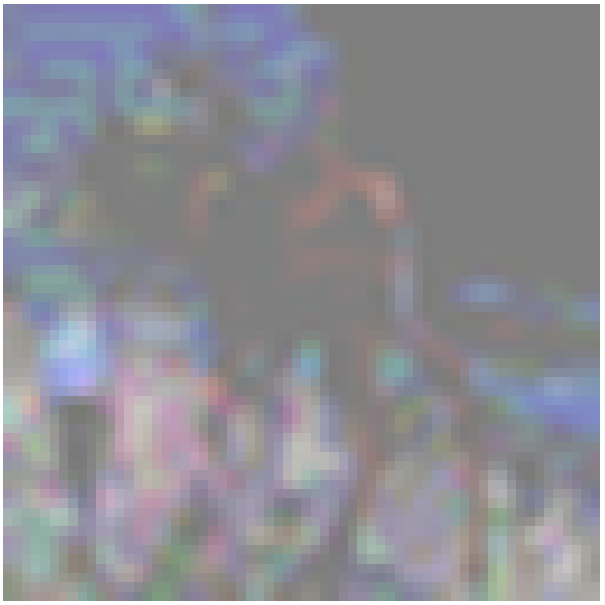
Original prediction: 1 (car)
Adversarial prediction: 0 (plane)
epsilon = 0.1

Original prediction: 1 (car)
Adversarial prediction: 0 (plane)
epsilon = 0.2

c) Index = 100



Original prediction: 5 (dog)
Adversarial prediction: 2 (bird)
epsilon = 0.2

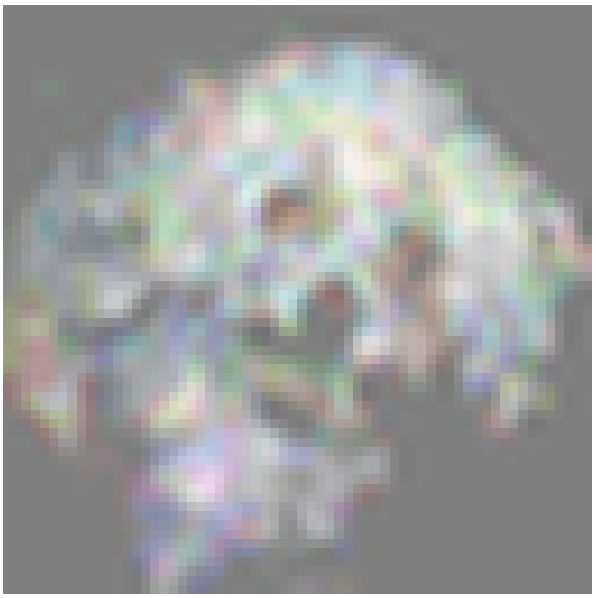


Original prediction: 5 (dog)
Adversarial prediction: 2 (bird)
epsilon = 0.1

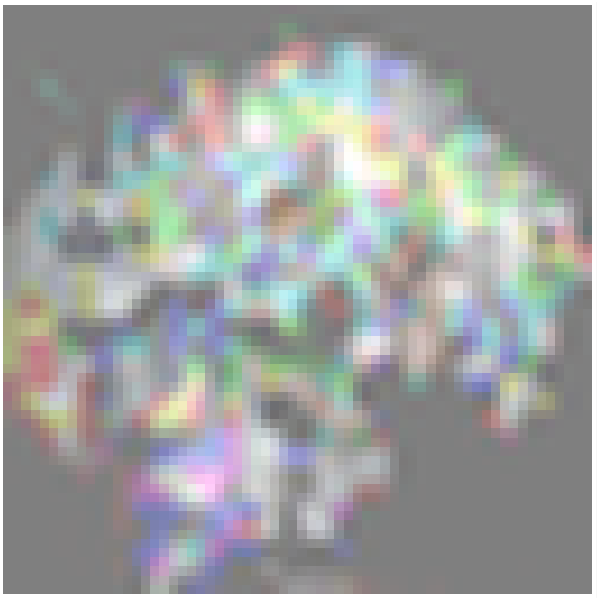
Original prediction: 5 (dog)
Adversarial prediction: 2 (bird)
epsilon = 0.07

Original prediction: 5 (dog)
Adversarial prediction: 0 (plane)
epsilon = 0.007

d) Index = 1000



Original prediction: 5 (dog)
Adversarial prediction: 8 (ship)
epsilon = 0.1



Original prediction: 5 (dog)
Adversarial prediction: 8 (ship)
epsilon = 0.2

Original prediction: 5 (dog)
Adversarial prediction: 5 (dog)
epsilon = 0.007

Original prediction: 5 (dog)
Adversarial prediction: 8 (ship)
epsilon = 0.07

- Tests on image recognition model

```
Saving image 6074: Predicted: deer, Actual: deer
Saving image 6352: Predicted: bird, Actual: bird
Saving image 6548: Predicted: plane, Actual: plane
Saving image 6914: Predicted: ship, Actual: ship
Saving image 7280: Predicted: car, Actual: car
Saving image 8246: Predicted: ship, Actual: ship
Saving image 8650: Predicted: bird, Actual: bird
Saving image 9730: Predicted: cat, Actual: frog
Accuracy of the network on the 10000 test images: 84 %
```

In the image above, we see how the accuracy of the image recognition model

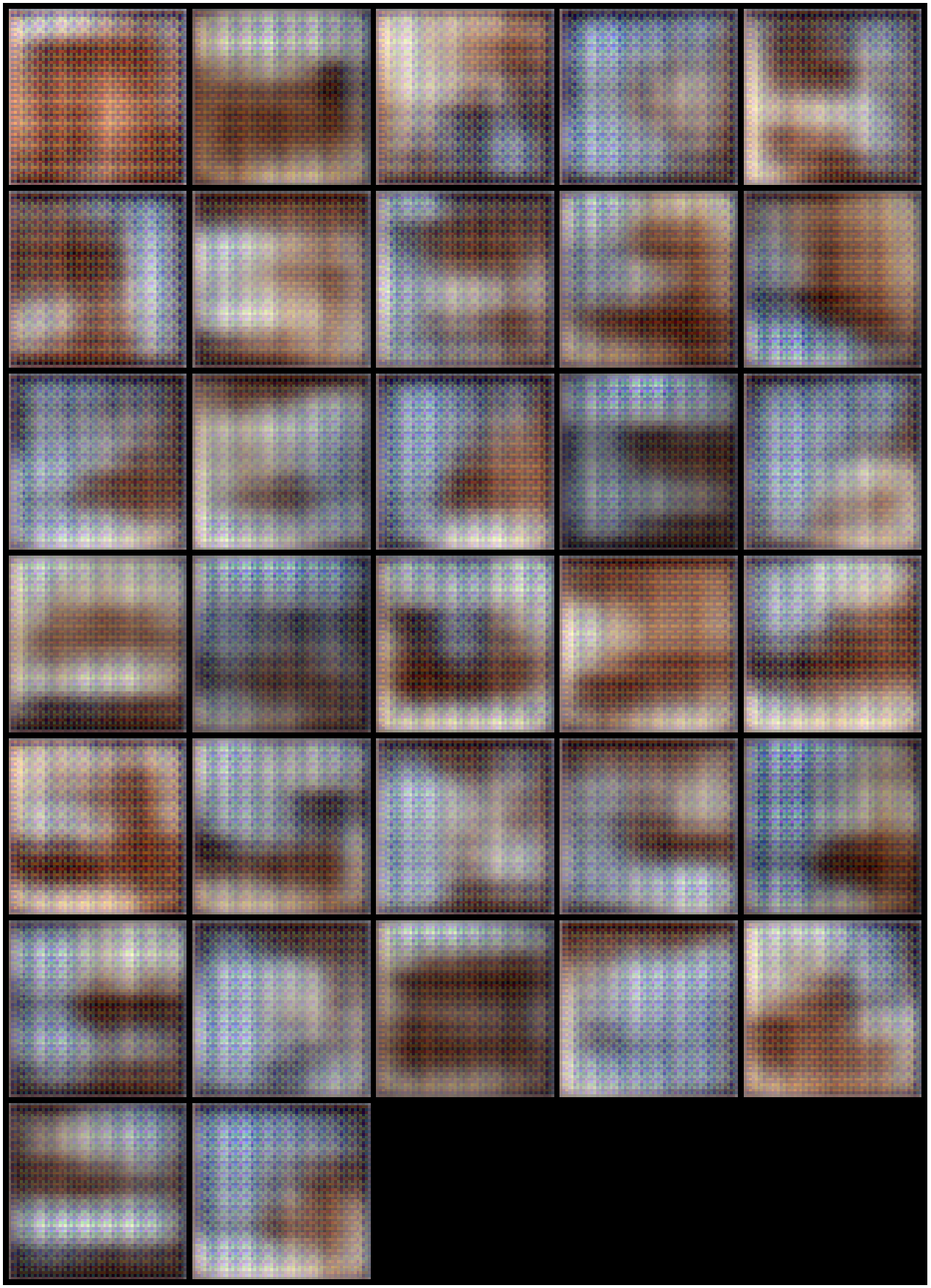
```
Image 99/100
Original prediction: airplane
Adversarial prediction: airplane
Image 100/100
Original prediction: horse
Adversarial prediction: dog

Accuracy on original images: 89.00%
Accuracy on adversarial images: 42.00%
```

In the image above, we see how the image recognition model behaves to the FGSM attack with the epsilon value 0.001

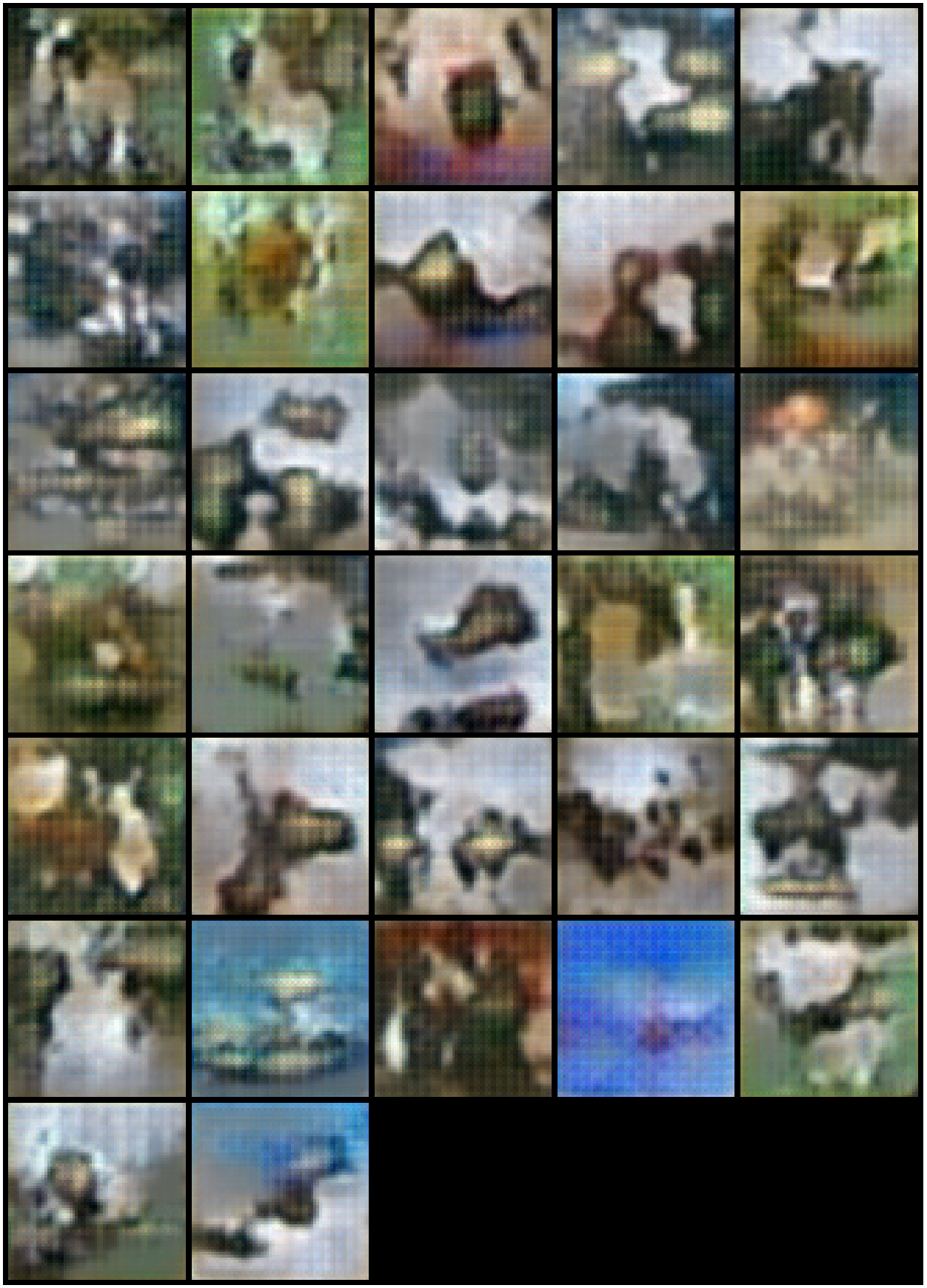
```
Image 99/100
Original prediction: airplane
Adversarial prediction: airplane
Image 100/100
Original prediction: horse
Adversarial prediction: dog

Accuracy on original images: 89.00%
Accuracy on adversarial images: 69.00%
```

In the image above, we see how the image recognition model behaves to the FGSM attack with the epsilon value 0.01

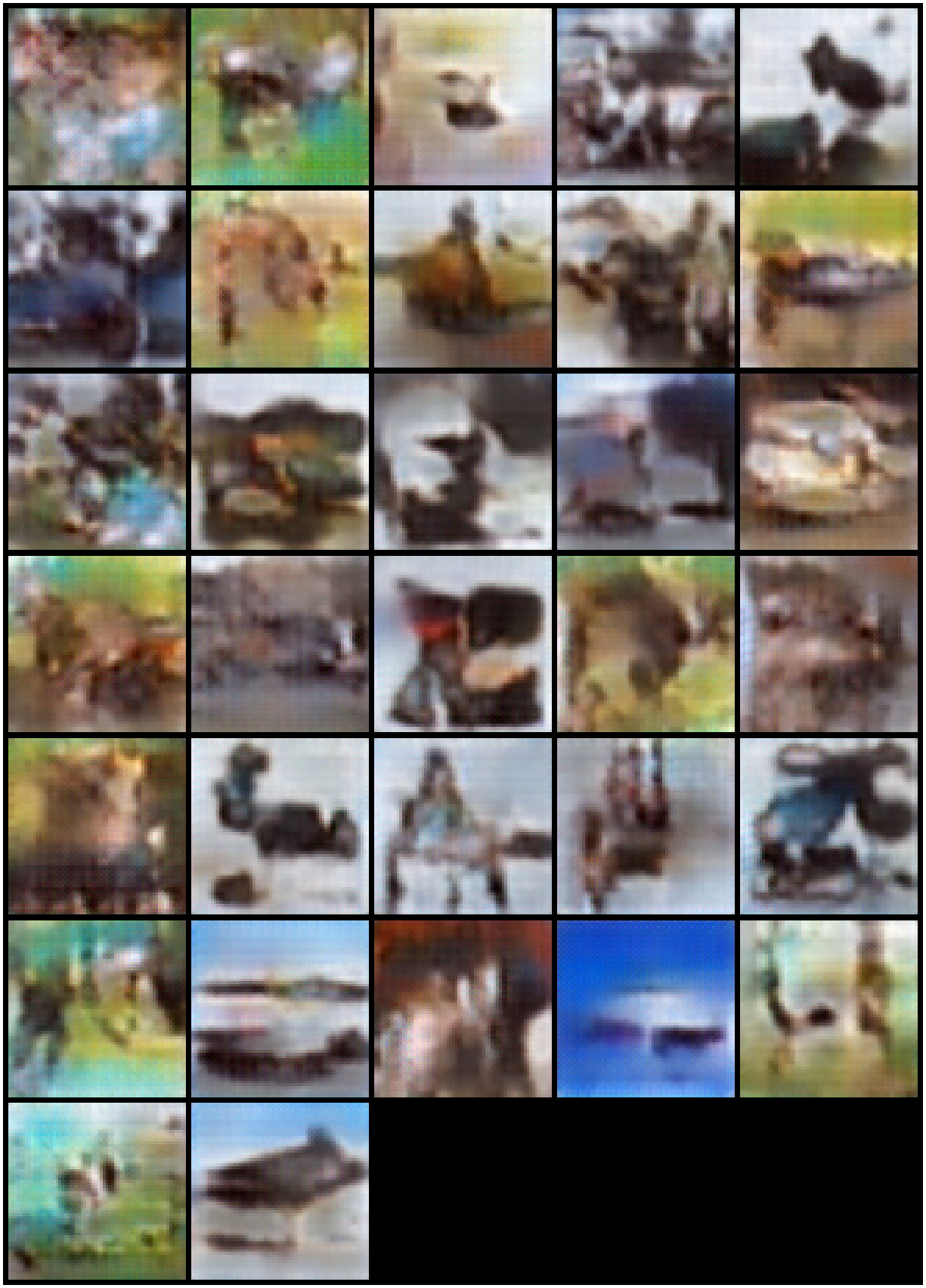
- In the images below we can see how the gan module generates the images and classifies.



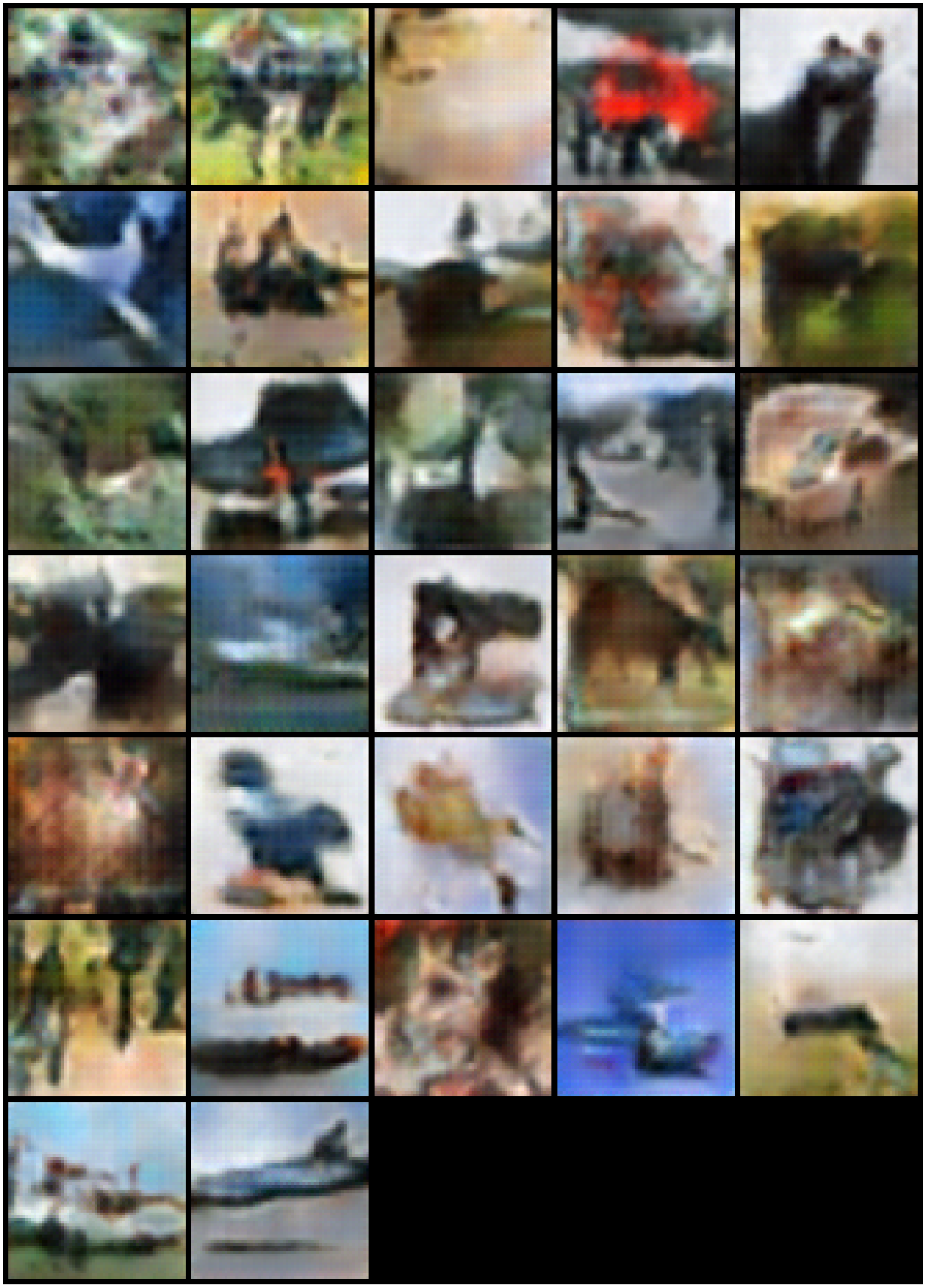
Here, we see the first 32 images that are generated with GAN (epoch 0).



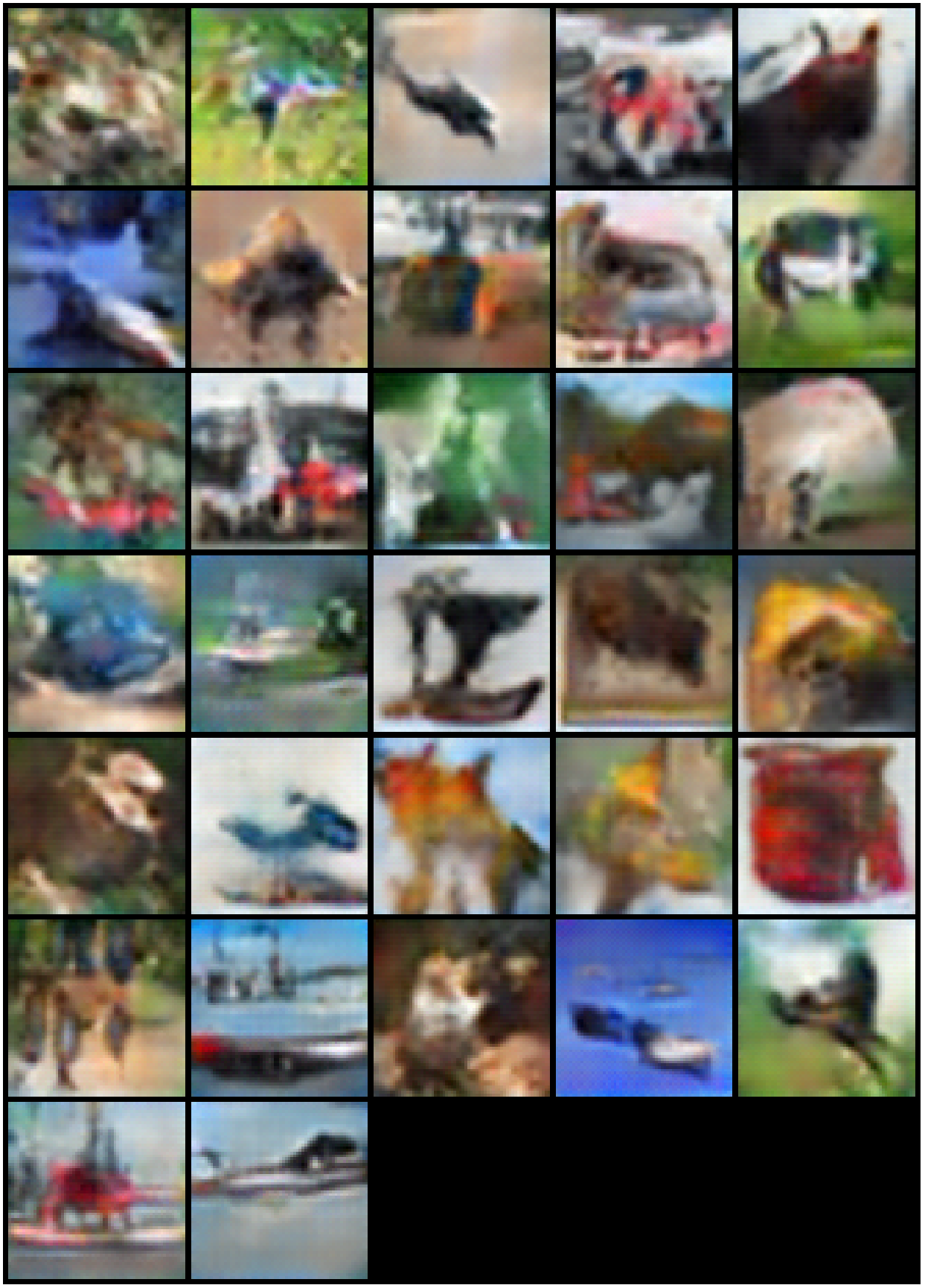
Here we see the images from Epoch 14.



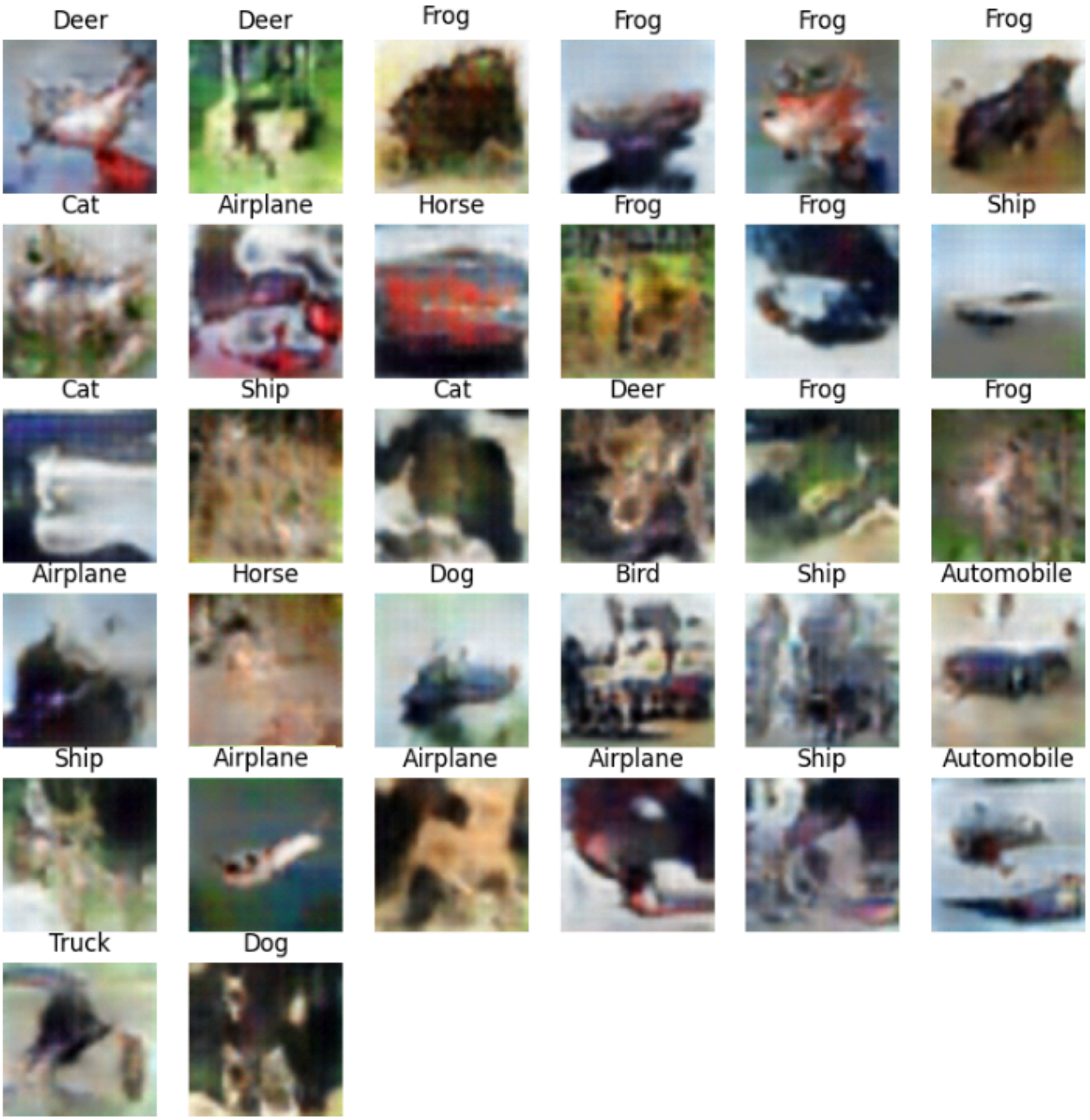
Here we see the images from epoch 25.



Here we see the images from Epoch 36.



Here we see the images from the 49th Epoch on a single run. even though the images doesn't seem like real ones, with training long enough the output of the generator creates more real like images.



This is also another 50 epoch trained GAN outputs.

ACKNOWLEDGEMENTS

We wish to thank our adviser, **Assist. Prof. Ece Gelal Soyak** for helping us through this study.

REFERENCES

1. [1] Selfridge, O. G. . *Pattern recognition and modern computers*. https://dl.acm.org/doi/abs/10.1145/1455292.1455310 (1955).
2. [2] Biggio, B., & Roli, F. Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning. https://doi.org/10.1145/3243734.3264418 (2018).
3. [3] Serban, A. C., Poll, E., & Visser, J. Adversarial Examples - A Complete Characterisation of the Phenomenon. arXiv preprint arXiv:1810.01185. https://arxiv.org/abs/1810.01185. (2019, February).
4. [4] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. Generative Adversarial Networks: An Overview. arXiv preprint arXiv:1710.07035. https://arxiv.org/abs/1710.07035 (2017).
5. [5] Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (2009).
6. [6] Krizhevsky, A., Sutskever, I., & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems). https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (2012, January).
7. [7] O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. Stanford University. arXiv preprint arXiv:1409.0575. https://arxiv.org/abs/1409.0575 (2015, January).
8. [8] Alhajjar, E., Maxwell, P., & Bastian, N. D. Adversarial Machine Learning in Network Intrusion Detection Systems. arXiv preprint arXiv:2004.11898. https://arxiv.org/abs/2004.11898 (2020, April).
9. [9] Goodfellow, I. J., Shlens, J., & Szegedy, C. Explaining and Harnessing Adversarial Examples. arXiv preprint arXiv:1412.6572. https://arxiv.org/abs/1412.6572 (2015).
10. [10] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., & Swami, A. The Limitations of Deep Learning in Adversarial Settings. arXiv preprint arXiv:1511.07528. https://arxiv.org/abs/1511.07528 (2015, November).
11. [11] Sheatsley, R., McDaniel, P., Papernot, N., Weisman, M. J., & Verma, G. Adversarial Examples in Constrained Domains. arXiv preprint arXiv:2011.01183. https://arxiv.org/abs/2011.01183 (2022, September).
12. [12] Moosavi-Dezfooli, S. M., Fawzi, A., & Frossard, P. DeepFool: a simple and accurate method to fool deep neural networks. arXiv preprint arXiv:1511.04599. https://arxiv.org/abs/1511.04599 (2016, July).
13. [13] Carlini, N., & Wagner, D. Towards Evaluating the Robustness of Neural Networks.

arXiv preprint arXiv:1608.04644. https://arxiv.org/abs/1608.04644 (2017, March).

14. [14] Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. The Space of Transferable Adversarial Examples. arXiv preprint arXiv:1704.03453. https://arxiv.org/abs/1704.03453 (2017, May).
15. [15] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. Intriguing Properties of Neural Networks. arXiv preprint arXiv:1312.6199. https://arxiv.org/abs/1312.6199 (2014, February).
16. [16] Google Brain. TensorFlow: A system for large-scale machine learning. arXiv preprint arXiv:1605.08695. https://arxiv.org/abs/1605.08695 (2016, May)
17. [17] Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv preprint arXiv:1912.01703. https://arxiv.org/abs/1912.01703 (2019, December).
18. [18] Wu, Y. et al. A Comparative Measurement Study of Deep Learning as a Service Framework. arXiv preprint arXiv:1810.12210. https://arxiv.org/abs/1810.12210 (2019, August).
19. [19] Goodfellow, I. J. et al. Generative Adversarial Nets. Département d'informatique et de recherche op´ rationnelle. https://arxiv.org/abs/1406.2661 (2014, June)
20. [20] Dash, A., Ye, J., & Wang, G. A review of Generative Adversarial Networks (GANs) and its applications in a wide variety of disciplines - From Medical to Remote Sensing. Department of Computer Science, New Jersey Institute of Technology, USA. https://arxiv.org/pdf/2110.01442.pdf (2021, October)

APPENDIX

GAN MODULES

```
import os
import torch
from torch import nn, optim
from torch.autograd import Variable
from torchvision import datasets, transforms, models
import torchvision.utils as vutils
import matplotlib.pyplot as plt
import numpy as np
import progressbar

class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog',
'Frog', 'Horse', 'Ship', 'Truck']

OUTPUT_DIR = './data/CIFAR-10'
MODEL_DIR = './models'
IMAGE_SAVE_DIR = './generated_images'
BATCH_SIZE = 100
LR = 0.001
NUM_EPOCHS = 1
NUM_TEST_SAMPLES = 32

os.makedirs(MODEL_DIR, exist_ok=True)
os.makedirs(IMAGE_SAVE_DIR, exist_ok=True)


def load_data():
    compose = transforms.Compose([
        transforms.Resize(64),
        transforms.ToTensor(),
        transforms.Normalize((.5, .5, .5), (.5, .5, .5))
    ])
    return datasets.CIFAR10(root=OUTPUT_DIR, train=False,
transform=compose, download=True)


data = load_data()
data_loader = torch.utils.data.DataLoader(data, batch_size=BATCH_SIZE,
shuffle=True)
NUM_BATCHES = len(data_loader)

print("No. of Batches =", NUM_BATCHES)


def noise(size, num_classes):
    n = Variable(torch.randn(size, 100)).to(device)
    labels = np.random.randint(0, num_classes, size)
    one_hot_labels = np.zeros((size, num_classes))
    one_hot_labels[np.arange(size), labels] = 1
    one_hot_labels = torch.tensor(one_hot_labels,
dtype=torch.float32).to(device)
    noise_with_labels = torch.cat((n, one_hot_labels), dim=1)
    return noise_with_labels, labels
```

```
def init_weights(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1 or classname.find('BatchNorm') != -1:
        m.weight.data.normal_(0.00, 0.02)


def real_data_target(size):
    data = Variable(torch.ones(size, 1)).to(device)
    return data


def fake_data_target(size):
    data = Variable(torch.zeros(size, 1)).to(device)
    return data


def train_discriminator(optimizer, real_data, fake_data):
    optimizer.zero_grad()
    prediction_real = discriminator(real_data)
    error_real = criterion(prediction_real,
real_data_target(real_data.size(0)))
    error_real.backward()
    prediction_fake = discriminator(fake_data)
    error_fake = criterion(prediction_fake,
fake_data_target(real_data.size(0)))
    error_fake.backward()
    optimizer.step()
    return error_real + error_fake, prediction_real, prediction_fake


def train_generator(optimizer, fake_data):
    optimizer.zero_grad()
    prediction = discriminator(fake_data)
    error = criterion(prediction, real_data_target(prediction.size(0)))
    error.backward()
    optimizer.step()
    return error


class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=128, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.LeakyReLU(0.2, inplace=True))
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(256), nn.LeakyReLU(0.2, inplace=True))
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=4,
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(512), nn.LeakyReLU(0.2, inplace=True))
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=4,
```

```
stride=2, padding=1, bias=False),
            nn.BatchNorm2d(1024), nn.LeakyReLU(0.2, inplace=True))
        self.output = nn.Sequential(nn.Linear(1024 * 4 * 4, 1),
nn.Sigmoid())

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = x.view(-1, 1024 * 4 * 4)
        x = self.output(x)
        return x


class Generator(nn.Module):
    def __init__(self, noise_dim, num_classes):
        super(Generator, self).__init__()
        self.linear = nn.Linear(noise_dim + num_classes, 1024 * 4 * 4)
        self.deconv1 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=1024, out_channels=512,
kernel_size=4, stride=2, padding=1, bias=False),
            nn.BatchNorm2d(512), nn.ReLU(inplace=True))
        self.deconv2 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=512, out_channels=256,
kernel_size=4, stride=2, padding=1, bias=False),
            nn.BatchNorm2d(256), nn.ReLU(inplace=True))
        self.deconv3 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=256, out_channels=128,
kernel_size=4, stride=2, padding=1, bias=False),
            nn.BatchNorm2d(128), nn.ReLU(inplace=True))
        self.deconv4 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=128, out_channels=3,
kernel_size=4, stride=2, padding=1, bias=False))
        self.output = nn.Tanh()

    def forward(self, x):
        x = self.linear(x)
        x = x.view(x.shape[0], 1024, 4, 4)
        x = self.deconv1(x)
        x = self.deconv2(x)
        x = self.deconv3(x)
        x = self.deconv4(x)
        return self.output(x)


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

noise_dim = 100
num_classes = len(class_names)
generator = Generator(noise_dim, num_classes).to(device)
discriminator = Discriminator().to(device)
generator.apply(init_weights)
discriminator.apply(init_weights)

d_optimizer = optim.Adam(discriminator.parameters(), lr=LR, betas=(0.5,
0.999))
g_optimizer = optim.Adam(generator.parameters(), lr=LR, betas=(0.5,
```

```
0.999))
criterion = nn.BCELoss()

test_noise, test_labels = noise(NUM_TEST_SAMPLES, num_classes)


def generate_and_save_images(generator, test_noise, test_labels, epoch,
save_dir, class_names,
                             num_images=NUM_TEST_SAMPLES):
    with torch.no_grad():
        test_images = generator(test_noise).cpu()

    rows = int(np.ceil(np.sqrt(num_images)))
    fig, axes = plt.subplots(rows, rows, figsize=(10, 10))
    fig.suptitle(f"Generated Images at Epoch {epoch}", fontsize=16)

    for i in range(num_images):
        image_tensor = test_images[i]
        class_name = class_names[test_labels[i]]  # Use the provided class
labels
        np_image = image_tensor.numpy().transpose((1, 2, 0))

        np_image = (np_image - np_image.min()) / (np_image.max() -
np_image.min())

        ax = axes[i // rows, i % rows]
        ax.imshow(np_image)
        ax.set_title(f"{class_name}")
        ax.axis('off')

        image_name = f'epoch_{epoch}_image_{i}_class_{class_name}.png'
        image_path = os.path.join(save_dir, image_name)
        plt.imsave(image_path, np_image)
        print(f'Saved image at {image_path}')

    for j in range(i + 1, rows * rows):
        fig.delaxes(axes[j // rows, j % rows])

    combined_image_path = os.path.join(save_dir,
f'epoch_{epoch}_combined.png')
    plt.savefig(combined_image_path)
    plt.show()
    plt.close()
    print(f'Saved combined image grid at {combined_image_path}')


def save_models(generator, discriminator):
    torch.save(generator.state_dict(), os.path.join(MODEL_DIR,
'generator.pth'))
    torch.save(discriminator.state_dict(), os.path.join(MODEL_DIR,
'discriminator.pth'))


def load_models(generator, discriminator):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    generator.load_state_dict(torch.load(os.path.join(MODEL_DIR,
'generator.pth'), map_location=device))
    discriminator.load_state_dict(torch.load(os.path.join(MODEL_DIR,
```

```
'discriminator.pth'), map_location=device))

    generator.to(device)
    discriminator.to(device)


if os.path.exists(os.path.join(MODEL_DIR, 'generator.pth')) and
os.path.exists(
        os.path.join(MODEL_DIR, 'discriminator.pth')):
    load_models(generator, discriminator)

for epoch in range(NUM_EPOCHS):
    print(f"\nEpoch #{epoch} in progress...")
    progress_bar = progressbar.ProgressBar()
    d_running_loss = 0
    g_running_loss = 0

    for n_batch, (real_batch, _) in enumerate(progress_bar(data_loader)):
        real_data = Variable(real_batch).to(device)
        fake_noise, _ = noise(real_data.size(0), num_classes)
        fake_data = generator(fake_noise).detach()
        d_error, d_pred_real, d_pred_fake =
train_discriminator(d_optimizer, real_data, fake_data)
        fake_noise, _ = noise(real_batch.size(0), num_classes)
        fake_data = generator(fake_noise)
        g_error = train_generator(g_optimizer, fake_data)
        d_running_loss += d_error.item()
        g_running_loss += g_error.item()

    print(f"Loss (Discriminator): {d_running_loss}")
    print(f"Loss (Generator): {g_running_loss}")

    generate_and_save_images(generator, test_noise, test_labels, epoch,
IMAGE_SAVE_DIR, class_names)

    save_models(generator, discriminator)
```

FGSM Attack

GAN Module has FGSM technique in it as well, this code is to test FGSM separately and implement the perturbation to CIFAR-10 datas.

```
import os
import torchvision.models as models
import torch
import torch.nn as nn
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np

net = models.resnet18(weights=False, num_classes=10)
net.load_state_dict(torch.load('image_recognition_model.pth',
map_location=torch.device('cpu')))
net.eval()

def fgsm_attack(data, epsilon, data_grad):
    sign_data_grad = data_grad.sign()
    perturbed_data = data + epsilon * sign_data_grad
```

```
    perturbed_data = torch.clamp(perturbed_data, 0, 1)
    return perturbed_data

def generate_fgsm_adversarial_example(model, data, target, epsilon):
    data_copy = data.clone().detach().requires_grad_(True)
    output = model(data_copy)
    loss = nn.CrossEntropyLoss()(output, target)
    model.zero_grad()
    loss.backward()
    data_grad = data_copy.grad.data
    perturbed_data = fgsm_attack(data_copy, epsilon, data_grad)
    return perturbed_data

def save_image_with_classification(image_data, classification, filepath):
    image = np.transpose(image_data.squeeze(0).detach().numpy(), (1, 2,
0))
    image = np.clip(image, 0, 1)
    plt.imshow(image)
    plt.title(classification)
    plt.axis('off')
    plt.savefig(filepath)
    plt.close()

class_names = datasets.CIFAR10(root='./data', train=True,
download=True).classes

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
test_data = datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
test_loader = DataLoader(test_data, batch_size=1, shuffle=False)

epsilon = 0.01
num_images = 100

output_dir = 'perturbed_images'
os.makedirs(output_dir, exist_ok=True)

correct_original = 0
correct_adversarial = 0

for i, (data, target) in enumerate(test_loader):
    if i >= num_images:
        break

    data, target = data.to('cpu'), target.to('cpu')

    data_copy = data.clone().detach()

    output = net(data_copy)
    original_pred = output.argmax(dim=1, keepdim=True)
    correct_original +=
original_pred.eq(target.view_as(original_pred)).sum().item()

    adversarial_data = generate_fgsm_adversarial_example(net, data_copy,
target, epsilon)
```

```
    output = net(adversarial_data)
    adversarial_pred = output.argmax(dim=1, keepdim=True)
    correct_adversarial +=
adversarial_pred.eq(target.view_as(adversarial_pred)).sum().item()

    original_class = class_names[original_pred.item()]
    adversarial_class = class_names[adversarial_pred.item()]

    image_dir = os.path.join(output_dir, f'image{i}')
    os.makedirs(image_dir, exist_ok=True)

    original_image_path = os.path.join(image_dir, f'image{i}.png')
    adversarial_image_path = os.path.join(image_dir,
f'image{i}_perturbed.png')

    save_image_with_classification(data, f'Original: {original_class}',
original_image_path)
    save_image_with_classification(adversarial_data, f'Adversarial:
{adversarial_class}', adversarial_image_path)

    print(f'Image {i+1}/{num_images}')
    print('Original prediction:',  original_class)
    print('Adversarial prediction:', adversarial_class)

accuracy_original = correct_original / num_images
accuracy_adversarial = correct_adversarial / num_images

print(f'\nAccuracy on original images: {accuracy_original * 100:.2f}%')
print(f'Accuracy on adversarial images: {accuracy_adversarial *
100:.2f}%')
```

Image Recognition Model

```
import os

import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import torchvision.models as models

transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True,
```

```
transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True,
transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                         shuffle=False, num_workers=0)

net = models.resnet18(weights=False, num_classes=10)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)

model_path = 'image_recognition_model.pth'
if os.path.exists(model_path):
   net.load_state_dict(torch.load(model_path))
   print("Loaded existing model from", model_path)
else:
   print("No existing model found, starting training from scratch")

num_epochs = 100
train_losses = []
val_losses = []

for epoch in range(num_epochs):
   net.train()
   running_loss = 0.0
   for inputs, labels in trainloader:
       optimizer.zero_grad()
       outputs = net(inputs)
       loss = criterion(outputs, labels)
       loss.backward()
       optimizer.step()
       running_loss += loss.item()

   train_losses.append(running_loss / len(trainloader))

   net.eval()
   val_loss = 0.0
   correct = 0
   total = 0
   with torch.no_grad():
       for inputs, labels in testloader:
           outputs = net(inputs)
           loss = criterion(outputs, labels)
           val_loss += loss.item()
           _, predicted = torch.max(outputs.data, 1)
           total += labels.size(0)
           correct += (predicted == labels).sum().item()
   val_losses.append(val_loss / len(testloader))

   scheduler.step()

   print(
       f'Epoch {epoch + 1}, Train Loss: {train_losses[-1]}, Val Loss:
```

```
{val_losses[-1]}, Accuracy: {100 * correct / total:.2f}%')

print('Finished Training')

torch.save(net.state_dict(), 'image_recognition_model.pth')

correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
        100 * correct / total))
```

CONFUSION MATRIX TEST

```
import os
import random

import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import torchvision.models as models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def main():
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform_test)
    testloader = torch.utils.data.DataLoader(testset, batch_size=64,
shuffle=False, num_workers=0)

    classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck')

    net = models.resnet18(pretrained=False, num_classes=10)

    model_path = 'image_recognition_model.pth'
    if os.path.exists(model_path):
        net.load_state_dict(torch.load(model_path))
        print("Loaded existing model from", model_path)
    else:
        print("No existing model found. Please train the model first.")
        return
```

```
    net.eval()

    all_labels = []
    all_preds = []

    with torch.no_grad():
        for data in testloader:
            images, labels = data
            outputs = net(images)
            _, predicted = torch.max(outputs, 1)
            all_labels.extend(labels.numpy())
            all_preds.extend(predicted.numpy())

    cm = confusion_matrix(all_labels, all_preds,
labels=list(range(len(classes))))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=classes)
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix of Image Recognition Model')
    plt.show()

if __name__ == '__main__':
    main()
```

TEST function to see classifications

```
def imshow(img, predicted_label, actual_label, index):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.title(f'Predicted: {predicted_label}, Actual: {actual_label}')
    plt.savefig(f'images/image_{index}.png')
    plt.show()
    plt.close()
```