In the Name of God

University of Tehran

Faculty of Electrical and Computer Engineering

Neural Networks and Deep Learning

Assignment 5

c

Assignment Details	
Designer TA (Q1)	Arian Firoozi
Email	arianfiroozim@gmail.com
Designer TA (Q2)	Mohammad Gorji
Email	mohamadgorjicode@gmail.com
Submission Deadline	1404/03/18 (June 8, 2025)

Submission Rules

- **Report Format:** Prepare your report in the format provided on the Elearn system, named `REPORTS_TEMPLATE.docx`.
- **Group Work:** It is recommended to complete the assignments in groups of two. (More than two people are not allowed, and individual submissions do not receive extra credit). Note that group members do not need to remain the same throughout the semester.
- **Report Quality:** The quality of your report is highly important in the grading process. Therefore, please include all the points and assumptions you considered in your implementations and calculations.
- **Code Details:** It is not necessary to provide detailed explanations of the code in the report, but the results obtained from it must be reported and analyzed.
- **Result Analysis:** Analysis of the results is mandatory, even if not explicitly asked for in the question.
- **Code Execution:** TAs are not required to run your code. Therefore, any results or analyses requested in the questions must be clearly and completely presented in the report. Failure to comply will result in a deduction of points.
- **Code Submission:** Codes must be submitted in a Jupyter Notebook with the `.ipynb` extension. At the end of the work, all code must be executed, and the output of each cell must be saved in the submitted file.
- **Academic Integrity:** In case of plagiarism, the score of all involved parties will be penalized by -100.
- **Programming Language:** The only authorized programming language is Python.
- **Use of Pre-written Code:** Using pre-written code for the assignments is strictly forbidden. If two groups use a common source and submit similar codes, it will be considered plagiarism.
- **Late Submission Policy:** After the submission deadline, there is a maximum of one week for late submission. After this one week, the score for that assignment will be zero. Penalties are as follows:
  - First three days: No penalty.
  - Day 4: 5% penalty.
  - Day 5: 10% penalty.
  - Day 6: 15% penalty.
  - Day 7: 20% penalty.
- **File Naming:** Please place the report, codes, and other attachments in a folder with the following name, compress it, and then upload it to the Elearn system: `HW[Number]_[Lastname]_[StudentNumber]_[Lastname]_[StudentNumber].zip`.

1 Question 1: Image Classification with ViT (100 Points)

1.1 Introduction (10 Points)

In agriculture, the timely diagnosis of diseases and pests in plants plays a crucial role in the health and productivity of crops. In this exercise, we want to use two types of neural networks to classify diseases using plant leaves.

- **Vision Transformer (ViT):** Models that use a transformer architecture and attention mechanism on image datasets.
- **Convolutional Neural Network (CNN):** As you have become familiar with the workings of this type of neural network before, CNNs extract image features using convolutional layers.

The goal of this exercise is to become familiar with the structure of the Vision Transformer and its differences from other neural networks. For this purpose, you will use the dataset provided in the paper, which is available for convenience from this link. To complete the exercise, you need to study this paper.

In relation to the paper and Vision Transformers, answer the following questions:

- According to the paper, what is the main difference and advantage of using Vision Transformer models compared to traditional models? (5 points)
- When the available dataset is limited, which model performs better? Justify your answer based on the structure and mechanisms used in the models. (5 points)

1.2 Data Preparation (15 Points)

Receive the data and follow the steps below:

- Display a sample from each of the 10 classes present in the dataset. (2 points)
- Examine the balance of the dataset: Compare the number of images for each class in a table. Is the number of images in the dataset balanced? If your answer is no, argue what type of data augmentation can be used to balance the data without seriously damaging the quality of the photos, and apply it. (7 points)
- If you suggest another preprocessing step that could improve performance, apply it to the data. According to the input size of the models and the size suggested in the paper, the photos must be in specific dimensions. For this purpose, the internal structure of the CNN used (which is Inception-V3) must be changed, but since the main topic of the exercise is Vision Transformer, you can use a different size. (3 points)
- Divide your data into two sets: training and validation. (3 points)

1.3 CNN Model Training (20 Points)

- First, load the Inception-V3 model raw and without pre-training. Adjust the number of outputs according to the dataset and set other parameters according to the paper. Describe the overall functionality of this model. (7 points)
- Explain the loss function used in the paper. What other functions are suitable for this task? (3 points)
- Train the model for at least 10 epochs and plot the accuracy and loss graphs for both datasets. Plot the model's confusion matrix. (10 points)

1.4 ViT Model Training (40 Points)

- Implement the model described in the paper. It is necessary to display the output of the model's layers, and if a layer acts contrary to what is stated in the paper, state the reason. (20 points)
- For what purpose is the Patch Embedding layer used in image transformer networks? What effect does reducing or increasing the size of each patch have on the output in this exercise? (5 points)
- (Bonus) Implement a pixel output capability in the Patch Embedding layer and show the output of this layer as an image using it. (5 points)
- Train the model for at least 20 epochs and plot the accuracy and loss graphs for both datasets. Plot the model's confusion matrix. (10 points)

1.5 Analysis and Conclusion (15 Points)

Compare the results of the two methods with each other.

- Compare the models in terms of accuracy, precision, and the number of parameters. (8 points)
- Considering which model performed better, explain under what conditions the weaker model could have performed better. (7 points)

2 Question 2: Robust Zero-Shot Classification (100 Points)

Adversarial attacks on neural networks are one of the fascinating and controversial challenges in deep learning and trustworthy AI. In these attacks, samples are designed that are practically indistinguishable from the original samples to the human eye but can deceive machine learning models. These samples, called **adversarial examples**, are created with small, targeted changes to the input data and cause the model to produce an incorrect output. For example, adding subtle noise to an image can cause an image classification model to misidentify a stop sign as a yield sign. (Figure 1)

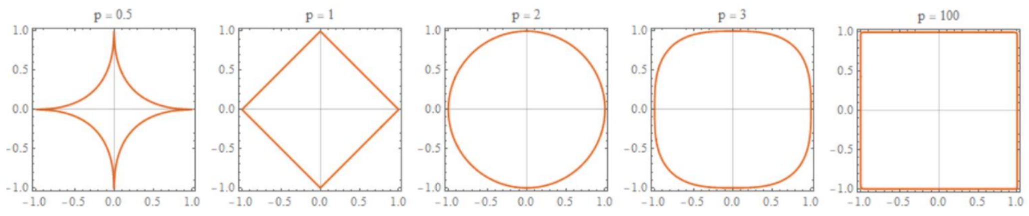


Figure 1: An AI model's mistake in identifying the class of an adversarial example.

2.1 Introduction to CLIP, One-Shot Classification, and Adversarial Attacks

First, study the paper and answer the following questions:

1. Explain the two methods for generating adversarial examples, FGSM and PGD, with details of their optimization. (6 points)
2. Describe the architecture of the CLIP model with a diagram and explain the loss function and the type of Contrastive training. (6 points)
3. Explain the key differences between normal classification and one-shot classification and, with a diagram, state how this is done with the CLIP model. (6 points)
4. There are two types of adversarial attacks in different scenarios: white-box and black-box. Explain each and then compare these two approaches from various desired aspects. (6 points)
5. Explain why transfer attacks are a serious threat for real-world problems compared to white-box attacks. (6 points)
6. Describe the LoRA fine-tuning method with a diagram and state three reasons for using this method over others. (10 points)
7. Find two research papers that extend or improve the CLIP loss function and summarize each in one or two paragraphs. Also, explain how the impact of these loss functions on increasing the robustness of the CLIP model has been. (10 points)

2.2 Implementation and Comparison of Adversarial Training Methods

In the second part of this project, you are to fully implement the training and evaluation process of one-shot CLIP models against adversarial attacks using the CIFAR-10 dataset and libraries such as PyTorch, torchvision, Transformers, and PEFT.

1. For this question, you must use the CIFAR-10 dataset. Download this data using the torchvision module and divide it into three sets: training, validation, and testing. Use a function to display 5 random samples from the dataset. Then, resize the images to 224x224 and normalize them with the mean and standard deviation specific to CLIP. (5 points)

2. After preparing the data, load the CLIP model from the HuggingFace repository and keep it in evaluation mode. Also, as a model for generating adversarial examples, download a pre-trained ResNet-20 model on CIFAR-10 from PyTorch Hub and also keep it in eval mode. Next, by generating text vectors for each of the ten CIFAR-10 classes (e.g., "a photo of a airplane", "a photo of an automobile", etc.), prepare the CLIP text space to be used for one-shot classification in the next steps. (5 points)

3. The next step is to evaluate the CLIP model on clean images. By writing a function that calculates the image feature vector and then normalizes it, and by dot product of these vectors with the text vectors, obtain the classification output and report the model's accuracy. Then, using a PGD attack on the ResNet-20 model ($\epsilon = 8/255, \alpha = 2/255, \text{steps} = 7$), create adversarial examples (using the torchattacks library) and run the same evaluation function to observe the accuracy of CLIP against these transfer attacks. To make the issue more tangible, take a test image, and display it alongside its adversarial version and the attack noise in a three-part figure. (5 points)

4. After that, it's time for standard adversarial fine-tuning with the LoRA method. For this, first apply LoraConfig settings with Low-Rank Adaptation on the vision module layers of CLIP (for example, $r = 8$ and $\alpha = 32$). Then, put the model in train mode and in a training epoch, for each batch of images, calculate the CLIP feature vector on the adversarial images and dot product it with the text vectors. With the CrossEntropyLoss function on the original labels, calculate the gradients and update the LoRA parameters. At the end of this stage, re-measure the clean and adversarial accuracy of the model to see how much standard adversarial training has been able to increase the model's robustness. (15 points)

5. In the next step, implement the TeCoA (Text-guided Contrastive Adversarial Training) algorithm according to equation (3) of the paper and, like the previous section, train the model with this loss function and perform the related tests. (15 points)

6. Finally, display the clean and adversarial accuracies of the three main states: the original CLIP model, the fine-tuned CLIP with LoRA and CrossEntropy, and the fine-tuned CLIP with LoRA and the TeCoA algorithm in a comparative table or graph. Briefly discuss the advantages and limitations of each method and the amount of reduction or increase in clean and adversarial accuracy. (5 points)

Student Report

Prepared by: Mohammad Taha Majlesi - 810101504

2 Question 2 – Robust Zero-Shot Classification

2.1 Introduction

2.1.1 The Concept of Robustness

Machine learning models can be extremely fragile. A very important topic today is increasing the **robustness** of models to ensure their reliable performance in critical and sensitive applications, such as autonomous vehicles and medicine, to prevent potential harm. By examining the weaknesses of models in the context of robustness, we can devise solutions and try to improve their resilience. Therefore, both creating samples that reveal model weaknesses and developing methods to counter these samples are of great importance.

First, let's define the concept of robustness. Robustness is a concept close to generalization, which indicates how well a model can withstand any kind of small adversarial or non-adversarial changes made to the input. In many problems, increasing robustness leads to a decrease in the model's performance on the training data, and the paper also introduces a method to prevent this performance drop.

Various methods exist for generating adversarial examples from data, especially image data. The general idea behind adversarial examples is that we have a sample X and we know that if we give X to the model, it will be assigned the label T. The goal is to solve the following optimization problem:

$$\min_{X'} d(X, X') + \lambda \cdot \text{Loss}(T, F(X')) \quad \text{s.t.} \quad T \neq F(X')$$

This means we want to find an X' that, while being close to X according to a distance function d, has a low loss for a different target T. Usually, cross-entropy is used for classification. The function F represents the model under attack. The parameter λ balances the importance of small changes versus a smaller error between $F(X')$ and T. This type of attack is called a targeted attack. Another type of attack is untargeted, where the only goal is for $F(X')$ to be anything other than $F(X)$, without caring what it becomes.

2.1.2 Methods for Generating Adversarial Examples

Before explaining the methods for generating adversarial examples, we will discuss neighborhood metrics. These metrics are usually used with the goal of creating minimal changes in the input. Generally, the p-norm of a vector x with n components is defined as:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}$$

We consider the change in a sample relative to its original state as $\|x - x'\|_p \leq \epsilon$. The L_0-norm indicates the number of components of x that have changed. The L_1-norm in a two-dimensional space corresponds to a diamond shape centered at x. The L_2-norm represents a circle, and finally, the L_∞-norm, which is one of the most common norms, represents the maximum component of x, which in a two-dimensional space is a square with side length 2ϵ.

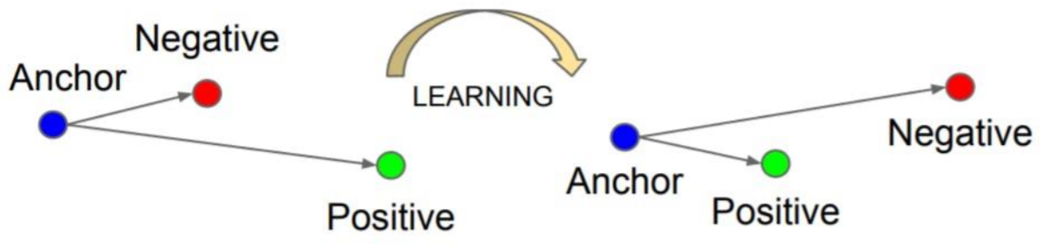


Figure 2: Geometric interpretation of various p-norms. Source: Slides from a reliable AI course.

Now we will describe two common methods for generating adversarial examples, which are also used in the paper: Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD).

FGSM In the FGSM method, we typically use the L_∞ neighborhood and aim for the distance between x and x' to be at most ϵ. To generate x' for an untargeted attack, we act according to the following relation:

$$x' = x + \epsilon \cdot \text{sign}[\nabla_x J(\theta, x, y)]$$

Here, y is the label the model gives to input x, θ represents the model's parameters, and J is the loss function. This is similar to the gradient ascent algorithm. After finding the gradient of the model's loss with respect to the input x and considering its direction with the sign function, we move with a magnitude of ϵ in the calculated direction to form x'.

PGD The next common method is PGD, which is an iterative method. In this method, at iteration zero, the value of x' is considered a point in the neighborhood of x. In each step, by making small changes, the model's loss with respect to the original label is increased. According to this method, x' at step $t + 1$ is obtained as follows:

$$x^{t+1} = \text{Proj}_\epsilon\{x^t + \alpha \cdot \text{sign}[\nabla_x J(\theta, x^t, y)]\}$$

Similar to before, by performing gradient ascent, we try to increase the model's loss on the sample x^t from the previous iteration for the label y. This time, the gradient direction vector is multiplied by α, where α is an arbitrary number and, like the learning rate in deep models, indicates the rate of change in each iteration. After adding the previous value of x' to the gradient vector, the resulting vector x' may go out of the defined range. Therefore, after the addition in each iteration, the resulting vector is projected onto the desired space, which in the L_∞ case is a square with side 2ϵ and center x.

2.2 Introduction to CLIP, One-Shot Classification, and Adversarial Attacks

2.2.1 Contrastive Learning

Now we will examine the model used in the paper, named CLIP. The CLIP model is one of the best models in terms of performance in the field of combining language and vision. This model has various applications in different fields such as generating images from text and searching for images from text or text from images. This model is trained using a contrastive method, which we will explain further.

The contrastive learning method is used in training large language and computer vision models to extract meaningful representations from unlabeled data. In this method, the assumption is that similar samples in the embedding space should have a smaller distance compared to dissimilar samples. Two different types of contrastive learning are proposed in the context of supervised and self-supervised learning.

In supervised contrastive learning, the model learns to distinguish between similar and dissimilar samples with the help of labels. In this method, the model is trained on pairs of data and labels to learn an embedding space where similar samples have a smaller distance than dissimilar samples. The common loss function for this is Information Noise Contrastive Estimation (InfoNCE), which works with the concept of mutual information between data. Its relation is as follows:

$$\text{Loss} = -\text{InfoNCE} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(f(v_i, v_i^+))}{\sum_j \exp(f(v_i, v_j))}$$

Here, f is a function that estimates the similarity of two samples, and in the CLIP model and many other models, cosine similarity or dot product is used. Also, v_i^+ represents a sample similar to v_i. Thus, by minimizing the loss, the model tries to maximize the numerator, which is the similarity between two similar samples, while reducing the similarity with other samples that are in the denominator.

2.2.2 The CLIP Model

Models for classifying specific datasets work well on them but do not have the ability to recognize classes they have not seen in their data. The CLIP model, with contrastive learning on hundreds of millions of images along with texts that are available with the images on the internet, not only performs similarly to models trained on those data on the classes it has seen, but can also recognize classes it has not seen in its data.

Training the CLIP model on hundreds of millions of data requires a very large computational cost. The CLIP model uses two algorithmic methods to reduce computational costs. The first method is the use of Vision Transformers, which, due to the use of the transformer architecture, can take advantage of the parallelization capabilities of GPUs and, in terms of computational cost, are about 3 times better than the model. The other algorithm for reducing computations is the use of contrastive learning, which is about 4 to 10 times more optimized, and thus the entire model could be trained in two weeks, similar to large language models.

Apart from being optimal in terms of computations, the CLIP model has a lot of generalizability, and as we will explain later, it has acquired the ability for one-shot classification well. Also, according to reports from OpenAI, which introduced this model, the best CLIP model performs better than ImageNet models on 20 out of 26 datasets.

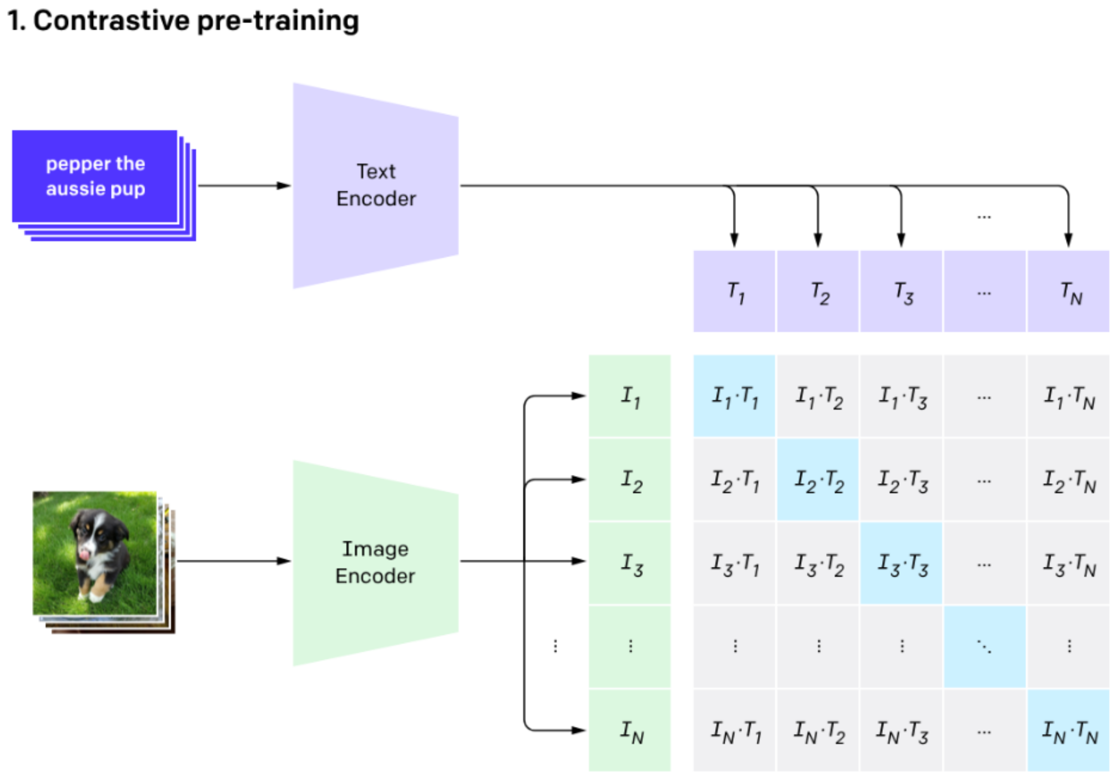

1. Contrastive pre-training
pepper the aussie pup
Text Encoder
Image Encoder
T_1 T_2 T_3 ... T_N
I_1 I_2 I_3 ⋮ I_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
$I_2 \cdot T_1$ $I_2 \cdot T_2$ $I_2 \cdot T_3$ ... $I_2 \cdot T_N$
$I_3 \cdot T_1$ $I_3 \cdot T_2$ $I_3 \cdot T_3$ ... $I_3 \cdot T_N$
$I_N \cdot T_1$ $I_N \cdot T_2$ $I_N \cdot T_3$ ... $I_N \cdot T_N$

Figure 3: The CLIP model architecture.

Now let's look at CLIP in more detail. This model consists of two encoders. One of them receives text as input and outputs a vector containing the semantic content of that text. Similarly, the other model, by receiving an image as input, takes it to an embedding space vector. According to contrastive learning, it tries to increase the cosine similarity of these two embedding vectors and decrease the similarity of other vectors present in the batch. For this, it uses the N-pair multi-class loss function, which we explained above.

2.2.3 One-Shot Classification

As we stated, the CLIP model can also recognize classes that it has not seen in its training data. This type of generalization is called Zero-Shot Generalization. In general, in models, to create this type of generalization, features are usually taken to an embedding space, or with the help of generative models, unrealistic features from unseen classes are generated to train the classifier.

This phenomenon also exists in language models, which gives them the ability to perform tasks whose likes they have not seen in their training texts. In the field of computer vision, CLIP is one of the models that has acquired this ability well and can even perform OCR.

In conventional image classification methods, the model learns the patterns of images by seeing various images from K classes and, by acquiring the ability to generalize, can also recognize images from the same K classes that it has not seen. In one-shot classification, the model, like conventional classification, learns to classify K different classes by seeing images, but besides that, it can also correctly classify images of other classes besides those K classes.

In CLIP, because for each image, the texts similar to that image are also learned, and given the one-shot capabilities created in language models with training on abundant data, the CLIP model can, by relying on the knowledge it has received from texts, also recognize images of new classes.

For classification with the help of CLIP, we give the desired image to the image encoder part, and after receiving the image embedding vector, for the desired classes, we generate desired text, which according to the paper itself, can be the text "a photo of a {object}". By giving this text to the text encoder part, we receive the text embedding vector. Then we calculate the cosine similarity of the image vector with all the text vectors, and the text whose corresponding vector had the most similarity to the image vector is considered as the image class.

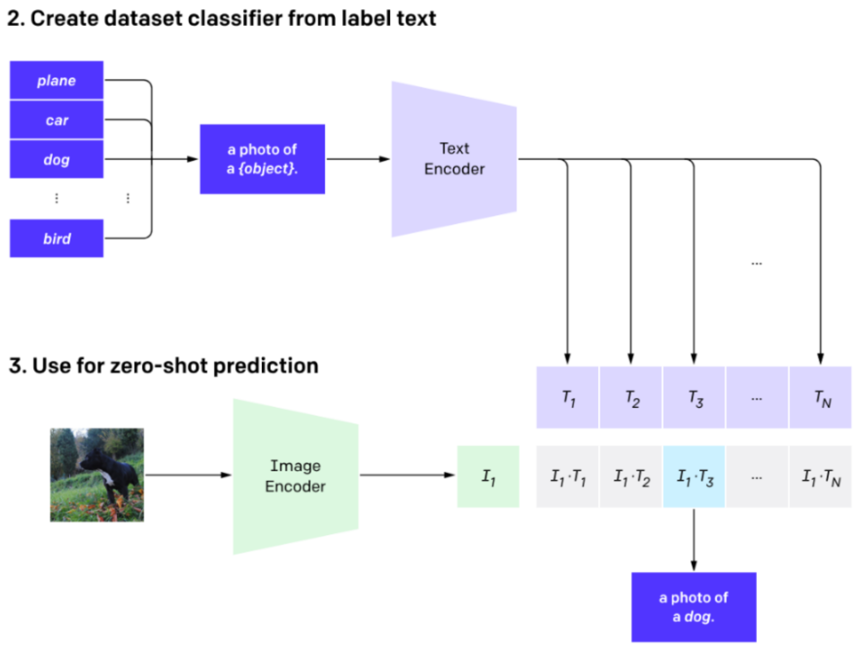

2. Create dataset classifier from label text
plane
car
dog
bird
a photo of a {object}.
Text Encoder
3. Use for zero-shot prediction
Image Encoder
I_1
T_1 T_2 T_3 ... T_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ ... $I_1 \cdot T_N$
a photo of a dog.

Figure 4: How to use CLIP as a classifier.

2.2.4 Types of Adversarial Attacks based on Model Access

In terms of access to the target model in the field of robustness, the type of adversarial attack is usually divided into three different types:

- **White-box attack:** In this type of attack, the attacker has complete information about the model and can use complete information about the weights, hyperparameters, gradients, architecture, and other information about the model whenever they want.
- **Gray-box attack:** In this type of attack, access to information is more limited. For example, the attacker may only know about the model's architecture or the training data but has no information about the training details and the model's weights.

- **Black-box attack:** In this type of attack, the attacker has no access to the weights, gradients, architecture, or other information about the model. The attacker can only get the output by giving input to the model, where the output itself can be in the form of the probability of each class or the predicted label. The goal in this type of attack is to create an adversarial example by giving the minimum number of inputs to the model.

2.2.5 Transfer Attacks

Transfer attack methods are one of the two main categories of black-box attacks. The other category of black-box attacks is optimization-based methods that estimate the gradient of the target model. Although this category of methods has less information about the target model and is more practical in terms of implementation, it has a lower probability of success than the other category. However, as we explained, like other black-box attacks, because they are closer to reality (since the attacker does not have much information about the model), their investigation has great importance compared to white-box attacks.

In general, in this type of attack, by giving input to the model and receiving its output, another model is trained to approximate the first model and act similarly to it. Then, by having complete information about the trained model, adversarial examples are generated with white-box methods, and we give them to the target model. It has been shown that adversarial examples generated for one model can also be adversarial for other models, and thus cause a drop in the quality of the other model. It has also been shown that even using adversarial examples from an inaccurate model of the target model, the probability of success is not negligible.

2.2.6 Fine-tuning with LoRA

The LoRA (Low-Rank Adaptation) fine-tuning method is one of the most popular and widely used methods for fine-tuning deep models, especially large language models. This method is part of the Reparameterized Fine-tuning methods, which in this category of methods, a low-rank matrix is used to update the weights of the pre-trained layers of the model. Thus, a balance is maintained between updating the weights and keeping the current weights, and it is computationally optimal.

With the growth of models and the increase in the number of parameters, fine-tuning them is not feasible because for fine-tuning models, we need about three times the model's own size in memory. For this reason, PEFT (Parameter-Efficient Fine-Tuning) methods were introduced so that by making changes in a small ratio of parameters, we can fine-tune the model in a way that its performance is close to fine-tuning the entire model.

Among these, LoRA is one of the most popular PEFT methods and has shown its advantages in various fields. First, let's examine how it works. Before the introduction of LoRA, another paper introduced a method for measuring the number of intrinsic dimensions of large language models and showed that the parameters of large language models are located in a low-dimensional space.

With the results of that paper, in another paper, the LoRA method was introduced for fine-tuning models. The presentation of this method had three motivations. First, according to the stated paper, the number of dimensions of language models is very small, and this can be used for optimal fine-tuning. Second, in many PEFT methods, increasing the number of parameters does not necessarily improve performance, while we prefer that with an increase in the number of parameters, the model's performance improves. Finally, we prefer that the PEFT method does not cause the model to slow down.

In this method, the pre-trained weights of the model, i.e., W_0, are kept fixed during fine-tuning. The weights $\Delta W = BA$ are defined in parallel with a part of the weights W_0, and by adding these new weights to the old weights, its weights are updated. Also, the matrix BA has rank r, which is much smaller than the number of input dimensions k and the number of output dimensions d. The initial value of the parameters of A is from a Gaussian distribution, and the initial values of the matrix B are considered zero so that initially ΔW is zero and the model only uses its own weights, and with fine-tuning, the weights are updated.

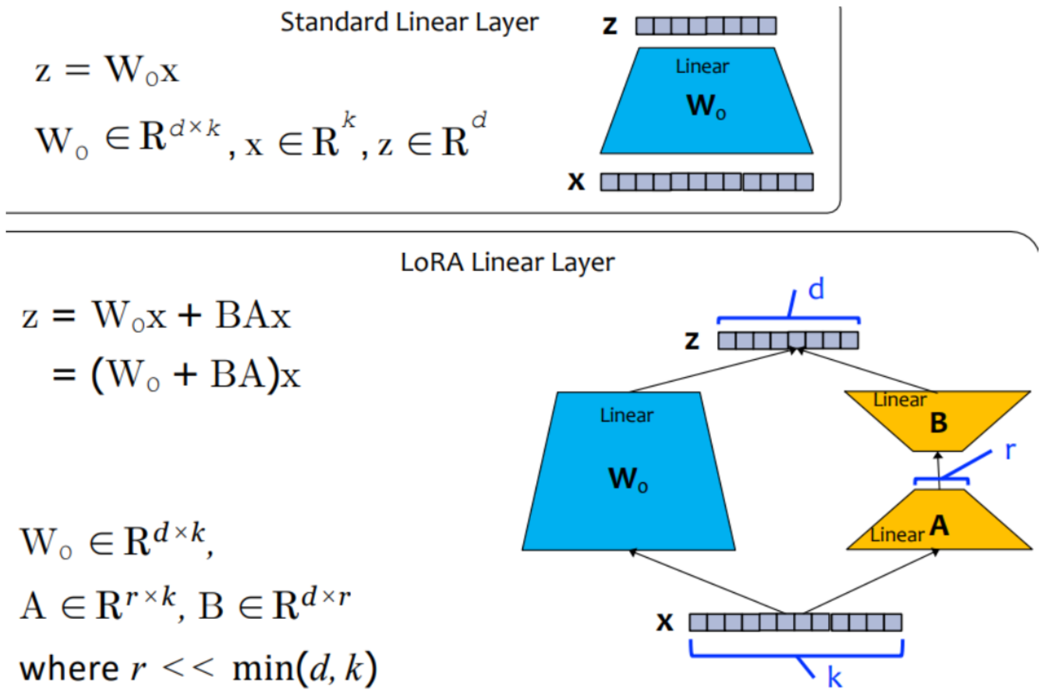


Figure 5: The LoRA mechanism. Source: Slides from a large language models course.

2.3 Preparation for Training

2.3.1 Dataset Preparation

The CIFAR-10 dataset is one of the most widely used datasets in the field of computer vision, which includes color images with dimensions of 32x32 pixels from 10 different classes, with each class containing 6000 unique images. The classes of this dataset are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

To use the images for the CLIP model, we first resize the images to 224 pixels and then normalize them with the mean and standard deviation of the CLIP model's images. Then we separate the images to have 10,000 evaluation images, 2,000 validation images, and 8,000 training images.

Figure 6: A few samples from the CIFAR-10 dataset with their labels.

2.3.2 Loading the Models

The Hugging Face site provides various tools and models in different fields, including natural language learning, to developers. We load two models, CLIP and ResNet20. The CLIP model is the model we intend to attack, and since we want our attacks to be as close to reality as possible, we perform a transfer attack with the help of the ResNet20 model, which is trained on the CIFAR-10 images. After loading the models, for each of the ten existing classes, we form a text similar to "a photo of a {class_category}" and with the help of the text encoder part of the CLIP model, we receive the ten vectors corresponding to each text and normalize them to use them later to obtain the class classified by the model.

2.4 Analysis of Results

2.4.1 Analysis of Results on Clean Images

First, by giving the clean validation images to the CLIP model and obtaining the similarity of the image vector to the texts by dot product of the normalized vectors, we examine the performance of the one-shot classification of the CLIP model on the CIFAR-10 images before performing the attack.

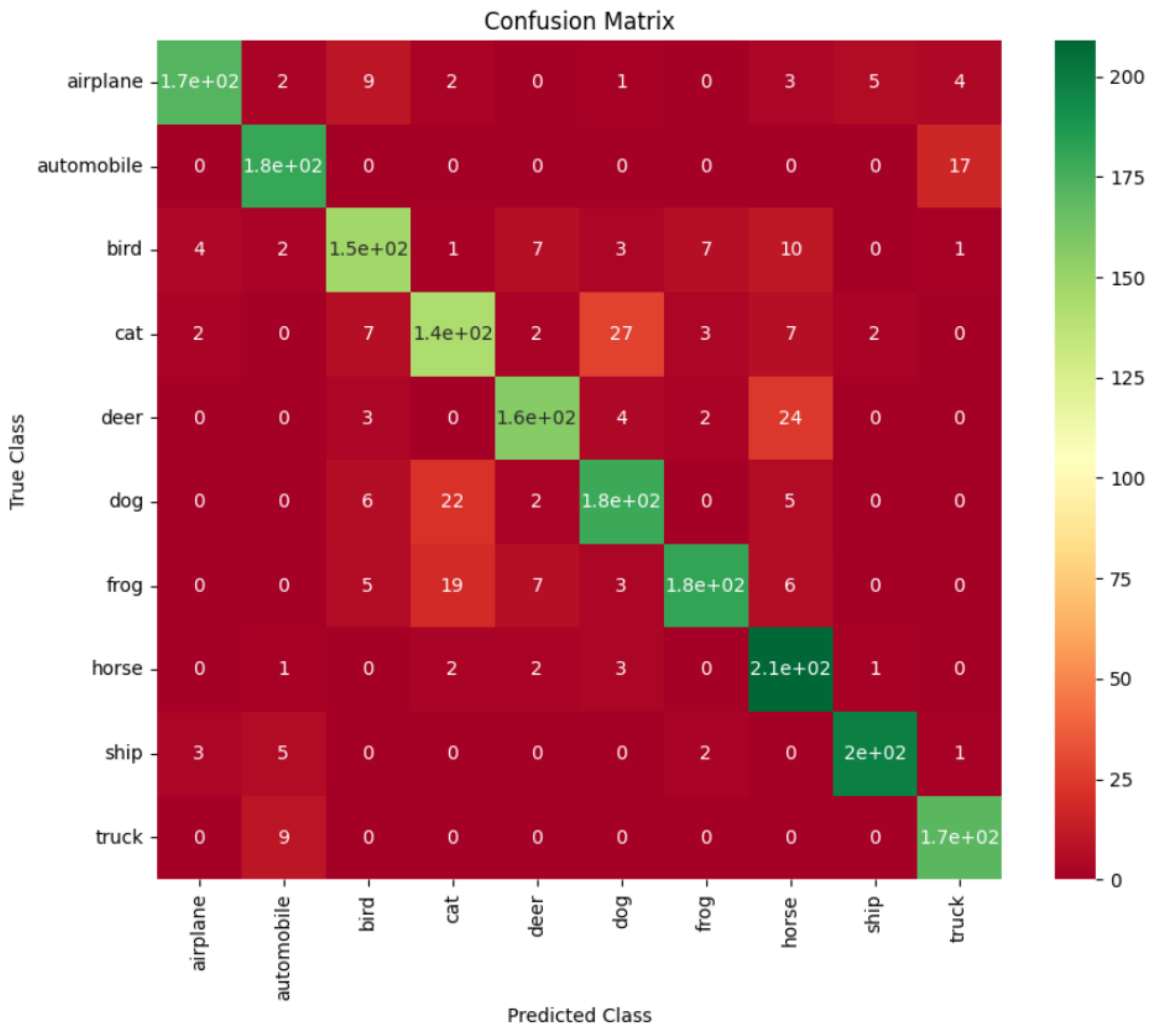


Figure 7: Confusion matrix of the CLIP model on clean images before fine-tuning.

It can be seen that the overall performance of the model is suitable, and the highest error rate is between the two classes of dog and cat, with 27 cases. Also, most of the other errors are between animal classes or the car and truck classes, which is due to the low quality of the model's images, which has a low error rate.

Table 1: Evaluation metrics of the model on clean images before fine-tuning.

Class	**Accuracy**	**Precision**	**Recall**	**F1 score**
Micro	0.8675	0.8675	0.8675	0.8675
Macro	0.8675	0.8703	0.8667	0.8669
Weighted	0.8675	0.8707	0.8675	0.8674
Airplane	0.9825	0.9503	0.8687	0.9077
Automobile	0.9820	0.9045	0.9137	0.9091
Bird	0.9675	0.8305	0.8077	0.8189
Cat	0.9520	0.7566	0.7409	0.7487
Deer	0.9735	0.8864	0.8254	0.8548
Dog	0.9620	0.8128	0.8357	0.8241
Frog	0.9730	0.9282	0.8190	0.8702
Horse	0.9680	0.7917	0.9587	0.8672
Ship	0.9905	0.9614	0.9476	0.9544
Truck	0.9840	0.8808	0.9497	0.9140

It can be seen that the model has reached an overall accuracy of 86 percent, and its accuracy, precision, recall, and F1 score for all classes except the cat class are above 80 percent.

2.4.2 Analysis of Results on Adversarial Images

Now that we have evaluated the model's performance on clean images and the model was able to classify the classes well despite the low quality of the images, we generate adversarial examples with the help of the ResNet20 model and give them to the model to examine its performance.

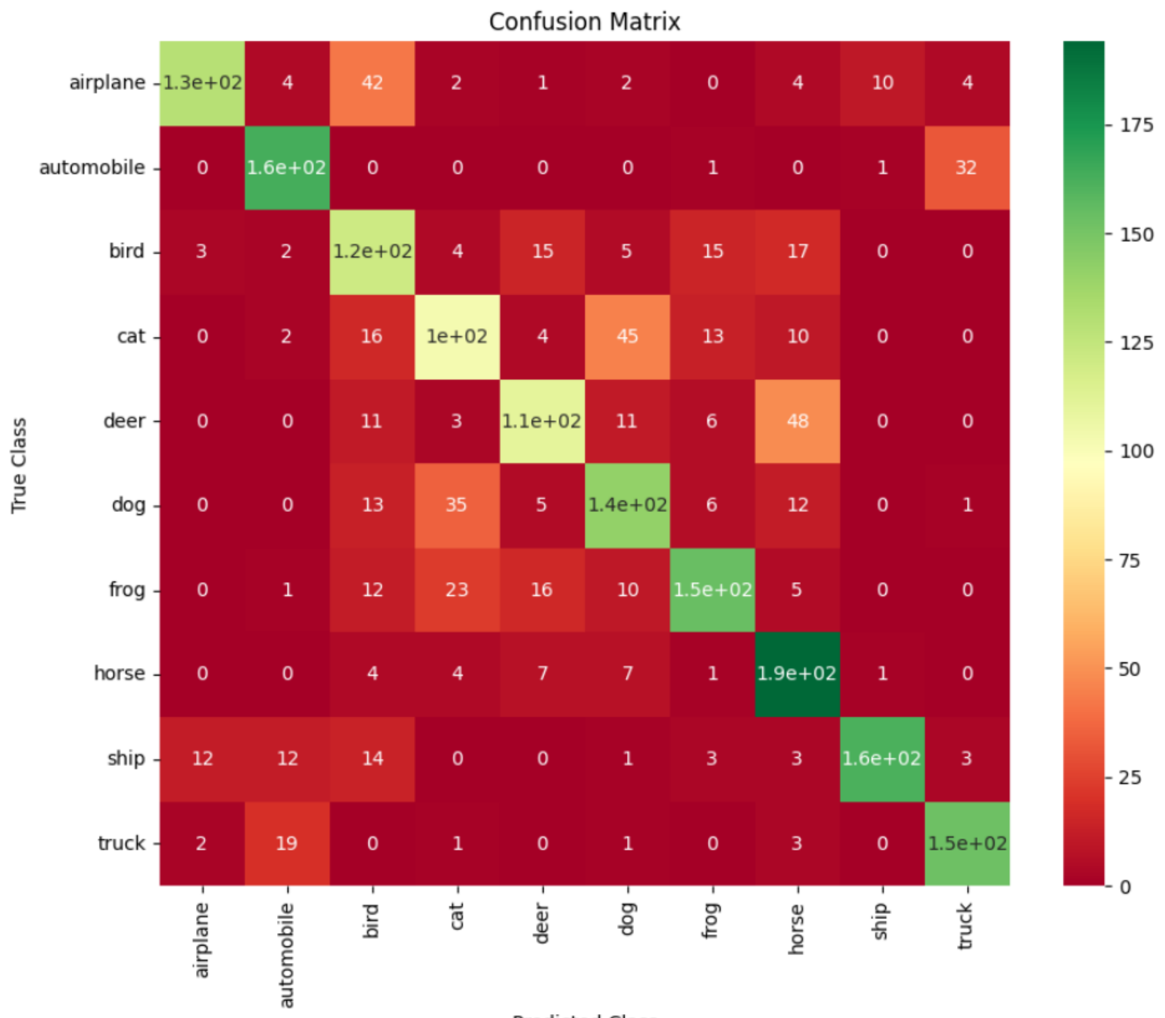


Figure 8: Confusion matrix of the model on adversarial examples before fine-tuning.

It can be seen that by giving adversarial examples, the model's performance decreases. For example, although the model could distinguish between the two classes of bird and airplane well, now there are 42 error cases between these two classes. Similarly, its performance on other classes, especially in distinguishing between different animals, has dropped.

Table 2: Evaluation metrics of the model on adversarial examples before fine-tuning.

Class	Accuracy	Precision	Recall	F1 score
Micro	0.7150	0.7150	0.7150	0.7150
Macro	0.7150	0.7276	0.7134	0.7143
Weighted	0.7150	0.7294	0.7150	0.7159
Airplane	0.9570	0.8836	0.6515	0.7500
Automobile	0.9630	0.8030	0.8274	0.8150
Bird	0.9135	0.5193	0.6648	0.5831
Cat	0.9190	0.5886	0.5337	0.5598
Deer	0.9365	0.6962	0.5820	0.6340
Dog	0.9230	0.6323	0.6620	0.6468
Frog	0.9440	0.7739	0.6968	0.7333
Horse	0.9370	0.6554	0.8899	0.7549
Ship	0.9700	0.9310	0.7714	0.8438
Truck	0.9670	0.7927	0.8547	0.8226

From the table above, it can be seen that the model's performance has generally dropped by about 15 percent. The weakest performance in terms of F1 score is related to the animal classes, especially cat and bird, with their values reaching 55 and 58 percent, respectively, while on clean images, their F1 scores were 74 and 81 percent.

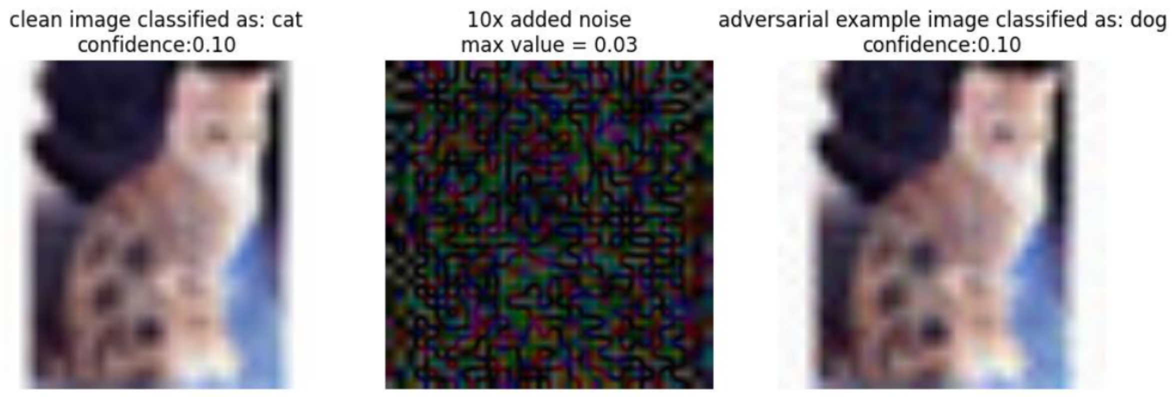

clean image classified as: cat
confidence:0.10
10x added noise
max value = 0.03
adversarial example image classified as: dog
confidence:0.10

Figure 9: A successful adversarial example along with the generated noise.

In the figure above, the model correctly identified the left image as a cat, but after adding the specified noise to the right image, we see that it does not differ much from the left image from a human perspective, but the model mistakenly identifies it as a dog. Also, in the middle, the added noise is shown, which, due to the possibility of being negative, we have normalized its values between zero and one, and to better show it, we have multiplied it by ten. Also, the maximum noise value was about 0.03, which matches the epsilon given to the attack function, and it is clear that if a white-box attack or with more noise were performed, the probability of success would increase.

2.5 Fine-tuning and Analysis of Results on Adversarial Validation Images

2.5.1 Analysis of Results after Training with Cross-Entropy Loss

In the paper, various methods for training the model against adversarial attacks are reviewed. Of course, in all methods, adversarial training is used. In this method, we show the model adversarial examples and try to reduce its error on these samples so that it can correctly identify these samples as well. Many methods have been introduced in defense against adversarial attacks, but for most of these methods, by creating more precise adversarial examples, the model's performance can be compromised. However, the adversarial training method has been one of the best methods, and although the model's performance on clean data also decreases slightly, the attack on the model becomes harder.

Given that the CLIP model has about 151.77 million parameters, training it on available GPUs with low memory is not possible. For this reason, we use PEFT methods, and more precisely, LoRA, to fine-tune a part of the model's parameters to achieve a performance close to fine-tuning all the weights.

Thus, we use LoRA with rank $r = 8$ and alpha parameter 32 in the query and value matrices of the attention mechanism to learn less than 500,000 parameters, which includes only about 0.32 percent of the model's parameters.

After writing the model training functions on adversarial images and putting the data loader in attack mode, we first use the second error reviewed in the paper. According to this error, the model, by receiving adversarial images, calculates the similarity between images and texts by dot product of each image's vector with each of the text vectors, and the text that has the most similarity to the image is considered as the class predicted by the model. Then, by having the predicted classes and the real classes, the model's weights are updated with the cross-entropy loss function.

For fine-tuning our model, we use a batch size of 64 and ‘weight$_{d}ecay`of 0.0001. We also set the mome$

We train the model on the training images and review its performance on the validation images. Initially, the model's loss is 2.25, and it eventually reaches 2.09 on the validation images.

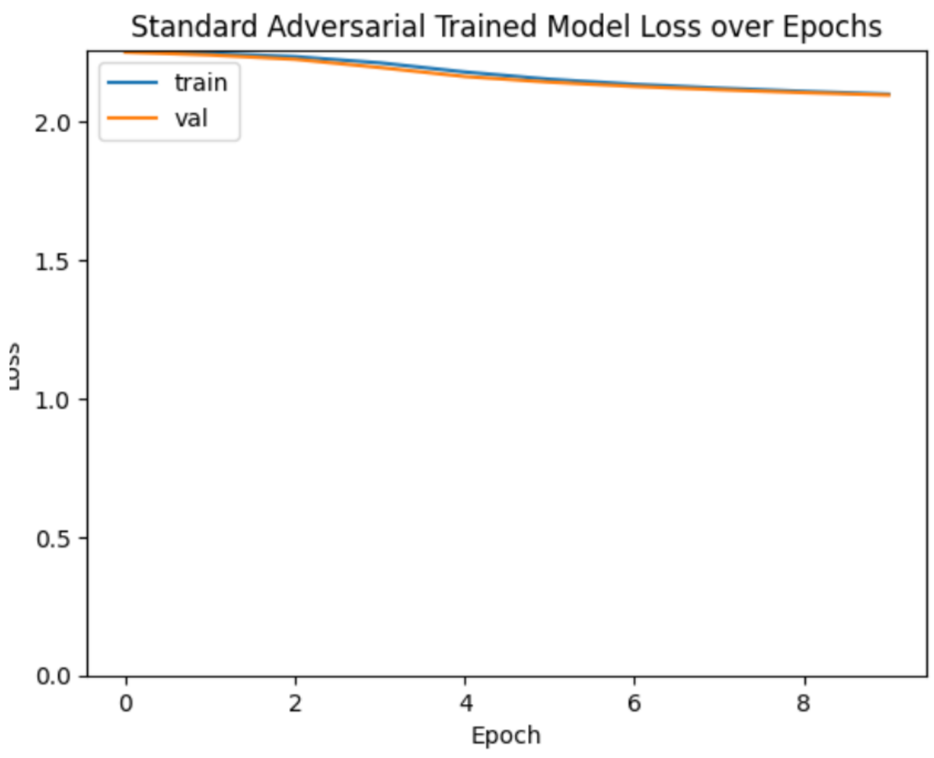
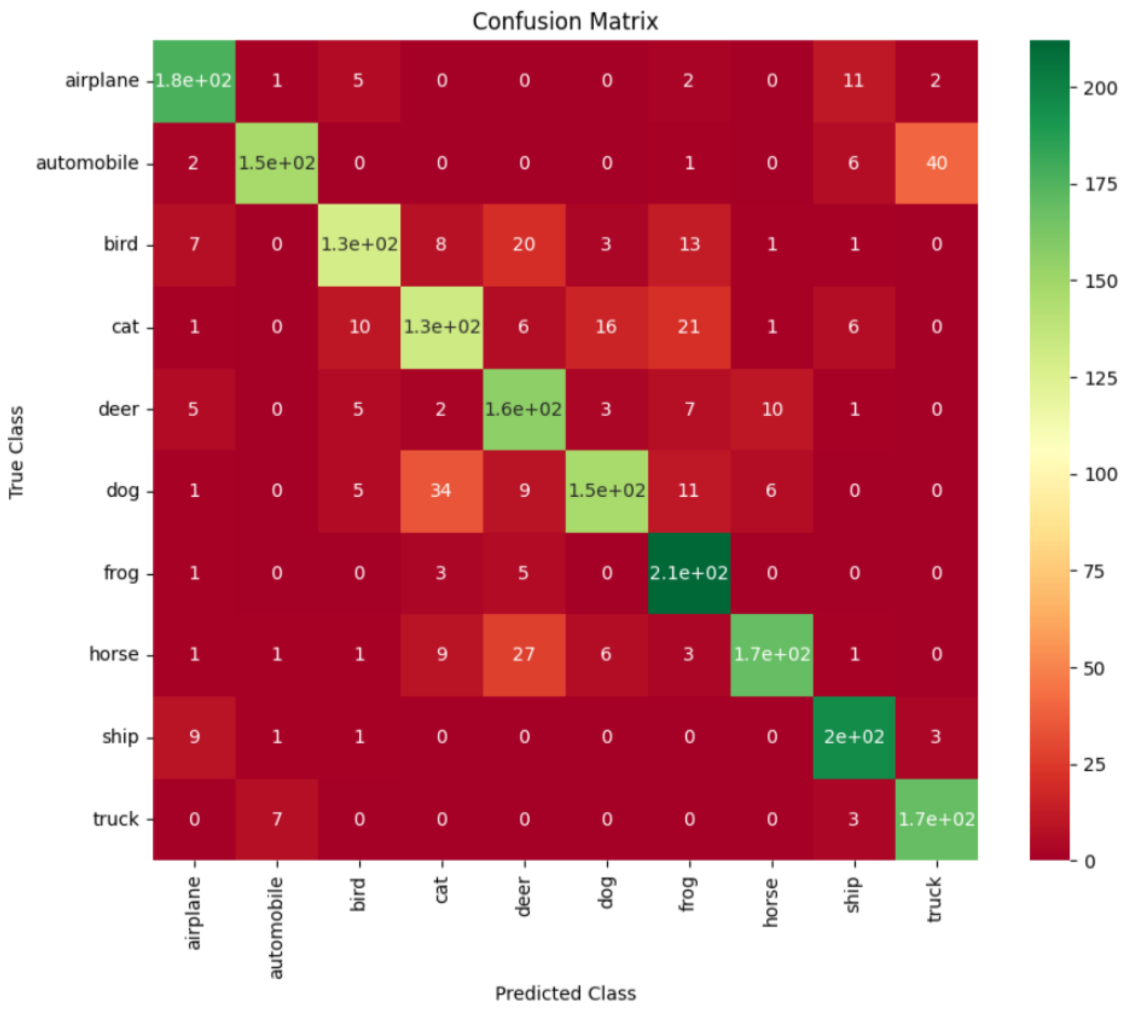


Figure 10: The model's loss changes during training.

From the graph above, it can be seen that the loss converges after about 10 epochs. So we use the same parameters for training the next models, and given the similarity of the loss changes in the next models, we do not plot their loss.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	1.8e+02	1	5	0	0	0	2	0	11	2
automobile	2	1.5e+02	0	0	0	0	1	0	6	40
bird	7	0	1.3e+02	8	20	3	13	1	1	0
cat	1	0	10	1.3e+02	6	16	21	1	6	0
deer	5	0	5	2	1.6e+02	3	7	10	1	0
dog	1	0	5	34	9	1.5e+02	11	6	0	0
frog	1	0	0	3	5	0	2.1e+02	0	0	0
horse	1	1	1	9	27	6	3	1.7e+02	1	0
ship	9	1	1	0	0	0	0	0	2e+02	3
truck	0	7	0	0	0	0	0	0	3	1.7e+02

Figure 11: Confusion matrix of the model on adversarial images after fine-tuning with cross-entropy.

From the confusion matrix, it can be seen that after fine-tuning with cross-entropy with the help of adversarial examples, the model's error has significantly decreased, and unlike before, it can correctly identify the bird and airplane classes. It still has some distance from the initial state.

Table 3: Evaluation metrics of the model on adversarial examples after training with cross-entropy.

Class	**Accuracy**	**Precision**	**Recall**	**F1 score**
Micro	0.8175	0.8175	0.8175	0.8175
Macro	0.8175	0.8223	0.8165	0.8145
Weighted	0.8175	0.8242	0.8175	0.8159
Airplane	0.9760	0.8676	0.8939	0.8806
Automobile	0.9705	0.9367	0.7513	0.8338
Bird	0.9600	0.8269	0.7088	0.7633
Cat	0.9415	0.7021	0.6839	0.6929
Deer	0.9500	0.6996	0.8254	0.7573
Dog	0.9530	0.8400	0.6901	0.7577
Frog	0.9665	0.7852	0.9593	0.8635
Horse	0.9665	0.9037	0.7752	0.8346
Ship	0.9785	0.8711	0.9333	0.9011
Truck	0.9725	0.7897	0.9441	0.8601

From the table above, it is clear that the model's performance has improved significantly, but it is still about 5 percent away from its performance on clean images. Again, similar to before, the most error is in distinguishing animals from each other.

2.5.2 Analysis of Results after Training with TeCoA Loss

Now we use the loss function introduced in the paper named TeCoA. This loss function, unlike the previous loss function, is contrastive, meaning it tries to bring the positive samples in each batch closer together. More precisely, the TeCoA loss function, for each image, considers only the text of that image as a positive sample for it and all other texts for other samples in the batch as negative samples, and calculates the loss function in this way:

$$L_s(x,t,y) = -E_{i,j}[y_{i,j} \log \frac{\exp(\cos(z_i^{(I)}, z_j^{(T)})/\tau)}{\sum_k \exp(\cos(z_i^{(I)}, z_k^{(T)})/\tau)}]$$

In the above relation, $z_i^{(I)}$ represents the model's output vector for the i-th image, and $z_j^{(T)}$ represents the model's output vector for the j-th text. Also, 'cos' represents cosine similarity, and $y_{i,j}$ is only 1 for each image and its corresponding text and is zero for the rest of the image texts. τ is also a scalar value that was not reported in the paper, and we first examine the value 0.01 and then 0.1. Thus, by reducing the loss function, the model tries to bring the vectors of the text and image of a specific class closer together, and due to the presence of other vectors in the denominator, their similarity decreases.

By writing the loss function, we train the model with its help, and the model's loss on the validation data starts from 2.32 and eventually reaches 2.19.

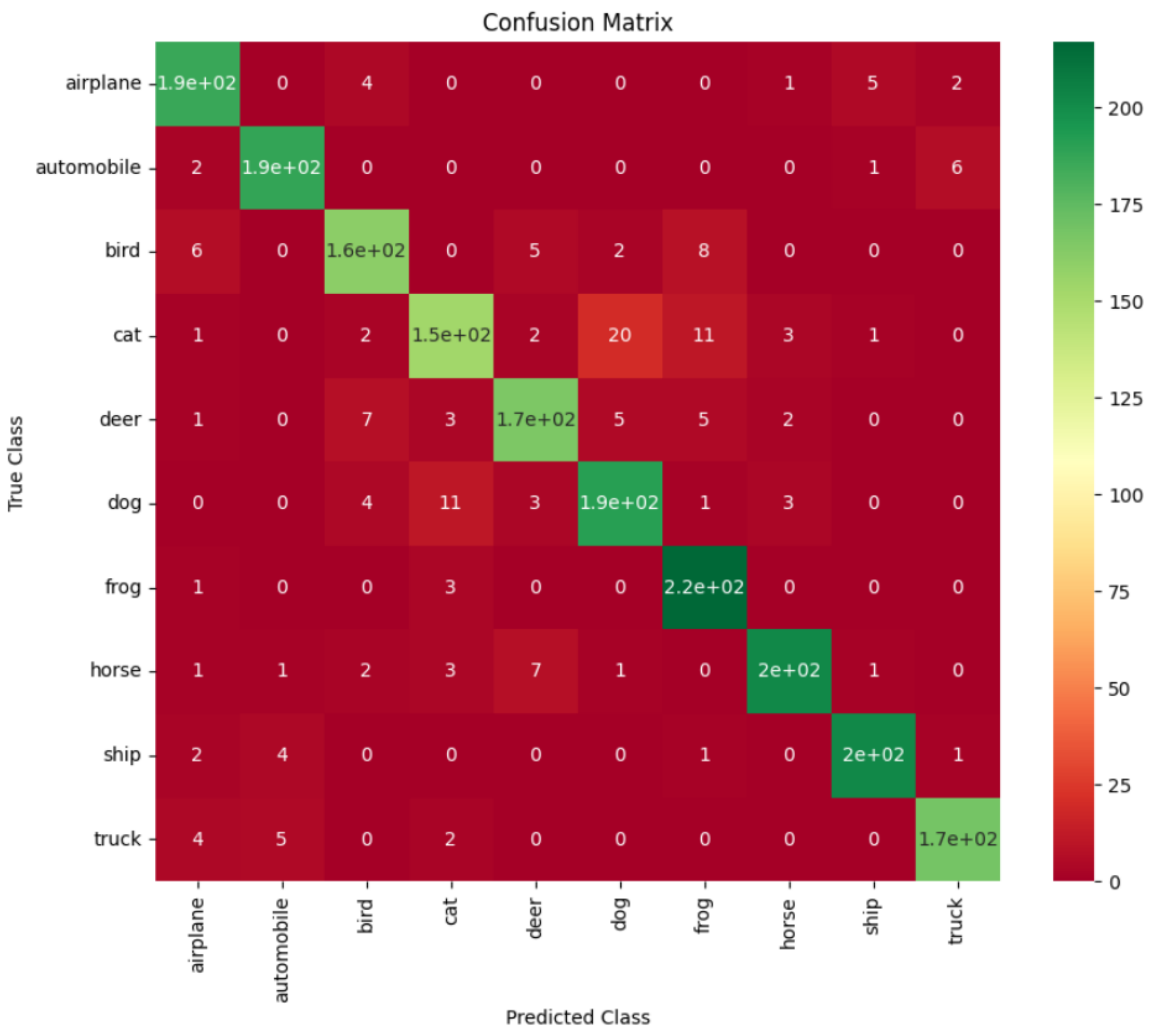


Figure 12: Confusion matrix of the model trained with TeCoA loss.

From the confusion matrix above, it can be seen that the diagnosis of the two classes of dog and cat is still difficult for the model, but apart from that, it seems that the model can distinguish all other classes well. Thus, to better examine the metrics, we calculate them.

Table 4: Evaluation metrics of the model on adversarial examples after training with TeCoA.

Class	Accuracy	Precision	Recall	F1 score
Micro	0.9170	0.9170	0.9170	0.9170
Macro	0.9170	0.9171	0.9155	0.9158
Weighted	0.9170	0.9171	0.9170	0.9165
Airplane	0.9850	0.9118	0.9394	0.9254
Automobile	0.9905	0.9495	0.9543	0.9519
Bird	0.9800	0.8944	0.8846	0.8895
Cat	0.9690	0.8743	0.7927	0.8315
Deer	0.9800	0.9071	0.8783	0.8925
Dog	0.9750	0.8721	0.8967	0.8843
Frog	0.9850	0.8930	0.9819	0.9353
Horse	0.9875	0.9573	0.9266	0.9417
Ship	0.9920	0.9619	0.9619	0.9619
Truck	0.9900	0.9492	0.9385	0.9438

From the table above, it is clear that the model's performance is even better than the original model on clean images. This is due to the appropriate fine-tuning with this loss function, which has caused the model's performance not only not to drop when seeing adversarial images but also to work even better than before on these images. Now the values of all metrics for all classes are above 80 percent, and the model has the ability to recognize classes.

2.5.3 Final Evaluation and Comparison

Now, the four evaluated methods on the validation images are trained on 10,000 training and validation images, and we evaluate the models' performance on the clean and adversarial test images. Given that in the previous part, the best performance on the adversarial validation data was for the model trained with TeCoA loss, we expect this model to perform best now as well.

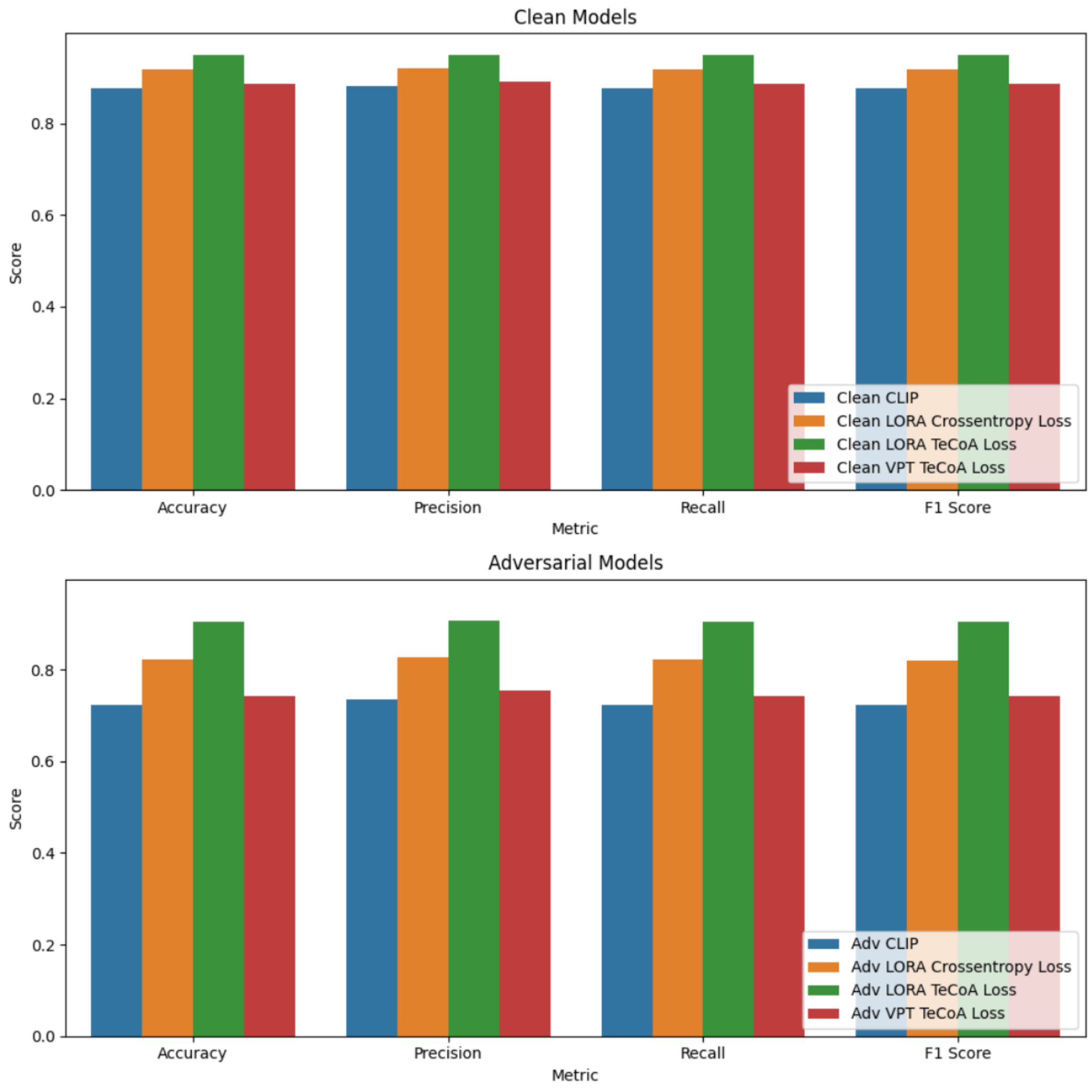


Figure 13: Comparison of the models' performance on clean and adversarial test images.

From the images above, it can be seen that, as we expected, the models' performance on clean images is better than on adversarial examples. Also, the best performance on both clean and adversarial images is, in order, from best to worst: first, the model trained with TeCoA loss with the LoRA method; second, the model trained with cross-entropy loss; third, the model trained with the VPT method; and finally, the model before training. Now, to examine the metrics more precisely, we review them numerically.

Table 5: Metric values for the models on adversarial and clean test data.

Model Configuration	Accuracy	Precision	Recall	F1 score
Clean Test Set				
Clean CLIP	0.8765	0.8810	0.8765	0.8765
Clean LORA Crossentropy Loss	0.9170	0.9187	0.9170	0.9164
Clean LORA TeCoA Loss	0.9485	0.9490	0.9485	0.9485
Clean VPT TeCoA Loss	0.8866	0.8907	0.8866	0.8869
Adversarial Test Set				
Adv CLIP	0.7231	0.7346	0.7231	0.7226
Adv LORA Crossentropy Loss	0.8208	0.8261	0.8208	0.8189
Adv LORA TeCoA Loss	0.9047	0.9065	0.9047	0.9050
Adv VPT TeCoA Loss	0.7417	0.7534	0.7417	0.7428

From the table above, it is also clear that training with VPT improves performance on adversarial images by about two percent. Training with cross-entropy loss improves performance by 10 percent, and training with LoRA with TeCoA loss improves performance by about 18 percent. Also, on clean images, training with VPT improves by 1 percent, training with cross-entropy by 4 percent, and training with TeCoA by 7 percent. The reason for the better performance of the model on clean images can be due to fine-tuning. We know that the CLIP model has been trained on much larger images, and by fine-tuning on a specific set of images, it can improve the model's performance on that set of images. Of course, it is better to evaluate the model's performance on other image sets as well to get a correct assessment of the model's overall performance for one-shot classification.

2.6 Conclusion

In this exercise, we became familiar with many concepts. First, with the concept of robustness in models and various types of adversarial attack and defense methods against them in models. We also became familiar with the CLIP model, which is a multi-modal model, how it works, and how the one-shot classification feature is created in it and what advantages this feature has. We also became familiar with fine-tuning methods, especially optimal fine-tuning or PEFT, and especially LoRA, which allows us to train very large models.

Then, by loading the models, we tried to, similar to reality, without having any information from the CLIP model except that we intend to evaluate it on CIFAR-10 images, generate adversarial examples with transfer attack methods and with the help of the ResNet20 model. Then, by examining the model's performance on the adversarial examples, we tried to make it robust against attacks with different methods.

We examined four main types of fine-tuning methods to make the model robust. In two methods, we used LoRA to fine-tune the model with two loss functions, cross-entropy and TeCoA, which were introduced in the paper, and we also used the VPT method to fine-tune the model with fewer parameters.

Finally, by examining the results, we confirmed the paper's claim about the performance of the model after training with the TeCoA loss compared to other fine-tuning methods in our training environment. Of course, there are other investigations to be done, such as examining the one-shot classification capability of the CLIP model on other dataset classes and using the rest of the methods presented in the paper, which we hope to be able to investigate in a suitable opportunity.