In the Name of God

University of Tehran

Faculty of Electrical and Computer Engineering

Neural Networks and Deep Learning

Assignment 5

c

Assignment Details	
Provider	Mohammad Taha Majlesi
STUNumber	810101504
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)

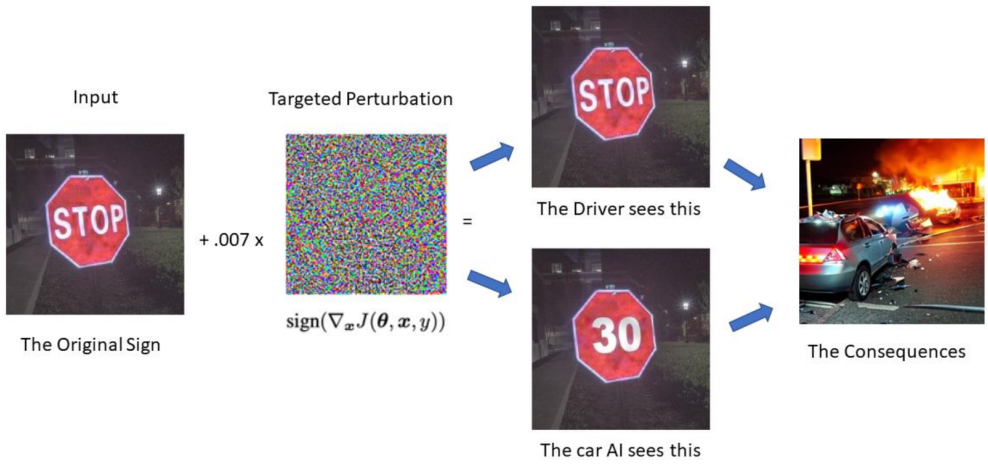

Figure 1: An AI model's mistake in identifying the class of an adversarial example.

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 ($\epsilon = 8/255, \alpha = 2/255,$ 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

1 Question 1: Image Classification with ViT

1.1 Introduction

In this exercise, we aim to use transfer learning with a pre-trained convolutional neural network named Inception V3 and train a Vision Transformer (ViT) network to diagnose diseases and pests in plants. This task helps in preventing crop damage, ensuring their health, and avoiding losses for farmers.

The conventional method for diagnosing plant diseases is visual inspection by a specialist, which can be costly, time-consuming, and prone to human error. Therefore, with the advancement of artificial intelligence models, efforts have been made to automate this process.

Convolutional networks, due to their suitable performance, have become the dominant architecture in computer vision over the years. The application of convolution and considering the correlation between adjacent pixels in images are among the main reasons for the success of these networks.

One of the newer architectures compared to convolutional networks is the Vision Transformer or ViT, which achieves remarkable performance in classification despite having a large number of images. Unlike CNNs, which have many applications in computer vision, ViT works with an attention mechanism and can thus learn the relationship between patches across the entire image.

In convolutional networks, after training the model's weights for each input image, features are obtained, but the weight of all these features remains constant. In contrast, ViT networks, by employing an attention mechanism, can pay more attention to important features and thus discard important information present in the image. By assigning appropriate weights to the image patches, the ViT model can give more importance to more significant features. Thus, these networks learn the global relationships of image patches from the very beginning, unlike convolutional models that require many layers.

It has also been shown in various tasks that the performance of ViT is better than convolutional networks. Also, due to the use of fixed-size patches from images, this model is more scalable than convolutional networks and can be used for images of different dimensions. Usually, the learning capacity of these networks is high, and their performance improves with an increase in the number of data.

1.2 Data Preparation

The dataset we use is a part of the Plant Village dataset, which includes images of various plants. The dataset used only contains images of tomato leaves with ten classes, consisting of nine different disease classes and one healthy class. In total, there are 9325 images, and a few samples of the images are shown in the figure below.

Figure 2: One sample image for each class.

From Figure 3, it is clear that all classes except the last two have exactly 1000 data points, while the last two classes have 952 and 373 images, respectively. To balance the classes, we intend to use the common method of augmentation. However, since according to the paper, the augmentation layers are inside the models, in this case, for the augmented images, the augmentation layers are applied twice. Therefore, we initially use oversampling, which means repeating the images of the two under-represented classes, to equalize the number of images in all classes. By placing the augmentation layers in the models, the necessary changes are applied. Since the paper states that the number of images in all classes is 1000, we can assume that the imbalance in our dataset was created. Therefore, to ensure that the number of validation data for each class is also the same, we take the same number of images from each class for validation and then apply oversampling to the images of the two classes in the training data.

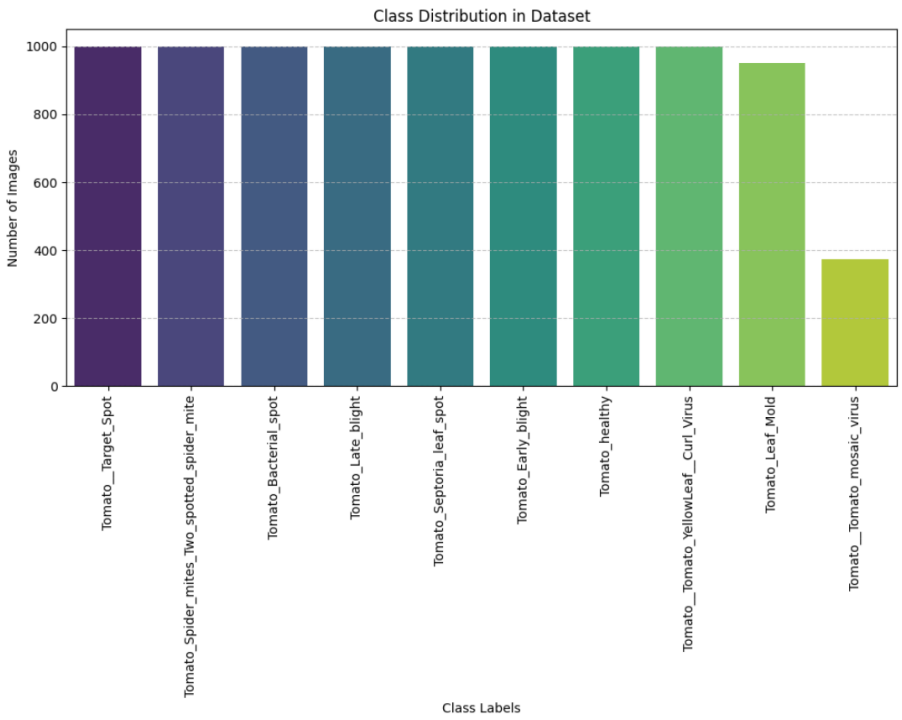

Figure 3: Class distribution of the dataset.

1.3 Inception V3 Model

1.3.1 Model Creation

The Inception V3 model is a deep architecture composed of convolutional and fully connected layers. This model uses a module called the Inception module, which allows the model to extract features at different sizes with filters of different dimensions. This model was introduced in 2015 with the aim of reducing the computations of the previous version and is more efficient in terms of computation and number of parameters compared to VGGNet.

This model, by using filters with different dimensions, can learn both detailed and general features well. Also, in the structure of the model, auxiliary classifiers are used in the middle layers, which force the model to learn features correctly in the initial layers as well and act like a regularizer.

We load the Inception V3 model with random weights along with a 75x75 pixel input layer. Also, as stated in the paper, we add an augmentation layer to its input layer. We use an average pooling layer at the end of the model. At the end of the model, we place a 10-neuron classifier. Finally, we use softmax so that the model's output is converted to the probability of each class.

Table 1: Summary of the CNN model.

```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ Augmentation_Layers             │ (None, 75, 75, 3)      │             0 │
│ (Sequential)                    │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ inception_v3 (Functional)       │ (None, 2048)           │    21,802,784 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 10)             │        20,490 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
Total params: 21,823,274 (83.25 MB)
Trainable params: 21,788,842 (83.12 MB)
Non-trainable params: 34,432 (134.50 KB)
```

The created model has nearly 22 million trainable parameters.

1.3.2 Analysis of Loss Functions

The paper uses categorical cross-entropy loss, which is the most common loss function in training multi-class classification models. This loss function uses the concept of entropy in statistics to compel the model to make correct and confident predictions. According to the Maximum Likelihood method in statistics, using this loss function is recommended. Of course, to use this loss function, the output must first be in One-hot format, which is done by the TensorFlow framework using sparse categorical cross-entropy. The calculation of this loss function is as follows:

$$\text{Loss}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Another loss function used in multi-class classification is Kullback-Leibler Divergence Loss, which generally uses the distance function used in statistics called KL Divergence to minimize the difference between the model's output distribution and the true distribution. This loss function is similar to the previous loss function and essentially shows how much information is lost if the model's distribution is used as an approximation for the true distribution. The loss value is calculated as follows:

$$\text{Loss}(y, \hat{y}) = y(\log y - \log \hat{y})$$

Another usable function is categorical hinge loss, which is used in SVM models and increases the model's loss if the score of the correct class is not greater than the other classes by a certain margin. Usually, the margin value is considered to be one so that the model is forced to classify the correct class with certainty.

1.3.3 Applying Augmentation

As we saw, two classes had fewer images than the other classes, and since we used oversampling, by creating changes in the images with the help of augmentation layers, we make the model robust to various changes and cause each class to have new images for each forward pass. By creating augmented data, we must try to apply changes that, while creating different images to prevent overfitting, we must be careful that the generated images are from the same distribution as the training images, meaning that

the possibility of such images existing in the real world is real, so that the model can learn real features. For this, we write the augmentation layers as follows:

```
def create_aug_layers(input_size, mean=[0,0,0], var=[1,1,1]):
    transforms = tf.keras.models.Sequential([
        layers.Input(shape=(input_size, input_size, 3)),
        layers.RandomBrightness(0.1, (0.0, 1.0)),
        layers.RandomRotation(0.125, fill_mode='reflect'),
        layers.RandomZoom((-.05, .05), (-.05, .05), fill_mode='reflect'),
        layers.RandomFlip("horizontal"),
        layers.Normalization(mean=mean, variance=var, axis=-1),
    ], name="Augmentation_Layers")
    return transforms
```

Listing 1: Augmentation Layers

The layers above, upon receiving an input image, first randomly change the image's brightness slightly. Then, they rotate the image by about 45 degrees randomly. Then, they zoom in or out on the images randomly. After that, there is the possibility of flipping the image around the vertical axis. The main reason we did not flip the images around the horizontal axis is that by examining the images, we realized that most are in a specific direction and only differ slightly in angle. Finally, as explained at the end of section 2-1, for preprocessing, we standardize the pixel values with the mean and standard deviation of the training images. This is generally done for faster stability and convergence, and we also, according to our investigation, found that the models had very poor performance without this preprocessing and could not learn the patterns. This could be due to the depth of the models. Because, like the paper, we also put the augmentation layers inside the models.

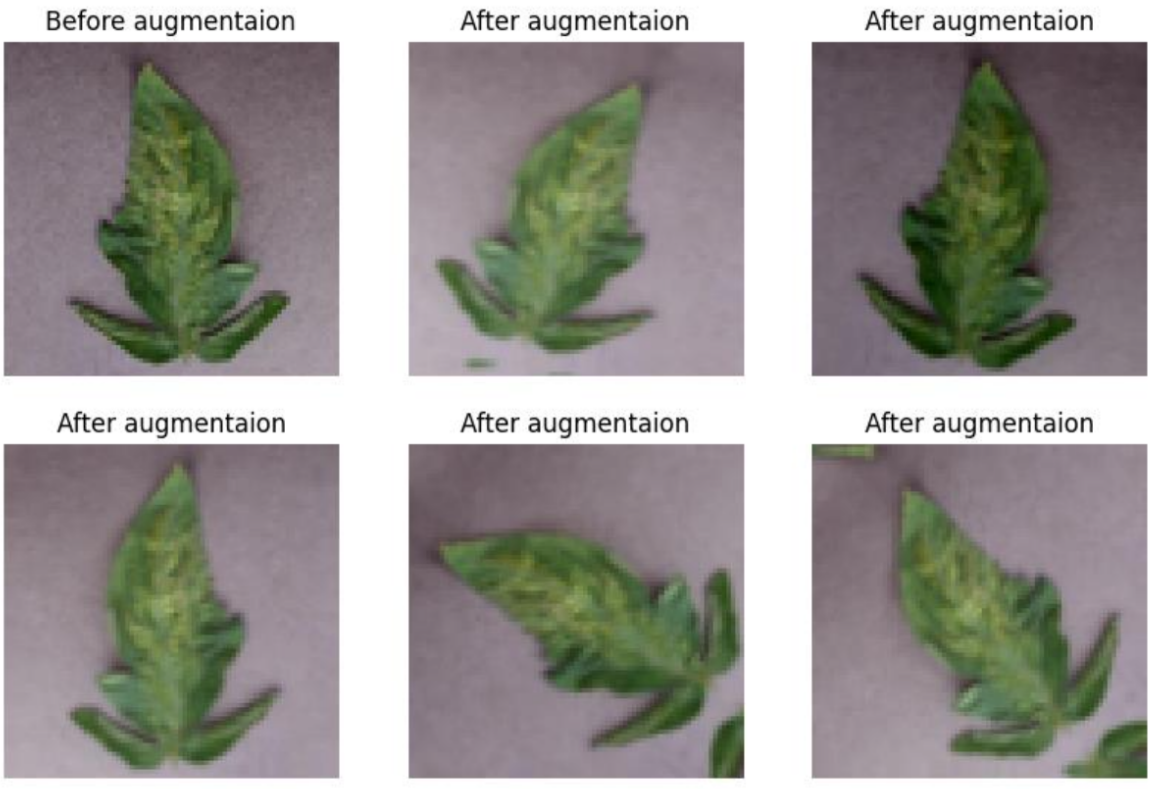

Figure 4: A few examples of image outputs after passing through the augmentation layer.

1.3.4 Model Training

In the paper, the details and hyperparameters of the ViT training are described. We also, similar to the paper, use a learning rate of 0.001 along with a weight decay of 0.0001 and the Adam optimizer. We also choose the sparse categorical cross-entropy loss function similar to the paper. Finally, we train the model with batches of size 256 for 30 epochs, similar to the paper.

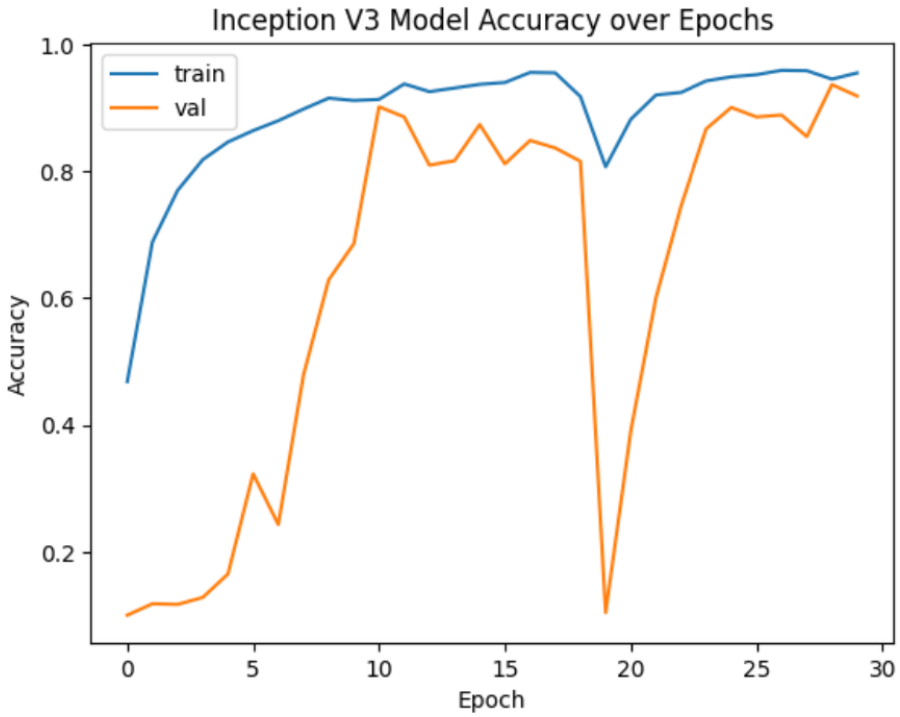

Figure 5: Accuracy changes of the Inception V3 model during training.

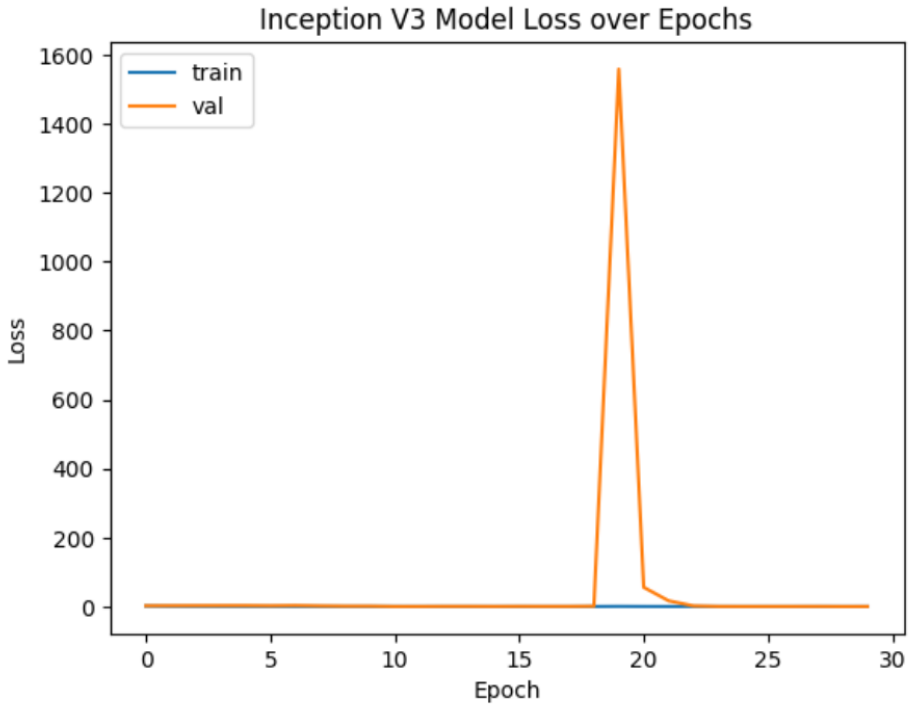

Figure 6: Loss changes of the Inception V3 model during training.

Initially, the model acts randomly on the validation data, and its accuracy is 10

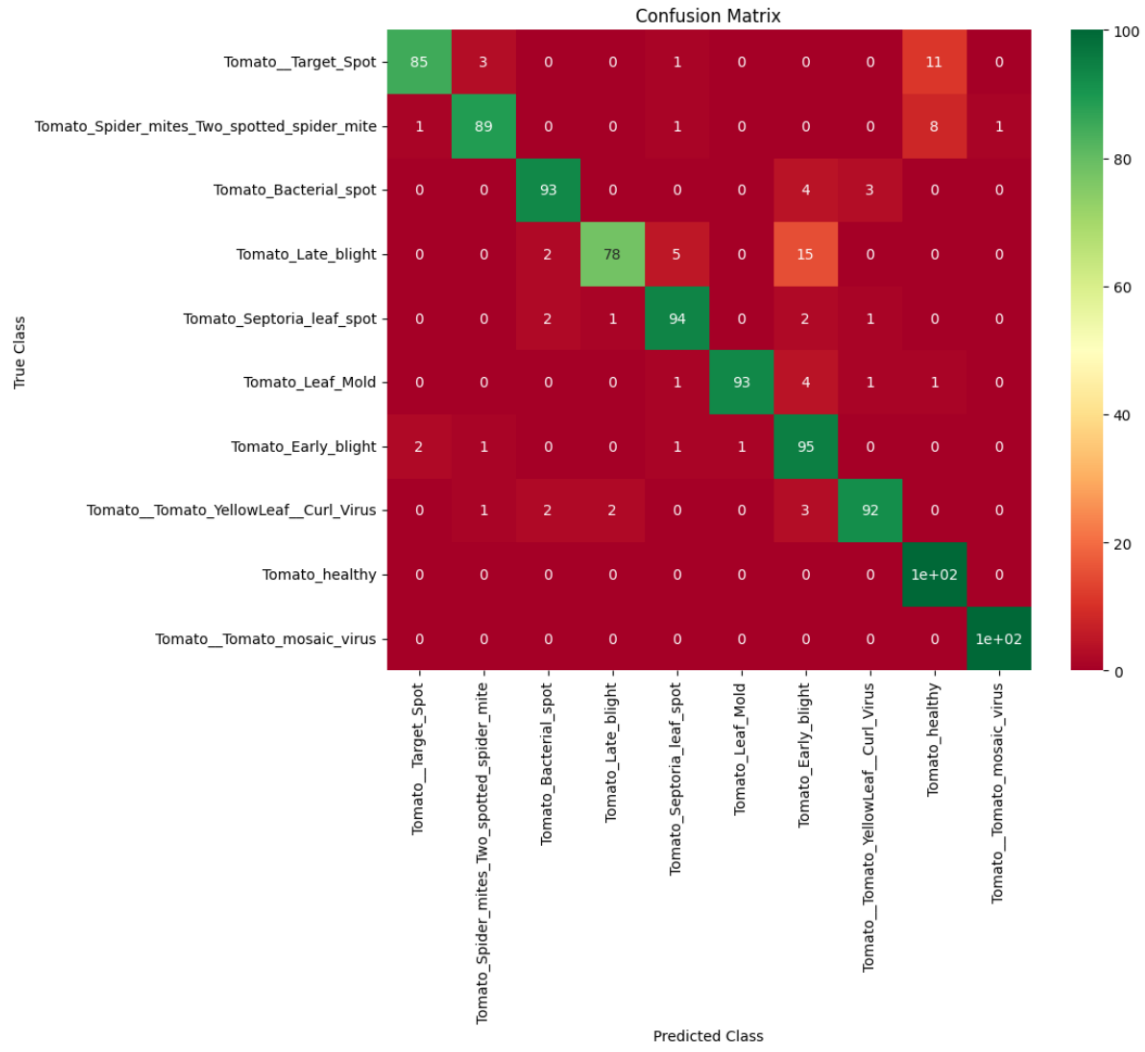

Figure 7: Confusion matrix of the Inception V3 model.

From the matrix above, it is clear for which classes the most errors occurred. It can be seen that the two classes, the fifth and the last, which had fewer images and their number of images was increased with oversampling, have very few errors, which suggests that the method used with the help of augmentation layers to balance the classes has produced different samples and has caused the model to learn the patterns of the images of the under-represented classes as well as other classes. The most error is between the two diseases Early blight and Late blight, which both have a similar nature. Also, errors exist in the diagnosis of the first two diseases.

Table 2: Evaluation metrics of the Inception V3 model.

Class	Accuracy	Precision	Recall	F1 score
Micro	0.919	0.9190	0.9190	0.9190
Macro	0.919	0.9261	0.9261	0.9193
Tomato Target Spot	0.850	0.9659	0.8500	0.9043
Tomato Spider mites	0.890	0.9468	0.8900	0.9175
Tomato Bacterial spot	0.930	0.9394	0.9300	0.9347
Tomato Late blight	0.780	0.9630	0.7800	0.8619
Tomato Septoria leaf spot	0.940	0.9126	0.9400	0.9261
Tomato Leaf Mold	0.930	0.9894	0.9300	0.9588
Tomato Early blight	0.950	0.7724	0.9500	0.8520
Tomato YellowLeaf Curl Virus	0.920	0.9485	0.9200	0.9340
Tomato healthy	1.000	0.8333	1.0000	0.9091
Tomato mosaic virus	1.000	0.9901	1.0000	0.9950

In Table 2, the performance of the model can be seen for each class and overall with two micro and macro averaging methods. It can be seen that the average performance of the model on all metrics is greater than 0.77. For most classes, the metric values are above 0.9.

It can also be seen that the model, with complete recall of being healthy, diagnoses the plant, meaning there is no case where it is healthy and the model mistakenly diagnoses it as sick. Of course, for health, the precision metric is more important because it shows the errors where the model mistakenly diagnoses a sick plant as healthy, and thus, by trusting the model, the farmer suffers a loss. In contrast, if it mistakenly diagnoses a healthy plant as sick, by consulting specialists, the treatment methods can be prevented, and less damage is incurred. In the next part, we try to improve the quality of classification by using the attention mechanism. Also, the error rate of the model on the class that had very few images is very low, so that it has complete recall and 99

1.4 ViT Model

1.4.1 Model Analysis

The ViT model has shown remarkable performance in various fields and has even in some cases shown better performance than the common models in computer vision, i.e., CNNs. Also, by taking image patches as tokens, it has more scalability because it can process images with different dimensions more easily. According to the paper, a ViT model generally works as follows:

- **Standardizing the input image:** By receiving the input image, we separate fixed-size patches from the image and, by passing the patches through fully connected layers, we consider the resulting one-dimensional vector as the input token for the model.

- **Embedding:** Each vector specific to a patch of the image is converted to an embedding with a fixed dimension, whose values are learned by the model during training.
- **Positional Information:** To diagnose the relative position of the image patch, a positional embedding is used. By considering a similar embedding as before but for each position instead of each type of token and adding the embedding value to the embedding obtained from the previous stage, we arrive at an embedding for each token that includes the meaning and position of the corresponding image patch.
- **Transformer Encoder:** The core of the ViT model is its transformer encoder. This module consists of a number of layers, each layer including an attention mechanism and fully connected networks, and thus the model can learn the relationships between tokens or the same image patches. More precisely, in this module, the tokens first pass through a multi-head attention module, and the resulting embedding vectors are added to the module's input with a skip connection, and the resulting vectors are normalized. Then the resulting vectors, after passing through deep layers, are again added to the results via a skip connection and form the output.
- **Classifier:** After passing through the transformer, a feature vector is created and goes to the fully connected layers to predict the model's class.
- **Training:** The weights used in the embedding and transformer layers are trained with the backpropagation algorithm and SGD or Adam methods.
- **Execution:** The input is given to the model, and after passing through the network, the probability of the input belonging to each class is available in the output.

Table 3: Hyperparameters of the model as mentioned in Table 3 of the paper.

Serial No.	Hyperparameter Name	Value
1	Input size	64
2	Learning rate	0.001
3	Weight decay	0.0001
4	Batch Size	256
5	Transformer Dim	2
6	Projection Layers	8

We also use the first four hyperparameters for training the model, but the meaning of the last two hyperparameters was not clear to us at first. However, by examining the next table, we understood the meaning of these two and will use them.

Table 4: Summary of the model's architecture from Table 4 of the paper.

Layers	Output Shape	Parameters
Add_14 (Add)	(None, 100, 64)	0
Layer Normalization	(None, 100, 64)	128
Dense Layer	(None, 100, 128)	8320
Dropout Layer	(None, 100, 128)	0
Dense Layer	(None, 100, 64)	8256
Dropout Layer	(None, 100, 64)	0
Add	(None, 100, 64)	0
Layer Normalization	(None, 100, 64)	128
Flatten	(None, 6400)	0
Dropout Layer	(None, 100, 64)	0
Dense Layer	(None, 2048)	13,109,248
Dropout Layer	(None, 2048)	2,098,176
Dense Layer	(None, 1024)	0
Dropout Layer	(None, 1024)	10,250
Dense Layer	(None, 10)	

In Table 4, a summary of the model's architecture is provided. By examining this table, we realized that the first layer is named Add_14, and we concluded that since in the TensorFlow framework, which is used in the paper, counting starts from zero, this is the fifteenth adder present in the model. By considering another adder that exists in the table, we have a total of 16 adders, and since the adders are located in the transformer encoder blocks and each block has two adders, one after the attention mechanism and the other after the fully connected layers, we conclude that the hyperparameter Projection layers indicates the number of consecutive transformer blocks. So we will also use 8 consecutive blocks. Also, given that the number of heads of the attention mechanism is not known, we think that Transformer Dim indicates the number of heads of the attention mechanism, which is two.

1.4.2 Patches Class

The initial layer that forms the Patch Embedding model. In this layer, images are divided into fixed-size patches, which are then used as input for the rest of the parts. By dividing the images into smaller patches, the model can take larger images as input. In general, there are two methods for implementing this layer:

- In the first method, a convolutional layer is used to extract patches from images. This method is more efficient computationally, especially for large images, because it uses the parallelization capability of the GPU, but it reduces the model's interpretability.

- The other method is to separate the patches by dividing the image with a grid and considering each cell as a patch. Although this method is justifiable, it is more costly computationally.

The paper refers to the use of the 'extract$_{p}atches$' $function from the TensorFlow framework, which uses$

In the first ViT model, 16x16 patches were used as input. By considering large patch dimensions, we lose some information at the pixel level. In some models, methods for using patches with different dimensions were presented to solve this problem. In some other models, with the help of convolutional layers, the features that are lost due to considering large patches were tried to be taken into account.

Another problem with large patches is the addition of inductive bias. A special type of bias that is strongly present in convolutional networks assumes that there is spatial invariance in images, and with this assumption, by moving the same kernels throughout the image, features can be extracted. Of course, in these models, there are many other biases, such as the assumption that with pooling layers, essential and important features can be obtained, the use of small-sized filters is sufficient to obtain the required features, and many other assumptions are also considered. Initially, these biases, due to human intuition about image features, seem problem-free and necessary, but in terms of generalizability, it is better to reduce the model's biases as much as possible so that in the presence of any pattern, the model itself can learn them to avoid the presence of incorrect assumptions.

In the ViT architecture, there is less inductive bias compared to convolutional networks, and the model itself learns the relationships between patches. But still, due to the use of image patches, this bias exists to some extent, and the larger the patches are considered, this bias also comes more into the model's predictions. Of course, in modified ViT models, by using patches with different dimensions or other changes, the introduction of this bias into the model is prevented.

Thus, it can be said that by considering small patches, we reduce the inductive bias in the model, and thus the model itself can learn the relationships and more detailed features correctly and achieve better performance. But on the other hand, the smaller the patch dimensions, the more patches are extracted from the image, and larger images cannot be processed. Also, with small patches, less meaning can be extracted from the patches, and thus a large part of the learning burden is removed from the embedding layer and transferred to the attention mechanism.

To better understand the effect of patch size, for a sample of images, we plot the patches with dimensions of 4, 6, and 8 pixels.

Figure 8: A sample image.

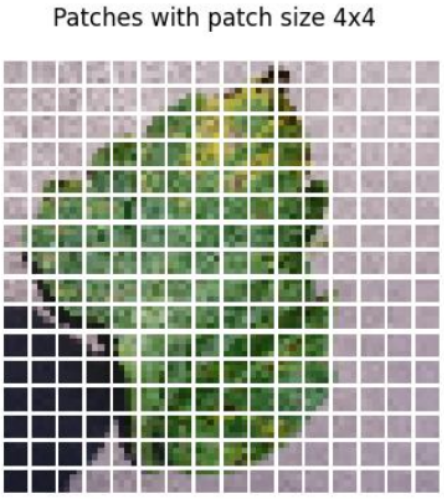

Figure 9: Image patches with size 4 pixels.

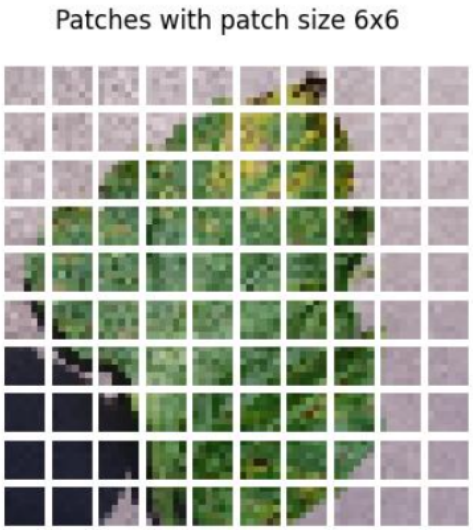

Figure 10: Image patches with size 6 pixels.

Figure 11: Image patches with size 8 pixels.

From the images above, it can be seen that the first image can be divided into 256 4-pixel patches, 100 6-pixel patches, or 64 8-pixel patches. It is also clear that in the 8-pixel patches, on the edges of the leaf, half of the patch is occupied by the surrounding environment, and also in the yellow spots, parts of the green color without disease symptoms are included. Whereas it can be seen that in the 4-pixel patches, each yellow part is almost in a separate patch, and on the edges of the leaf, most of the patch is occupied by the leaf or by the environment, and thus it seems that each patch contains a more uniform meaning.

1.4.3 Model Creation

After creating the different modules, we connect them to create the final model. First, as stated in the paper and like the previous model, we pass the images through the

augmentation layer to create new samples with changes. After passing the image through Patches and passing the obtained patches through PatchEncoder, the embeddings are given to the first transformer block. In total, eight transformer blocks are placed consecutively, and after eight transformer layers, the output of the last block goes to the flatten layer to be converted to a one-dimensional vector for all tokens. Then the resulting vector, after passing through a dropout, goes to a fully connected layer with 2048 neurons and then another dropout and then a fully connected layer with 1024 neurons, and the output after passing through the last dropout goes to 10 neurons to predict the image class.

In all fully connected layers, the Relu activation function is used, except for the last layer, which, according to the paper, we used softmax to obtain the probability of each class. We also set the rate of all dropouts to 0.1.

Table 5: Summary of the implemented ViT model architecture.

Layer (type)	Output Shape	Param #
input_layer_16 (InputLayer)	(None, 64, 64, 3)	0
Augmentation_Layers (Sequential)	(None, 64, 64, 3)	0
patches_16 (Patches)	(None, 100, 108)	0

Table 6: Summary of the implemented ViT model architecture.

patch_encoder_7 (PatchEncoder)	(None, 100, 64)	13,376
transformer_blocks * 8 (TransformerBlock)	(None, 100, 64)	50,048
flatten_7 (Flatten)	(None, 6400)	0
dropout_213 (Dropout)	(None, 6400)	0
dense_160 (Dense)	(None, 2048)	13,109,248
dropout_214 (Dropout)	(None, 2048)	0
dense_161 (Dense)	(None, 1024)	2,098,176
dropout_215 (Dropout)	(None, 1024)	0
dense_162 (Dense)	(None, 10)	10,250

Total params: 15,631,434 (59.63 MB)
Trainable params: 15,631,434 (59.63 MB)
Non-trainable params: 0 (0.00 B)

Table 7: Summary of the transformer block architecture.

Layer (type)	Output Shape	Param #
multi_head_attention (MultiHeadAttention)	(256, 100, 64)	33,216
layer_normalization (LayerNormalization)	(256, 100, 64)	128
dense (Dense)	(256, 100, 128)	8,320
dropout (Dropout)	?	0
dense_1 (Dense)	(256, 100, 64)	8,256
dropout_1 (Dropout)	?	0
layer_normalization_1 (LayerNormalization)	(256, 100, 64)	128

Total params: 50,048 (195.50 KB)
Trainable params: 50,048 (195.50 KB)
Non-trainable params: 0 (0.00 B)

1.4.4 Model Training

Now that we have ensured the similarity of the model with the paper's model, we load and divide the images again with a size of 64 pixels to train the model for 30 epochs with the sparse categorical cross-entropy loss function and the Adam optimizer and a batch size of 256.

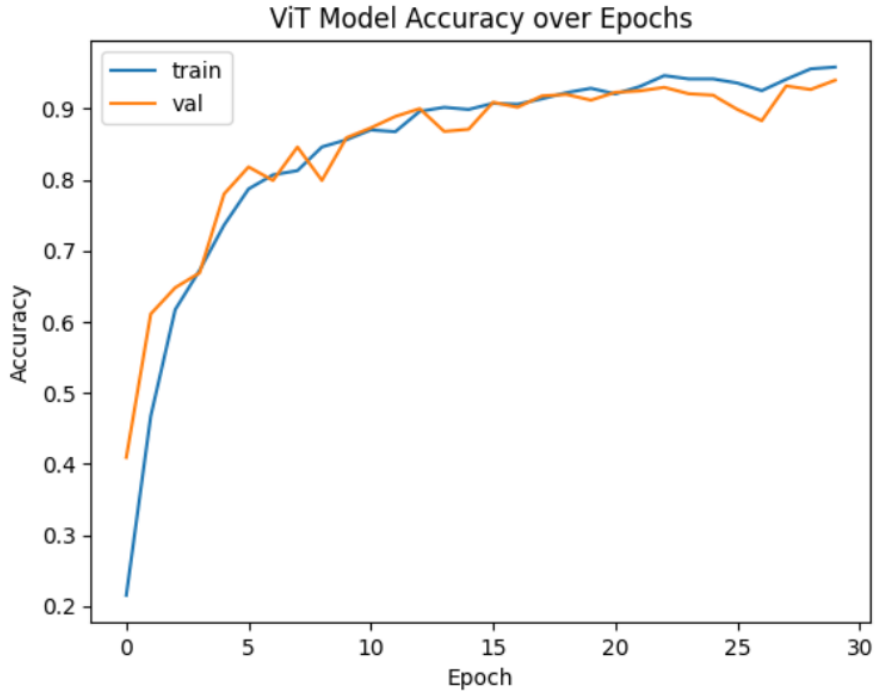

Figure 12: Accuracy changes of the ViT model during training.

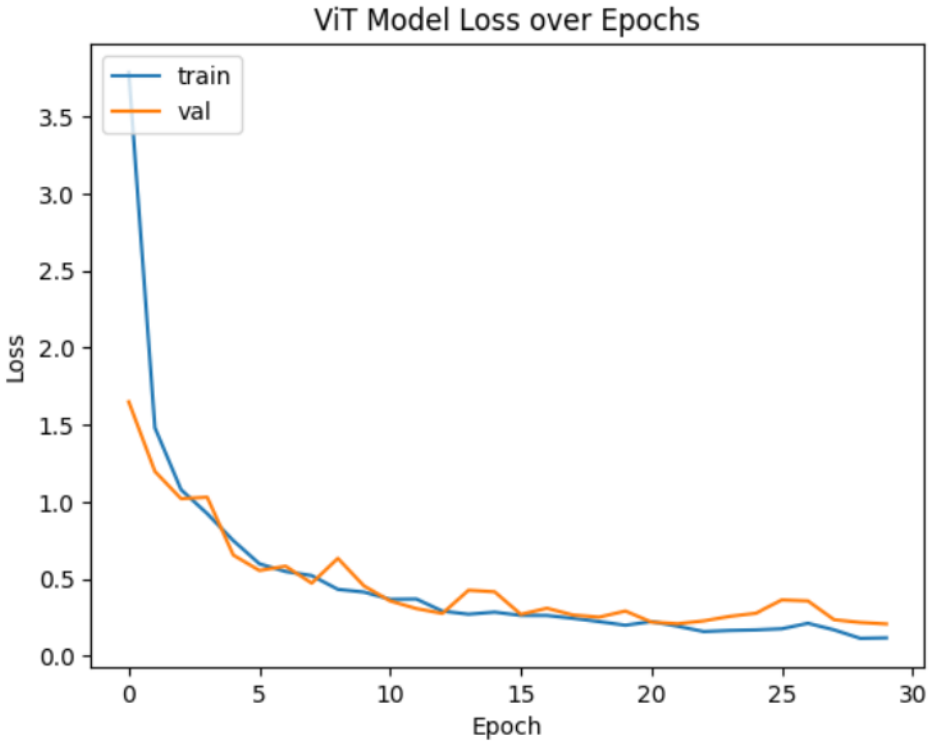

Figure 13: Loss changes of the ViT model during training.

The model performs better than the previous model in the first few epochs, and although the previous model still had 10

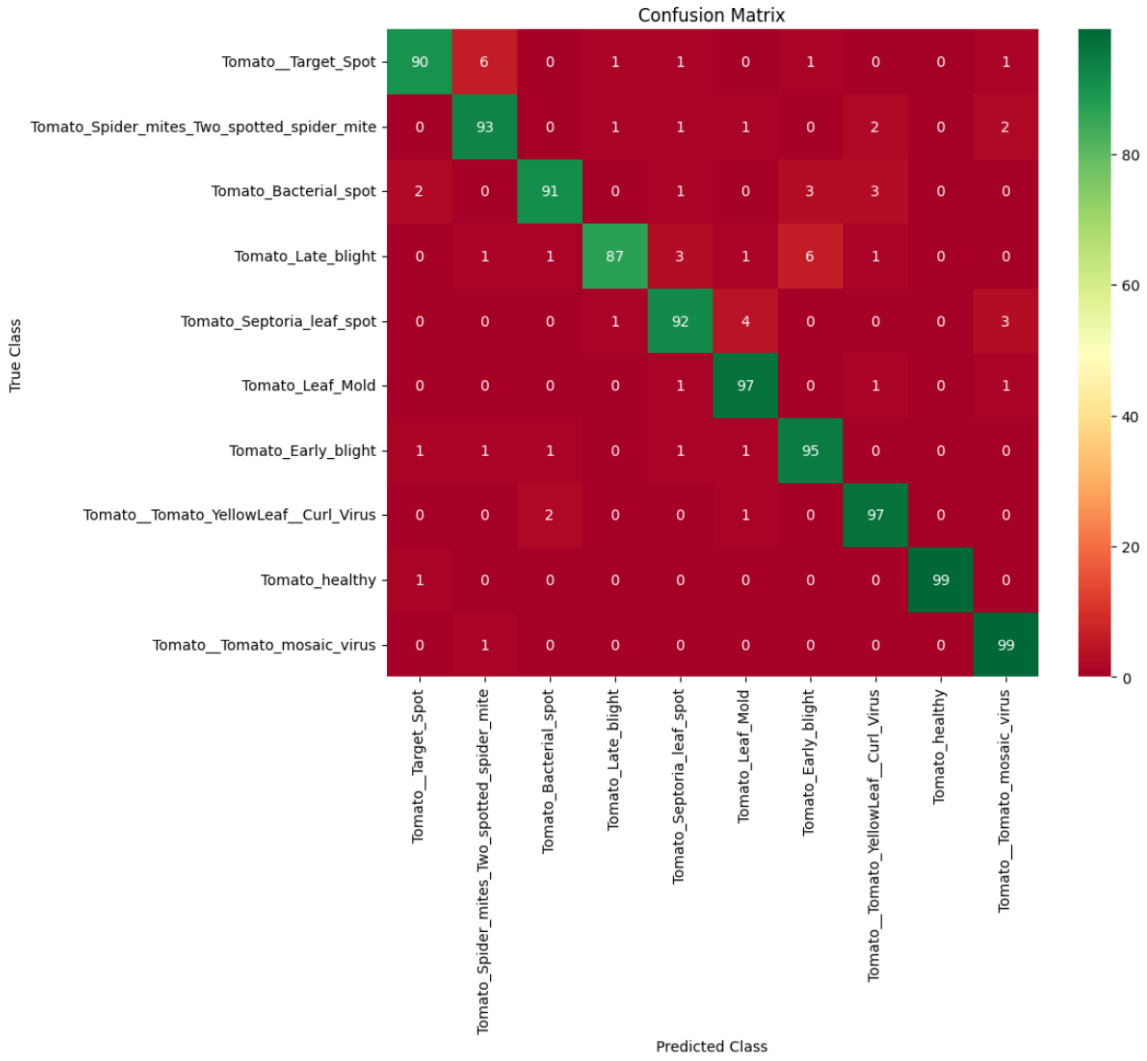

Figure 14: Confusion matrix of the ViT model.

From the confusion matrix above, the improved performance of the model compared to the previous model is evident, so that in the classes where the previous model performed poorly, this model has fewer errors, and the most error is still between the two classes Late blight and Early blight, but unlike before, where there were 15 errors, in this model, only 6 errors exist between these two classes.

Table 8: Evaluation metrics for the ViT model.

Class	Accuracy	Precision	Recall	F1 score
Micro	0.94	0.9400	0.9400	0.9400
Macro	0.94	0.9409	0.9409	0.9398
Tomato Target Spot	0.90	0.9574	0.9000	0.9278
Tomato Spider mites	0.93	0.9118	0.9300	0.9208
Tomato Bacterial spot	0.91	0.9579	0.9100	0.9333
Tomato Late blight	0.87	0.9667	0.8700	0.9158
Tomato Septoria leaf spot	0.92	0.9200	0.9200	0.9200
Tomato Leaf Mold	0.97	0.9238	0.9700	0.9463
Tomato Early blight	0.95	0.9048	0.9500	0.9268
Tomato YellowLeaf Curl Virus	0.97	0.9327	0.9700	0.9510
Tomato healthy	0.99	1.0000	0.9900	0.9950
Tomato mosaic virus	0.99	0.9340	0.9900	0.9612

From the table above, the much better performance of the model compared to the previous model is evident. Although in the previous model, the prediction of the weakest class had an accuracy of 0.77, in this model, the weakest recall metric for the Late blight class is 0.87. Also, it can be seen that the evaluation metrics for the healthy class are greater than 0.99, and the model's precision in diagnosing health has increased, meaning there is no image that indicates a sick plant and the model mistakenly diagnoses it as healthy, which means reducing losses and diagnosing pests and diseases in 100

1.5 Analysis and Conclusion

By examining the confusion matrix and evaluation metrics, it is clear that the ViT model has performed much better than the Inception V3 model, so that it can distinguish all classes with an accuracy of over 90 percent. This is while the number of parameters of the ViT is about 15.6 million and the number of parameters of Inception V3 is about 21.8 million. Also, in general, with both micro and macro averaging methods, all four metrics have about two to three percent more value for the ViT model. The possible reasons for the better performance of the model were examined in the model's components.

According to similar works, it has been shown that usually if the number of data is small and cannot be increased with methods, the performance of convolutional models due to inductive bias and assumptions about images can be better. Also, if the images have little complexity and the assumptions considered in convolutional networks are valid, which is usually the case, convolutional models can also achieve suitable performance. But in general, models based on the attention mechanism are more powerful, and even if the assumptions present in convolutional architectures are not valid, these models, in the presence of sufficient and high-quality data, can learn the patterns.

1.6 Summary

In this exercise, we tried to train a raw Inception V3 model as a baseline model and then train a ViT model based on the attention mechanism on the images of tomato plant leaves to diagnose 9 different diseases, so that we can show that with the help of deep models, pests and plant diseases can be prevented.

The available images included unbalanced classes, with one of the classes having one-third the images of the rest. We first equalized the number of images in all classes with the oversampling method and, by placing augmentation layers in the models, applied changes to the images to prevent overfitting.

In general, we tried to proceed exactly like the paper, and for this reason, we divided the data into two parts, training and validation, to use the hyperparameters present in the paper. For preprocessing the images, we normalized them between zero and one and then used the mean and variance of the training images to scale all the images.

We also examined the differences, advantages, and disadvantages of the ViT model compared to convolutional networks during implementation and finally analyzed and compared the results. By examining the results, we realized that, as stated in the paper, the use of ViT led to improved diagnoses, so that in all validation images, including the sick plant leaf, the model was able to detect the presence of the disease.