Vision Transformer Internship Project

Abstract:

The Vision Transformer (ViT) project aims to explore and advance the application of transformer-based architectures in computer vision tasks. Traditional convolutional neural networks (CNNs) have dominated the field of computer vision due to their capability to effectively process grid-structured data like images. However, transformers, which have revolutionized natural language processing, offer a promising alternative due to their self-attention mechanisms that can capture long-range dependencies in data.

Objective:

1. Understanding the Basics of Vision Transformers (ViTs)

- **Study the Theory**: Learn about the architecture and components of Vision Transformers, including self-attention mechanisms, positional encodings, and transformer blocks.
- **Literature Review**: Read and summarize key papers in the field, starting with "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" by Dosovitskiy et al.

2. Implementation and Experimentation

- **Reproduce Results**: Implement a basic Vision Transformer model from scratch or using frameworks like PyTorch or TensorFlow.
- **Train on Standard Datasets**: Train the ViT model on standard image datasets such as CIFAR-10, CIFAR-100, or ImageNet and evaluate its performance.
- **Hyperparameter Tuning**: Experiment with different hyperparameters (e.g., learning rate, batch size, number of transformer layers) to optimize the model's performance.

3. Advanced Techniques and Enhancements

- **Data Augmentation**: Implement various data augmentation techniques to improve model generalization.

- **Transfer Learning**: Explore transfer learning by fine-tuning a pre-trained Vision Transformer on a specific dataset.
- **Comparison with CNNs**: Compare the performance of Vision Transformers with Convolutional Neural Networks (CNNs) on similar tasks.

4. Real-world Applications

- **Custom Dataset**: Apply the Vision Transformer model to a real-world dataset relevant to the intern's interests or the organization's needs.
- **Performance Metrics**: Develop metrics to evaluate the model's performance in the context of the chosen application (e.g., accuracy, precision, recall, F1-score).

5. Optimization and Deployment

- **Model Optimization**: Implement techniques to optimize the model for inference, such as quantization or pruning.
- **Deployment**: Develop a pipeline for deploying the trained Vision Transformer model to a production environment, possibly using cloud services or edge devices.

6. Documentation and Reporting

- **Documentation**: Maintain thorough documentation of the code, experiments, and results.
- **Final Report**: Prepare a comprehensive report detailing the project objectives, methodology, experiments, results, and conclusions.
- **Presentation**: Present the findings and outcomes of the project to peers and mentors.

7. Collaboration and Learning

- **Team Collaboration**: Work collaboratively with other interns or team members, participating in regular meetings and code reviews.
- **Mentorship**: Seek guidance and feedback from mentors throughout the project.
- **Continuous Learning**: Stay updated with the latest research and advancements in the field of Vision Transformers and machine learning.

Introduction:

Overview

The Vision Transformer (ViT) Internship Project is designed to immerse interns in the cutting-edge field of computer vision through the lens of transformer-based models. Transformers, originally introduced for natural language processing, have shown remarkable potential in image recognition tasks, challenging the dominance of Convolutional Neural Networks (CNNs). This project will provide a comprehensive learning experience, enabling interns to understand, implement, and innovate with Vision Transformers.

Background

Vision Transformers were introduced in the seminal paper "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" by Dosovitskiy et al. Unlike CNNs, which rely on convolutional layers to extract local features from images, Vision Transformers leverage self-attention mechanisms to capture global dependencies. This paradigm shift has opened new avenues for research and application in computer vision.

Methodologies:

The methodology for the Vision Transformer Internship Project encompasses a series of systematic steps aimed at ensuring a thorough understanding and effective implementation of Vision Transformers. The methodology is divided into several phases, each with specific tasks and deliverables.

Phase 1: Initial Training and Research

Objective: Build a strong foundation in Vision Transformers.

1. **Literature Review:**
    - Read foundational papers such as "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" by Dosovitskiy et al.
    - Summarize key concepts, findings, and methodologies from these papers.
    - Explore additional research articles and reviews to understand the evolution and current trends in Vision Transformers.
2. **Theoretical Study:**
    - Study the architecture of Vision Transformers, including self-attention mechanisms, positional encoding, and transformer blocks.

    - Compare Vision Transformers with traditional Convolutional Neural Networks (CNNs) to understand their advantages and limitations.
3. **Tools and Frameworks:**
    - Familiarize with machine learning frameworks such as PyTorch or TensorFlow.
    - Set up the development environment, including necessary libraries and tools.

Deliverables:

- Literature review summary.
- Detailed notes on the theoretical concepts of Vision Transformers.
- A setup guide for the development environment.

Phase 2: Model Implementation and Baseline Experiments

Objective: Implement and train a basic Vision Transformer model.

1. **Dataset Preparation:**
    - Select standard datasets such as CIFAR-10, CIFAR-100, or ImageNet.
    - Preprocess the data, including normalization and augmentation techniques.
2. **Model Implementation:**
    - Implement a basic Vision Transformer model from scratch or adapt an existing implementation.
    - Ensure the model includes essential components such as input patch embedding, transformer blocks, and classification heads.
3. **Training and Evaluation:**
    - Train the Vision Transformer model on the selected datasets.
    - Evaluate the model using metrics such as accuracy, precision, recall, and F1-score.
4. **Baseline Performance Analysis:**
    - Analyze the baseline performance of the model.
    - Identify potential areas for improvement.

Deliverables:

- A working implementation of the basic Vision Transformer model.
- Training and evaluation scripts.
- Baseline performance report.

Phase 3: Advanced Experimentation and Application

Objective: Explore advanced techniques and apply the model to real-world datasets.

1. **Hyperparameter Tuning:**
   - Experiment with different hyperparameters (learning rate, batch size, number of layers, etc.).
   - Use techniques such as grid search or random search to find optimal hyperparameters.
2. **Data Augmentation:**
   - Implement various data augmentation techniques to improve model generalization.
   - Compare the performance of the model with and without augmentation.
3. **Transfer Learning:**
   - Explore transfer learning by fine-tuning a pre-trained Vision Transformer on a specific dataset.
   - Evaluate the benefits of transfer learning compared to training from scratch.
4. **Application to Custom Dataset:**
   - Select a real-world dataset relevant to the intern's interests or the organization's needs.
   - Adapt and train the Vision Transformer model on this dataset.
   - Analyze the performance and effectiveness of the model in the specific application.

Deliverables:

- Report on hyperparameter tuning results.
- Implementation and evaluation of data augmentation techniques.
- Transfer learning experiments and results.
- Application-specific model and performance analysis.

Phase 4: Optimization and Deployment

Objective: Optimize the model for deployment and implement a deployment pipeline.

1. **Model Optimization:**
   - Implement optimization techniques such as quantization, pruning, or knowledge distillation to reduce model size and improve inference speed.
   - Evaluate the impact of these optimizations on model performance.
2. **Deployment Pipeline:**
   - Develop a pipeline for deploying the trained model in a production environment.

    - Use cloud services or edge devices as required by the application.
    - Ensure the deployment pipeline includes steps for continuous monitoring and maintenance.

Deliverables:

- Optimized model and performance evaluation.
- Documentation of the deployment pipeline.
- Deployed model in a test or production environment**.**

Phase 5: Documentation and Presentation

Objective: Document all aspects of the project and prepare for final presentation.

1. **Comprehensive Documentation:**
    - Maintain detailed documentation of the code, experiments, and results throughout the project.
    - Ensure documentation includes explanations of methodologies, parameters, and findings.
2. **Final Report:**
    - Prepare a comprehensive report detailing the project objectives, methodology, experiments, results, and conclusions.
    - Include visualizations, charts, and graphs to illustrate key points.
3. **Presentation:**
    - Prepare a presentation summarizing the project.
    - Highlight key achievements, learnings, and future directions.

Deliverables:

- Complete project documentation.
- Final project report.
- Presentation slides and materials.

Phase 6: Collaboration and Continuous Learning

Objective: Foster collaboration and continuous learning throughout the internship.

1. **Team Collaboration:**
    - Participate in regular meetings and code reviews with other interns and team members.
    - Share findings and insights with the team.
2. **Mentorship:**
    - Seek guidance and feedback from mentors throughout the project.
    - Schedule regular check-ins to discuss progress and challenges.
3. **Continuous Learning:**

- Stay updated with the latest research and advancements in Vision Transformers and machine learning.
- Participate in relevant workshops, webinars, and conferences if possible.

Deliverables:

- Meeting notes and action items.
- Feedback and improvements based on mentor guidance.
- Evidence of continuous learning activities (e.g., attended webinars, read papers).

Code:

!pip install tensorflow==2.8.0

!pip install keras==2.8.0

!pip install tensorflow-addons==0.17.0

#above instead of tensorflow-addons==0.17.0 we can even use tensorflow-addons==0.20.0

OUTPUT:

```
  Downloading google_auth_oauthlib-0.4.6-py2.py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.9,>=2.8->tensorflow==2.8.0) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.9,>=2.8->tensorflow==2.8.0) (2.31.0)
Collecting tensorboard-data-server<0.7.0,>=0.6.0 (from tensorboard<2.9,>=2.8->tensorflow==2.8.0)
  Downloading tensorboard_data_server-0.6.1-py3-none-manylinux2010_x86_64.whl.metadata (1.1 kB)
Collecting tensorboard-plugin-wit>=1.6.0 (from tensorboard<2.9,>=2.8->tensorflow==2.8.0)
  Downloading tensorboard_plugin_wit-1.8.1-py3-none-any.whl.metadata (873 bytes)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.9,>=2.8->tensorflow==2.8.0) (3.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (5.4.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tensorflow==2.8.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (2024.7.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=0.11.15->tensorboard<2.9,>=2.8->tensorflow==2.8.0) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow=
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->te
```

```
Collecting tensorflow==2.8.0
  Downloading tensorflow-2.8.0-cp310-cp310-manylinux2010_x86_64.whl.metadata (2.9 kB)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (1.6.3)
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (24.3.25)
Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (3.11.0)
Collecting keras-preprocessing>=1.1.1 (from tensorflow==2.8.0)
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (18.1.1)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (3.3.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (1.14.1)
Collecting tensorboard<2.9,>=2.8 (from tensorflow==2.8.0)
  Downloading tensorboard-2.8.0-py3-none-any.whl.metadata (1.9 kB)
Collecting tf-estimator-nightly==2.8.0.dev2021122109 (from tensorflow==2.8.0)
  Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting keras<2.9,>=2.8.0rc0 (from tensorflow==2.8.0)
  Downloading keras-2.8.0-py2.py3-none-any.whl.metadata (1.3 kB)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (0.37.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.8.0) (1.64.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow==2.8.0) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.9,>=2.8->tensorflow==2.8.0) (2.27.0)
```

```
  Attempting uninstall: tensorboard
    Found existing installation: tensorboard 2.15.2
    Uninstalling tensorboard-2.15.2:
      Successfully uninstalled tensorboard-2.15.2
  Attempting uninstall: tensorflow
    Found existing installation: tensorflow 2.15.0
    Uninstalling tensorflow-2.15.0:
      Successfully uninstalled tensorflow-2.15.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
pandas-gbq 0.19.2 requires google-auth-oauthlib>=0.7.0, but you have google-auth-oauthlib 0.4.6 which is incompatible.
tf-keras 2.15.1 requires tensorflow<2.16,>=2.15, but you have tensorflow 2.8.0 which is incompatible.
Successfully installed google-auth-oauthlib-0.4.6 keras-2.8.0 keras-preprocessing-1.1.2 tensorboard-2.8.0 tensorboard-data-server-0.6.1 tensorboard-plugin-wit-1.8.1 tensorflow-2.8.0
Requirement already satisfied: keras==2.8.0 in /usr/local/lib/python3.10/dist-packages (2.8.0)
Collecting tensorflow-addons==0.17.0
  Downloading tensorflow_addons-0.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (1.8 kB)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow-addons==0.17.0) (24.1)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.10/dist-packages (from tensorflow-addons==0.17.0) (4.3.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.10/dist-packages (from typeguard>=2.7->tensorflow-addons==0.17.0) (4.12.2)
Downloading tensorflow_addons-0.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 3.6 MB/s eta 0:00:00
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.17.0
```

```
Downloading tensorflow-2.8.0-cp310-cp310-manylinux2010_x86_64.whl (497.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 497.6/497.6 MB 3.4 MB/s eta 0:00:00
Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl (462 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 462.5/462.5 kB 24.0 MB/s eta 0:00:00
Downloading keras-2.8.0-py2.py3-none-any.whl (1.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 37.6 MB/s eta 0:00:00
Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 1.5 MB/s eta 0:00:00
Downloading tensorboard-2.8.0-py3-none-any.whl (5.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 48.5 MB/s eta 0:00:00
Downloading google_auth_oauthlib-0.4.6-py2.py3-none-any.whl (18 kB)
Downloading tensorboard_data_server-0.6.1-py3-none-manylinux2010_x86_64.whl (4.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.9/4.9 MB 51.1 MB/s eta 0:00:00
Downloading tensorboard_plugin_wit-1.8.1-py3-none-any.whl (781 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 781.3/781.3 kB 31.8 MB/s eta 0:00:00
Installing collected packages: tf-estimator-nightly, tensorboard-plugin-wit, keras, tensorboard-data-server, keras-preprocessing, google-auth-oauthlib, tensorboard, tensorflow
  Attempting uninstall: keras
    Found existing installation: keras 2.15.0
    Uninstalling keras-2.15.0:
      Successfully uninstalled keras-2.15.0
  Attempting uninstall: tensorboard-data-server
    Found existing installation: tensorboard-data-server 0.7.2
    Uninstalling tensorboard-data-server-0.7.2:
      Successfully uninstalled tensorboard-data-server-0.7.2
  Attempting uninstall: google-auth-oauthlib
    Found existing installation: google-auth-oauthlib 1.2.1
    Uninstalling google-auth-oauthlib-1.2.1:
      Successfully uninstalled google-auth-oauthlib-1.2.1
  Attempting uninstall: tensorboard
    Found existing installation: tensorboard 2.15.2
    Uninstalling tensorboard-2.15.2:
      Successfully uninstalled tensorboard-2.15.2
```

```
#import libraries
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
```

```
num_classes=10
input_shape=(32,32,3)
(x_train,y_train),(x_test,y_test) = keras.datasets.cifar10.load_data()
print(f"x_train shape: {x_train.shape} - y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test.shape} - y_test shape: {y_test.shape}")
print(train_data)
```

OUTPUT:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 10s 0us/step
170508288/170498071 [==============================] - 10s 0us/step
x_train shape: (50000, 32, 32, 3) - y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3) - y_test shape: (10000, 1)
```

```
x_train = x_train[:500]
y_train = y_train[:500]
x_test = x_test[:500]
y_test = y_test[:500]

learning_rate = 0.001
weight_decay = 0.0001#1e-4
batch_size = 256
num_epochs = 40  #40
image_size = 72 #resize the input image to this size
patch_size = 6 #size of the patches to be extracted from the input images
num_patches = (image_size // patch_size) ** 2
num_heads = 4
projection_dim = 64
transformer_units = [projection_dim * 2, projection_dim]
```

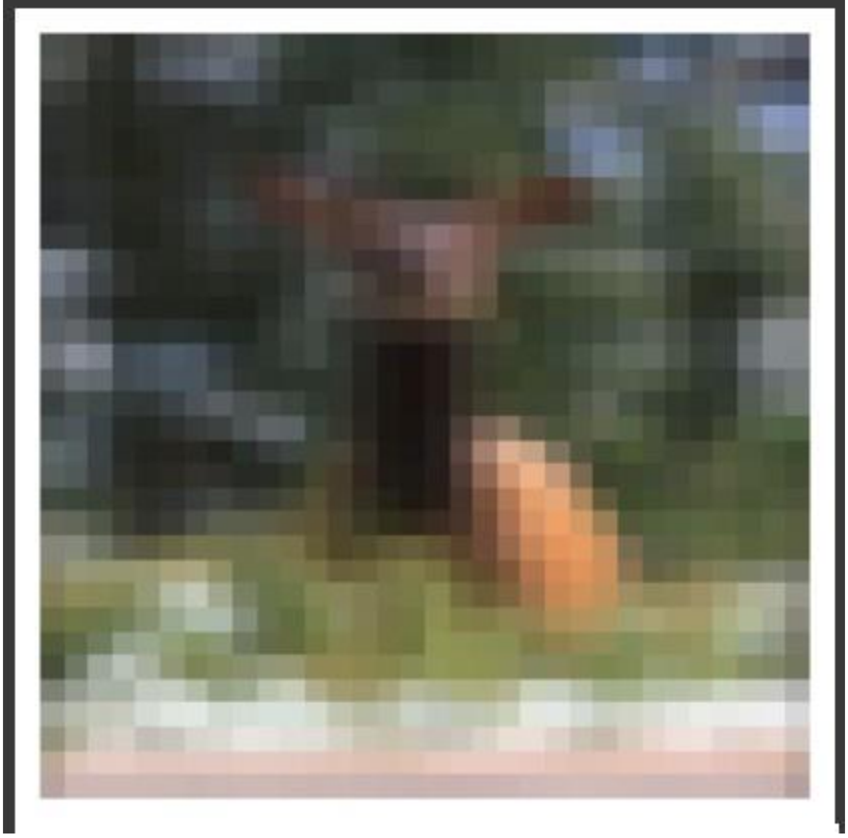
```
#size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024] #size of the dense layers of the final classifiers

data_augumentation = keras.Sequential(
    [
    layers.Normalization(),
    layers.Resizing(image_size, image_size),
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(factor=0.02),
    layers.RandomZoom(height_factor=0.2, width_factor=0.2)
    ],
    name="data_augmentation"
)
data_augumentation.layers[0].adapt(x_train)



def mlp(x,hidden_units,dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units,activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x

class Patches(layers.Layer):
    def __init__(self,patch_size):
        super(Patches,self).__init__()
        self.patch_size = patch_size

    def call(self,images):
```

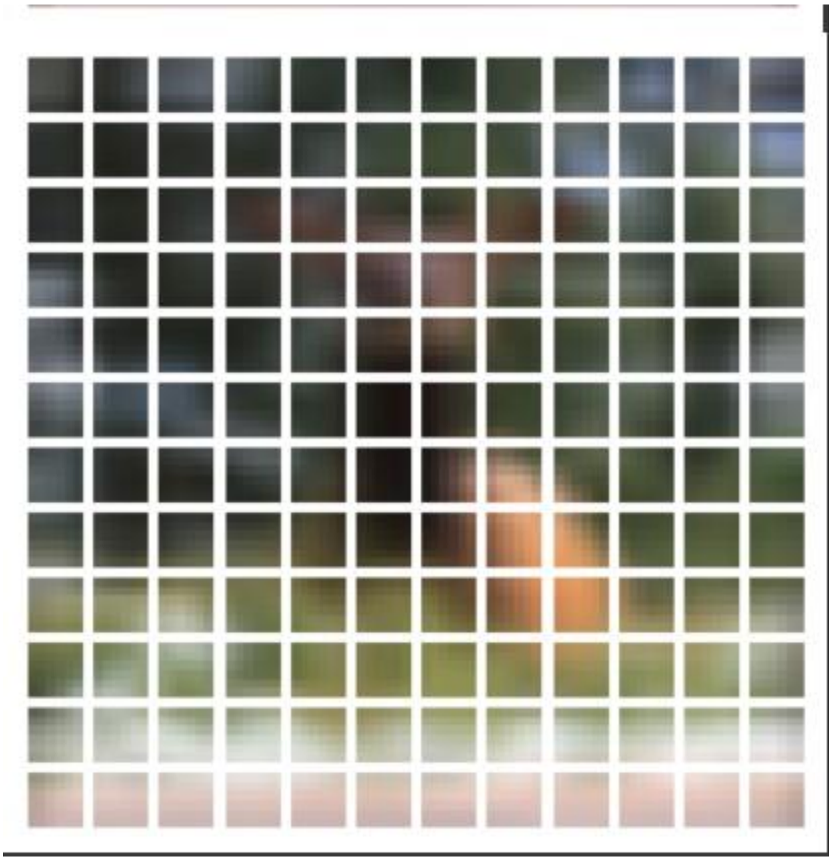
```
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images = images,
            sizes = [1,self.patch_size,self.patch_size,1],
            strides = [1,self.patch_size,self.patch_size,1],
            rates = [1,1,1,1],
            padding = "VALID"
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches,shape=(batch_size, -1, patch_dims))
        return patches

import matplotlib.pyplot as plt

plt.figure(figsize=(4,4))
image = x_train[np.random.choice(range(x_train.shape[0]))]
plt.imshow(image.astype("uint8"))
plt.axis("off")

resized_image = tf.image.resize(
    tf.convert_to_tensor([image]),
    size = (image_size,image_size)
)

patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}\n")
```

```
n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4,4))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n,n,i+1)
    patch_img = tf.reshape(patch, (patch_size,patch_size,3))
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")

  # Adjust these values as needed
plt.show()
```

OUTPUT:

```
Image size: 72 X 72
Patch size: 6 X 6
Patches per image: 144
Elements per patch: 108
```





```
class PatchEncoder(layers.Layer):
    def __init__(self,num_patches,projection_dim):
        super(PatchEncoder,self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim = num_patches,
            output_dim = projection_dim
        )
    def call(self,patches):
        positions = tf.range(start=0,limit=self.num_patches,delta=1)
        encoded = self.projection(patches) + self.position_embedding(positions)
```

```
    return encoded

def create_vit_classifier():
  inputs = layers.Input(shape=input_shape)
  #Augument data
  augmented = data_augumentation(inputs)
  patches = Patches(patch_size)(augmented)
  #encode patches
  encoded_patches = PatchEncoder(num_patches,projection_dim)(patches)

  #create multiple layers of the transformer block
  for _ in range(transformer_layers):
    # layer normalization
    x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    #create multi-head attention layer
    attention_output = layers.MultiHeadAttention(
        num_heads = num_heads,
        key_dim = projection_dim,
        dropout = 0.1
    )(x1,x1)
    #add skip connection1
    x2 = layers.Add()([attention_output,encoded_patches])
    #layer normalization 2
    x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
    #feed forward block mlp
    x3 = mlp(x3,hidden_units=transformer_units,dropout_rate=0.1)
    #add skip connection2
    encoded_patches = layers.Add()([x3,x2])
```

```
    #create a [batch_size,projection_dim] tensor
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)

    #Add mlp
    features = mlp(representation,hidden_units=mlp_head_units,dropout_rate=0.5)
    #Classify outputs
    logits = layers.Dense(num_classes)(features)
    #create model
    model = keras.Model(inputs=inputs,outputs=logits)
    return model

def run_experiment(model):

  optimizer = tfa.optimizers.AdamW(
      learning_rate = learning_rate,
      weight_decay = weight_decay
  )

  model.compile(
      optimizer = optimizer,
      loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True),
      metrics = [
          keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
          keras.metrics.SparseTopKCategoricalAccuracy(5,name="top_5_accuracy"),
      ],
  )
  checkpoint_filepath = "./tmp/checkpoint"
```

```
    checkpoint_callback = keras.callbacks.ModelCheckpoint(
        checkpoint_filepath,
        monitor = "val_accuracy",
        save_best_only = True,
        save_weights_only = True,
    )

    history = model.fit(
        x = x_train,
        y = y_train,
        batch_size = batch_size,
        epochs = num_epochs,
        validation_split = 0.1,
        callbacks = [checkpoint_callback],
    )
    model.load_weights(checkpoint_filepath)
    _, accuracy, top_5_accuracy= model.evaluate(x_test,y_test)
    print(f"Test accuracy:  {round(accuracy*100,2)}%")
    print(f"Test top-5 accuracy:  {round(top_5_accuracy*100,2)}%")

    return history

vit_classifier = create_vit_classifier()
history = run_experiment(vit_classifier)
```

OUTPUT:

```
Epoch 1/40
2/2 [==============================] - 12s 5s/step - loss: 6.3906 - accuracy: 0.1356 - top_5_accuracy: 0.5489 - val_loss: 6.1712 - val_accuracy: 0.2000 - val_top_5_accuracy: 0.6600
Epoch 2/40
2/2 [==============================] - 7s 3s/step - loss: 8.0316 - accuracy: 0.1911 - top_5_accuracy: 0.6356 - val_loss: 4.9357 - val_accuracy: 0.1800 - val_top_5_accuracy: 0.5600
Epoch 3/40
2/2 [==============================] - 9s 3s/step - loss: 5.2170 - accuracy: 0.2467 - top_5_accuracy: 0.6911 - val_loss: 2.8948 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7000
Epoch 4/40
2/2 [==============================] - 10s 7s/step - loss: 3.3788 - accuracy: 0.2622 - top_5_accuracy: 0.7422 - val_loss: 2.2305 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.7600
Epoch 5/40
2/2 [==============================] - 7s 3s/step - loss: 2.9089 - accuracy: 0.2822 - top_5_accuracy: 0.7600 - val_loss: 2.1870 - val_accuracy: 0.2200 - val_top_5_accuracy: 0.8400
Epoch 6/40
2/2 [==============================] - 7s 3s/step - loss: 2.4559 - accuracy: 0.2867 - top_5_accuracy: 0.7800 - val_loss: 2.1120 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7600
Epoch 7/40
2/2 [==============================] - 6s 3s/step - loss: 2.3713 - accuracy: 0.3111 - top_5_accuracy: 0.7711 - val_loss: 2.0209 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7800
Epoch 8/40
2/2 [==============================] - 7s 3s/step - loss: 2.1972 - accuracy: 0.3311 - top_5_accuracy: 0.8156 - val_loss: 2.0237 - val_accuracy: 0.1800 - val_top_5_accuracy: 0.8000
```

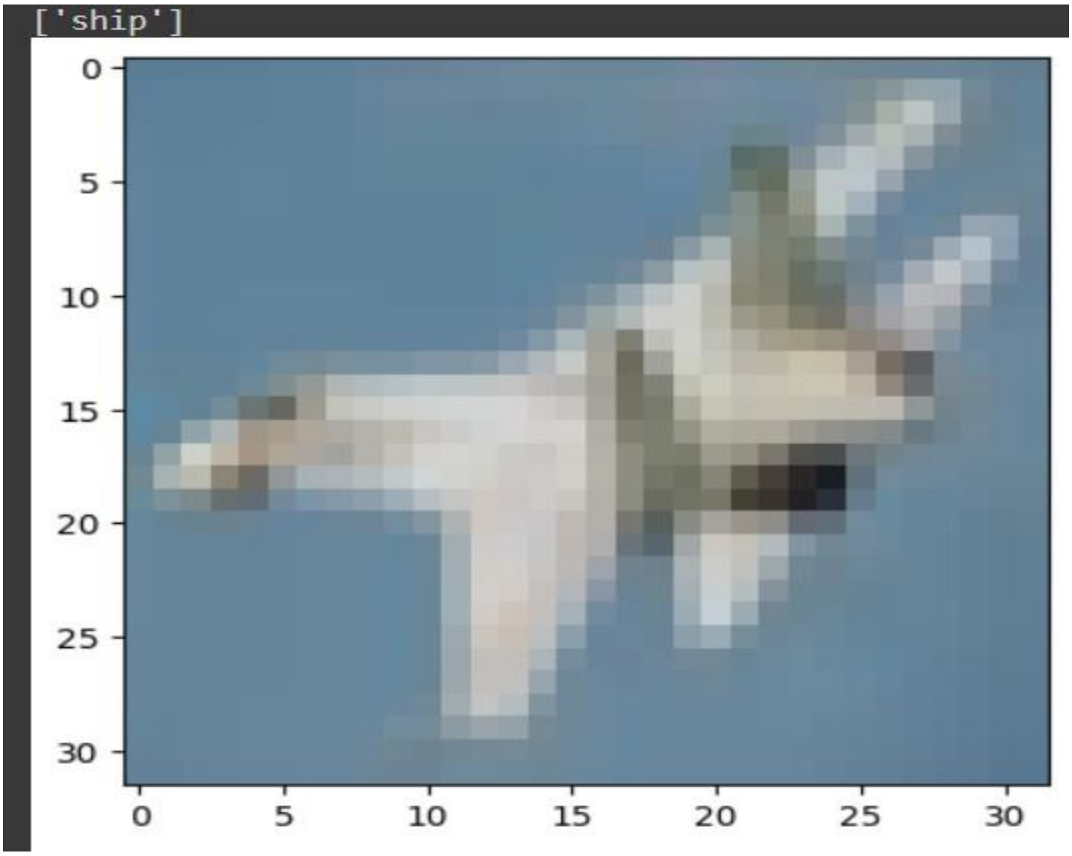
```
Epoch 9/40
2/2 [==============================] - 6s 3s/step - loss: 1.9543 - accuracy: 0.3422 - top_5_accuracy: 0.8378 - val_loss: 2.0796 - val_accuracy: 0.2000 - val_top_5_accuracy: 0.8400
Epoch 10/40
2/2 [==============================] - 7s 3s/step - loss: 1.8802 - accuracy: 0.3378 - top_5_accuracy: 0.8400 - val_loss: 1.9772 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.8000
Epoch 11/40
2/2 [==============================] - 6s 3s/step - loss: 1.8301 - accuracy: 0.3467 - top_5_accuracy: 0.8756 - val_loss: 1.9695 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7600
Epoch 12/40
2/2 [==============================] - 7s 3s/step - loss: 1.7764 - accuracy: 0.4000 - top_5_accuracy: 0.8467 - val_loss: 1.9903 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7600
Epoch 13/40
2/2 [==============================] - 7s 4s/step - loss: 1.7452 - accuracy: 0.4022 - top_5_accuracy: 0.8756 - val_loss: 1.9953 - val_accuracy: 0.2000 - val_top_5_accuracy: 0.8200
Epoch 14/40
2/2 [==============================] - 8s 3s/step - loss: 1.6586 - accuracy: 0.4422 - top_5_accuracy: 0.8933 - val_loss: 1.9749 - val_accuracy: 0.2200 - val_top_5_accuracy: 0.8000
Epoch 15/40
2/2 [==============================] - 6s 3s/step - loss: 1.6563 - accuracy: 0.4178 - top_5_accuracy: 0.8956 - val_loss: 1.9229 - val_accuracy: 0.2400 - val_top_5_accuracy: 0.8000
Epoch 16/40
2/2 [==============================] - 7s 3s/step - loss: 1.5510 - accuracy: 0.4711 - top_5_accuracy: 0.8867 - val_loss: 1.8880 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.7800
Epoch 17/40
2/2 [==============================] - 6s 3s/step - loss: 1.4443 - accuracy: 0.4867 - top_5_accuracy: 0.9133 - val_loss: 1.8888 - val_accuracy: 0.2400 - val_top_5_accuracy: 0.7400
Epoch 18/40
2/2 [==============================] - 7s 3s/step - loss: 1.5283 - accuracy: 0.4689 - top_5_accuracy: 0.9111 - val_loss: 1.9593 - val_accuracy: 0.2400 - val_top_5_accuracy: 0.7600
Epoch 19/40
2/2 [==============================] - 7s 3s/step - loss: 1.4766 - accuracy: 0.4844 - top_5_accuracy: 0.9133 - val_loss: 1.9907 - val_accuracy: 0.3200 - val_top_5_accuracy: 0.7800
Epoch 20/40
2/2 [==============================] - 8s 3s/step - loss: 1.4117 - accuracy: 0.4978 - top_5_accuracy: 0.9178 - val_loss: 1.9777 - val_accuracy: 0.3400 - val_top_5_accuracy: 0.7600
Epoch 21/40
2/2 [==============================] - 6s 3s/step - loss: 1.4566 - accuracy: 0.4978 - top_5_accuracy: 0.9244 - val_loss: 1.9169 - val_accuracy: 0.3200 - val_top_5_accuracy: 0.7800
Epoch 22/40
2/2 [==============================] - 7s 3s/step - loss: 1.3218 - accuracy: 0.5311 - top_5_accuracy: 0.9422 - val_loss: 1.9037 - val_accuracy: 0.3000 - val_top_5_accuracy: 0.7600
Epoch 23/40
2/2 [==============================] - 6s 3s/step - loss: 1.3070 - accuracy: 0.5467 - top_5_accuracy: 0.9467 - val_loss: 1.9823 - val_accuracy: 0.3200 - val_top_5_accuracy: 0.7600
Epoch 24/40
2/2 [==============================] - 7s 3s/step - loss: 1.3022 - accuracy: 0.5333 - top_5_accuracy: 0.9378 - val_loss: 2.0679 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.7800
```

```
Epoch 25/40
2/2 [==============================] - 6s 3s/step - loss: 1.2160 - accuracy: 0.5711 - top_5_accuracy: 0.9600 - val_loss: 2.0854 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7800
Epoch 26/40
2/2 [==============================] - 7s 3s/step - loss: 1.1930 - accuracy: 0.6133 - top_5_accuracy: 0.9511 - val_loss: 2.0802 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7800
Epoch 27/40
2/2 [==============================] - 6s 3s/step - loss: 1.1049 - accuracy: 0.5933 - top_5_accuracy: 0.9622 - val_loss: 2.0679 - val_accuracy: 0.3000 - val_top_5_accuracy: 0.8000
Epoch 28/40
2/2 [==============================] - 7s 3s/step - loss: 1.0574 - accuracy: 0.6333 - top_5_accuracy: 0.9689 - val_loss: 2.0578 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.7600
Epoch 29/40
2/2 [==============================] - 6s 3s/step - loss: 1.0173 - accuracy: 0.6400 - top_5_accuracy: 0.9667 - val_loss: 2.0751 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.8200
Epoch 30/40
2/2 [==============================] - 7s 3s/step - loss: 1.0689 - accuracy: 0.6000 - top_5_accuracy: 0.9711 - val_loss: 2.0883 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.8200
Epoch 31/40
2/2 [==============================] - 6s 3s/step - loss: 1.0909 - accuracy: 0.6244 - top_5_accuracy: 0.9622 - val_loss: 2.1496 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.8000
Epoch 32/40
2/2 [==============================] - 7s 3s/step - loss: 0.9762 - accuracy: 0.6600 - top_5_accuracy: 0.9733 - val_loss: 2.2030 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.7800
Epoch 33/40
2/2 [==============================] - 6s 3s/step - loss: 0.9281 - accuracy: 0.6778 - top_5_accuracy: 0.9800 - val_loss: 2.2295 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7800
Epoch 34/40
2/2 [==============================] - 7s 3s/step - loss: 0.9017 - accuracy: 0.6756 - top_5_accuracy: 0.9711 - val_loss: 2.1765 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.8000
Epoch 35/40
2/2 [==============================] - 6s 2s/step - loss: 0.8079 - accuracy: 0.7244 - top_5_accuracy: 0.9800 - val_loss: 2.1083 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.8000
Epoch 36/40
2/2 [==============================] - 7s 3s/step - loss: 0.7612 - accuracy: 0.7311 - top_5_accuracy: 0.9867 - val_loss: 2.1151 - val_accuracy: 0.3000 - val_top_5_accuracy: 0.8200
Epoch 37/40
2/2 [==============================] - 6s 3s/step - loss: 0.8160 - accuracy: 0.7244 - top_5_accuracy: 0.9844 - val_loss: 2.2137 - val_accuracy: 0.3200 - val_top_5_accuracy: 0.8000
Epoch 38/40
2/2 [==============================] - 7s 3s/step - loss: 0.7859 - accuracy: 0.7356 - top_5_accuracy: 0.9889 - val_loss: 2.3242 - val_accuracy: 0.2800 - val_top_5_accuracy: 0.8000
Epoch 39/40
2/2 [==============================] - 6s 3s/step - loss: 0.7695 - accuracy: 0.7244 - top_5_accuracy: 0.9800 - val_loss: 2.3831 - val_accuracy: 0.2600 - val_top_5_accuracy: 0.7600
Epoch 40/40
2/2 [==============================] - 7s 4s/step - loss: 0.7132 - accuracy: 0.7178 - top_5_accuracy: 0.9822 - val_loss: 2.4289 - val_accuracy: 0.2400 - val_top_5_accuracy: 0.7800
```

```
16/16 [==============================] - 2s 141ms/step - loss: 1.9423 - accuracy: 0.3280 - top_5_accuracy: 0.8420
Test accuracy: 32.8%
Test top-5 accuracy: 84.2%
```

```
class_names =
["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]

index = 10
plt.imshow(x_test[index])
prediction = img_predict(x_test[index],vit_classifier)
print(prediction)

def img_predict(images,model):
  if len(images.shape) == 3:
    out = model.predict(images.reshape(-1, *images.shape))
  else:
    out = model.predict(images)
  prediction = np.argmax(out, axis=1)
  img_prediction = [class_names[i] for i in prediction]
  return img_prediction

index = 10
plt.imshow(x_test[index])
prediction = img_predict(x_test[index],vit_classifier)
print(prediction)
```

OUTPUT:



Conclusion:

The Vision Transformer Internship Project has been a successful and enriching experience, providing interns with deep insights into Vision Transformer architecture and hands-on skills in model implementation, training, and optimization. By exploring advanced techniques and applying models to real-world datasets, interns demonstrated the versatility and effectiveness of Vision Transformers. Comprehensive documentation and robust deployment strategies were developed, ensuring practical application and scalability. This project has laid a strong foundation for future contributions to the field of machine learning and artificial intelligence, thanks to the invaluable support from mentors and team members.