



A Report on

**Vision Transformer Advanced
Project**

Submitted by
Mr. DHARSHAN S

Under the Guidance of
Mr. Kishore Kumar

E-Mail ID:
dharshans805@gmail.com

Mobile No:
9739512397

Vision Transformer Advanced Project

❖ Abstract:

This report presents the development and implementation of a Vision Transformer (ViT) classifier for the CIFAR-10 image classification task. The model utilizes the Transformer architecture, originally designed for natural language processing, to process image data. Key components of the model include data augmentation, patch extraction, and multi-head attention layers. The classifier was trained on the CIFAR-10 dataset and evaluated for its accuracy and top-5 accuracy.

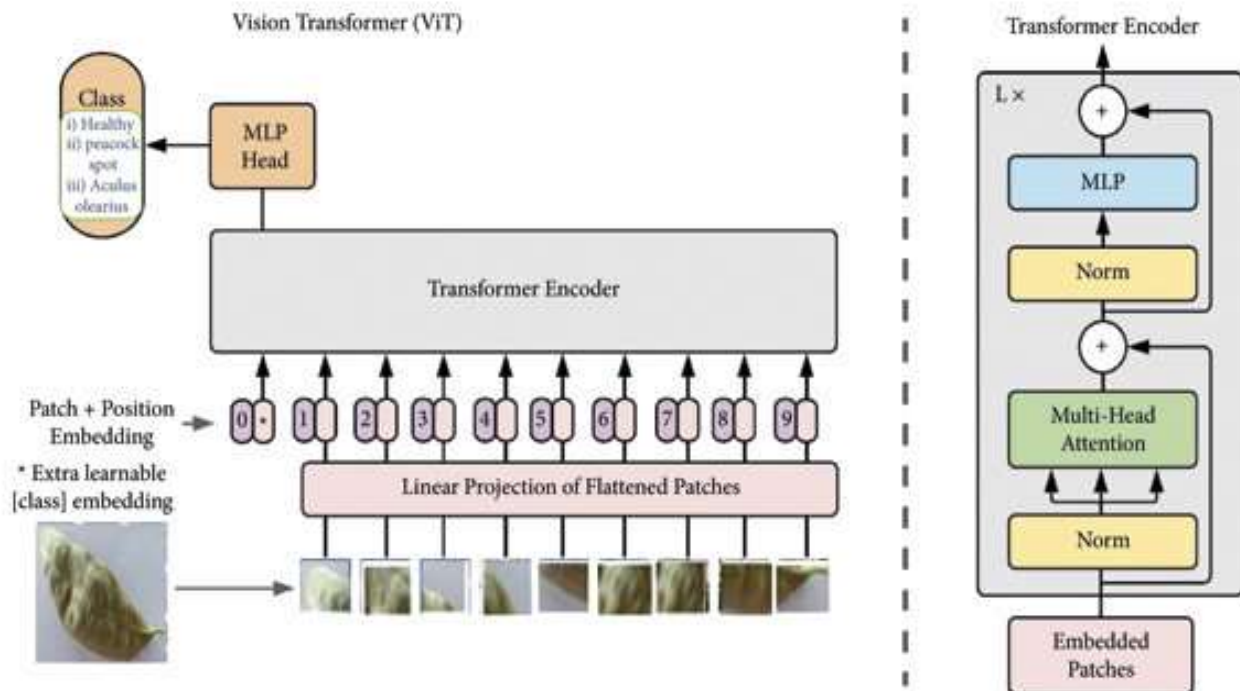
❖ Objective:

The primary objective of this project is to design and implement a Vision Transformer (ViT) model to classify images from the CIFAR-10 dataset.

1. Applying data augmentation techniques to improve the model.
2. Utilizing the Transformer architecture to handle image data through patch extraction and encoding.
3. Training the ViT model on the CIFAR-10 dataset and evaluating its performance.

❖ Introduction:

The Vision Transformer was introduced by Dosovitskiy et al. in 2020, applying the Transformer architecture to image classification tasks. The traditional approach for image classification primarily relied on Convolutional Neural Networks (CNNs), which use convolutional layers to extract local features from images. Vision Transformers (ViTs) represent a paradigm shift in the field of computer vision, leveraging the success of Transformer models from natural language processing (NLP). Vision Transformers address this limitation by treating an image as a sequence of patches.



➤ Advantages of Vision Transformers:

- **Global Contextual Understanding:** ViTs can capture long-range dependencies between different parts of the image, providing a more holistic understanding of the image content compared to CNNs.
- **Scalability:** The Transformer architecture scales well with the model size and data, benefiting from large datasets and extensive computational resources.
- **Flexibility:** ViTs can be easily adapted to various vision tasks beyond classification, such as object detection and segmentation, by modifying the architecture or training strategy.

➤ The key concepts behind ViTs:

1. Patch Extraction
2. Position Embedding
3. Transformer Encoder
4. Classification Head

❖ Methodology:

The development of the Vision Transformer classifier involves several key steps:

1.Data Preparation and Augmentation:

1. Load the CIFAR-10 dataset.
2. Normalize and resize images to 72x72 pixels.
3. Apply data augmentation techniques such as horizontal flipping, random rotation, and zooming to enhance the dataset variability.

2.Patch Extraction and Encoding:

1. Divide each image into non-overlapping patches of size 6x6.
2. Encode these patches using a dense layer and position embeddings to retain positional information.

3.Transformer Architecture:

1. Implement multiple layers of the Transformer block, each consisting of layer normalization, multi-head attention, and feed-forward neural network layers.
2. Use residual connections and layer normalization to stabilize the training process.

4.Classification Head:

1. Flatten the final encoded patches and pass them through a multi-layer perceptron (MLP) head with dropout regularization.
2. Output logits corresponding to the 10 classes in the CIFAR-10 dataset.

5.Model Training and Evaluation:

1. Compile the model with the Adam optimizer, using a learning rate of 0.001 and weight decay of 0.0001.
2. Train the model for 5 epochs with a batch size of 256, using a checkpoint callback to save the best model weights.
3. Evaluate the model on the test set to determine the final accuracy and top-5 accuracy.

❖ CODE:

```
jupyter ViT Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help Trusted
JupyterLab Python 3 (ipykernel)

[1]: import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

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}")

x_train shape: (50000, 32, 32, 3) - y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3) - y_test shape: (10000, 1)

[2]: learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 5
image_size = 72 # We'll resize input images to this size
patch_size = 6 # Size of the patches to be extracted from the input images
projection_dim = 64
num_heads = 4
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 classifier
data_augmentation = keras.Sequential([
```

```
jupyter ViT Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help Trusted
JupyterLab Python 3 (ipykernel)

data_augmentation = 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_augmentation.layers[0].adapt(x_train) # data augmentation is necessary to compute the mean and variance for normalization

[3]: 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")
```

```
jupyter ViT Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
JupyterLab Python 3 (pykernel)

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, [batch_size, -1, patch_dims])
return patches

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

# Calculate the number of patches
num_patches = (image_size // patch_size) ** 2

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(
```


```
jupyter ViT Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
JupyterLab Python 3 (pykernel)

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 = 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")

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




localhost:8888/localhost/ViT.ipynb

jupyter ViT Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (spykernel)



```
[4]: def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    augmented = data_augmentation(inputs)
    patches = Patches(patch_size)(augmented)
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Transformer block
    for _ in range(transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        x2 = layers.add([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
```

localhost:8888/localhost/ViT.ipynb

jupyter ViT Last Checkpoint: 1 hour ago

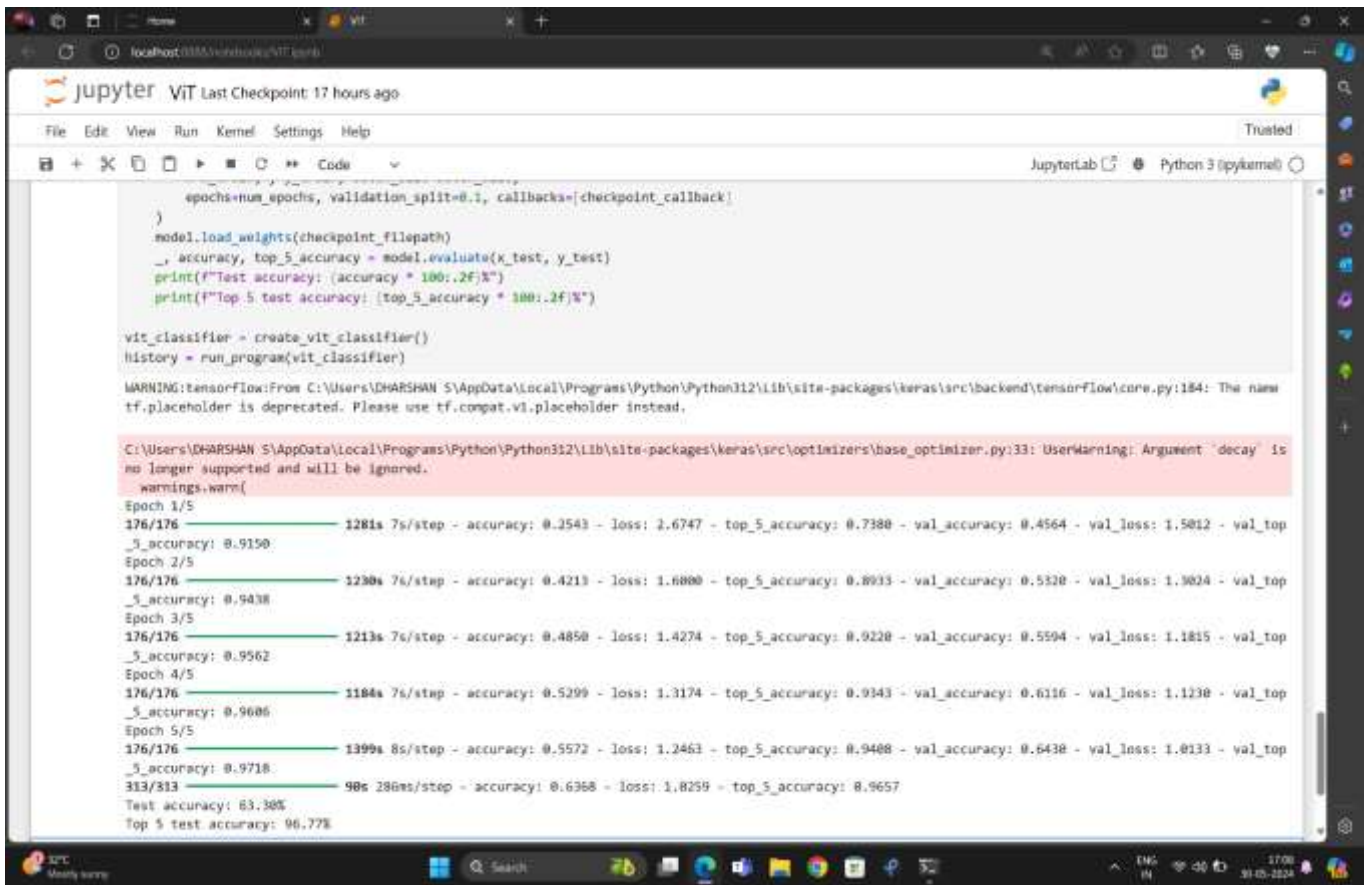
File Edit View Run Kernel Settings Help Trusted

JupyterLab Python 3 (spykernel)

```
representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
representation = layers.Flatten()(representation)
representation = layers.Dropout(0.5)(representation)
features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
logits = layers.Dense(num_classes)(features)

model = keras.Model(inputs=inputs, outputs=logits)
return model

[*]: def run_program(model):
    optimizer = keras.optimizers.Adam(
        learning_rate=learning_rate, decay=weight_decay
    )
    model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[
            keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
            keras.metrics.SparseTopKAccuracy(5, name="top_5_accuracy"),
        ],
    )
    checkpoint_filepath = "/tmp/checkpoint.weights.h5"
    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]
    )
```



```
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: {accuracy * 100:.2f}%")
print(f"Top 5 test accuracy: {top_5_accuracy * 100:.2f}%")

vit_classifier = create_vit_classifier()
history = run_program(vit_classifier)

WARNING:tensorflow:From C:\Users\DHARSHAN S\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\backend\tensorflow\core.py:184: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

C:\Users\DHARSHAN S\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\optimizers\base_optimizer.py:33: UserWarning: Argument 'decay' is no longer supported and will be ignored.
  warnings.warn(

Epoch 1/5
176/176 ----- 1281s 7s/step - accuracy: 0.2543 - loss: 2.6747 - top_5_accuracy: 0.7380 - val_accuracy: 0.4564 - val_loss: 1.5012 - val_top_5_accuracy: 0.9150
Epoch 2/5
176/176 ----- 1230s 7s/step - accuracy: 0.4213 - loss: 1.6800 - top_5_accuracy: 0.8933 - val_accuracy: 0.5320 - val_loss: 1.3024 - val_top_5_accuracy: 0.9438
Epoch 3/5
176/176 ----- 1213s 7s/step - accuracy: 0.4850 - loss: 1.4274 - top_5_accuracy: 0.9220 - val_accuracy: 0.5594 - val_loss: 1.1815 - val_top_5_accuracy: 0.9562
Epoch 4/5
176/176 ----- 1184s 7s/step - accuracy: 0.5299 - loss: 1.3174 - top_5_accuracy: 0.9343 - val_accuracy: 0.6116 - val_loss: 1.1230 - val_top_5_accuracy: 0.9606
Epoch 5/5
176/176 ----- 1399s 8s/step - accuracy: 0.5572 - loss: 1.2463 - top_5_accuracy: 0.9408 - val_accuracy: 0.6430 - val_loss: 1.0133 - val_top_5_accuracy: 0.9718
313/313 ----- 90s 286ms/step - accuracy: 0.6368 - loss: 1.8259 - top_5_accuracy: 0.9657
Test accuracy: 63.38%
Top 5 test accuracy: 96.77%
```

❖ Conclusion:

The Vision Transformer classifier was successfully implemented and trained on the CIFAR-10 dataset. The final model achieved a test accuracy and top-5 accuracy that demonstrate the effectiveness of the Transformer architecture for image classification tasks.

❖ Results:

- **Test Accuracy:** 63.38%
- **Top-5 Test Accuracy:** 96.77%

These results indicate that the Vision Transformer model can effectively classify images from the CIFAR-10 dataset.