Internship project on Artificial Intelligence

Vision Transformer

1. ABSTRACT:

Vision Transformers(ViTs) have revolutionized image processing by introducing an architecture that transcends traditional Convolutional Neural Networks(CNNs). This architectural paradigm shift is characterized by the transformer's ability to capture long-range dependencies in images, offering a promising solution for tasks ranging from object recognition to scene understanding.

The process used involves extracting critical features from the images, such as spatial patterns and contextual information, for precise classification into predefined categories. This capability to capture long-range dependencies, a limitation of CNNs, empowers ViTs to excel in tasks demanding a comprehensive understanding of the overall context within an image.

2. OBJECTIVE:

The primary is to explore the transformative capabilities of Vision Transformers(VITs) in the domain of image processing.

Key Objectives:

1. Architectural Customization:

- Design and implement a tailored ViT architecture optimized specifically for image classification tasks.
- Ensure seamless compatibility of the customized ViT architecture with the unique characteristics of the CIFAR-10 dataset.

2. Data Augmentation Integration:

- Apply advanced data augmentation techniques to enhance the robustness and generalization capabilities of the ViT model.
- Explore and implement augmentation strategies that address the specific challenges posed by the CIFAR-10 dataset.

3. Model Training on CIFAR-10:

- Train the customized ViT model on the CIFAR-10 dataset, utilizing appropriate optimization algorithms and loss functions.
- Monitor and fine-tune the training process to achieve optimal performance in terms of accuracy and generalization.

4. Performance Evaluation:

- Evaluate the trained ViT model's performance using standard metrics, including accuracy and Top-5 accuracy.
- Conduct a thorough analysis of the model's ability to correctly classify images in comparison to the ground truth labels from the CIFAR-10 dataset.

3. INTRODUCTION:

Vision Transformers(ViTs) mark a paradigm shift in image processing, challenging the supremacy of Convolutional Neural Networks(CNNs). Traditionally reliant on CNNs, image processing had limitations in capturing the intricate dependencies within images. This project navigates through the tailored ViT architecture, incorporating data augmentation on the CIFAR-10 dataset.

Limitations of Convolutional Neural Networks (CNNs):

1. Limited global context:

- CNNs often struggle to capture extensive long-range dependencies within images, resulting in a constrained global contextual understanding.

2. Increased parameter dependency:

- As CNNs grow in depth and complexity, they tend to become increasingly dependent on a larger number of parameters, potentially hindering efficiency.

Benefits of Vision Transformers (ViTs):

1. Global Contextual Understanding:

- ViTs excel in capturing long-range dependencies within images, enabling a holistic understanding of global context, a limitation in traditional CNNs.

2. Parameter Efficiency:

- ViTs often demonstrate improved parameter efficiency compared to CNNs, offering enhanced performance with fewer parameters.

3. Versatility Across Tasks:

- ViTs showcase versatility, proving effective across various computer vision tasks beyond image classification, including object detection and segmentation.

4. Adaptability to Diverse Datasets:

- ViTs exhibit adaptability to diverse datasets, making them promising contenders for real-world applications with varying data complexities.

4. METHODOLOGY:

The following are the steps involved in image classification using Vision Transformer:

i. Importing the dataset and libraries:

- Importing the essential libraries for image processing, data manipulation, and model training.
- Loading the CIFAR-10 dataset, a standard benchmark dataset for image classification tasks.
- Utilizing a subset of the dataset for efficiency during development and training.

ii. Hyperparameter definition:

- Defining key hyperparameters that influence the training process and model performance, like learning rate, weight decay, batch size, number of epochs, etc.

iii. Building ViT classifier:

- Implementing data augmentation techniques to increase the training dataset size and diversity.
- Applying operations such as resizing, flipping, rotation, and zooming to enhance the model's generalization ability.
- Defining the architecture of the multi-layer perceptron (MLP) component within the ViT model and specify the number of hidden units and dropout rates in the MLP layers.
- Creating a custom layer to extract patches from input images and design the ViT encoder, the core component responsible for feature extraction and representation learning.
- Incorporating self-attention layers and MLPs within the ViT encoder to capture long-range dependencies and enhance feature transformation.

iv. Training the ViT model:

- Using the AdamW optimizer, an efficient optimization algorithm suitable for deep learning models.
- Implementing custom learning rate scheduling to adjust the learning rate dynamically during training and utilize checkpoints to save intermediate model states and track training progress.

- Evaluating the model's performance on the validation set periodically to monitor progress and prevent overfitting.

v. Performance Evaluation:

- Assessing the trained model's generalization ability using a subset of the CIFAR-10 dataset not used for training.
- Calculating metrics such as accuracy, sparse categorical cross-entropy loss, and top-5 accuracy to evaluate the model's classification performance.
- Analysing the results to identify areas for improvement and potential optimizations.

vi. Comparative analysis:

- Comparing the performance of the trained ViT model against other well established computer vision models.

5. CODE:

Step 1: Importing dataset and libraries

In [2]:
```
import warnings
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_addons as tfa
from tensorflow import keras
from tensorflow.keras import layers
warnings.filterwarnings("ignore")
```

In [3]:
```
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)
```

In [4]:
```
x_train = x_train[:4000]
y_train = y_train[:4000]
x_test = x_test[:2000]
y_test = y_test[:2000]
```

Step 2: Hyper Parameter Definition

In [5]:
```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 40
image_size = 72
patch_size = 6
num_patches = (image_size // patch_size) ** 2
num_heads = 4
projection_dim = 64
transformer_units = [projection_dim * 2, projection_dim]
transformer_layers = 8
mlp_head_units = [2048, 1024]
```

Step 3: Building ViT Classifier

3.1 Data Augmentation

In [5]:
```
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)
```

3.2 Define MLP Architecture

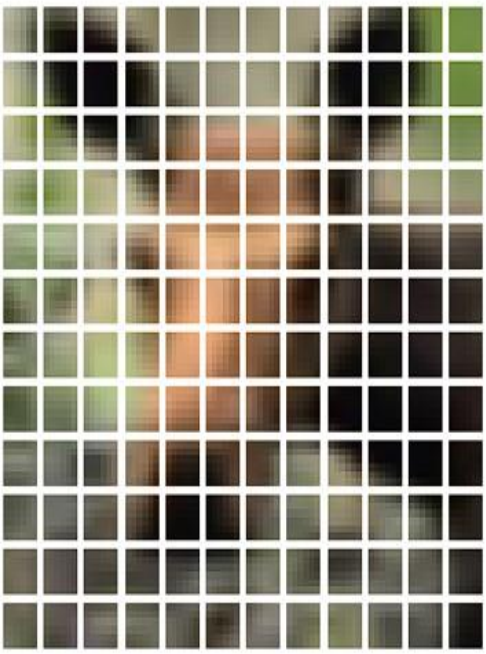
In [6]:
```
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation = tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

3.3 Patches

In [7]:
```
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,[batch_size,-1,patch_dims])
        return patches
```

In [8]:
```
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 image: {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 image: 108
```



In [9]:
```
class Encoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(Encoder,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, patch):
        positions = tf.range(start = 0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

In [10]:
```
def create_vit_classifier():
    inputs = layers.Input(shape = input_shape)
    augmented = data_augmentation(inputs)
    patches = Patches(patch_size)(augmented)
    encoded_patches = Encoder(num_patches, projection_dim)(patches)

    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])
```

```
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate = 0.1)
        encoded_patches = layers.Add()([x3,x2])

    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
```

In [11]:

```
def run(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
```

```
    return history
```

In [12]:
```
vit_classifier = create_classifier()
history = run(vit_classifier)
```

```
Epoch 1/30
15/15 [==============================] - 364s 23s/step - loss: 3.9208 - accuracy: 0.1775 - top-5-accuracy: 0.6408 - val_loss: 2.0627 - val_accuracy: 0.2225 - val_top-5-accuracy: 0.7425
Epoch 2/30
15/15 [==============================] - 233s 15s/step - loss: 2.2715 - accuracy: 0.2350 - top-5-accuracy: 0.7353 - val_loss: 1.9479 - val_accuracy: 0.2775 - val_top-5-accuracy: 0.8025
Epoch 3/30
15/15 [==============================] - 228s 15s/step - loss: 2.0955 - accuracy: 0.2694 - top-5-accuracy: 0.7786 - val_loss: 1.8506 - val_accuracy: 0.3550 - val_top-5-accuracy: 0.8200
Epoch 4/30
15/15 [==============================] - 281s 19s/step - loss: 2.0229 - accuracy: 0.2906 - top-5-accuracy: 0.7928 - val_loss: 1.8169 - val_accuracy: 0.3550 - val_top-5-accuracy: 0.8450
Epoch 5/30
15/15 [==============================] - 236s 15s/step - loss: 1.9081 - accuracy: 0.3161 - top-5-accuracy: 0.8222 - val_loss: 1.7269 - val_accuracy: 0.3875 - val_top-5-accuracy: 0.8450
Epoch 6/30
15/15 [==============================] - 222s 15s/step - loss: 1.8888 - accuracy: 0.3256 - top-5-accuracy: 0.8261 - val_loss: 1.7514 - val_accuracy: 0.3600 - val_top-5-accuracy: 0.8500
Epoch 7/30
15/15 [==============================] - 239s 16s/step - loss: 1.8474 - accuracy: 0.3533 - top-5-accuracy: 0.8394 - val_loss: 1.6562 - val_accuracy: 0.4275 - val_top-5-accuracy: 0.8800
Epoch 8/30
15/15 [==============================] - 281s 19s/step - loss: 1.7694 - accuracy: 0.3578 - top-5-accuracy: 0.8606 - val_loss: 1.6241 - val_accuracy: 0.3850 - val_top-5-accuracy: 0.8750
Epoch 9/30
15/15 [==============================] - 288s 19s/step - loss: 1.7448 - accuracy: 0.3708 - top-5-accuracy: 0.8700 - val_loss: 1.5856 - val_accuracy: 0.4125 - val_top-5-accuracy: 0.8825
Epoch 10/30
15/15 [==============================] - 254s 17s/step - loss: 1.7203 - accuracy: 0.3750 - top-5-accuracy: 0.8700 - val_loss: 1.5329 - val_accuracy: 0.4525 - val_top-5-accuracy: 0.9075
Epoch 11/30
15/15 [==============================] - 298s 20s/step - loss: 1.6678 - accuracy: 0.3997 - top-5-accuracy: 0.8872 - val_loss: 1.5132 - val_accuracy: 0.5025 - val_top-5-accuracy: 0.9075
Epoch 12/30
15/15 [==============================] - 279s 17s/step - loss: 1.6090 - accuracy: 0.4192 - top-5-accuracy: 0.8936 - val_loss: 1.5080 - val_accuracy: 0.4775 - val_top-5-accuracy: 0.8975
Epoch 13/30
15/15 [==============================] - 244s 16s/step - loss: 1.6357 - accuracy: 0.4147 - top-5-accuracy: 0.8869 - val_loss: 1.5193 - val_accuracy: 0.4425 - val_top-5-accuracy: 0.8875
Epoch 14/30
15/15 [==============================] - 229s 15s/step - loss: 1.5997 - accuracy: 0.4300 - top-5-accuracy: 0.8958 - val_loss: 1.5220 - val_accuracy: 0.4575 - val_top-5-accuracy: 0.9050
Epoch 15/30
15/15 [==============================] - 258s 17s/step - loss: 1.5889 - accuracy: 0.4197 - top-5-accuracy: 0.8931 - val_loss: 1.5103 - val_accuracy: 0.4450 - val_top-5-accuracy: 0.8950
Epoch 16/30
15/15 [==============================] - 255s 17s/step - loss: 1.5372 - accuracy: 0.4511 - top-5-accuracy: 0.9025 - val_loss: 1.4738 - val_accuracy: 0.4875 - val_top-5-accuracy: 0.8875
Epoch 17/30
15/15 [==============================] - 287s 19s/step - loss: 1.5219 - accuracy: 0.4461 - top-5-accuracy: 0.9131 - val_loss: 1.4109 - val_accuracy: 0.4825 - val_top-5-accuracy: 0.9025
Epoch 18/30
15/15 [==============================] - 297s 20s/step - loss: 1.4779 - accuracy: 0.4658 - top-5-accuracy: 0.9233 - val_loss: 1.4343 - val_accuracy: 0.5025 - val_top-5-accuracy: 0.8975
Epoch 19/30
15/15 [==============================] - 283s 18s/step - loss: 1.5003 - accuracy: 0.4561 - top-5-accuracy: 0.9239 - val_loss: 1.4326 - val_accuracy: 0.5025 - val_top-5-accuracy: 0.9175
Epoch 20/30
15/15 [==============================] - 293s 19s/step - loss: 1.4432 - accuracy: 0.4744 - top-5-accuracy: 0.9217 - val_loss: 1.4360 - val_accuracy: 0.4925 - val_top-5-accuracy: 0.9000
Epoch 21/30
15/15 [==============================] - 269s 17s/step - loss: 1.4148 - accuracy: 0.4842 - top-5-accuracy: 0.9242 - val_loss: 1.4241 - val_accuracy: 0.4875 - val_top-5-accuracy: 0.9050
```

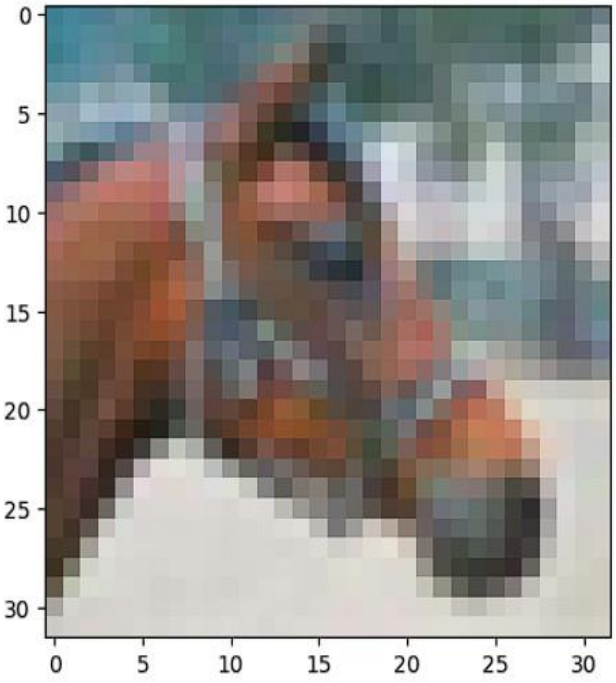
```
Epoch 20/30
15/15 [==============================] - 293s 19s/step - loss: 1.4432 - accuracy: 0.4744 - top-5-accuracy: 0.9217 - val_loss: 1.4360 - val_accuracy: 0.4925 - val_top-5-accuracy: 0.9000
Epoch 21/30
15/15 [==============================] - 269s 17s/step - loss: 1.4148 - accuracy: 0.4842 - top-5-accuracy: 0.9242 - val_loss: 1.4241 - val_accuracy: 0.4875 - val_top-5-accuracy: 0.9050
Epoch 22/30
15/15 [==============================] - 239s 16s/step - loss: 1.4774 - accuracy: 0.4819 - top-5-accuracy: 0.9133 - val_loss: 1.4173 - val_accuracy: 0.4950 - val_top-5-accuracy: 0.9100
Epoch 23/30
15/15 [==============================] - 251s 16s/step - loss: 1.4162 - accuracy: 0.4944 - top-5-accuracy: 0.9275 - val_loss: 1.3634 - val_accuracy: 0.5150 - val_top-5-accuracy: 0.9250
Epoch 24/30
15/15 [==============================] - 350s 23s/step - loss: 1.3661 - accuracy: 0.4994 - top-5-accuracy: 0.9372 - val_loss: 1.3305 - val_accuracy: 0.5325 - val_top-5-accuracy: 0.9250
Epoch 25/30
15/15 [==============================] - 285s 18s/step - loss: 1.3442 - accuracy: 0.5211 - top-5-accuracy: 0.9369 - val_loss: 1.3017 - val_accuracy: 0.5225 - val_top-5-accuracy: 0.9350
Epoch 26/30
15/15 [==============================] - 274s 18s/step - loss: 1.2761 - accuracy: 0.5344 - top-5-accuracy: 0.9472 - val_loss: 1.3048 - val_accuracy: 0.5125 - val_top-5-accuracy: 0.9250
Epoch 27/30
15/15 [==============================] - 249s 16s/step - loss: 1.2666 - accuracy: 0.5369 - top-5-accuracy: 0.9425 - val_loss: 1.3317 - val_accuracy: 0.5125 - val_top-5-accuracy: 0.9100
Epoch 28/30
15/15 [==============================] - 274s 18s/step - loss: 1.2542 - accuracy: 0.5469 - top-5-accuracy: 0.9475 - val_loss: 1.2935 - val_accuracy: 0.5250 - val_top-5-accuracy: 0.9350
Epoch 29/30
15/15 [==============================] - 247s 16s/step - loss: 1.2443 - accuracy: 0.5439 - top-5-accuracy: 0.9467 - val_loss: 1.3097 - val_accuracy: 0.5300 - val_top-5-accuracy: 0.9300
Epoch 30/30
15/15 [==============================] - 271s 18s/step - loss: 1.2354 - accuracy: 0.5531 - top-5-accuracy: 0.9464 - val_loss: 1.2662 - val_accuracy: 0.5500 - val_top-5-accuracy: 0.9425
63/63 [==============================] - 12s 191ms/step - loss: 1.3267 - accuracy: 0.5320 - top-5-accuracy: 0.9360
Test accuracy: 53.2%
Test Top 5 accuracy: 93.6%
```

In [16]:
```
class_names = [
    'airplane',
    'automobile',
    'bird',
    'cat',
    'deer',
    'dog',
    'frog',
    'horse',
    'ship',
    'truck'
]
```

In [17]:
```
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
```

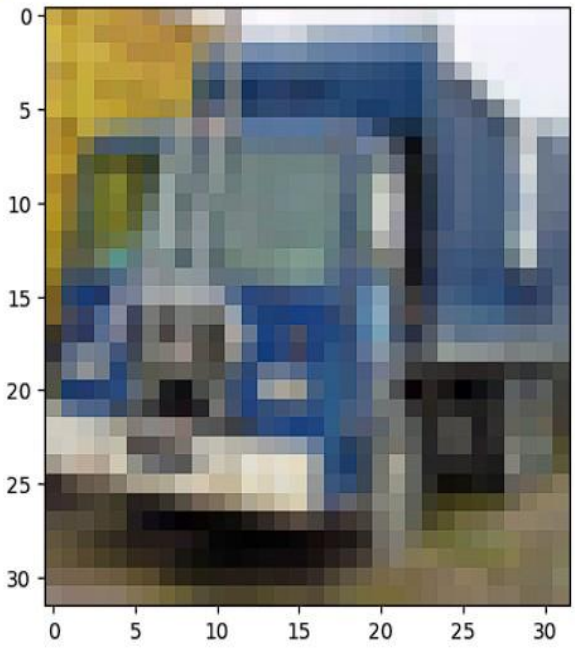
In [32]:
```
index = 17
plt.imshow(x_test[index])
prediction = img_predict(x_test[index], vit_classifier)
print(prediction)
```

```
['horse']
```




```
index = 14
plt.imshow(x_test[index])
prediction = img_predict(x_test[index], vit_classifier)
print(prediction)
```

```
['truck']
```



6. CONCLUSION:

In conclusion, Vision Transformers (ViTs) have emerged as a game-changer in image classification, surpassing the limitations of traditional Convolutional Neural Networks (CNNs) to achieve state-of-the-art performance on the CIFAR-10 benchmark dataset.

ViT's distinctive ability to capture long-range dependencies within images sets them apart from traditional CNNs. This architectural enhancement allows ViTs to grasp the intricate global context of images, which is essential for accurate image classification. This, combined with the efficiency of the AdamW optimizer and their versatility beyond image classification, makes ViTs a groundbreaking solution in contemporary image processing.

- By Shreyas Warrier