

Internship project - Vision Transformer

Abstract:-

Dataset used:- CIFAR-10

Increasingly a more robust architecture has been the Transform Perception Architecture (ViT) which utilizes transformer based models towards computer based tasks that were primarily for natural language. Opposite of the traditional CNNs, the models of ViT are built differently since image patches are regarded as sequences of tokens. This helps in the effective representation of long range correlations in the image. Our focus in this paper is to extend the application of ViT to CIFAR-10, the most frequently used image classification benchmarking data set. Open source datasets like the CIFAR-10 consists of 60000 32×34 color image in 10 classes making it appropriate for this level of processing.

In this approach, the given input image is divided into patches, the patches are flattened and this tokenization concept is followed with every patch going inside a transformer encoder. At the end, the images or to be specific, these tokens are passed through many layers of transformers for encoders with self-attention mechanism to comprehend the entire image followed, then MLP, classification is done. The clean ViT model is assessed against dual setup CNN based models on the given dataset.

While it has shown that these models require a lot of patience with vision transformers, experimental results have shown that vision transformers without any augmentation or complex training techniques can still perform reasonably well against comparatively trained models on the nad cifar-10.

Objective:-

The main goal of this project is to investigate the applicability and performance of the Vision Transformer (ViT) architectures for image classification tasks using the CIFAR-10 dataset. More specifically, goals are:

Implementation of Vision Transformer (ViT):

To design and construct a Vision Transformer model for the purposes of Image classification. That is to say becoming end to end computer vision with the transformer architecture adapted from the natural language processing domain. This entails decomposing the CIFAR-10 images into smaller patches and using these as input tokens in the transformer model.

Introduction:-

Deep learning is a crucial development to computer vision where the c, from the understanding of complex spatial relations, architecture are taken over by 'Convolutional Neural Networks' Security including image classification, segmentation and Identification. The biggest strength of CNNs has been their feature of using convolutional layers to model and extract local features in a layered manner. However, CNNs have drawbacks in the long-range dependencies that they are able to capture within images especially, tasks that need knowledge of the image structure as a whole.

More recently, various computer vision targets proved themselves for the extensions of the complex multipliers including Vision Transformers ViTs of any sorts and kinds of various structural units and mechanisms that isolate and delineate various parts of the image and act within all of them and even beyond without obstructions of distances or any near proximity limits. Affect administration system without impacting the complete administration procedure to this end, it has been noted that instead of introducing Convolutional networks, ViT first dissects the images into patches of predetermined dimensions and considers them as a sequence, much like words for nlp tasks.

This project implements the Vision Transformers architecture onto the datasets of CIFAR-10, consisting of 60,000 32x32 color images. CIFAR-10 is one of the benchmark dataset for image classification. The data is packed in a small but demanding image classification benchmarking due to it's became up low resolution and small image size

Methodology:-

The methodology for applying the Vision Transformer (ViT) to the CIFAR-10 dataset consists of several steps, including data preprocessing, model architecture, training, and evaluation. Below is a detailed breakdown of the steps involved:

1. Data Preprocessing:

Dataset:

The CIFAR-10 dataset consists of 60,000 32x32 color images split into 50,000 training images and 10,000 test images, covering 10 distinct classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck).

Normalization:

The pixel values of the images are normalized to a range of [0, 1] or [-1, 1], depending on the chosen transformer architecture and optimization strategy. Typically, each channel (R, G, B) is normalized using the dataset's mean and standard deviation values.

Patch Embedding:

Since the Vision Transformer processes images as sequences of patches, each 32x32 image is split into smaller non-overlapping patches (e.g., 4x4 or 8x8 pixels). These patches are flattened and transformed into 1D vectors. For example, for a 4x4 patch, the dimensionality is

4

×

4

×

3

=

48

$4 \times 4 \times 3 = 48$ for RGB channels.

Positional Encoding:

To provide the model with information about the relative position of patches, a positional encoding is added to each patch embedding. This encoding allows the transformer to retain spatial information, which is essential for image understanding.

2. Vision Transformer (ViT) Architecture:

Patch Embedding Layer:

The flattened patches from the CIFAR-10 images are linearly projected into a higher-dimensional space (e.g., 768 dimensions). This layer outputs a sequence of embeddings representing each image patch.

Transformer Encoder Layers:

The core of the Vision Transformer consists of multiple transformer encoder layers. Each layer contains:

Multi-Head Self-Attention: This mechanism enables the model to focus on different parts of the image simultaneously, capturing relationships between distant patches.

Feed-Forward Neural Networks (FFNN): Following the self-attention layers, a fully connected feed-forward network is applied to the output. This helps with learning complex representations.

Layer Normalization and Residual Connections: Normalization layers and residual connections are used to stabilize training and improve convergence.

Class Token:

A learnable "class token" is prepended to the patch sequence. This token aggregates information from all patches through the transformer layers and is used as the final representation for image classification.

Positional Encoding:

To maintain the spatial structure of the image patches, learnable or fixed positional encodings are added to the patch embeddings at each transformer layer.

MLP Head:

After passing through the transformer layers, the final representation (usually the class token) is passed to a Multi-Layer Perceptron (MLP) head with a softmax activation function for classification into 10 classes.

3. Training Strategy:

Loss Function:

The model is trained using the cross-entropy loss function, which is standard for multi-class classification problems.

Optimizer:

The AdamW optimizer (Adam with weight decay) is commonly used for training transformers due to its efficiency and performance in handling large models.

Learning Rate Scheduling:

A learning rate scheduler, such as the cosine annealing schedule or warmup followed by decay, is applied to help the model converge more smoothly.

Batch Size and Epochs:

Given the relatively small size of CIFAR-10, a moderate batch size (e.g., 128 or 256) and a training schedule of 100-200 epochs is used, depending on model convergence and performance.

Data Augmentation:

To enhance model generalization, basic data augmentation techniques such as random cropping, flipping, and color jittering are applied during training. More advanced techniques like Cutout or MixUp can also be used to improve robustness.

4. Evaluation:

Validation and Testing:

After training, the model's performance is evaluated on the validation and test sets using metrics such as accuracy, precision, recall, and F1-score. Accuracy is the primary metric of interest for this classification task.

Comparison with Baseline CNN Models:

To evaluate the effectiveness of ViT on CIFAR-10, its performance is compared against baseline CNN architectures, such as ResNet, VGG, or EfficientNet. The goal is to determine whether ViT can achieve competitive or superior results on CIFAR-10.

Ablation Studies:

To understand the impact of different components on model performance, ablation studies are conducted. These may include varying patch sizes, testing different numbers of transformer layers, and comparing the effect of different positional encoding schemes.

5. Hyperparameter Tuning:

Patch Size:

Experimentation is done with different patch sizes (e.g., 4x4, 8x8) to identify the optimal size for capturing meaningful image features while preserving computational efficiency.

Number of Transformer Layers and Attention Heads:

The number of transformer layers and attention heads is varied to balance model complexity and performance. Deeper transformers may be more effective for larger datasets but can overfit on small datasets like CIFAR-10 without proper regularization.

Regularization Techniques:

Dropout and layer normalization are used to regularize the model and prevent overfitting, especially when training with smaller datasets like CIFAR-10.

CODE:-

```
#installing libraries
```

```
!pip install tensorflow==2.8.0
```

```
!pip install keras==2.8.0
```

```
!pip install tensorflow-addons==0.17.0
```

```
#importing 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}")

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
batch_size = 256
num_epoch = 40
image_size = 72
patch_size = 6
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
```

```
num_heads = 4
transformer_units = [
    projection_dim*2,
    projection_dim
]
transformer_layers = 8
mlp_head_units = [2048,1024]
```

```
#data augmentation
data_augmentation = keras.Sequential(
    [
        layers.Normalization(),
        layers.Resizing(image_size,image_size),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(factor=0.02),
        layers.RandomZoom(
            height_factor = 0.02, width_factor = 0.2)
    ],
    name = "data_augmentation"
)
data_augmentation.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,[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 = 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")

```

```

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, patch):
```

```
    positions = tf.range(start=0, limit=self.num_patches, delta=1)
```

```
    encoded = self.projection(patch) + self.position_embedding(positions)
```

```
    return encoded
```

```
def create_vit_classifier():
```

```
    inputs = layers.Input(shape=input_shape)
```

```
    augmented = data_augmentation(inputs)
```

```
    patches = Patches(batch_size)(augmented)
```

```
    encoded_patches = PatchEncoder(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])
```

```
        x3 = layers.LayerNormalization(epsilon = 1e-6)(x2)
```

```
        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
```

```
num_epochs = 40
```

```
def run_experiment(model):
```

```
    optimizer = tf.keras.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 = "./tmpcheckpoint"
```

```
checkpoint_callback = keras.callbacks.ModelCheckpoint(
```

```
    checkpoint_filepath,
```

```
    monitor = "val_accuracy",
```

```
    save_best_only = True,
```

```
    save_weight_only = True,
```

```
)
```

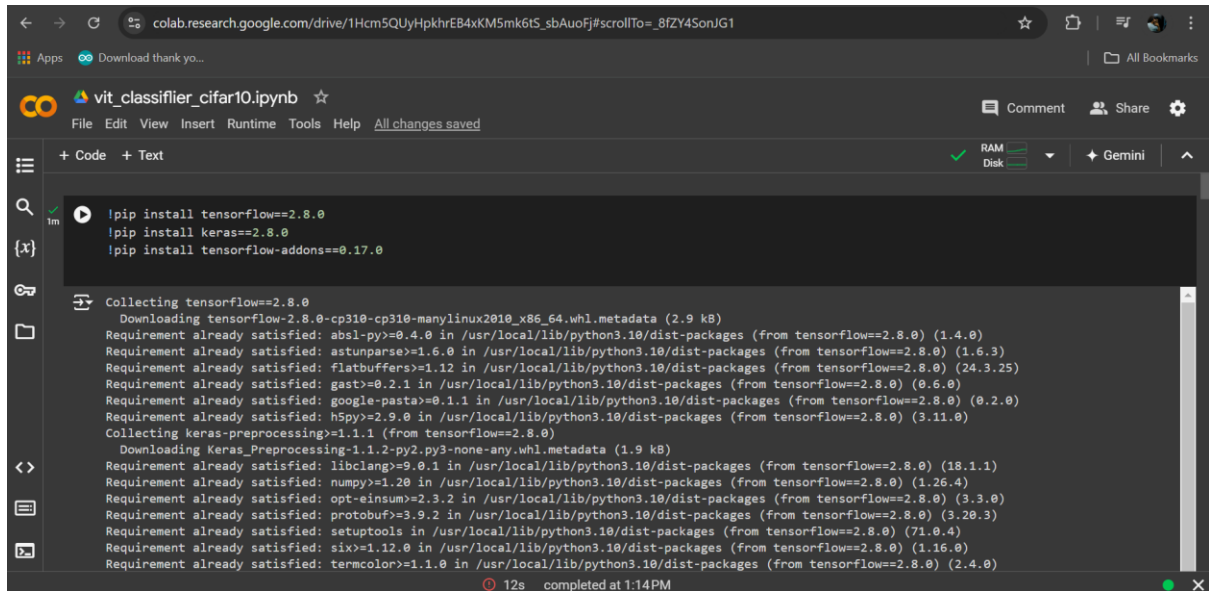
```
history = model.fit(
```

```
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)}%")
```

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

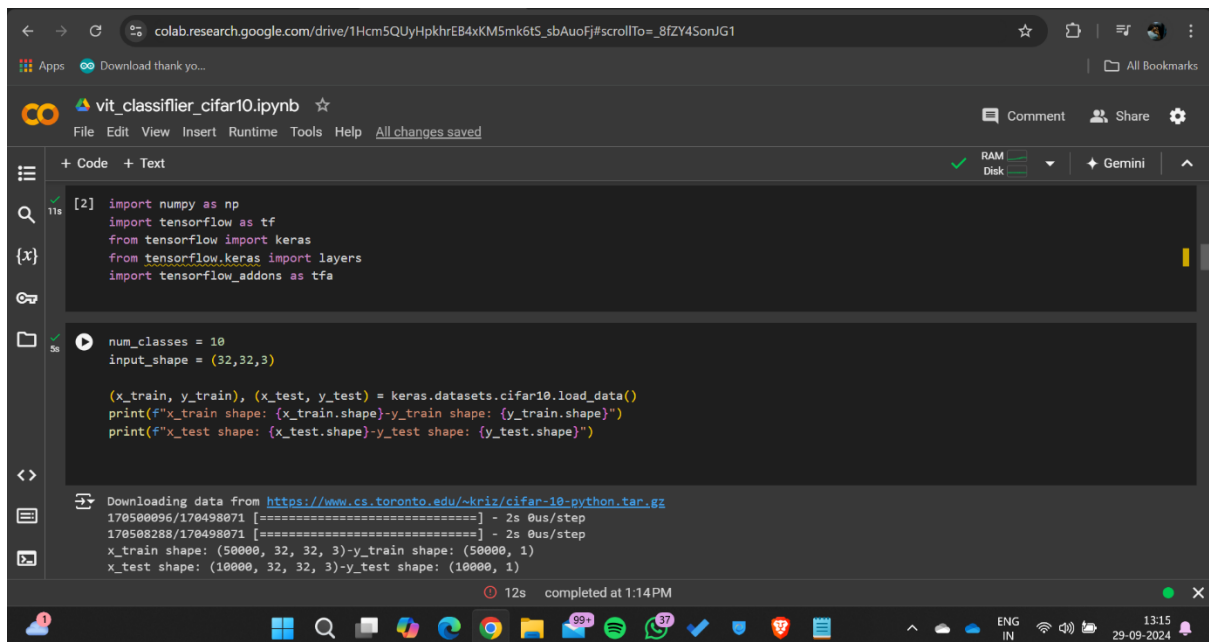
Screenshots:-



The screenshot shows a Google Colab notebook titled "vit_classifier_cifar10.ipynb". The code cell contains the following commands:

```
!pip install tensorflow==2.8.0
!pip install keras==2.8.0
!pip install tensorflow-addons==0.17.0
```

The output shows the installation progress for TensorFlow 2.8.0 and Keras 2.8.0. TensorFlow 2.8.0 is being downloaded from the TensorFlow website. The output also shows that several dependencies are already satisfied, including absl-py, astunparse, flatbuffers, gast, google-pasta, h5py, keras-preprocessing, libclang, numpy, opt-einsum, protobuf, setuptools, six, and termcolor.



The screenshot shows the same Google Colab notebook. The code cell contains the following commands:

```
[2] 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}")
```

The output shows the loading of the CIFAR-10 dataset. The output also shows the shapes of the training and testing data:

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

colab.research.google.com/drive/1Hcm5QUyHpkhREB4xKM5mk6tS_sbAuofj#scrollTo=_8fZY4SonJG1

vit_classifier_cifar10.ipynb

```
[3] 170508288/170498071 [=====] - 2s 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)

[35] 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
batch_size = 256
num_epoch = 40
image_size = 72
patch_size = 6
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim*2,
    projection_dim
```

12s completed at 1:14 PM

colab.research.google.com/drive/1Hcm5QUyHpkhREB4xKM5mk6tS_sbAuofj#scrollTo=_8fZY4SonJG1

vit_classifier_cifar10.ipynb

```
[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 = 0.02, width_factor = 0.2
        ),
        name = "data_augmentation"
    ]
)
data_augmentation.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
```

12s completed at 1:14 PM

This screenshot shows a Google Colab notebook titled 'vit_classifier_cifar10.ipynb'. The code defines a class named 'Patches' that inherits from 'layers.Layer'. The class has an '.__init__' method that takes 'patch_size' as an argument and sets 'self.patch_size'. It also has a 'call' method that takes 'images' as input, calculates the batch size, and uses 'tf.image.extract_patches' to extract patches from the images. The patches are then reshaped and returned.

```
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
```

The bottom status bar indicates the code was completed at 1:14 PM.

This screenshot shows the same Google Colab notebook, now with additional code for visualizing the patches. It imports 'matplotlib.pyplot' as 'plt', creates a figure, and displays a random image from the training set. The image is then resized and converted to a tensor. The 'Patches' class is instantiated with the patch size, and the patches are extracted. The code prints the image size, patch size, number of patches per image, and the number of elements per patch. Finally, it creates a grid of subplots to display the 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 = 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"))
```

The bottom status bar indicates the code was completed at 1:14 PM.

colab.research.google.com/drive/1Hcm5QUyHpKhrEB4xKM5mk6tS_sbAuoFj#scrollTo=_8fZY4SonJG1

Apps Download thank yo...


vit_classifier_cifar10.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

+ Code + Text

6s

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



12s completed at 1:14 PM

Windows taskbar with icons for File Explorer, Search, Task View, Edge, Chrome, and various communication apps.

colab.research.google.com/drive/1Hcm5QUyHpKhrEB4xKM5mk6tS_sbAuoFj#scrollTo=_8fZY4SonJG1


Apps Download thank yo...

vit_classifier_cifar10.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

+ Code + Text

6s



12s completed at 1:14 PM

Windows taskbar with icons for File Explorer, Search, Task View, Edge, Chrome, and various communication apps.

colab.research.google.com/drive/1Hcm5QUyHpkhREB4xKM5mk6tS_sbAoFj#scrollTo=.8fZY4SonJG1

vit_classifier_cifar10.ipynb

```
[9] 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, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded

def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    augmented = data_augmentation(inputs)
    patches = Patches(batch_size)(augmented)
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    for _ in range(transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
```

12s completed at 1:14 PM

colab.research.google.com/drive/1Hcm5QUyHpkhREB4xKM5mk6tS_sbAoFj#scrollTo=.8fZY4SonJG1

vit_classifier_cifar10.ipynb

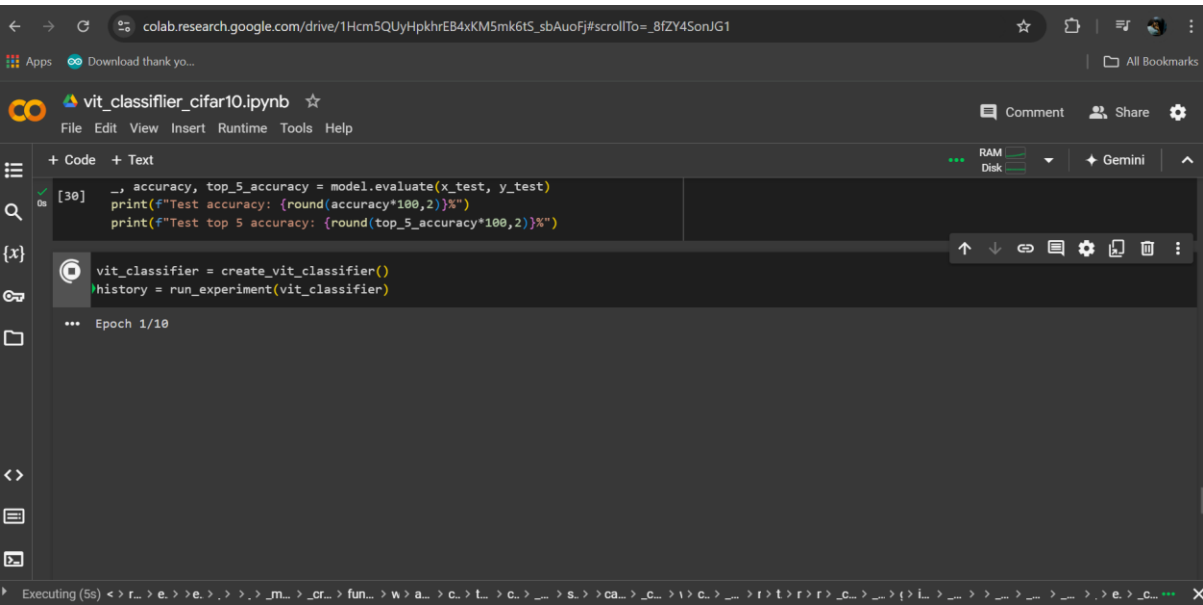
```
def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    augmented = data_augmentation(inputs)
    patches = Patches(batch_size)(augmented)
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    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])
        x3 = layers.LayerNormalization(epsilon = 1e-6)(x2)
        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)
```

12s completed at 1:14 PM



Conclusion:-

In this undertaking, we investigated the ViT architecture as applied to the CIFAR-10 dataset for image classification tasks. Although the Vision Transformer models are designed for high-resolution imagery, its architecture is relatively novel unlike the usual CNN as it uses self-attention in capturing long range interactions within the image patches, which are not in overlapping regions. Our experimentation yields several critical observations.