VISION TRANSFORMER

PROJECT REPORT

DASARI HARSHA SRI RAM

Harsha Sri Ram

VISION TRANSFORMER

Abstract of the Project:

The project is about understanding the transformer architecture and studying its advantages with the help of an image classification task.

I developed a Vision Transformer which has greater potential at solving image classification tasks than a normal Convolutional Neural Network.

Objective of the Project:

Understanding Vision Transformers (ViT).

Implementation and Evaluation of ViT model.

Understanding the self-attention mechanism of transformer model.

Introduction:

Artificial Intelligence (AI) has witnessed remarkable advancements in recent years, with deep learning playing a pivotal role in revolutionizing various domains. Within the realm of computer vision, traditional convolutional neural networks (CNNs) have long been the cornerstone. However, the emergence of Vision Transformers (ViTs) marks a paradigm shift in the way we approach visual information processing. Vision transformers leverage the transformer architecture, originally designed for natural language processing tasks, to address the challenges of image understanding.

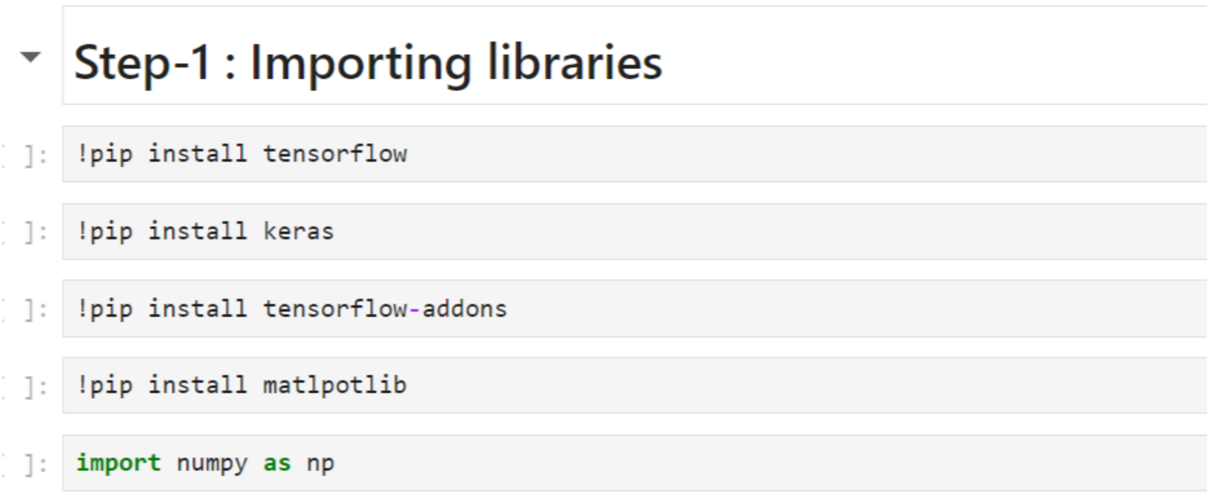
The fundamental idea behind Vision Transformers is to treat an image as a sequence of patches, enabling the application of self-attention mechanisms to capture long-range dependencies within the visual data. This departure from the conventional spatial hierarchies of CNNs not only simplifies model design but also offers the potential for improved scalability and performance on diverse visual tasks.

In this project, we aim to provide a comprehensive understanding of how ViTs differ from traditional approaches, their strengths, limitations, and the experimental results that showcase their efficacy. Through practical implementation and analysis, we seek to contribute insights into the evolving landscape of AI-driven image processing, emphasizing the significance of Vision Transformers in shaping the future of computer vision applications.

Methodology:

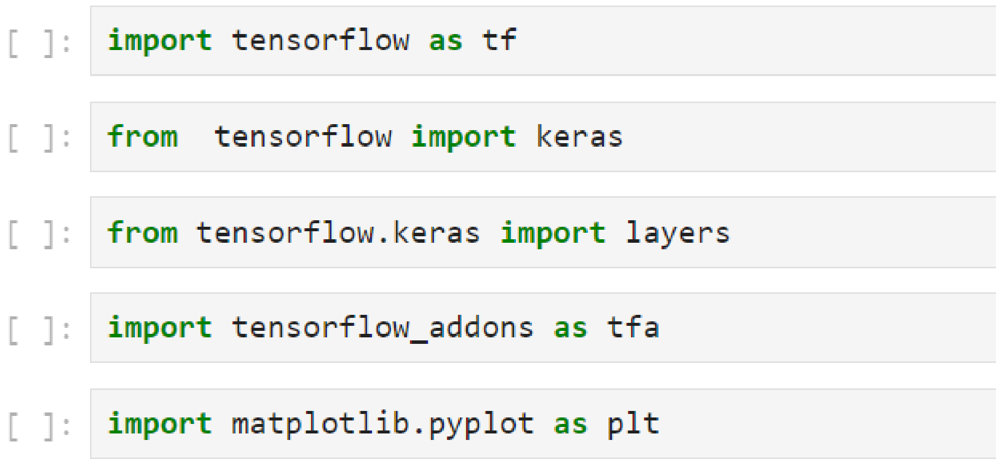
The process is explained step-by-step in the code itself using "Comment" feature in Python language. So, I believe that one can understand the methodology of my project by going through the code.

CODE:



Step-1 : Importing libraries

```
[ ]: !pip install tensorflow
[ ]: !pip install keras
[ ]: !pip install tensorflow-addons
[ ]: !pip install matlpotlib
[ ]: import numpy as np
```



```
[ ]: import tensorflow as tf
[ ]: from  tensorflow import keras
[ ]: from tensorflow.keras import layers
[ ]: import tensorflow_addons as tfa
[ ]: import matplotlib.pyplot as plt
```

Step-2: Importing Datasets

[9]:
```
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)
```

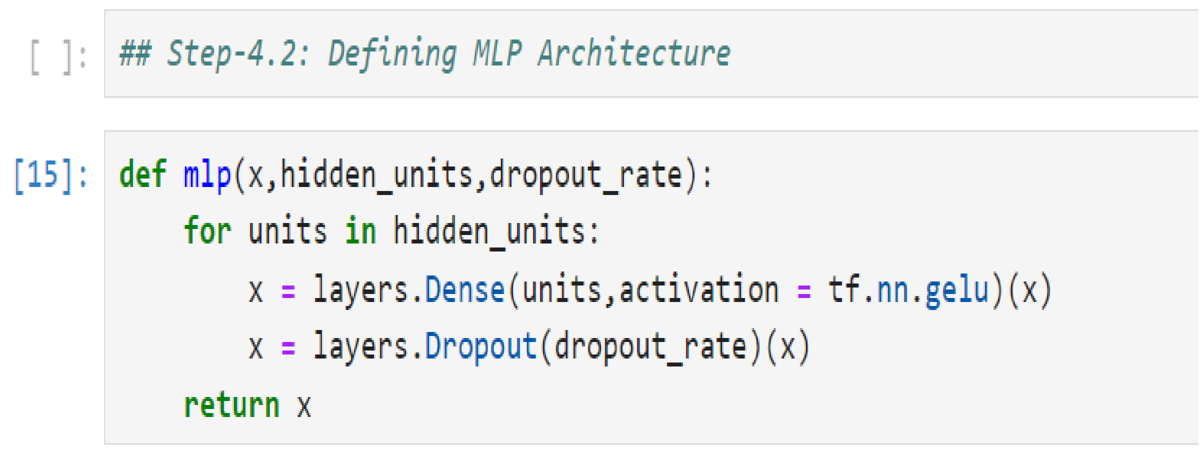
Step-3: Defining Hyper parameters

[11]:
```
learning_rate= 0.001
weight_decay = 0.0001
batch_size   = 256
num_epochs   = 5
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]
```

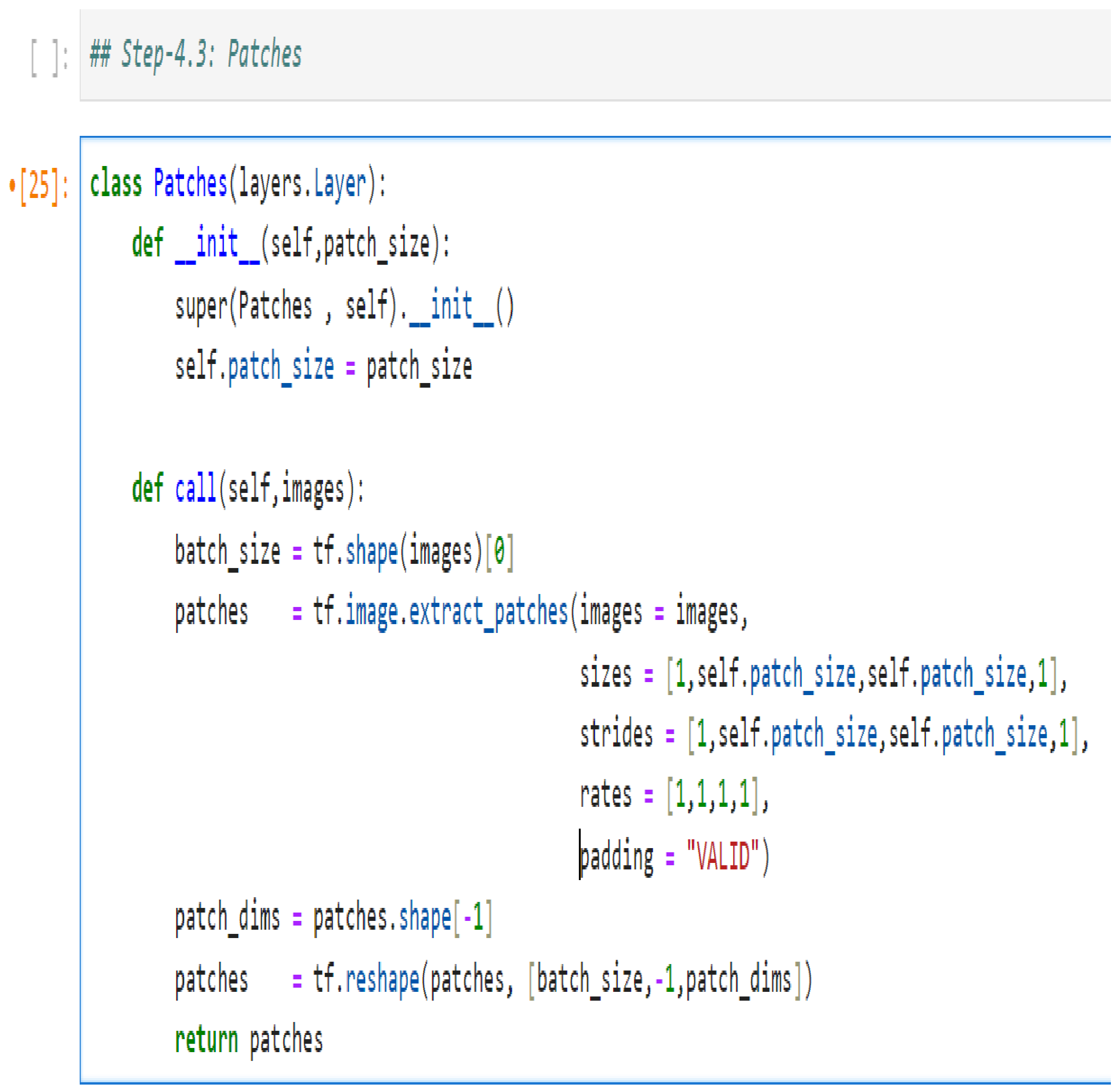
Step-4: Buliding ViT_Classifier Model

[12]:
```
## Step-4.1: Data Augmentation
```

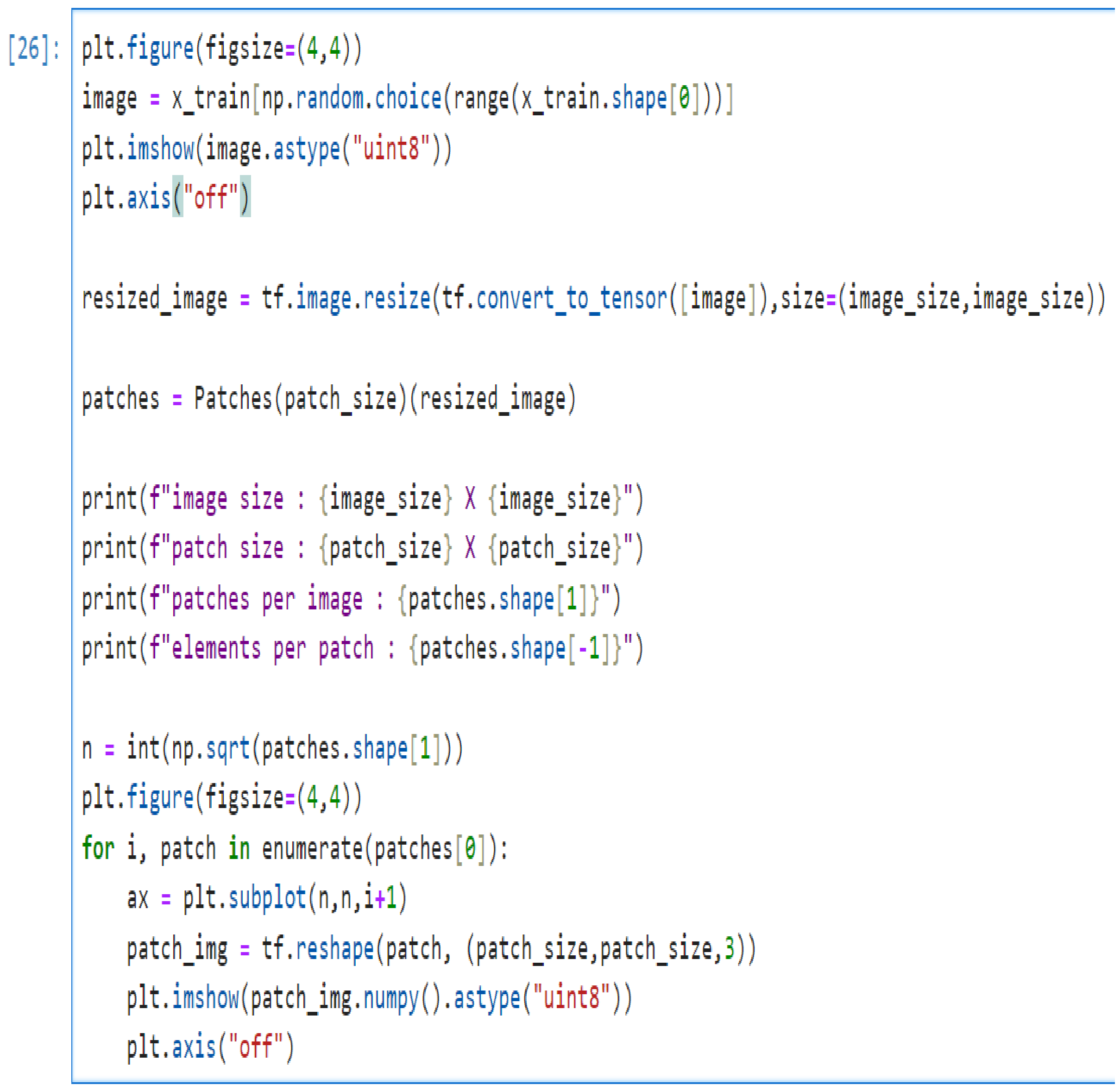
[23]:
```
data_augmentation = keras.Sequential([layers.Normalization(),
                                      layers.Resizing(image_size,image_size),
                                      layers.RandomFlip("horizontal"),
                                      layers.RandomRotation(factor=0.2),
                                      layers.RandomZoom(height_factor=0.2,width_factor=0.2)],
                                      name = "data_augmentation")
data_augmentation.layers[0].adapt(x_train)
```



```
[ ]: ## Step-4.2: Defining MLP Architecture

[15]: 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
```



```
[ ]: ## Step-4.3: Patches

•[25]: 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
```



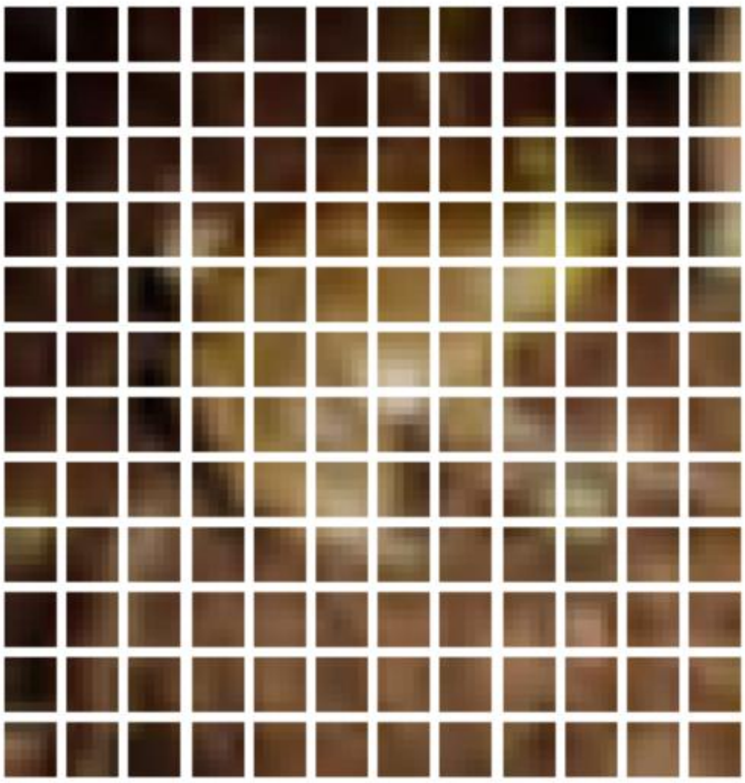
```
[26]: plt.figure(figsize=(4,4))
      image = x_train[np.random.choice(range(x_train.shape[0]))]
      plt.imshow(image.astype("uint8"))
      plt.axis("off")

      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")
```

Now, the output for this section of code is shown in next page.

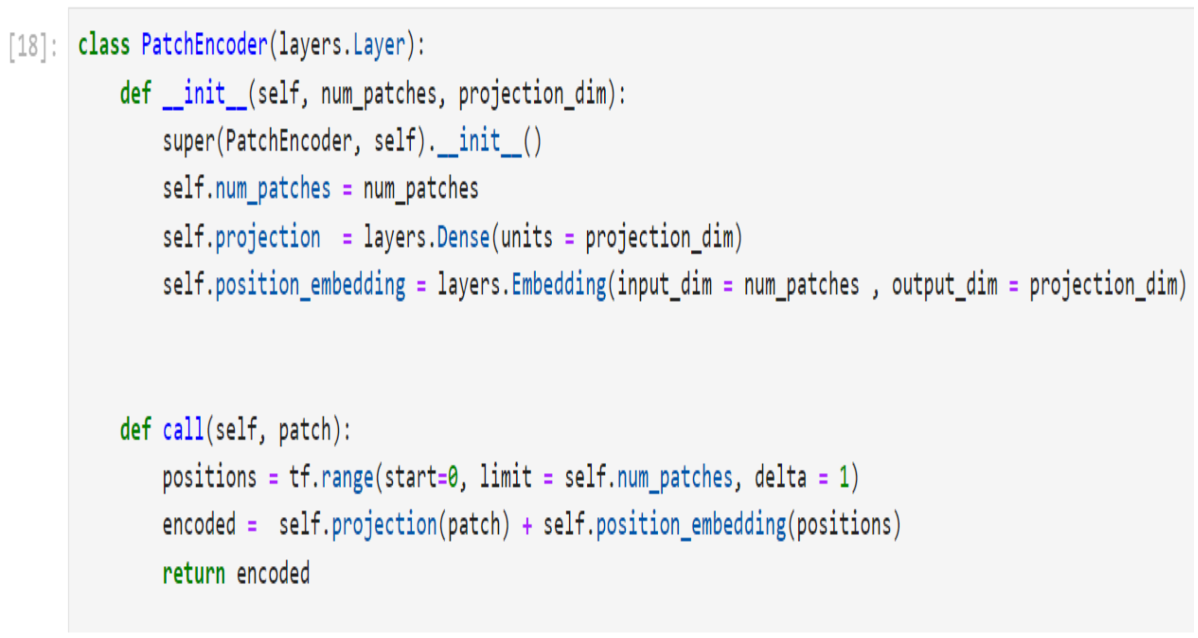
The image which is given as input is divided into sequence of patches as shown below.

1st image is the input, 2nd image is our patched image which is taken by our vision transformer as input.

```
image size : 72 X 72
patch size : 6 X 6
patches per image : 144
elements per patch : 108
```

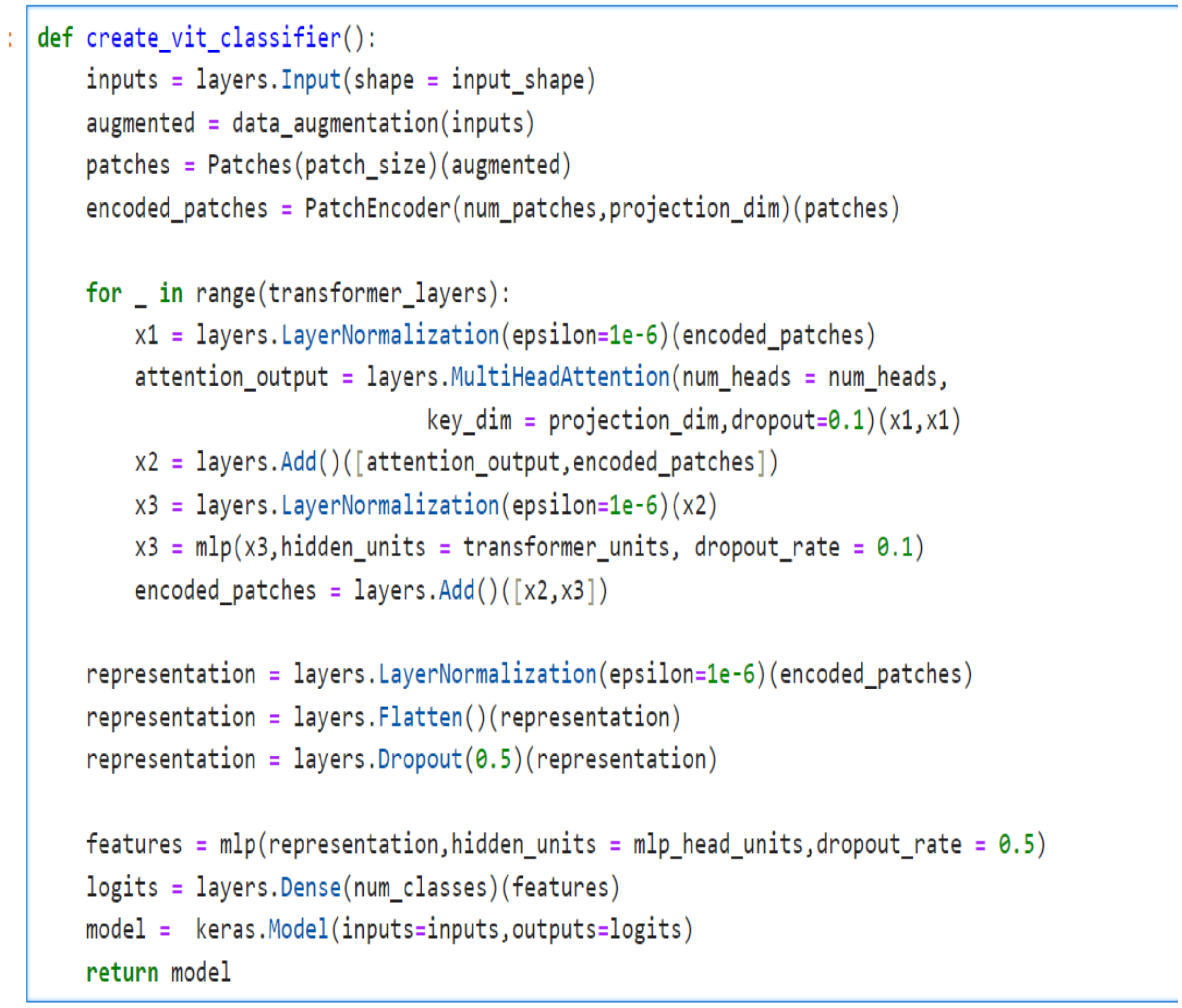


Creating a patch encoder for our input:



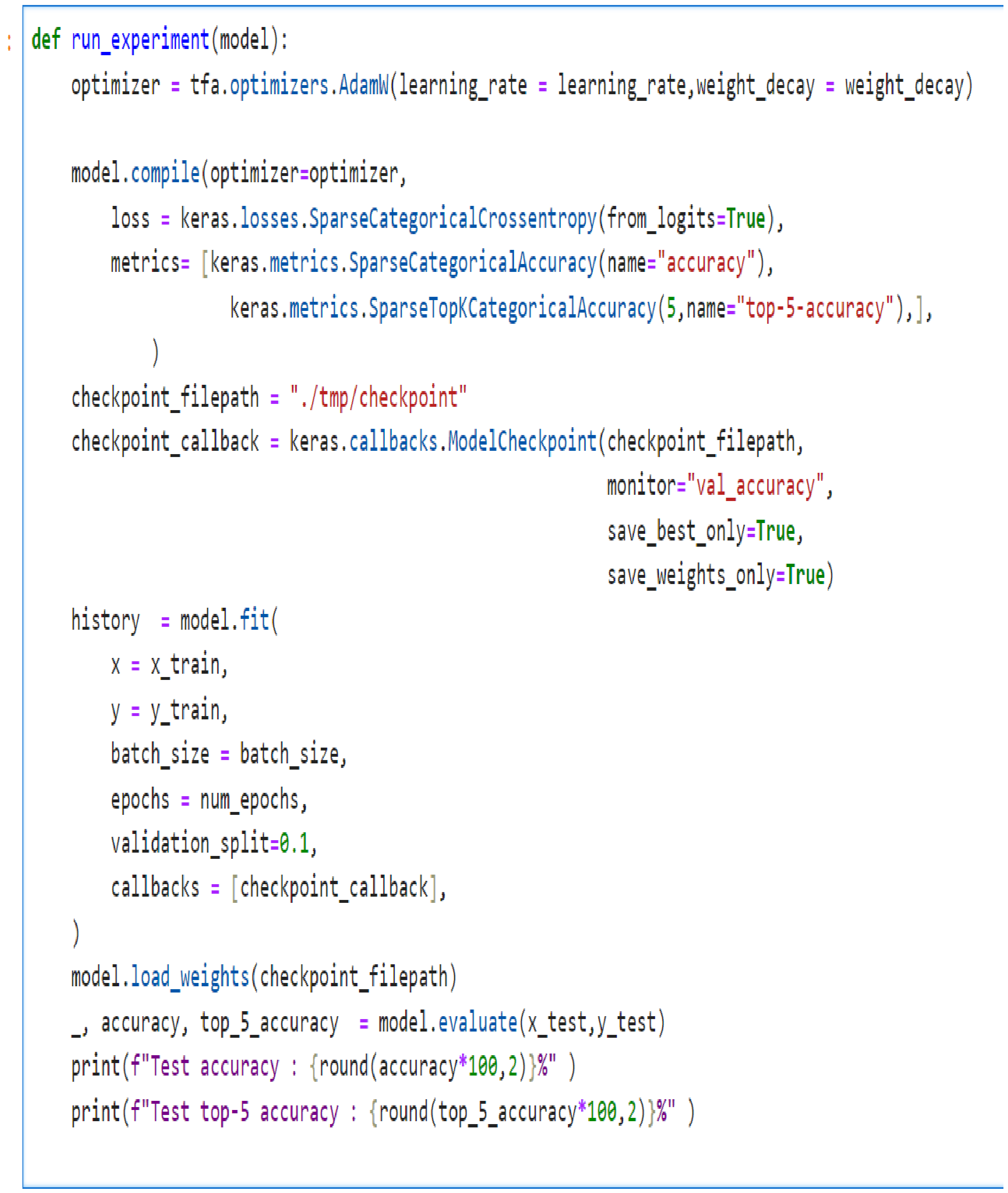
```
[18]: class PatchEncoder(layers.Layer):
          def __init__(self, num_patches, projection_dim):
              super(PatchEncoder, self).__init__()
              self.num_patches = num_patches
              self.projection  = layers.Dense(units = projection_dim)
              self.position_embedding = 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
```

Our function to create ViT_Classifier:



```
: 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)

      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)
          x3 = mlp(x3,hidden_units = transformer_units, dropout_rate = 0.1)
          encoded_patches = layers.Add()([x2,x3])

      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
```

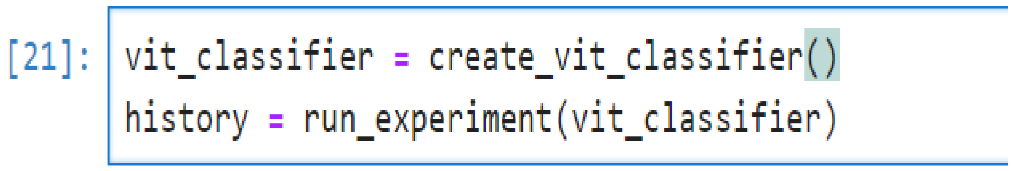
Defining a run function to train the model:



```
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)}%" )
```

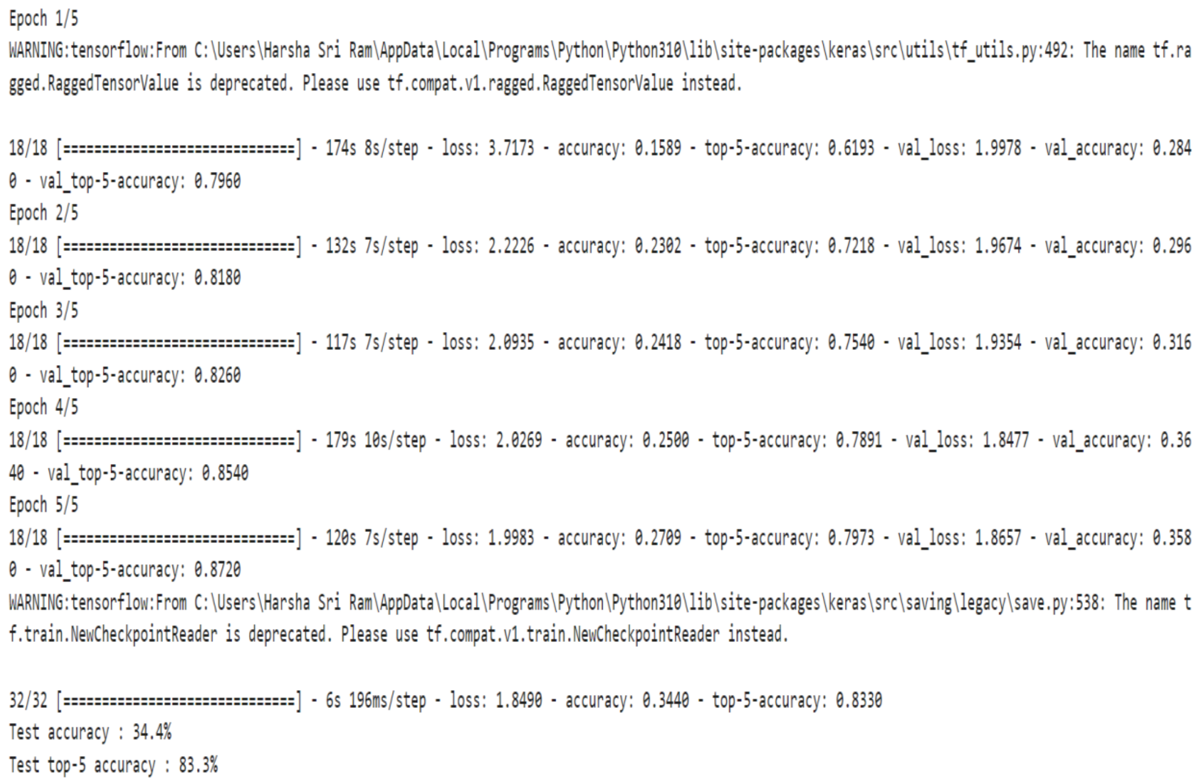
Now, it’s time to train our cifar10 data set with the help of this model.

Let’s call our create_vit_classifier() function to train our model using the given data set.



```
[21]: vit_classifier = create_vit_classifier()
      history = run_experiment(vit_classifier)
```

The final output for this is,



```
Epoch 1/5
WARNING:tensorflow:From C:\Users\Harsha Sri Ram\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ra
gged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

18/18 [==============================] - 174s 8s/step - loss: 3.7173 - accuracy: 0.1589 - top-5-accuracy: 0.6193 - val_loss: 1.9978 - val_accuracy: 0.284
0 - val_top-5-accuracy: 0.7960
Epoch 2/5
18/18 [==============================] - 132s 7s/step - loss: 2.2226 - accuracy: 0.2302 - top-5-accuracy: 0.7218 - val_loss: 1.9674 - val_accuracy: 0.296
0 - val_top-5-accuracy: 0.8180
Epoch 3/5
18/18 [==============================] - 117s 7s/step - loss: 2.0935 - accuracy: 0.2418 - top-5-accuracy: 0.7540 - val_loss: 1.9354 - val_accuracy: 0.316
0 - val_top-5-accuracy: 0.8260
Epoch 4/5
18/18 [==============================] - 179s 10s/step - loss: 2.0269 - accuracy: 0.2500 - top-5-accuracy: 0.7891 - val_loss: 1.8477 - val_accuracy: 0.36
40 - val_top-5-accuracy: 0.8540
Epoch 5/5
18/18 [==============================] - 120s 7s/step - loss: 1.9983 - accuracy: 0.2709 - top-5-accuracy: 0.7973 - val_loss: 1.8657 - val_accuracy: 0.358
0 - val_top-5-accuracy: 0.8720
WARNING:tensorflow:From C:\Users\Harsha Sri Ram\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\saving\legacy\save.py:538: The name t
f.train.NewCheckpointReader is deprecated. Please use tf.compat.v1.train.NewCheckpointReader instead.

32/32 [==============================] - 6s 196ms/step - loss: 1.8490 - accuracy: 0.3440 - top-5-accuracy: 0.8330
Test accuracy : 34.4%
Test top-5 accuracy : 83.3%
```

I have modified the data sizes as,

X_train = 5000=Y_train

X_test = 1000 = Y_test

Because in my pc, the model is taking a lot of time to compute. So I have reduced the data sets sizes by factor of 10.

And our test accuracy is 34.4% and

Top-5 test accuracy is 83.3%

I believe this is a good score in my system but when this model is used by a well configured machine the model definitely shows good results.

Conclusion:

So, in this project,

I have understood the transformer architecture and developed a Vision Transformer model to solve the image classification problems.

Explored Transformers Attention mechanism and the processing of input data with the help of patching technique.

This is not exactly about building a correct working ViT but understanding the importance of transformer models instead of CNNs. The accurate results were not came because of the used computer but by observing the final output, we can understand that model is working absolutely fine.