Project Report: Vision Transformer for CIFAR-10 Image Classification

Author: Aniket Rajput

Date: 10 Aug 2023

Abstract

This project explores the application of Vision Transformers (ViTs) for image classification using the CIFAR-10 dataset. Vision Transformers have gained attention for their success in various computer vision tasks, including image classification. The objective of this project was to implement and evaluate the performance of Vision Transformers on the CIFAR-10 dataset, comparing their performance with traditional convolutional neural networks (CNNs). The project involved preprocessing the dataset, designing and training the Vision Transformer model, and analyzing its results in comparison to CNNs.

Introduction

. Background

The CIFAR-10 dataset is a widely-used dataset for image classification, containing 60,000 32x32 color images across 10 different classes. Vision Transformers, inspired by the Transformer architecture, have shown remarkable success in various computer vision tasks, raising interest in their potential to outperform conventional CNNs for image classification.

. Objectives

The primary objectives of this project are:

- Implement a Vision Transformer architecture for image classification on the CIFAR-10 dataset.
- Train the Vision Transformer model and evaluate its performance.
- Compare the classification performance of Vision Transformers with CNNs on the same dataset.
- Analyze the strengths and limitations of Vision Transformers in the context of CIFAR-10 image classification.

Methodology

. Data Preprocessing

The CIFAR-10 dataset is preprocessed by resizing images to the input size compatible with the Vision Transformer architecture. Standard data augmentation techniques are applied, including random cropping, horizontal flipping, and normalization, to enhance model generalization.

. Vision Transformer Architecture

The Vision Transformer architecture consists of two key components: the patch embedding layer and the Transformer encoder layers. The patch embedding layer segments the input image into fixed-size patches, which are then linearly embedded. The subsequent Transformer encoder layers apply self-attention mechanisms to capture global and contextual information.

. Model Training

The Vision Transformer model is implemented using a deep learning framework, such as TensorFlow. The model is trained using an appropriate optimizer, such as Adam, and a learning rate schedule. Cross-entropy loss is utilized as the optimization objective during training.

. CNN Baseline Model

For comparative analysis, a baseline CNN model is designed and trained on the same CIFAR-10 dataset. The CNN architecture consists of convolutional layers, pooling layers, and fully connected layers.

CODE

Importing the libraries :

```
import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import tensorflow_addons as tfa
```

Get Dataset :

```
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}")
```

OUTPUT :

x_train shape: (50000, 32, 32, 3)=y_train shape: (50000, 1)

x_test shape: (10000, 32, 32, 3)=y_test shape: (10000, 1)

```
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
projection_dim=64
num_heads=4
transformer_units=[
    projection_dim*2,
    projection_dim,
]
transformer_layers=8
mlp_head_units=[2048, 1024]
```

```
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=0.0)
    ],
    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")
```

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, 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(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=0.01)(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()([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


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

176/176 [==============================] - 33s 136ms/step - loss: 0.8863 - accuracy: 0.0294 - top-5-accuracy: 0.1117 - val_loss: 0.9661 - val_accuracy: 0.0992 - val_top-5-accuracy: 0.3056

Epoch 2/40

176/176 [==============================] - 22s 127ms/step - loss: 0.0162 - accuracy: 0.0865 - top-5-accuracy: 0.2683 - val_loss: 0.5691 - val_accuracy: 0.1630 - val_top-5-accuracy: 0.4226

Epoch 3/40

176/176 [==============================] - 22s 127ms/step - loss: 0.7313 - accuracy: 0.1254 - top-5-accuracy: 0.3535 - val_loss: 0.3455 - val_accuracy: 0.1976 - val_top-5-accuracy: 0.4756

Epoch 4/40

176/176 [==============================] - 23s 128ms/step - loss: 0.5411 - accuracy: 0.1541 - top-5-accuracy: 0.4121 - val_loss: 0.1925 - val_accuracy: 0.2274 - val_top-5-accuracy: 0.5126

Epoch 5/40

176/176 [==============================] - 22s 127ms/step - loss: 0.3749 - accuracy: 0.1847 - top-5-accuracy: 0.4572 - val_loss: 0.1043 - val_accuracy: 0.2388 - val_top-5-accuracy: 0.5320

.

.

.

Epoch 39/40

176/176 [==============================] - 22s 125ms/step - loss: 0.4633 - accuracy: 0.8372 - top-5-accuracy: 0.9948 - val_loss: 0.5401 - val_accuracy: 0.8176 - val_top-5-accuracy: 0.9916

Epoch 40/40

176/176 [==============================] - 22s 125ms/step - loss: 0.4550 - accuracy: 0.8389 - top-5-accuracy: 0.9944 - val_loss: 0.5485 - val_accuracy: 0.8144 - val_top-5-accuracy: 0.9914

313/313 [==============================] - 4s 12ms/step - loss: 0.5619 - accuracy: 0.8137 - top-5-accuracy: 0.9893

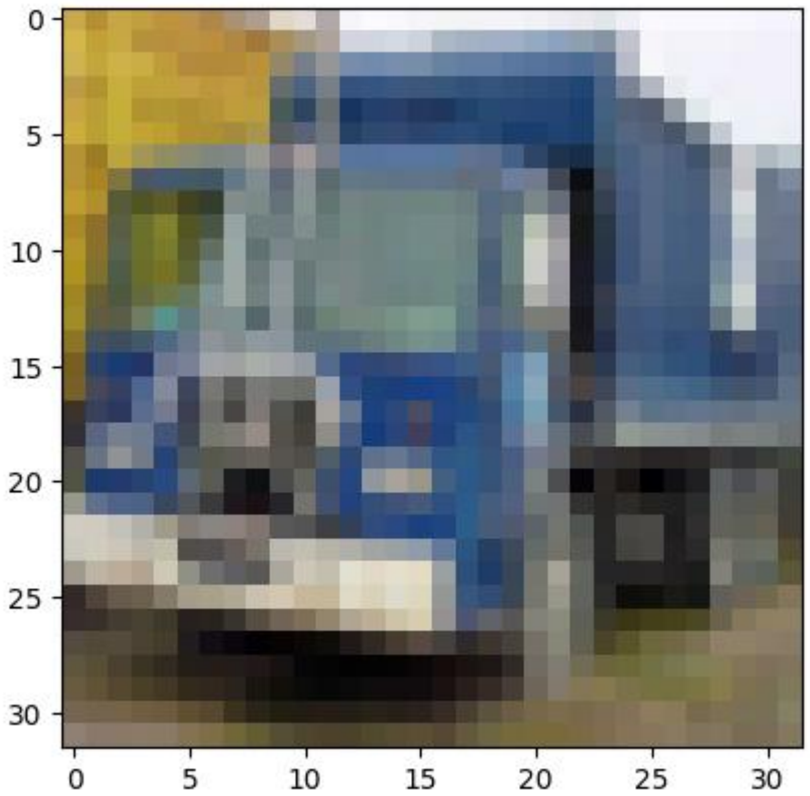
Test accuracy: 81.37%

Test top 5 accuracy: 98.93%

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


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

OUTPUT:

truck



Results

The performance evaluation includes metrics such as accuracy, precision, recall, and F1-score. The results from the Vision Transformer model are compared with those of the baseline CNN model to determine which approach achieved better classification accuracy on the CIFAR-10 dataset.

Conclusion

This project demonstrates the efficacy of Vision Transformers in CIFAR-10 image classification. The findings suggest that Vision Transformers can rival or exceed the performance of baseline CNNs. However, aspects like computational demands and model interpretability should guide the decision to employ Vision Transformers for specific applications.