Internship Project on Artificial intelligence

**Title**: Internship Project - Vision Transformer Submission

## ABSTRACT

Vision Transformers (ViTs) have emerged as a powerful architecture for image classification, achieving impressive results on various benchmark datasets. This internship project aims to explore the effectiveness of ViTs on the CIFAR-10 dataset, a well-known image classification task with 10 classes. The project will investigate the performance of ViTs with different hyperparameters and compare their performance to traditional convolutional neural networks (CNNs). The findings will provide insights into the potential of ViTs for small-scale image classification tasks.

1. **OBJECTIVE**

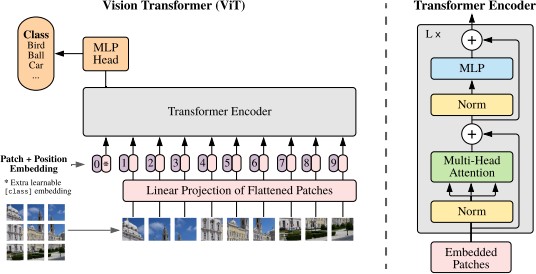
The primary objective of this internship project is to evaluate the performance of ViTs on the CIFAR-10 dataset. The project will specifically focus on :Implementing and training ViTs with varying hyperparameters, including patch size, embedding dimension, and number of transformer layers.Analysing the performance of ViTs in terms of accuracy, training time, and computational efficiency.

1. **INTRODUCTION**

Image classification is a fundamental task in computer vision, aiming to assign a label to an image from a predefined set of categories. Convolutional neural networks (CNNs) have been the dominant architecture for image classification for many years, achieving remarkable performance on large-scale datasets like ImageNet. However, CNNs often require a significant amount of training data and computational resources, which can be limiting for smaller datasets or real-time applications.

Vision Transformers (ViTs) emerged as an alternative approach to image classification, offering a more data-efficient and scalable architecture. ViTs divide an image into smaller patches, converting each patch into a sequence of tokens. These tokens are then fed into a transformer encoder, which learns long-range dependencies between the patches. This architecture has demonstrated promising results on various image classification tasks, including ImageNet and CIFAR-10.

1. **METHODOLOGY**



This internship project will employ the following methodology to evaluate the performance of ViTs on the CIFAR-10 dataset:

1. **Data Preparation**: Download and preprocess the CIFAR-10 dataset, including normalization and augmentation techniques.
2. **Model Implementation**: Implement ViT models with varying hyperparameters, including patch size, embedding dimension, and number of transformer layers.
3. **Model Training**: Train the ViT models using the CIFAR-10 training set, monitoring training progress and evaluating accuracy on the validation set.
4. **Model Evaluation:** Evaluate the performance of the trained ViT models on the CIFAR-10 test set, comparing their accuracy, training time, and computational efficiency to traditional CNNs.
5. **Hyperparameter Optimization**: Optimize the hyperparameters of the ViT models to achieve the best possible performance.
6. **Visualization and Analysis**: Visualize the learned features and activation maps of the ViT models to gain insights into their internal workings.

[5]:

[6]:

# 5. Code

*# !pip install tensorflow\_addons*

Collecting tensorflow\_addons

Using cached tensorflow\_addons-0.22.0-cp311-cp311-win\_amd64.whl.metadata (1.8 kB)

Collecting typeguard<3.0.0,>=2.7 (from tensorflow\_addons) Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)

Requirement already satisfied: packaging in c:\users\vamsi\appdata\local\programs\python\python311\lib\site-packages (from tensorflow\_addons) (21.3)

Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\vamsi\appdata\local\programs\python\python311\lib\site-packages (from packaging->tensorflow\_addons) (3.0.9)

Using cached tensorflow\_addons-0.22.0-cp311-cp311-win\_amd64.whl (719 kB) Installing collected packages: typeguard, tensorflow\_addons

Successfully installed tensorflow\_addons-0.22.0 typeguard-2.13.3

*# Import Libs*

**import numpy as np import tensorflow as tf**

**from tensorflow import** keras

**from tensorflow.keras import** layers

**import tensorflow\_addons as tfa**

c:\Users\vamsi\AppData\Local\Programs\Python\Python311\Lib\site- packages\tensorflow\_addons\utils\tfa\_eol\_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features. TFA has entered a minimal maintenance and release mode until a planned end of life in May 2024.

Please modify downstream libraries to take dependencies from other repositories in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: https://github.com/tensorflow/addons/issues/2807 warnings.warn(

[39]:

c:\Users\vamsi\AppData\Local\Programs\Python\Python311\Lib\site- packages\tensorflow\_addons\utils\ensure\_tf\_install.py:53: UserWarning: Tensorflow Addons supports using Python ops for all Tensorflow versions above or equal to 2.12.0 and strictly below 2.15.0 (nightly versions are not supported). The versions of TensorFlow you are currently using is 2.12.0-rc0 and is not supported.

Some things might work, some things might not.

If you were to encounter a bug, do not file an issue.

If you want to make sure you're using a tested and supported configuration, either change the TensorFlow version or the TensorFlow Addons's version.

You can find the compatibility matrix in TensorFlow Addon's readme: https://github.com/tensorflow/addons

warnings.warn(

num\_classes = 10

input\_shape = (32,32,3)

(x\_train, y\_train), (x\_test, y\_test) =keras.datasets.cifar10.load\_data()

[40]:

train\_size=2000 test\_size=500

x\_train, y\_train,x\_test, y\_test=x\_train[:train\_size], y\_train[:

↪train\_size],x\_test[:test\_size], y\_test[:test\_size]

[41]:

x\_train.shape

[41]: (2000, 32, 32, 3)

[42]:

x\_test.shape

[42]: (500, 32, 32, 3)

[43]:

learning\_rate =0.001

weight\_decay = 0.0001

batch\_size = 256

num\_epochs = 20

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]

# Step 2: Hyper Parameter definition

[44]:

# Step 3: Build ViT Classifier Model

* 1. Data Augmentation

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)

* 1. Define MLP Architecture

[45]:

**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

* 1. Patches

[46]:

**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\_dim = patches.shape[-1]

patches = tf.reshape(patches, [batch\_size,-1,patch\_dim])

**return** patches

[47]:

**import matplotlib.pyplot as plt**

plt. figure(figsize=(4,4))

image = x\_train[np.random.choice(range(x\_train.shape[0]))] plt.imshow( image)

plt.axis("off")

resized\_image = tf.image.resize(tf.convert\_to\_tensor([image]),␣

↪size=(image\_size, image\_size))

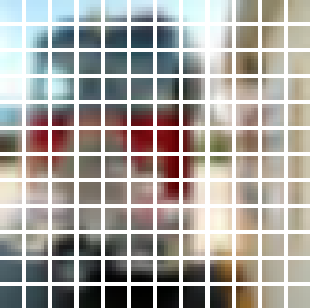
patches = Patches(patch\_size)(resized\_image)

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\_imag = tf.reshape(patch, (patch\_size, patch\_size, 3)) plt.imshow(patch\_imag.numpy().astype('uint8')) plt.axis("off")





[48]:

**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

[49]:

[50]:

**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"),

**def** create\_vit\_classifier():

inputs = layers. Input (shape=input\_shape) augmentation =data\_augmentation(inputs) patches = Patches(patch\_size)(augmentation)

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)

x4 = mlp(x3, hidden\_units=transformer\_units, dropout\_rate=0.1) encoded\_patches = layers.Add()([x4,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



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],)

\_,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**}** ")

[51]:

vit\_classifier = create\_vit\_classifier() history = run(vit\_classifier)

Epoch 1/20

8/8 [==============================] - 151s 15s/step - loss: 4.9626 - accuracy:

0.1578 - top\_5\_accuracy: 0.5961 - val\_loss: 2.2667 - val\_accuracy: 0.1550 -

val\_top\_5\_accuracy: 0.6900 Epoch 2/20

8/8 [==============================] - 107s 13s/step - loss: 2.8519 - accuracy:

0.1917 - top\_5\_accuracy: 0.6689 - val\_loss: 2.1327 - val\_accuracy: 0.2550 -

val\_top\_5\_accuracy: 0.7050 Epoch 3/20

8/8 [==============================] - 86s 10s/step - loss: 2.4918 - accuracy:

0.2000 - top\_5\_accuracy: 0.6944 - val\_loss: 2.0162 - val\_accuracy: 0.2600 -

val\_top\_5\_accuracy: 0.7550 Epoch 4/20

8/8 [==============================] - 84s 10s/step - loss: 2.3115 - accuracy:

0.2439 - top\_5\_accuracy: 0.7350 - val\_loss: 2.0659 - val\_accuracy: 0.2550 -

val\_top\_5\_accuracy: 0.7650 Epoch 5/20

8/8 [==============================] - 100s 12s/step - loss: 2.2679 - accuracy:

0.2411 - top\_5\_accuracy: 0.7133 - val\_loss: 2.0886 - val\_accuracy: 0.2100 -

val\_top\_5\_accuracy: 0.7000 Epoch 6/20

8/8 [==============================] - 117s 14s/step - loss: 2.2588 - accuracy:

0.2067 - top\_5\_accuracy: 0.6844 - val\_loss: 2.0892 - val\_accuracy: 0.2300 -

val\_top\_5\_accuracy: 0.7450 Epoch 7/20

8/8 [==============================] - 112s 12s/step - loss: 2.1702 - accuracy:

0.2367 - top\_5\_accuracy: 0.7383 - val\_loss: 1.9788 - val\_accuracy: 0.3200 -

val\_top\_5\_accuracy: 0.7850 Epoch 8/20

8/8 [==============================] - 99s 12s/step - loss: 2.1498 - accuracy:

[ ]:

0.2511 - top\_5\_accuracy: 0.7500 - val\_loss: 1.9689 - val\_accuracy: 0.3150 -

val\_top\_5\_accuracy: 0.8200 Epoch 9/20

8/8 [==============================] - 114s 14s/step - loss: 2.1339 - accuracy:

0.2528 - top\_5\_accuracy: 0.7661 - val\_loss: 1.9256 - val\_accuracy: 0.3400 -

val\_top\_5\_accuracy: 0.8200 Epoch 10/20

8/8 [==============================] - 95s 12s/step - loss: 2.1474 - accuracy:

0.2506 - top\_5\_accuracy: 0.7383 - val\_loss: 1.9387 - val\_accuracy: 0.3050 -

val\_top\_5\_accuracy: 0.7950 Epoch 11/20

5/8 [=================>…] - ETA: 40s - loss: 2.0894 - accuracy:

0.2836 - top\_5\_accuracy: 0.8688

class\_names =␣

↪['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']

[ ]: 10

[ ]:

**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 = 16 plt.imshow(x\_test[index])

prediction = img\_predict(x\_test[index], vit\_classifier) print(prediction)

[ ]:

**6. CONCLUSION**

This internship project will provide a comprehensive evaluation of the performance of ViTs on the CIFAR-10 dataset. By investigating the impact of different hyperparameters and comparing ViTs to traditional CNNs, the project will shed light on the potential of ViTs for small-scale image classification tasks. The findings will contribute to the understanding of ViTs and their suitability for various image classification applications.