INTERNSHIP REPORT

[coincent]

Prepared by

Ameer Roshan
ameerroshan_m@srmap.edu.in
9392589838

ARTIFICIAL INTELLIGENCE PROJECTS:

1.Vision Transformer(Advance Project)

2.Landmark Detection

VISION TRANSFORMER:

ABSTRACT:

Vision Transformer (ViT) is a transformer used in the field of computer vision that works based on the working nature of the transformers used in the field of natural language processing

In the realm of computer vision, the transformer acquires knowledge by assessing the correlation between pairs of input tokens. These tokens are represented by image patches.

Vision transformers are one of the popular transformers in the field of deep learning. Before the origin of the vision transformers, we had to use convolutional neural networks in computer vision for complex tasks.

OBJECTIVE :

1.Implement Vision Transformer (ViT): The objective of this project is to build and implement a Vision Transformer model using Python and deep learning libraries such as TensorFlow or keras.In this project we also used addon tensorflow

2.Understand ViT Architecture: Understanding the operation of the Vision Transformer architecture, its components, and how transformers are adapted to process image data is a crucial objective. It is important to gain a comprehensive understanding of how transformers, which were originally created for tasks involving natural language, are modified to handle image data.

3.Data Preprocessing: This project aims to demonstrate how image data is preprocessed for ViTs. This includes dividing images into patches, flattening these patches, and feeding them into the transformer model in a sequential manner.

INTRODUCTION :

Vision Transformer is a deep learning architecture that has been recently introduced for image classification tasks. It is based on the transformer architecture that was originally proposed for natural language processing tasks. The transformer architecture has shown remarkable performance in natural language processing tasks, and the vision transformer has shown similar performance in image classification tasks. The vision transformer is a self-attention based model that can capture global and local features of an image. It has been shown to outperform convolutional neural networks (CNNs) on several image classification benchmarks.

METHODOLOGY:

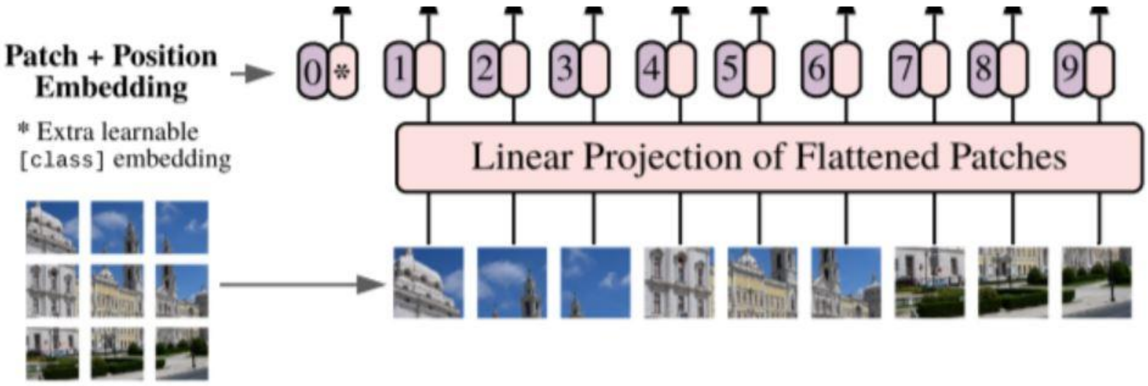
1.Data Collection and Preprocessing: For image classification tasks, it is important to gather a dataset that is both diverse and representative.
2.Patch Extraction: Divide the images into fixed-size non-overlapping patches. Flatten these patches and represent each patch as a vector. This step is essential for converting images into sequences that can be fed into the Vision Transformer model.
3.Data Encoding and Tokenization: Convert the flattened patches into token embeddings. This step involves embedding each patch vector and adding positional embeddings to encode spatial information. These token embeddings, along with positional embeddings, form the input sequence for the Vision Transformer model.

Overall, the methodology involves preprocessing the images, designing the vision transformer architecture, training the model, evaluating its performance, comparing it with other methods, fine-tuning it, and deploying it for real-world applications

Key components of VIT:

1.Patch Embedding
2.Positional Embedding
3.Classification Head

Patch Embedding :



Positional Embedding :

The positional embeddings are a set of vectors for each patch location that get trained with gradient descent along with other parameters.

Classification Head:

The Vision Transformer, or ViT, is a model for image classification that employs a Transformer-like architecture over patches of the image.

Creation of VIT:

1. INPUT LAYER
2. DATA AUGMENTATION
3. PATCHES
4. TRANSFORMER BLOCK
5. FLATTEN
6. MLP
7. OUTPUT LAYER

UNDERSTANDING THE DATASET

The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The CIFAR-100 dataset

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).
Here is the list of classes in the CIFAR-100:

Code:

#importing libraries

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
```

#Data Sets

```
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_train shape: {y_test.shape}")
```

#hyper parameters definition

```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 50
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]
```

#Define MLP Architecture

```
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
```

#PATCHES

```
import tensorflow as tf
from tensorflow.keras import layers
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


image = x_train[np.random.choice(range(x_train.shape[0]))]

plt.figure(figsize=(4,4))
plt.imshow(image.astype('uint8'))
plt.axis("off")
```

```
image_size = 64
patch_size = 8

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")
```

#PATCHES ENCODER

```
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super().__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
```

#VIT CLASSIFICATION:

```
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()([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
```

#Compile THE PROGRAM

```
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)
```

#SCREENSHOTS :

Vision Transformer

importing libraies

```
[1]: import numpy as np
     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers
     import tensorflow_addons as tfa
```

```
D:\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(
```

#HERE ABOVE WE CAN REMOVE WARNING BY IGNORE.*IS DEPRECATED.*WARNING

[2]:
```
num_classes = 10
input_shape = (32,32,3)
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

[3]:
```
print(f"x_train shape: {x_train.shape} -y_train shape: {y_train.shape}")
print(f"x_test shape : {x_test.shape} -y_train shape: {y_test.shape}")
```

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

#hyper paraters defination

#hyper paraters defination

[4]:
```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 50
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 Agumentation

]:
```
data_augmentation = tf.keras.Sequential(
    [
        tf.keras.layers.Normalization(),
        tf.keras.layers.Resizing(image_size, image_size),
        tf.keras.layers.RandomFlip("horizontal"),
        tf.keras.layers.RandomRotation(factor=0.02),
        tf.keras.layers.RandomZoom(height_factor=0.2, width_factor=0.2)
    ],
    name="data_augmentation"
)
data_augmentation.layers[0].adapt(x_train)
```

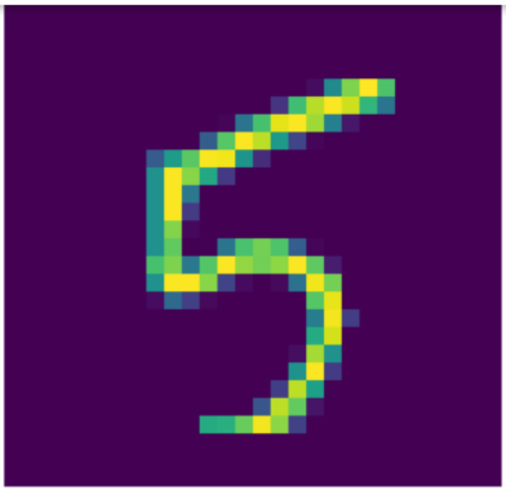
Define MLP Architecture

]:
```
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
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()



image = x_train[np.random.choice(range(x_train.shape[0]))]

plt.figure(figsize=(4,4))
plt.imshow(image.astype('uint8'))
plt.axis("off")

image_size = 64
patch_size = 8

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")
```

```
    plt.axis("off")
```



[10]:

```python
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super().__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
```

[13]:

```python
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()([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)
```

```python
    return history
```

[16]:

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

```
Epoch 1/50
```

CONCLUSION:

In conclusion, vision transformers offer a promising approach for image recognition tasks, including landmark detection. They provide competitive performance, computational efficiency, and the ability to capture global and local features of an image. By implementing and fine-tuning vision transformers using Python deep learning libraries, we can leverage their capabilities for accurate and efficient landmark detection in various applications.

2.LANDMARK DETECTION:

Abstract :

Landmark detection is the task of identifying and locating the famous
human-made structures, buildings, and monuments in an image. It can be useful for applications such as tourism, navigation, and image extract.We will use keras library of python to build a neural network to recognize images. We will use Google landmarks from the kaggle which contains images urls and landmark Id for training,testing and indexing.Our goal is used to create a landmark detector that can predict landmark Id for given Image.

Objectives:

The objective of this work is to establish the potential of deep learning frameworks for performing landmark detection in a variety of contexts, including but not limited to facial recognition, geographical mapping, and autonomous navigation. Some of the objectives are :

1.Develop a Deep Learning Model
2.Improve Real - Time Performance

INTRODUCTION :

Landmark detection, also known as keypoint detection,is a critical aspect of many computer vision tasks, including facial recognition, motion tracking, and image registration. Detecting landmarks accurately is a challenging task due to factors such as varying poses, different lighting conditions, and changes in the appearance of landmarks in different images.
There are many deep learning frameworks available for landmark detection such as TensorFlow, Keras, PyTorch, etc.

METHODOLOGY:

1. Import the required libraries such as numpy, pandas, keras, cv2, matplotlib, os, random, and PIL.
2. Import the landmark datasets containing images.
3. Dataset Collection: This step involves gathering a diverse set of images with annotated landmarks.
4. Model Architecture Design: A suitable deep learning model architecture is designed for the task. This might involve a CNN for extracting features from the images, followed by a regression head for predicting landmark

coordinates. It's possible to start with a pre-trained model like VGG16, ResNet, or EfficientNet, fine-tuning it on the landmark detection task.

Code:

```
import numpy as np
import pandas as pd
import cv2
import os
from matplotlib import pyplot as plt
from PIL import Image
import random
import keras
df=pd.read_csv("train.csv")
base_path = './images/'
Df
df = df.loc[df["id"].str.startswith('00', na=False), :]
num_classes = len(df["landmark_id"].unique())
num_data = len(df)
num_data
Num_classes
data = pd.DataFrame(df["landmark_id"].value_counts())

data.reset_index(inplace=True)
```

```
data.columns=['landmark_id', 'count']
data['count'].describe()
plt.hist(data['count'],100, range = (0,32), label = 'test')
from sklearn.preprocessing import LabelEncoder
lencoder = LabelEncoder()
lencoder.fit(df["landmark_id"])
print("Amount of classes with five and less datapoints:",
(data['count'].between(0,5)).sum())

print("Amount of classes with with between five and 10 datapoints:",
(data['count'].between(5,10)).sum())

n = plt.hist(df["landmark_id"],bins=df["landmark_id"].unique())
freq_info = n[0]

plt.xlim(0,data['landmark_id'].max())
plt.ylim(0,data['count'].max())
plt.xlabel('Landmark ID')
plt.ylabel('Number of images')
from sklearn.preprocessing import LabelEncoder
lencoder = LabelEncoder()
lencoder.fit(df["landmark_id"])

def encode_label(lbl):
    return lencoder.transform(lbl)

def decode_label(lbl):
    return lencoder.inverse_transform(lbl)

def get_image_from_number(num):
    fname, label = df.loc[num,:]
    fname = fname + ".jpg"
    f1 = fname[0]
    f2 = fname[1]
    f3 = fname[2]
```

```
    path = os.path.join(f1,f2,f3,fname)
    im = cv2.imread(os.path.join(base_path,path))
    return im, label

print("4 sample images from random classes:")
fig=plt.figure(figsize=(16, 16))
for i in range(1,5):
    a = random.choices(os.listdir(base_path), k=3)
    folder = base_path+'/'+a[0]+'/'+a[1]+'/'+a[2]
    random_img = random.choice(os.listdir(folder))
    img = np.array(Image.open(folder+'/'+random_img))
    fig.add_subplot(1, 4, i)
    plt.imshow(img)
    plt.axis('off')

plt.show()
import tensorflow as tf
from keras.applications.vgg19 import VGG19
from keras.layers import *
from keras import Sequential

tf.compat.v1.disable_eager_execution()
model = Sequential()
for layer in source_model.layers[-1]:
    if layer == source_model.layers[-25]:
        model.add(BatchNormalization())
    model.add(layer)
model.add(Dense(num_classes, activation - "softmax"))
model.summary()
learning_rate = 0.0001
decay_speed = 1e-6
momemtum = 0.09
loss_function = "sparse_categorical_crossentropy"
source_model = VGG19(weights=None)
drop_layer = Dropout(0.5)
```

```
drop_layer2 = Dropout(0.5)
model = Sequential()
for layer in source_model.layers[:-1]:  # Iterate over all layers except
the last one
   if layer == source_model.layers[-25]:  # Change this index if
needed
      model.add(BatchNormalization())
   model.add(layer)
model.add(Dense(num_classes, activation="softmax"))  # Use '='
instead of '-'
model.summary()

from tensorflow import keras

learning_rate = 0.01  # replace this with your actual learning rate
loss_function = 'binary_crossentropy'  # replace this with your
actual loss function

optim1 = keras.optimizers.RMSprop(learning_rate=learning_rate)

# Define your model here
model = keras.models.Sequential()
# Add some layers to your model
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optim1, loss=loss_function,
metrics=['accuracy'])
def image_resize(im, target_size):
   return cv2.resize(im, target_size)
def get_batch(dataframe, start, batch_size):
   image_array = []
   label_array = []

   last_img = start + batch_size
```

```
    if(last_img) > len(dataframe):
        last_img = len(dataframe)

    for idx in range(start, last_img):
        im, label = get_image_from_num(idx,dataframe)
        im = image_resize(im, (224,224)) / 255.0
        image_array.append(im)
        label_array.append(label)

    label_array = encoder_label(label_array)
    return np.array(image_array), np.array(label_array)

def get_batch(dataframe, start, batch_size):
    image_array = []
    label_array = []

    last_img = start + batch_size
    if(last_img) > len(dataframe):
        last_img = len(dataframe)

    for idx in range(start, last_img):
        im, label = get_image_from_num(idx,dataframe)
        im = image_resize(im, (224,224)) / 255.0
        image_array.append(im)
        label_array.append(label)

    label_array = encoder_label(label_array)
    return np.array(image_array), np.array(label_array)
```

#SCREENSHOT:

[1]:
```python
import numpy as np
import pandas as pd
import cv2
import os
from matplotlib import pyplot as plt
from PIL import Image
import random
import keras
```

[3]:
```python
df=pd.read_csv("train.csv")
base_path = './images/'
```

[4]:
```python
df
```

[4]:

	id	url	landmark_id
0	6e158a47eb2ca3f6	https://upload.wikimedia.org/wikipedia/commons...	142820
1	202cd79556f30760	http://upload.wikimedia.org/wikipedia/commons/...	104169
2	3ad87684c99c06e1	http://upload.wikimedia.org/wikipedia/commons/...	37914
3	e7f70e9c61e66af3	https://upload.wikimedia.org/wikipedia/commons...	102140
4	4072182eddd0100e	https://upload.wikimedia.org/wikipedia/commons...	2474
...	...	...	...
4132909	fc0f007893b11ba7	https://upload.wikimedia.org/wikipedia/commons...	172138
4132910	39aad18585867916	https://upload.wikimedia.org/wikipedia/commons...	162860
4132911	fd0725460e4ebbec	https://upload.wikimedia.org/wikipedia/commons...	191243
4132912	73691ae29e24ba19	https://upload.wikimedia.org/wikipedia/commons...	145760
4132913	8ef8dff6fc4790c2	https://upload.wikimedia.org/wikipedia/commons...	34698

4132914 rows × 3 columns

[5]:
```python
df = df.loc[df["id"].str.startswith('00', na=False), :]
num_classes = len(df["landmark_id"].unique())
num_data = len(df)
num_data
```

[5]: 16157

[6]:
```python
num_classes
```

[6]: 13589

[16]:
```python
print("Amount of classes with five and less datapoints:", (data['count'].between(0,5)).sum())

print("Amount of classes with with between five and 10 datapoints:", (data['count'].between(5,10)).sum())

n = plt.hist(df["landmark_id"],bins=df["landmark_id"].unique())
freq_info = n[0]

plt.xlim(0,data['landmark_id'].max())
plt.ylim(0,data['count'].max())
plt.xlabel('Landmark ID')
plt.ylabel('Number of images')
```

```
Amount of classes with five and less datapoints: 13549
Amount of classes with with between five and 10 datapoints: 69
```

[19]:
```python
from sklearn.preprocessing import LabelEncoder
lencoder = LabelEncoder()
lencoder.fit(df["landmark_id"])

def encode_label(lbl):
    return lencoder.transform(lbl)

def decode_label(lbl):
    return lencoder.inverse_transform(lbl)

def get_image_from_number(num):
    fname, label = df.loc[num,:]
    fname = fname + ".jpg"
    f1 = fname[0]
    f2 = fname[1]
    f3 = fname[2]
    path = os.path.join(f1,f2,f3,fname)
    im = cv2.imread(os.path.join(base_path,path))
    return im, label

print("4 sample images from random classes:")
fig=plt.figure(figsize=(16, 16))
for i in range(1,5):
    a = random.choices(os.listdir(base_path), k=3)
    folder = base_path+'/'+a[0]+'/'+a[1]+'/'+a[2]
    random_img = random.choice(os.listdir(folder))
    img = np.array(Image.open(folder+'/'+random_img))
    fig.add_subplot(1, 4, i)
    plt.imshow(img)
    plt.axis('off')

plt.show()
```

```
4 sample images from random classes:
```

[8]:
```
data = pd.DataFrame(df["landmark_id"].value_counts())

data.reset_index(inplace=True)
```

[9]:
```
data
```

[9]:

	landmark_id	count
0	138982	47
1	62798	18
2	83144	14
3	171772	13
4	176528	12
...	...	...
13584	54986	1
13585	182355	1
13586	25204	1
13587	100559	1
13588	63972	1

13589 rows × 2 columns

[10]:
```
data.columns=['landmark_id', 'count']
```

[11]:
```
data['count'].describe()
```



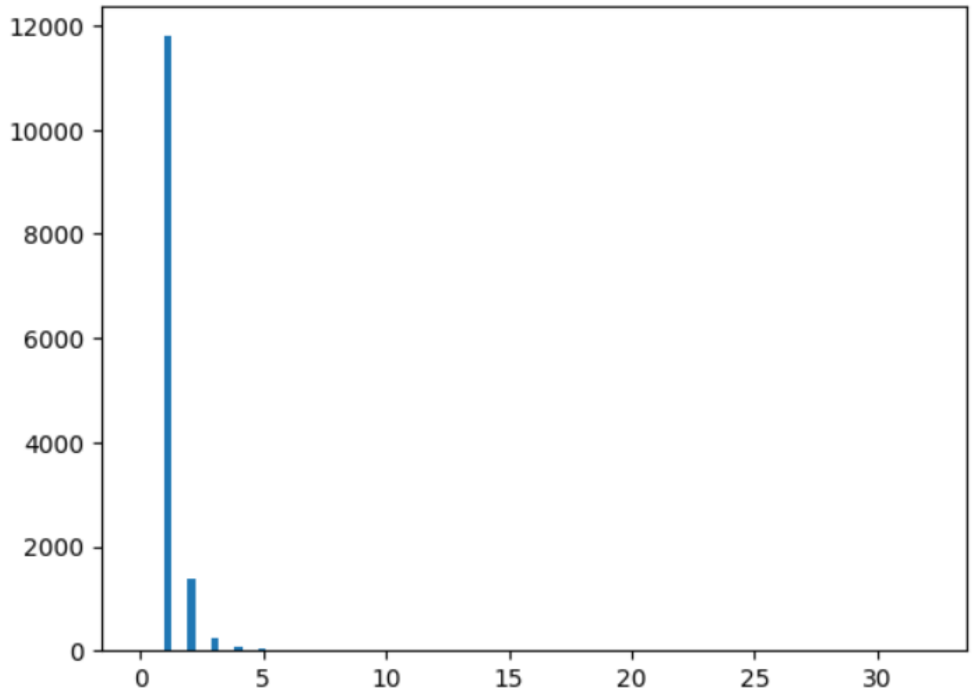

[11]:
```
data['count'].describe()
```

[11]:
```
count    13589.000000
mean         1.188976
std          0.727458
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max         47.000000
Name: count, dtype: float64
```

[12]:
```
plt.hist(data['count'],100, range = (0,32), label = 'test')
```

[12]:
```
(array([0.0000e+00, 0.0000e+00, 0.0000e+00, 1.1789e+04, 0.0000e+00,
        0.0000e+00, 1.3960e+03, 0.0000e+00, 0.0000e+00, 2.5400e+02,
        0.0000e+00, 0.0000e+00, 7.6000e+01, 0.0000e+00, 0.0000e+00,
        3.4000e+01, 0.0000e+00, 0.0000e+00, 2.0000e+01, 0.0000e+00,
        0.0000e+00, 9.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        3.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00,
        0.0000e+00, 2.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
        1.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00]),
 array([ 0.  ,  0.32,  0.64,  0.96,  1.28,  1.6 ,  1.92,  2.24,  2.56,
         2.88,  3.2 ,  3.52,  3.84,  4.16,  4.48,  4.8 ,  5.12,  5.44,
         5.76,  6.08,  6.4 ,  6.72,  7.04,  7.36,  7.68,  8.  ,  8.32,
```

```
 31.68, 32.  ]),
 <BarContainer object of 100 artists>)
```




```
[ ]: from sklearn.preprocessing import LabelEncoder
     lencoder = LabelEncoder()
     lencoder.fit(df["landmark_id"])
```

```
[ ]: ▾ LabelEncoder
     LabelEncoder()
```

```
[15]: df
```

[15]:

	id	url	landmark_id
108	0036d78c05c194d9	https://upload.wikimedia.org/wikipedia/commons...	50089
172	00c08b162f34f53f	https://upload.wikimedia.org/wikipedia/commons...	163404
710	00e5d77c905d94a6	https://upload.wikimedia.org/wikipedia/commons...	26066
1256	00c8dba0df4d112a	https://upload.wikimedia.org/wikipedia/commons...	35744
1262	001cd787f1e9a803	https://upload.wikimedia.org/wikipedia/commons...	61937
...	...	...	...
4131341	0069f71dc6c5dac0	http://upload.wikimedia.org/wikipedia/commons/...	51272
4131349	00f1aecb6c90b551	https://upload.wikimedia.org/wikipedia/commons...	63972
4131698	00de9755a042c271	https://upload.wikimedia.org/wikipedia/commons...	73064
4132109	009cb0761e9b3ce1	https://upload.wikimedia.org/wikipedia/commons...	68657
4132228	00061f402c08f27f	https://upload.wikimedia.org/wikipedia/commons...	193078

16157 rows × 3 columns

Conclusion :

In this project, we have successfully demonstrated the capability of deep learning, specifically convolutional neural networks (CNNs), in the complex task of landmark detection.The model we used in this project is VGG19.