Image_ranking_using_Autoencoders

December 12, 2022

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Base path for project

[2]:
```
base_path = "/content/drive/MyDrive/USML_Project"
```

Libraries import

[ ]:
```
import keras, tensorflow as tf
from keras import layers
from keras.utils import load_img, img_to_array
import shutil,csv,os,imageio,PIL,glob
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import keras.backend as K
from numpy import dot
from numpy.linalg import norm
```

Creating a training set by copying images from photos mapping image file names to different classes. And saving it to a file for later use.

[ ]:
```
pending_list =␣
  ↪["canyon","castle","coast","fish","flower","Horses","plane","skyline","sunset","Yosemite"]
train_directory = base_path + "/Train_set/Train_set_rgb"
mapping = []
for item in pending_list:
  directory = base_path + "/photos/" + item
  dir_list = os.listdir(directory)
  for file_t in dir_list:
    if file_t.endswith(".DS_Store"):
      continue
    mapping.append((file_t, item.split("_")[0].lower()))
    new_data = imageio.imread(directory + "/" + file_t)
    imageio.imsave(train_directory + "/" + file_t, new_data)
with open(base_path + '/train_mapping.csv','w', encoding='utf-8') as out:
```

```
    csv_out=csv.writer(out)
    csv_out.writerow(['file_name','class'])
    for row in mapping:
        csv_out.writerow(row)
```

Creating a dictionary to map classes with file names, will be used later.

[ ]:
```
dictionary = {}
csv_file_path = base_path + "/Train_set/train_mapping.csv"
mapping = []
with open(csv_file_path, newline='') as csvfile:
    reader = csv.reader(csvfile)
    counter = 0
    for row in reader:
        if counter == 0:
          counter += 1
          continue
        dictionary.setdefault(row[1],set()).add(row[0])
```

Loading mapping from saved mappings

[ ]:
```
csv_file_path = base_path + "/train_mapping.csv"
mapping = []
with open(csv_file_path, newline='') as csvfile:
    reader = csv.reader(csvfile)
    counter = 0
    for row in reader:
        if counter == 0:
          counter += 1
          continue
        mapping.append((row[0],row[1]))
```

Generating train set by reading images and converting them to vectors.

[ ]:
```
train_x_set = []
train_y_set = []
train_directory = base_path + "/Train_set/Train_set_rgb"
for item in mapping:
  file_name , cls = item
  directory = train_directory + "/" + file_name
  image = load_img(directory)
  img_array = img_to_array(image)
  train_x_set.append(img_array)
  train_y_set.append(cls)
```

Saving array representations on disk to be used later.

[ ]:
```
with open(base_path + "/train_x_dataset_rgb",'wb') as out:
    np.save(out, train_x_set)
```

```
with open(base_path + "/train_y_dataset_rgb",'wb') as out:
    np.save(out,train_y_set)
```

Function to load train dataset

[ ]:
```
def load_local_train_dataset():
  train_x = []
  train_y = []
  with open(base_path + "/train_x_dataset_rgb",'rb') as out:
      train_x = np.load(out)

  with open(base_path + "/train_y_dataset_rgb",'rb') as out:
      train_y = np.load(out)
  return np.array(train_x),np.array(train_y)
```

Loading training data

[ ]:
```
train_x_set,train_y_set = load_local_train_dataset()
```

[ ]:
```
train_x_set.shape
```

[ ]:
```
(1020, 256, 256, 3)
```

[ ]:
```
train_x_set[0].shape
```

[ ]:
```
(256, 256, 3)
```

[ ]:
```
train_x_set[0].shape
```

[ ]:
```
(256, 256, 1)
```

Building autoencoder model using keras Sequential

[ ]:
```
autoencoder_rgb = keras.Sequential([
    layers.Input(shape=(256,256,3)),
    layers.Conv2D(64,(4,4),padding='same',activation='relu'),
    layers.MaxPooling2D((2,2), padding='same'),
    layers.Conv2D(32,(4,4),padding='same',activation='relu'),
    layers.MaxPooling2D((2,2), padding='same'),
    layers.Conv2D(16,(4,4),padding='same',activation='relu'),
    layers.MaxPooling2D((2,2), padding='same'),
    layers.Conv2D(8,(4,4),padding='same',activation='relu'),
    layers.MaxPooling2D((2,2), padding='same'),
    layers.Conv2DTranspose(8,(4,4),padding='same',activation='relu'),
    layers.UpSampling2D((2,2)),
    layers.Conv2DTranspose(16,(4,4),padding='same',activation='relu'),
    layers.UpSampling2D((2,2)),
    layers.Conv2DTranspose(32,(4,4),padding='same',activation='relu'),
```

```
    layers.UpSampling2D((2,2)),
    layers.Conv2DTranspose(64,(4,4),padding='same',activation='relu'),
    layers.UpSampling2D((2,2)),
    layers.Conv2DTranspose(3,(4,4), padding='same'),
]);
```

Used only when loading the saved state of the model

[ ]:
```
autoencoder_rgb = keras.models.load_model(base_path+"/rgb_model")
```

structure of the model

[ ]:
```
autoencoder_rgb.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 64)      3136

 max_pooling2d (MaxPooling2D  (None, 128, 128, 64)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      32800

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 16)        8208

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 conv2d_3 (Conv2D)           (None, 32, 32, 8)         2056

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 8)        0
 2D)

 conv2d_transpose (Conv2DTra  (None, 16, 16, 8)        1032
 nspose)

 up_sampling2d (UpSampling2D  (None, 32, 32, 8)        0
 )

 conv2d_transpose_1 (Conv2DT  (None, 32, 32, 16)       2064
 ranspose)

 up_sampling2d_1 (UpSampling  (None, 64, 64, 16)       0
 2D)
```

```
conv2d_transpose_2 (Conv2DT  (None, 64, 64, 32)       8224
ranspose)

up_sampling2d_2 (UpSampling  (None, 128, 128, 32)     0
2D)

conv2d_transpose_3 (Conv2DT  (None, 128, 128, 64)     32832
ranspose)

up_sampling2d_3 (UpSampling  (None, 256, 256, 64)     0
2D)

conv2d_transpose_4 (Conv2DT  (None, 256, 256, 3)      3075
ranspose)

=================================================================
Total params: 93,427
Trainable params: 93,427
Non-trainable params: 0
_________________________________________________________________
```

compiling model with MSE as loss and adam optimizer

[ ]:
```
autoencoder_rgb.compile(loss='mse', optimizer='adam')
```

Creating a callback for early stopping of training

[ ]:
```
from keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss',min_delta=0, patience=10,␣
 ↪verbose=1,mode='auto')
```

Training the model

[ ]:
```
import  tensorflow as tf
tf.config.run_functions_eagerly(True)
history = autoencoder_rgb.fit(train_x_set, train_x_set, epochs=100,␣
 ↪callbacks=[early_stop], validation_split=0.1)
```

```
Epoch 1/100
29/29 [==============================] - 11s 385ms/step - loss: 1057.4625 -
val_loss: 999.4291
Epoch 2/100
29/29 [==============================] - 11s 382ms/step - loss: 1085.3124 -
val_loss: 1004.3936
Epoch 3/100
29/29 [==============================] - 11s 383ms/step - loss: 911.4535 -
val_loss: 834.5466
Epoch 4/100
29/29 [==============================] - 11s 385ms/step - loss: 848.9406 -
```

```
val_loss: 801.3876
Epoch 5/100
29/29 [==============================] - 11s 386ms/step - loss: 803.9859 -
val_loss: 792.2330
Epoch 6/100
29/29 [==============================] - 11s 388ms/step - loss: 789.1319 -
val_loss: 783.6547
Epoch 7/100
29/29 [==============================] - 11s 389ms/step - loss: 787.7593 -
val_loss: 780.7302
Epoch 8/100
29/29 [==============================] - 11s 378ms/step - loss: 776.4568 -
val_loss: 774.2553
Epoch 9/100
29/29 [==============================] - 11s 383ms/step - loss: 766.4744 -
val_loss: 766.3949
Epoch 10/100
29/29 [==============================] - 11s 382ms/step - loss: 758.9109 -
val_loss: 766.1528
Epoch 11/100
29/29 [==============================] - 11s 395ms/step - loss: 754.7207 -
val_loss: 758.1953
Epoch 12/100
29/29 [==============================] - 11s 395ms/step - loss: 751.1190 -
val_loss: 757.5303
Epoch 13/100
29/29 [==============================] - 11s 394ms/step - loss: 747.5236 -
val_loss: 754.9614
Epoch 14/100
29/29 [==============================] - 11s 382ms/step - loss: 748.8703 -
val_loss: 751.3146
Epoch 15/100
29/29 [==============================] - 11s 381ms/step - loss: 745.4163 -
val_loss: 752.8387
Epoch 16/100
29/29 [==============================] - 11s 384ms/step - loss: 740.0399 -
val_loss: 760.0253
Epoch 17/100
29/29 [==============================] - 11s 383ms/step - loss: 769.3708 -
val_loss: 777.1637
Epoch 18/100
29/29 [==============================] - 11s 394ms/step - loss: 739.3761 -
val_loss: 751.8456
Epoch 19/100
29/29 [==============================] - 11s 396ms/step - loss: 727.7024 -
val_loss: 739.3745
Epoch 20/100
29/29 [==============================] - 11s 395ms/step - loss: 728.1013 -
```

```
val_loss: 742.7032
Epoch 21/100
29/29 [==============================] - 11s 394ms/step - loss: 731.4163 -
val_loss: 742.2903
Epoch 22/100
29/29 [==============================] - 11s 384ms/step - loss: 746.4634 -
val_loss: 744.1838
Epoch 23/100
29/29 [==============================] - 11s 394ms/step - loss: 734.8120 -
val_loss: 743.8179
Epoch 24/100
29/29 [==============================] - 11s 385ms/step - loss: 733.6616 -
val_loss: 736.1763
Epoch 25/100
29/29 [==============================] - 11s 387ms/step - loss: 725.1230 -
val_loss: 737.6731
Epoch 26/100
29/29 [==============================] - 11s 396ms/step - loss: 723.0688 -
val_loss: 747.5722
Epoch 27/100
29/29 [==============================] - 11s 384ms/step - loss: 715.1095 -
val_loss: 729.5648
Epoch 28/100
29/29 [==============================] - 11s 385ms/step - loss: 725.6134 -
val_loss: 760.8331
Epoch 29/100
29/29 [==============================] - 11s 394ms/step - loss: 795.6917 -
val_loss: 832.7447
Epoch 30/100
29/29 [==============================] - 11s 393ms/step - loss: 810.5403 -
val_loss: 886.5372
Epoch 31/100
29/29 [==============================] - 11s 393ms/step - loss: 777.3632 -
val_loss: 785.6868
Epoch 32/100
29/29 [==============================] - 11s 394ms/step - loss: 724.9393 -
val_loss: 730.7233
Epoch 33/100
29/29 [==============================] - 11s 395ms/step - loss: 708.5316 -
val_loss: 725.4989
Epoch 34/100
29/29 [==============================] - 11s 397ms/step - loss: 702.6773 -
val_loss: 721.1461
Epoch 35/100
29/29 [==============================] - 11s 385ms/step - loss: 703.0277 -
val_loss: 719.0341
Epoch 36/100
29/29 [==============================] - 11s 394ms/step - loss: 701.9953 -
```

```
val_loss: 718.3367
Epoch 37/100
29/29 [==============================] - 11s 385ms/step - loss: 699.8366 -
val_loss: 719.7173
Epoch 38/100
29/29 [==============================] - 11s 386ms/step - loss: 699.6588 -
val_loss: 718.1902
Epoch 39/100
29/29 [==============================] - 11s 385ms/step - loss: 702.0925 -
val_loss: 717.6069
Epoch 40/100
29/29 [==============================] - 11s 395ms/step - loss: 697.9637 -
val_loss: 713.5549
Epoch 41/100
29/29 [==============================] - 12s 397ms/step - loss: 694.5691 -
val_loss: 724.8242
Epoch 42/100
29/29 [==============================] - 11s 395ms/step - loss: 703.0222 -
val_loss: 724.1604
Epoch 43/100
29/29 [==============================] - 11s 384ms/step - loss: 706.3488 -
val_loss: 717.3315
Epoch 44/100
29/29 [==============================] - 11s 395ms/step - loss: 696.2139 -
val_loss: 717.7568
Epoch 45/100
29/29 [==============================] - 11s 395ms/step - loss: 706.0579 -
val_loss: 709.9274
Epoch 46/100
29/29 [==============================] - 11s 385ms/step - loss: 697.4196 -
val_loss: 715.5581
Epoch 47/100
29/29 [==============================] - 11s 386ms/step - loss: 688.9799 -
val_loss: 707.9075
Epoch 48/100
29/29 [==============================] - 11s 394ms/step - loss: 690.9079 -
val_loss: 708.7930
Epoch 49/100
29/29 [==============================] - 11s 396ms/step - loss: 692.1851 -
val_loss: 709.3361
Epoch 50/100
29/29 [==============================] - 11s 394ms/step - loss: 692.0615 -
val_loss: 707.2184
Epoch 51/100
29/29 [==============================] - 11s 384ms/step - loss: 698.3539 -
val_loss: 715.7022
Epoch 52/100
29/29 [==============================] - 11s 395ms/step - loss: 710.7752 -
```

```
val_loss: 713.9114
Epoch 53/100
29/29 [==============================] - 11s 384ms/step - loss: 695.3658 -
val_loss: 713.4513
Epoch 54/100
29/29 [==============================] - 11s 394ms/step - loss: 691.3173 -
val_loss: 715.0610
Epoch 55/100
29/29 [==============================] - 11s 394ms/step - loss: 698.3843 -
val_loss: 716.5940
Epoch 56/100
29/29 [==============================] - 11s 394ms/step - loss: 719.3029 -
val_loss: 741.0532
Epoch 57/100
29/29 [==============================] - 11s 395ms/step - loss: 695.3605 -
val_loss: 715.4051
Epoch 58/100
29/29 [==============================] - 11s 394ms/step - loss: 679.4717 -
val_loss: 702.9741
Epoch 59/100
29/29 [==============================] - 11s 395ms/step - loss: 684.6108 -
val_loss: 708.6980
Epoch 60/100
29/29 [==============================] - 11s 395ms/step - loss: 709.0217 -
val_loss: 712.3141
Epoch 61/100
29/29 [==============================] - 11s 383ms/step - loss: 696.9835 -
val_loss: 719.4006
Epoch 62/100
29/29 [==============================] - 11s 395ms/step - loss: 684.9367 -
val_loss: 706.6915
Epoch 63/100
29/29 [==============================] - 11s 393ms/step - loss: 676.7970 -
val_loss: 702.6554
Epoch 64/100
29/29 [==============================] - 11s 385ms/step - loss: 674.3150 -
val_loss: 699.0760
Epoch 65/100
29/29 [==============================] - 11s 394ms/step - loss: 897.3273 -
val_loss: 1428.6724
Epoch 66/100
29/29 [==============================] - 11s 392ms/step - loss: 1026.4155 -
val_loss: 855.6586
Epoch 67/100
29/29 [==============================] - 11s 390ms/step - loss: 731.0219 -
val_loss: 714.7662
Epoch 68/100
29/29 [==============================] - 11s 380ms/step - loss: 683.1685 -
```

```
val_loss: 703.7599
Epoch 69/100
29/29 [==============================] - 11s 391ms/step - loss: 676.1260 -
val_loss: 701.1719
Epoch 70/100
29/29 [==============================] - 11s 382ms/step - loss: 671.2949 -
val_loss: 704.2927
Epoch 71/100
29/29 [==============================] - 11s 393ms/step - loss: 665.7676 -
val_loss: 699.3527
Epoch 72/100
29/29 [==============================] - 11s 393ms/step - loss: 663.1533 -
val_loss: 700.8326
Epoch 73/100
29/29 [==============================] - 11s 395ms/step - loss: 663.2010 -
val_loss: 692.9922
Epoch 74/100
29/29 [==============================] - 11s 394ms/step - loss: 656.7831 -
val_loss: 712.6542
Epoch 75/100
29/29 [==============================] - 11s 393ms/step - loss: 658.1022 -
val_loss: 705.5965
Epoch 76/100
29/29 [==============================] - 11s 382ms/step - loss: 657.8465 -
val_loss: 691.5184
Epoch 77/100
29/29 [==============================] - 11s 383ms/step - loss: 647.4288 -
val_loss: 693.8780
Epoch 78/100
29/29 [==============================] - 11s 393ms/step - loss: 638.5247 -
val_loss: 685.8611
Epoch 79/100
29/29 [==============================] - 11s 393ms/step - loss: 643.4665 -
val_loss: 692.4497
Epoch 80/100
29/29 [==============================] - 11s 382ms/step - loss: 644.2767 -
val_loss: 697.1867
Epoch 81/100
29/29 [==============================] - 11s 394ms/step - loss: 645.5698 -
val_loss: 688.7878
Epoch 82/100
29/29 [==============================] - 11s 383ms/step - loss: 634.1896 -
val_loss: 687.6802
Epoch 83/100
29/29 [==============================] - 11s 381ms/step - loss: 640.0814 -
val_loss: 711.1776
Epoch 84/100
29/29 [==============================] - 11s 381ms/step - loss: 656.4458 -
```

```
val_loss: 701.8090
Epoch 85/100
29/29 [==============================] - 11s 392ms/step - loss: 632.5416 -
val_loss: 684.5847
Epoch 86/100
29/29 [==============================] - 11s 393ms/step - loss: 625.5182 -
val_loss: 679.2968
Epoch 87/100
29/29 [==============================] - 11s 394ms/step - loss: 646.3860 -
val_loss: 815.4092
Epoch 88/100
29/29 [==============================] - 11s 393ms/step - loss: 676.3460 -
val_loss: 693.9429
Epoch 89/100
29/29 [==============================] - 11s 391ms/step - loss: 629.3938 -
val_loss: 680.6908
Epoch 90/100
29/29 [==============================] - 11s 392ms/step - loss: 622.2228 -
val_loss: 681.7325
Epoch 91/100
29/29 [==============================] - 11s 393ms/step - loss: 626.4916 -
val_loss: 684.1144
Epoch 92/100
29/29 [==============================] - 11s 394ms/step - loss: 624.9645 -
val_loss: 679.4354
Epoch 93/100
29/29 [==============================] - 11s 382ms/step - loss: 631.0356 -
val_loss: 692.3539
Epoch 94/100
29/29 [==============================] - 11s 383ms/step - loss: 625.2288 -
val_loss: 683.0153
Epoch 95/100
29/29 [==============================] - 11s 393ms/step - loss: 629.0433 -
val_loss: 673.4877
Epoch 96/100
29/29 [==============================] - 11s 382ms/step - loss: 621.3344 -
val_loss: 684.8497
Epoch 97/100
29/29 [==============================] - 11s 382ms/step - loss: 627.7305 -
val_loss: 700.4859
Epoch 98/100
29/29 [==============================] - 11s 391ms/step - loss: 675.0001 -
val_loss: 699.8235
Epoch 99/100
29/29 [==============================] - 11s 382ms/step - loss: 761.4604 -
val_loss: 700.2941
Epoch 100/100
29/29 [==============================] - 11s 391ms/step - loss: 631.3830 -
```

```
val_loss: 674.1476
```

Saving the model on to disk

[ ]:
```
autoencoder_rgb.save(base_path+"/rgb_model")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
5 of 9). These functions will not be directly callable after loading.
```

Plotting decoder output for one train image

[ ]:
```
plot_image(autoencoder_rgb.predict(train_x_set[:1000])[700])
```

```
32/32 [==============================] - 3s 92ms/step
```

Plotting actual image

[ ]:
```
plot_image(train_x_set[700])
```

defining utility functions

[ ]:
```
def plot_image(image):
  plt.imshow(image.astype("uint8"), cmap='binary')
  plt.axis('off')

def show_reconstruction(model,data_set, n_image=5):
  reconstructions = model.predict(data_set[:n_image])

  plt.figure(figsize=(n_image * 5, 10))
  for image_idx in range(n_image):
    plt.subplot(2,n_image, image_idx + 1)
    plot_image(data_set[image_idx].reshape(256,256,3))
    plt.subplot(2, n_image, n_image + image_idx + 1)
    plot_image(reconstructions[image_idx])

def plot_images(array):
  n_image = len(array)
  plt.figure(figsize=(n_image * 5, 15))
  for idx in range(len(array)):
    img = array[idx]
    path = base_path + "/" + img[1] + "/" + img[2]
    plt.subplot(2, n_image/2, idx+1)
    img = mpimg.imread(path)
    imgplot = plt.imshow(img)
    # plt.show()
    plt.axis('off')
```

Creating utility functions for extracting feature vector that will be used as inverted index of test images.

```
[ ]: base_path = "/content/drive/MyDrive/USML_Project/Test_set"
classes =␣
 ↪["canyon","castle","coast","fish","flower","horses","plane","skyline","sunset","yosemite"]
inverted_index = []

def get_image_as_array(path):
  image = load_img(path)
  img_array = img_to_array(image)
  img_array = [img_array]
  img_array = np.array(img_array)
  img_array = img_array.reshape(-1, 256, 256, 3)
  return img_array

def get_vector(img_array):
  get_all_layer_outputs = K.function([autoencoder_rgb.layers[0].input],
                                     [l.output for l in autoencoder_rgb.layers[1:
 ↪]])
  layer_output = get_all_layer_outputs([img_array])
  return layer_output[7].flatten()
```

Calculating inverted indices of all the test images

```
[ ]: for item in classes:
  directory_path = base_path + "/" + item
  for file_t in os.listdir(directory_path):
    img_array = get_image_as_array(directory_path + "/" + file_t)
    vector = get_vector(img_array)
    inverted_index.append((vector, item, file_t))
```

saving inverted index to file

```
[ ]: with open(base_path + '/inverted_index.csv','w', encoding='utf-8') as out:
    csv_out=csv.writer(out, delimiter=";")
    csv_out.writerow(['vector','class','file_name'])
    for row in inverted_index:
        v = ",".join(map(str, row[0].tolist()))
        # v = ",".join(row[0].tolist())
        csv_out.writerow([v, row[1],row[2]])
```

loading inverted indices from csv

```
[ ]: csv_file_path = base_path + "/Test_set/inverted_index.csv"
inverted_index_read = []
with open(csv_file_path, newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=";")
    counter = 0
```

```
    for row in reader:
        if counter == 0:
          counter += 1
          continue
        vector = row[0].split(",")
        vector = [float(i) for i in vector]
        inverted_index_read.append((vector,row[1],row[2]))
```

Defining cosine similarity function

[ ]:
```
def cosine_sim(a,b):
  cos_sim = dot(a, b)/(norm(a)*norm(b))
  return cos_sim
```

Defining function to ranking images on query and fetch top n items

[ ]:
```
def get_top_images(query_v,n_images=10):
  ranking_list = []
  for row in inverted_index_read:
    c_sim = cosine_sim(query_v, row[0])
    ranking_list.append((c_sim, row[1], row[2]))

  ranking_list = sorted(ranking_list,key=lambda x: x[0], reverse=True)
  return ranking_list[:n_images]
```

Loading query image for Castle

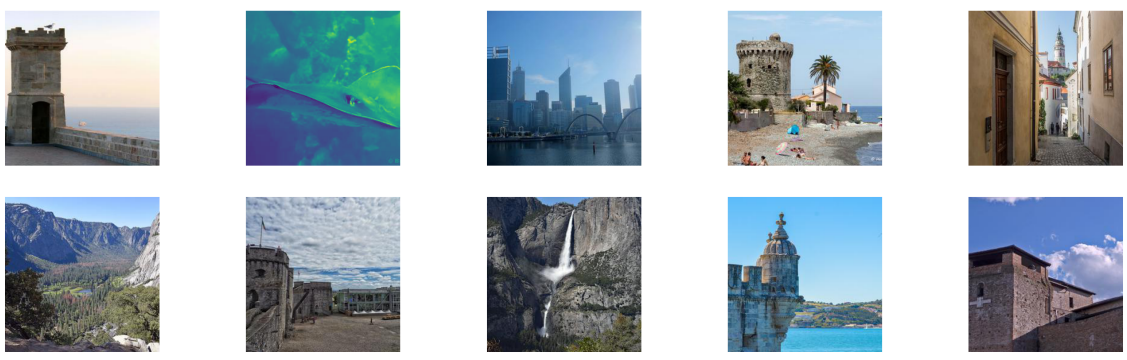
[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/castle/34403103904_9e829d548b_b.jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

Fetching top ten results for Castle

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.9161879799863897, 'castle', '36425144026_39d50f2865_b.jpg'),
 (0.9055926804287009, 'fish', '38553224962_0c2d03096c_b.jpg'),
 (0.8940093424955537, 'skyline', '42111801552_acfe8ebfec_b.jpg'),
 (0.8933622710024751, 'castle', '35790918262_489c4edb64_b.jpg'),
 (0.8932932785771729, 'castle', '38783856302_7824a35f29_b.jpg'),
 (0.892049411684046, 'yosemite', '34511381583_2f260b41c7_b.jpg'),
 (0.8902909883208547, 'castle', '36333957990_370882dc58_b.jpg'),
 (0.8902571055658706, 'yosemite', '34478469674_fa96a4568a_b.jpg'),
 (0.8896620794263659, 'castle', '35826774830_99ed30826e_b.jpg'),
 (0.8892161772682207, 'castle', '35609254672_a99bf6f4e7_b.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 6/10

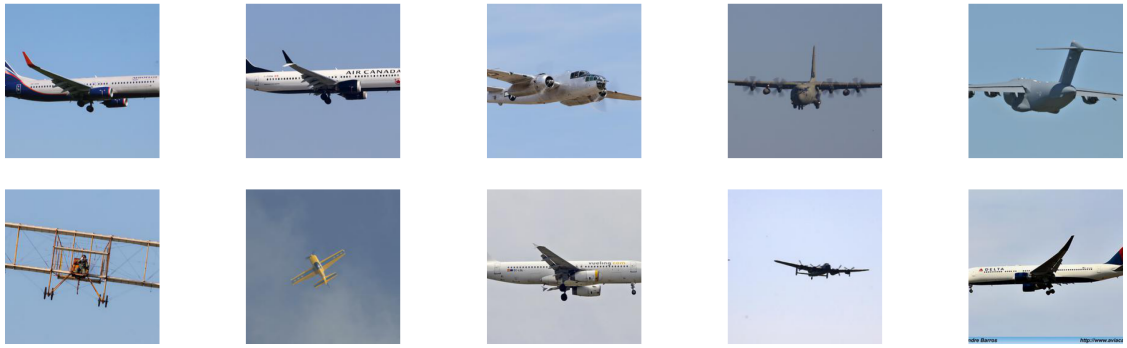
Loading query image for plane

[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/plane/27111282547_57c0757e7a_b.jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.9879310543500209, 'plane', '28088869348_69ef8b1908_b.jpg'),
 (0.9834426547474893, 'plane', '41254620434_0bc73915c2_b.jpg'),
 (0.9823121392969798, 'plane', '28290446988_0c373b6ce9_b.jpg'),
 (0.9800514541915849, 'plane', '28085617228_f7a5238251_b.jpg'),
 (0.9800364313593162, 'plane', '40152180770_64630c3288_b.jpg'),
 (0.9795508915790013, 'plane', '41156999705_4f50af642a_b.jpg'),
 (0.9789947153924176, 'plane', '41118633245_f00c234d24_o.jpg'),
 (0.9785830615265536, 'plane', '40294067180_48ebe10441_b.jpg'),
 (0.9773045002161095, 'plane', '41238385874_89bb73d51a_b.jpg'),
 (0.9765119411732779, 'plane', '28264757578_641f4b6768_o.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 10/10

Loading query image for canyon

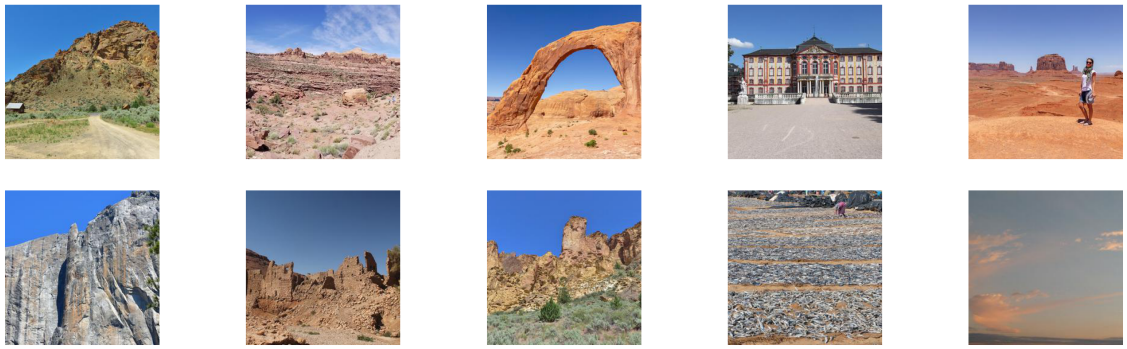
[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/canyon/27657642367_21a8409a11_b.jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

Fetching top ten results for Canyon

```
[ ]: result = get_top_images(query_vector, n_images = 10)
     result
```

```
[ ]: [(0.9615842576677092, 'canyon', '28980380768_f10dd70926_b.jpg'),
      (0.9565154376857008, 'canyon', '41960826744_ea66e0a596_b.jpg'),
      (0.9538129782971234, 'canyon', '42073916605_6632bbe812_b.jpg'),
      (0.9516893947167181, 'castle', '36269543091_daccbccb7b_b.jpg'),
      (0.9484740142364605, 'canyon', '42074252235_de80a40c05_b.jpg'),
      (0.9483136585881974, 'yosemite', '35036118031_04434e8269_b.jpg'),
      (0.9444713680253227, 'canyon', '41091205855_d937768c0e_b.jpg'),
      (0.944175703222832, 'canyon', '42135157694_4ccd884658_b.jpg'),
      (0.9439877943219503, 'coast', '41248266091_6dc888305f_b.jpg'),
      (0.9429715054462495, 'sunset', '34519933926_8a6dbf1c4d_b.jpg')]
```

```
[ ]: plot_images(result)
```



Precision @10 = 7/10

Loading query image for sunset

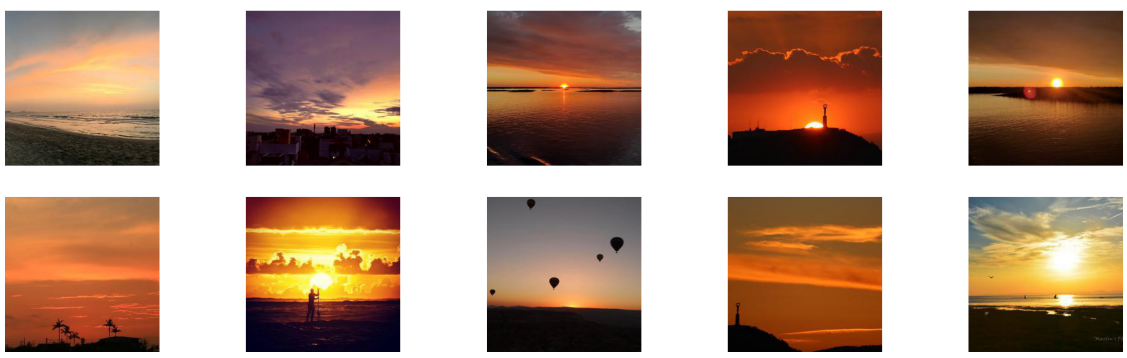
```
[ ]: train_directory = "/content/drive/MyDrive/USML_Project"
     query_img_path = train_directory + "/photos/sunset/34105340450_8aa36d5319_b.jpg"
     query_img = get_image_as_array(query_img_path)
     query_vector = get_vector(query_img)
     query_vector = query_vector.tolist()
     plot_image(query_img.reshape(256,256,3))
```

Fetching top ten results for Sunset

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.9224282986926251, 'sunset', '34212829550_64baeeb0b5_b.jpg'),
 (0.921541497627681, 'sunset', '34829757200_db5023f7a2_b.jpg'),
 (0.9197896627909949, 'sunset', '34966352085_343967c307_b.jpg'),
 (0.9196461196401741, 'sunset', '34892731462_c46311ddb1_b.jpg'),
 (0.9172314655282914, 'sunset', '34966360235_922e601045_b.jpg'),
 (0.9168220780900928, 'sunset', '35021274062_cb2eb4f150_b.jpg'),
 (0.9165008291064468, 'sunset', '34750844126_2506774849_o.jpg'),
 (0.9154585675737733, 'sunset', '34286161712_e89de32aea_o.jpg'),
 (0.9154455494457998, 'sunset', '35093138891_96fe34f899_b.jpg'),
 (0.9116963891672631, 'sunset', '34455495592_645113cda1_b.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 10/10

Loading query image for Yosemite

```
[ ]: train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/Yosemite/23473046368_f487a8effa_b.
  ↪jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

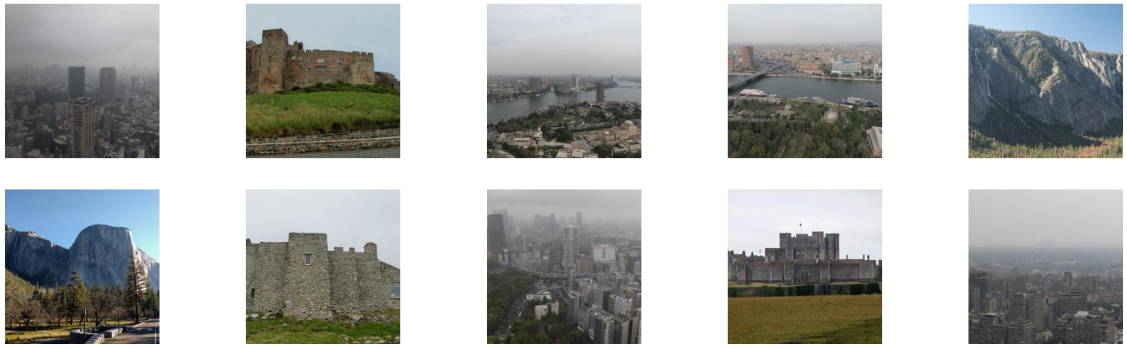
Fetching top ten results for Yosemite

```
[ ]: result = get_top_images(query_vector, n_images = 10)
result
```

```
[ ]: [(0.9549944769647133, 'skyline', '42157683772_45ee94fc3c_b.jpg'),
 (0.9489920456171191, 'castle', '35764582541_8d5c9271dc_b.jpg'),
 (0.9488606437748035, 'skyline', '41728233182_a7a15a2b45_b.jpg'),
 (0.9478340277299144, 'skyline', '41728172902_6f79e75e65_b.jpg'),
 (0.9475128953278644, 'yosemite', '35128316542_8ed07ccc50_b.jpg'),
 (0.9473908350377312, 'yosemite', '33874569516_3a89577127_b.jpg'),
 (0.9467348682327347, 'castle', '35710651516_df47a6e54b_b.jpg'),
 (0.9466159230507061, 'skyline', '41483429554_c85cc5b142_b.jpg'),
```

```
 (0.9463839876364306, 'castle', '36819291832_aa924a79cf_o.jpg'),
 (0.945904155891236, 'skyline', '41769452391_a440a73b66_b.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 2/10

Loading query image for Horses

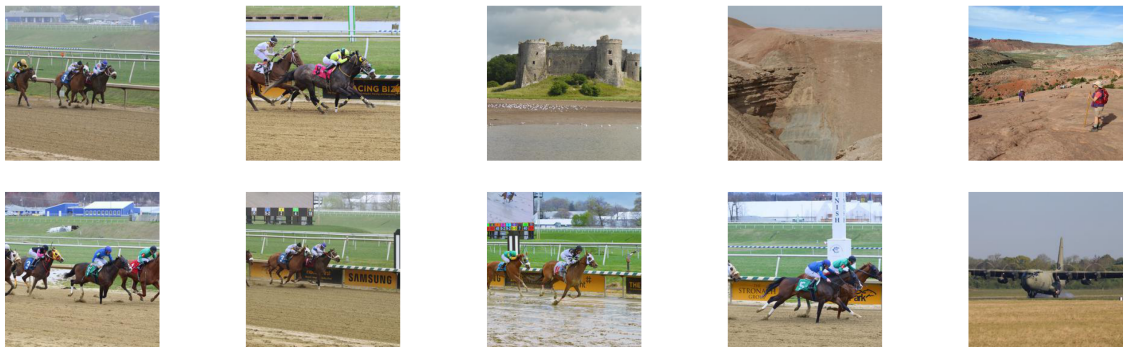
[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/Horses/27960469738_5270859e5c_b.jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

Fetching top ten results for Horses

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.9607788622023624, 'horses', '41580827871_3941404f68_b.jpg'),
 (0.9600595373881117, 'horses', '40631252834_cef4910330_b.jpg'),
 (0.9574282415398612, 'castle', '35019748294_d17bc55437_o.jpg'),
 (0.9549532277497674, 'canyon', '28529107368_4044ddcfc1_b.jpg'),
 (0.9532622549574957, 'canyon', '42245181621_0c2b38a635_b.jpg'),
 (0.9529217715918447, 'horses', '40631280294_8b5d8709c1_b.jpg'),
 (0.9528778065353486, 'horses', '41580827261_b02b9aac22_b.jpg'),
 (0.9528262485774248, 'horses', '27960473818_ae099cb2c8_b.jpg'),
 (0.9507074339018043, 'horses', '40449378415_085a068029_b.jpg'),
 (0.9495776677840234, 'plane', '28085618898_f76564e128_b.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 7/10

Loading query image for flower

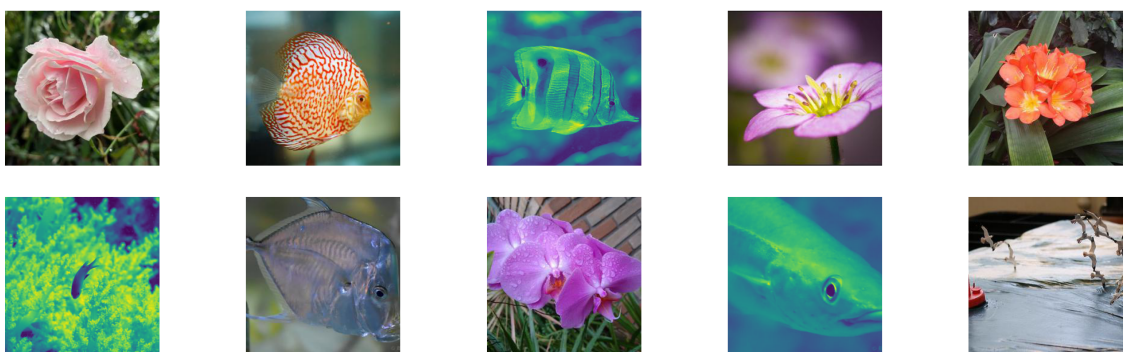
[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/flower/25789056418_2536536e75_b.jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

Fetching top ten results for Flower

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.900163699829225, 'flower', '40059779311_392636352c_b.jpg'),
 (0.8978430950548324, 'fish', '26049196157_1013095bed_b.jpg'),
 (0.8934617870219087, 'fish', '32759423671_fb01507dee_b.jpg'),
 (0.892709223028908, 'flower', '40029902422_0d6d377ac8_b.jpg'),
 (0.8925785414373374, 'flower', '40516257802_3757c0d127_b.jpg'),
 (0.8919650596149049, 'fish', '33249504823_245d4eb6aa_b.jpg'),
 (0.8910652792126867, 'fish', '31130007220_b86309bf69_b.jpg'),
 (0.8902181305345618, 'flower', '40284231121_7015704512_b.jpg'),
 (0.8893390224677884, 'fish', '38883088392_9ce22ac81e_b.jpg'),
 (0.8878112652818358, 'coast', '27931033039_5a13d089ba_b.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 4/10

Loading query image for skyline

[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/skyline/27301864387_43a7103e27_b.
 ↪jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

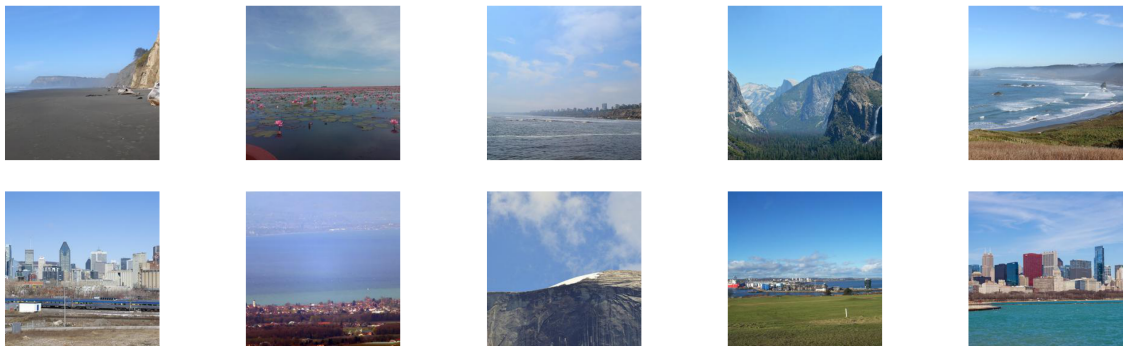
Fetching top ten results for skyline

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.9530390995413773, 'coast', '39258702704_e23a015606_b.jpg'),
 (0.9497426230091861, 'flower', '39721488281_5b9a5af2a8_b.jpg'),
 (0.9462258454991901, 'skyline', '28030620078_f4451e11b7_b.jpg'),
 (0.9458846762261296, 'yosemite', '34792238923_fc2c659c45_b.jpg'),
 (0.9456278342776787, 'coast', '39071104455_f646e30cb4_b.jpg'),
 (0.9443481415933743, 'skyline', '40431716245_6126403530_b.jpg'),
 (0.9440901684463675, 'skyline', '40746365164_9df9296082_b.jpg'),
 (0.9436220433090594, 'yosemite', '34357957463_5fdaaca553_b.jpg'),
```

```
(0.9435748951753284, 'coast', '39731892024_90a55df3af_b.jpg'),
(0.9434607401632831, 'skyline', '40497332855_ca848c8152_o.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 9/10

Loading query image for coast

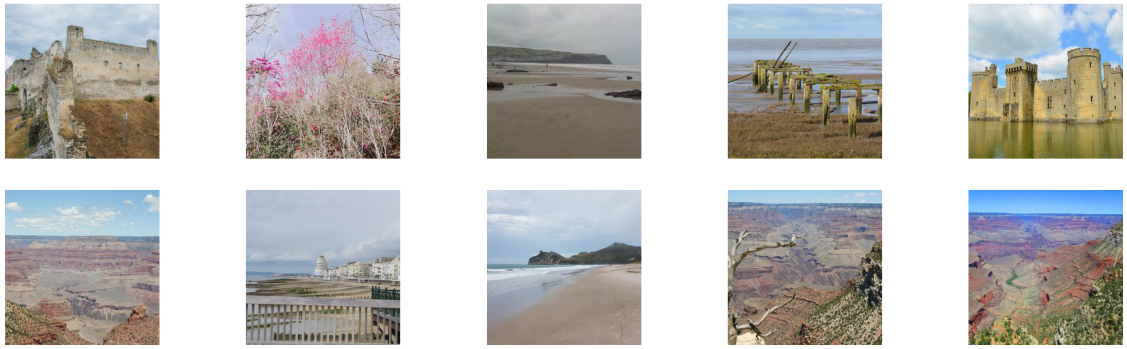
[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/coast/25284499637_1bd4ac8fd5_b.jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

Fetching top ten results for coast

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.9585521414313244, 'castle', '35794127462_6fe5020cdf_b.jpg'),
 (0.9557335534280901, 'flower', '39041016765_cc1b119ef3_b.jpg'),
 (0.955660503488693, 'coast', '40344363281_3dc8eeb6d0_b.jpg'),
 (0.9553558246472469, 'coast', '38783679600_0e54c9519d_b.jpg'),
 (0.9534968064636672, 'castle', '38741690921_fc4055d6f0_b.jpg'),
 (0.9531727480699462, 'canyon', '42092735292_09d9776d4f_b.jpg'),
 (0.9531548500393566, 'castle', '35740809966_4c3a6b566c_b.jpg'),
 (0.9520081911518117, 'coast', '40179395152_2870c4ec85_b.jpg'),
 (0.9518382527361359, 'canyon', '42092743222_df8fd66f1c_b.jpg'),
 (0.951006931559627, 'canyon', '42092802252_a91a7f3f35_b.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 5/10

Loading query image for fish

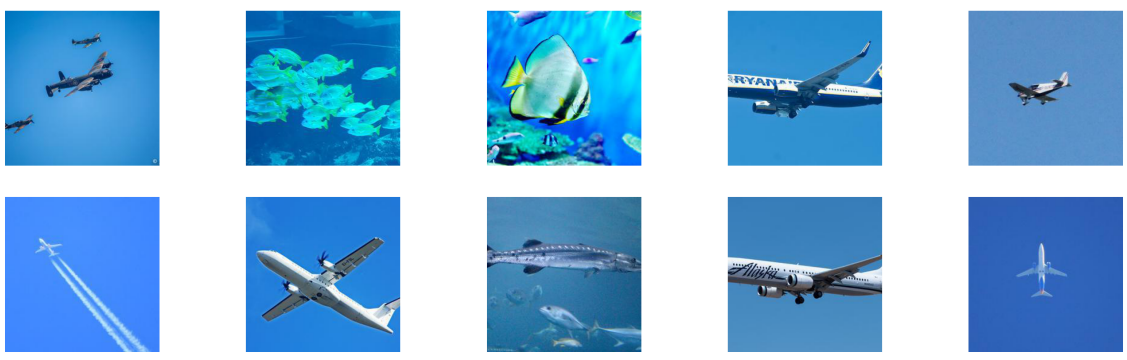
[ ]:
```
train_directory = "/content/drive/MyDrive/USML_Project"
query_img_path = train_directory + "/photos/fish/24708571066_c8e2f3fcdd_b.jpg"
query_img = get_image_as_array(query_img_path)
query_vector = get_vector(query_img)
query_vector = query_vector.tolist()
plot_image(query_img.reshape(256,256,3))
```

Fetching top ten results for fish

[ ]:
```
result = get_top_images(query_vector, n_images = 10)
result
```

[ ]:
```
[(0.9307845070484568, 'plane', '41034094805_40414840d3_b.jpg'),
 (0.9296254175064248, 'fish', '38117891421_711dc44fec_b.jpg'),
 (0.9219321579399836, 'fish', '28632956983_c2ac050539_b.jpg'),
 (0.9213055839893747, 'plane', '28105475198_75cf2729ff_b.jpg'),
 (0.915253184473874, 'plane', '41033470135_cc90dd0fd6_b.jpg'),
 (0.9134859157673431, 'plane', '40361150690_c4805a02ac_b.jpg'),
 (0.9120556481205442, 'plane', '28151100478_75eb93df64_b.jpg'),
 (0.9111363293929324, 'fish', '27265533355_4549a8c770_b.jpg'),
 (0.9104304709183497, 'plane', '41940567882_900be28d6d_b.jpg'),
 (0.9097856338749392, 'plane', '28099494658_3d6dfcfb9c_b.jpg')]
```

[ ]:
```
plot_images(result)
```



Precision @10 = 3/10