|  |
| --- |
| Image Featurization Applied to Find Similar Images:- |
| https://github.com/Microsoft/ML-Server-Python-Samples/blob/master/microsoftml/101/plot\_image\_featurizer\_match.py | ================================================== |
|  |  |
|  | Here is the scenario this sample addresses: You have a catalog |
|  | of images in a repository. When you get a new image, you want |
|  | to find the image from your catalog that most closely matches |
|  | this new image. |
|  | The procedure for finding the best match has the following steps: |
|  |  |
|  | - Locate the images in the catalogue and get their feature vectors. |
|  | - Locate the new image and get its feature vector. |
|  | - Find out which image or set of images from the catalog has the |
|  | smallest "distance" from the new image. There are a number of |
|  | ways to calculate this distance. A simple one is the Euclidean |
|  | distance, which we use in this sample. |
|  |  |
|  | In this sample, our intial catalog consists a set of pictures of fish and helicopters. |
|  | First, create a dataframe with the locations of these images: |
|  |  |
|  | .. index:: image, similarity |
|  | """ |
|  | import os |
|  |  |
|  | try: |
|  | root = os.path.dirname(\_\_file\_\_) |
|  | except NameError: |
|  | # \_\_file\_\_ does not exist in a notebook |
|  | root = "." |
|  |  |
|  | # An absolute path must be used if the current folder |
|  | # is not the script's one. |
|  | image\_location = os.path.abspath(os.path.join(root, "Data", "Pictures")) |
|  |  |
|  | ############################ |
|  | # Specify paths to the images we want to featurize. |
|  |  |
|  | images = [] |
|  | for im in ["Fish/Fish1.jpg", "Fish/Fish2.jpg", |
|  | "Helicopter/Helicopter1.jpg", "Helicopter/Helicopter2.jpg"]: |
|  | images.append(os.path.join(image\_location, im)) |
|  |  |
|  | ############################### |
|  | # Let's plot the image to see what they look like. |
|  |  |
|  | import matplotlib.pyplot as plt |
|  | from PIL import Image |
|  | fig, ax = plt.subplots(2, 2) |
|  | for i, im in enumerate(images): |
|  | ax[i // 2, i % 2].imshow(Image.open(im)) |
|  |  |
|  | ################ |
|  | # Setup a dataframe with the path to the image. |
|  |  |
|  | import pandas |
|  | image\_df = pandas.DataFrame(data=dict(image=images)) |
|  | print(image\_df) |
|  |  |
|  | ######################### |
|  | # Then, get the corresponding feature vectors for each |
|  | # of the catalog images into a dataframe. |
|  | # We follow the process mentioned at :ref:`l-imgfeat`. |
|  | # We load, resize, convert into pixels and finally build |
|  | # vectors from images. |
|  |  |
|  | from microsoftml import rx\_featurize, load\_image, resize\_image, extract\_pixels, featurize\_image |
|  | image\_vector = rx\_featurize(data=image\_df, ml\_transforms=[ |
|  | load\_image(cols=dict(Features="image")), |
|  | resize\_image(cols="Features", width=227, height=227), |
|  | extract\_pixels(cols="Features"), |
|  | featurize\_image(cols="Features", dnn\_model="Alexnet")]) |
|  |  |
|  | print(image\_vector.head()) |
|  |  |
|  | ################################### |
|  | # Secondly, create a dataframe with the location of |
|  | # the new image to match and get its feature vector into a dataframe. |
|  |  |
|  | images\_match = [] |
|  | for im in ["Fish/Fish4.jpg"]: |
|  | images\_match.append(os.path.join(image\_location, im)) |
|  |  |
|  | fig, ax = plt.subplots(1, 1) |
|  | ax.imshow(Image.open(images\_match[0])) |
|  |  |
|  | image\_match\_df = pandas.DataFrame(data=dict(image=images\_match)) |
|  |  |
|  | image\_match\_vectors = rx\_featurize(data=image\_match\_df, ml\_transforms=[ |
|  | load\_image(cols=dict(Features="image")), |
|  | resize\_image(cols="Features", width=227, height=227), |
|  | extract\_pixels(cols="Features"), |
|  | featurize\_image(cols="Features", dnn\_model="Alexnet")]) |
|  |  |
|  | print(image\_match\_vectors.head()) |
|  |  |
|  | ########################### |
|  | # Thirdly, compare the new image with the images in the |
|  | # catalogue to find the best match. |
|  | # We have 2 sets of feature vectors: |
|  | # |
|  | # - ``image\_vectors`` contains the feature vectors for the catalog images; |
|  | # - ``image\_match\_vectors`` contains the feature vector of the new image to be compared. |
|  | # |
|  | # The best match is defined (for our purposes) as the image pair |
|  | # with the least Euclidean distance between their image feature |
|  | # vectors where one of the feature vectors is for the new image. |
|  | # We implement these calculations using |
|  | # `cdist <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html#scipy.spatial.distance.cdist>`\_. |
|  |  |
|  | matimg = image\_vector.drop("image", axis=1).as\_matrix() |
|  | matmat = image\_match\_vectors.drop("image", axis=1).as\_matrix() |
|  |  |
|  | from scipy.spatial.distance import cdist |
|  | distance = cdist(matimg, matmat) |
|  | print(distance) |
|  |  |
|  | ####################### |
|  | # It contains 4 values corresponding to the Euclidian |
|  | # distance between the new image and the first four images |
|  | # we used as reference. |
|  | # |
|  | # .. note:: The actual values can change slightly depending on the machine |
|  | # used to run the code, but the order relations between the distance |
|  | # values should be invarient. |
|  | # |
|  | # And the winner is... |
|  |  |
|  | arg = distance.argmin() |
|  | print(arg) |
|  |  |
|  | fig, ax = plt.subplots(1, 1) |
|  | ax.imshow(Image.open(images[arg])) |

Method 2

Camera and Capture Frame using AForge library

Method 3

# Percentage difference between images

https://rosettacode.org/wiki/Percentage\_difference\_between\_images#Fortran