Image Similarity Detection in Action with Tensorflow 2.0

tds towardsdatascience.com/image-similarity-detection-in-action-with-tensorflow-2-0-b8d9a78b2509

29 de dezembro de 2019

Read More

Ready to use pipeline for your web application

In this post, I will show you how I implemented the ' **Image Similarity Detection**' task in my '**Fashion Price Comparison'** web application. I will use image similarity to suggest users visually similar products based on what they searched.

The full source code of the implementation is available in my GitHub repository.

Throughout the post, there will be dedicated sections for each one of the following subjects;

- How to use **Tensorflow 2.0** and **Tensorflow Hub** to generate **'image feature vectors'** of the product images.
- How to use **Spotify/annoy** library and **image feature vectors** to calculate the image similarity scores.
- Storing similarity scores and related product identification numbers in a **JSON file** to enable visual search in our web application.

What Is 'Image Similarity Detection' and Why It Is Important?

Image similarity detection is used to quantify the degree of visual and semantic similarity of the images.

Duplicate product detection, image clustering, visual search, and recommendation tasks are performed with this technology in modern applications.

"The future of search will be about pictures rather than keywords." — ***Ben Silbermann, Pinterest CEO***

"An advantage of visual search is that it relies entirely on item appearance. There is no need for other data such as bar codes, QR codes, product names, or other product metadata." — ***Brent Rabowsky,Amazon Web Services***

"Customers are increasingly using social media platforms, such as Instagram and Pinterest, as a source of inspiration so the visual search has the potential to transform how we shop for the home." — ***Mark Steel, Digital Director, Argos***

How to Use Tensorflow 2.0 and Tensorflow Hub to Generate 'Image Feature Vectors'

Tensorflow 2.0 and Tensorflow Hub

Tensorflow is an end-to-end open-source platform for machine learning developed by Google. It has tools, libraries and community resources that let developers easily build and deploy machine learning applications.

TensorFlow Hub provides many reusable machine learning models. It makes transfer learning very easy as it provides pre-trained models for different problem domains and different tasks such as image classification, image segmentation, pose detection, text embeddings, text classification, video generation, etc.

For further information about the transfer learning, you can check my previous article.

Machine Learning in the Browser: Train and Serve a Mobilenet Model for Custom Image Classification

Training Mobilenet Based Custom Image Classification Model on the Browser with Tensorflow.js and Angular

towardsdatascience.com

What is an image feature vector?

An **image feature vector** is a list of numbers that represents a whole **image,** typically used for image similarity calculations or image classification tasks.

In general, low-level image features are minor details of the image, such as lines, edges, corners or dots. High-level features are built on top of low-level features to detect objects and larger shapes in the image.

We can extract both types of features using convolutional neural networks: the first couple of convolutional layers will learn filters for finding low-level features while the later layers will learn to recognize common shapes and objects.

In our case, we will extract **high-level features of product images** using a pre-trained convolutional neural network which is **mobilenet_v2_140_224** stored in Tensorflow Hub.

MobilenetV2 is a simple neural network architecture suitable for mobile and resource-constrained applications. Follow this link to the original paper for further information on MobilenetV2.

Before start coding, it is required to install the Tensorflow 2.0, Tensorflow Hub and Spotify/Annoy libraries on our local computer.

```
$ virtualenv --system-site-packages -p python3 ./TFvenv
$ source ./TFvenv/bin/activate$ pip install tensorflow
$ pip install tensorflow-hub
$ pip install annoy
```

Let's Generate Image Feature Vectors: get_image_feature_vectors.py

The main purpose of this script is to generate image feature vectors by reading image files located in a local folder.

It has two functions: **load_img()** and **get_image_feature_vectors()**.

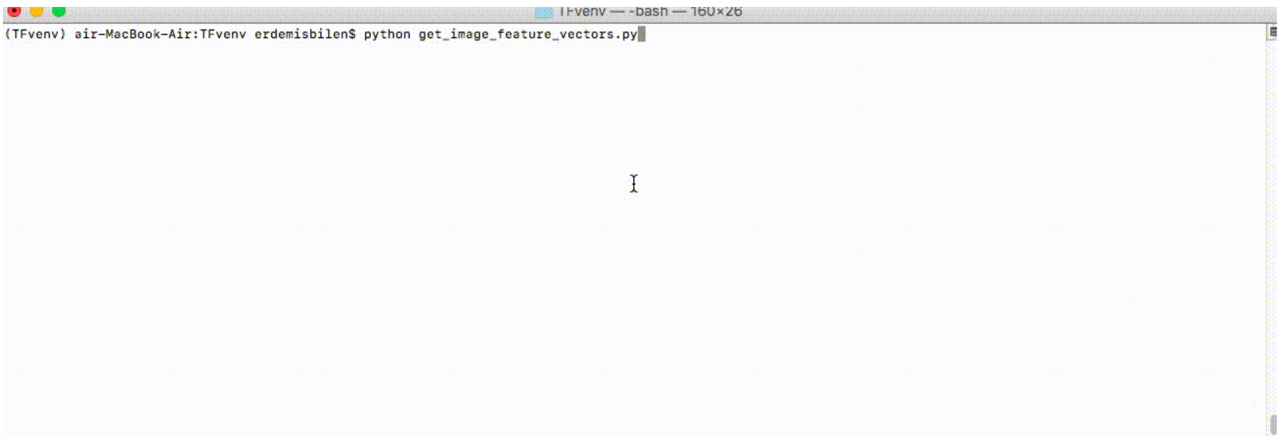
load_img(path) gets file names which are provided as an argument of the function. Then loads and pre-process the images so that we can use them in our MobilenetV2 CNN model.

Pre-processing steps are as follows;

- Decoding the image to W x H x 3 shape tensor with the data type of integer.
- Resizing the image to 224 x 224 x 3 shape tensor as the version of the MobilenetV2 model we use expects that specific image size.
- Converting the data type of tensor to **float** and adding a new axis to make tensor shape 1 x 224 x 224 x 3. This is the exact input shape expected by the model.

get_image_feature_vectors() function is where I extract the image feature vectors. You can see below, step by step definition of what this function does;

- Loads the MobilenetV2 model using Tensorflow Hub
- Loops through all images in a local folder and passing them to **load_img(path)** function
- Infers the image feature vectors
- Saves each one of the feature vectors to a separate file for later use



get_image_feature_vectors.py in action

```
#
get_image_feature_vectors.py####################################
# Imports and function definitions
##################################################
# For running inference on the TF-Hub module with Tensorflow
import tensorflow as tf
import tensorflow_hub as hub# For saving 'feature vectors' into a txt file
import numpy as np# Glob for reading file names in a folder
import glob
import os.path
##########################################################
# This function:
# Loads the JPEG image at the given path
# Decodes the JPEG image to a uint8 W X H X 3 tensor
# Resizes the image to 224 x 224 x 3 tensor
# Returns the pre processed image as 224 x 224 x 3 tensor
##################################################
def load_img(path):# Reads the image file and returns data type of string
 img = tf.io.read_file(path)# Decodes the image to W x H x 3 shape tensor with type of
uint8
 img = tf.io.decode_jpeg(img, channels=3)# Resizes the image to 224 x 224 x 3 shape
tensor
 img = tf.image.resize_with_pad(img, 224, 224)

# Converts the data type of uint8 to float32 by adding a new axis
 # img becomes 1 x 224 x 224 x 3 tensor with data type of float32
 # This is required for the mobilenet model we are using
 img = tf.image.convert_image_dtype(img,tf.float32)[tf.newaxis, ...]

  return img

##################################################
# This function:
# Loads the mobilenet model in TF.HUB
# Makes an inference for all images stored in a local folder
# Saves each of the feature vectors in a file
##################################################
def get_image_feature_vectors():

  # Definition of module with using tfhub.dev
 module_handle = "https://tfhub.dev/google/imagenet/
                 mobilenet_v2_140_224/feature_vector/4"
 # Loads the module
 module = hub.load(module_handle)

# Loops through all images in a local folder
 for filename in glob.glob('/Users/erdemisbilen/Angular/
         fashionWebScraping/images_scraped/full/*.jpg'):

   print(filename)# Loads and pre-process the image
  img = load_img(filename)# Calculate the image feature vector of the img
  features = module(img)

# Remove single-dimensional entries from the 'features' array
  feature_set = np.squeeze(features)
```

```
# Saves the image feature vectors into a file for later use
outfile_name = os.path.basename(filename) + ".npz"

 out_path = os.path.join('/Users/erdemisbilen/Angular/
        fashionWebScraping/images_scraped/feature-vectors/',
        outfile_name)# Saves the 'feature_set' to a text file
np.savetxt(out_path, feature_set, delimiter=',')get_image_feature_vectors()
```

How to Use Spotify/Annoy Library to Calculate the Similarity Scores

What is Spotify/Annoy Library?

Annoy(Approximate **N**earest **N**eighbor **O**h **Y**eah)**,** is an open-sourced library for approximate nearest neighbor implementation.

I will use it to find the image feature vectors in a given set that is closest (or most similar) to a given feature vector.

> There are just two main parameters needed to tune Annoy: the number of trees `n_trees` and the number of nodes to inspect during searching `search_k`.
>
> `n_trees` is provided during build time and affects the build time and the index size. A larger value will give more accurate results, but larger indexes.
>
> `search_k` is provided in runtime and affects the search performance. A larger value will give more accurate results, but will take longer time to return.
>
> **from Spotify/Annoy**

Let's Calculate Similarity Scores: cluster_image_feature_vectors.py

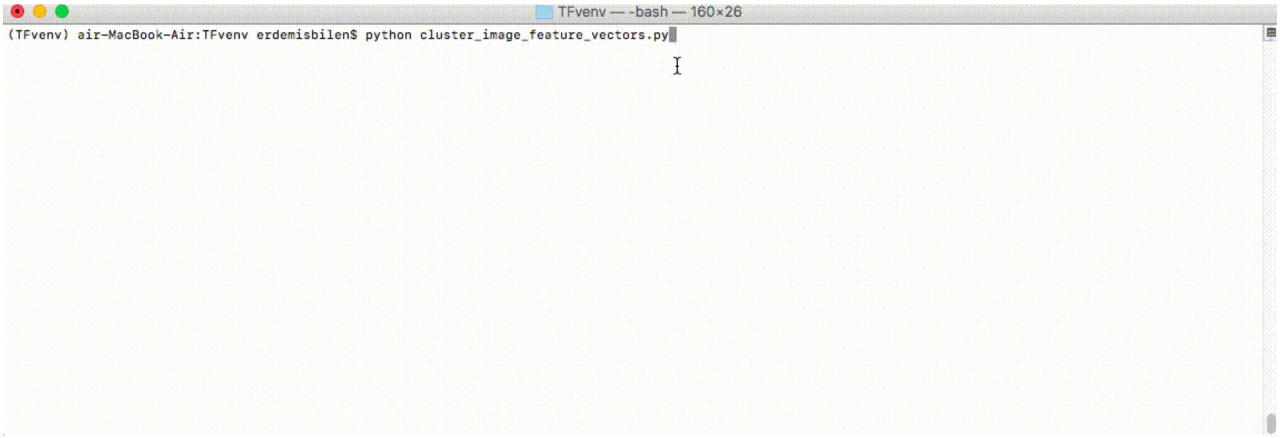
The main purpose of this script is to calculate image similarity scores using image feature vectors we have just generated in the previous chapter.

It has two functions: **match_id(filename)** and **cluster()**.

cluster() function does the image similarity calculation with the following process flow:

- Builds an annoy index by appending all image feature vectors stored in the local folder
- Calculates the nearest neighbors and similarity scores
- Saves and stores the information in a JSON file for later use.

match_id(filename) is a helper function as I need to match images with the product id's to enable visual product search in my web application. There is a JSON file that contains all the product id information matched with the product image names. This function retrieves the product id information for a given image file name using that JSON file.



cluster_image_feature_vectors.py in action

```
#
cluster_image_feature_vectors.py###################################
# Imports and function definitions
#################################################
# Numpy for loading image feature vectors from file
import numpy as np# Time for measuring the process time
import time# Glob for reading file names in a folder
import glob
import os.path# json for storing data in json file
import json# Annoy and Scipy for similarity calculation
from annoy import AnnoyIndex
from scipy import spatial
#########################################################
# This function reads from 'image_data.json' file
# Looks for a specific 'filename' value
# Returns the product id when product image names are matched
# So it is used to find product id based on the product image name
#################################################
def match_id(filename):
 with open('/Users/erdemisbilen/Angular/fashionWebScraping
 /jsonFiles/image_data.json') as json_file:for file in json_file:
   seen = json.loads(file)

for line in seen:

         if filename==line['imageName']:
      print(line)
      return line['productId']
      break
#################################################

#################################################
# This function:
# Reads all image feature vectores stored in /feature-vectors/*.npz
```

```
# Adds them all in Annoy Index
# Builds ANNOY index
# Calculates the nearest neighbors and image similarity metrics
# Stores image similarity scores with productID in a json file
#################################################
def cluster():
 start_time = time.time()

  print("---------------------------------")
 print ("Step.1 - ANNOY index generation - Started at %s"
 %time.ctime())
 print("---------------------------------")# Defining data structures as empty dict
 file_index_to_file_name = {}
 file_index_to_file_vector = {}
 file_index_to_product_id = {}# Configuring annoy parameters
 dims = 1792
 n_nearest_neighbors = 20
 trees = 10000

# Reads all file names which stores feature vectors
 allfiles = glob.glob('/Users/erdemisbilen/Angular
 /fashionWebScraping/images_scraped/feature-vectors/*.npz')

  t = AnnoyIndex(dims, metric='angular')for file_index, i in enumerate(allfiles):# Reads
feature vectors and assigns them into the file_vector
  file_vector = np.loadtxt(i)# Assigns file_name, feature_vectors and corresponding
product_id
  file_name = os.path.basename(i).split('.')[0]
  file_index_to_file_name[file_index] = file_name
  file_index_to_file_vector[file_index] = file_vector
  file_index_to_product_id[file_index] = match_id(file_name)# Adds image feature vectors
into annoy index
  t.add_item(file_index, file_vector)print("---------------------------------")
  print("Annoy index     : %s" %file_index)
  print("Image file name : %s" %file_name)
  print("Product id      : %s"
  %file_index_to_product_id[file_index])
  print("--- %.2f minutes passed ---------" % ((time.time() -
  start_time)/60))# Builds annoy index
 t.build(trees)print ("Step.1 - ANNOY index generation - Finished")
 print ("Step.2 - Similarity score calculation - Started ")named_nearest_neighbors = []

# Loops through all indexed items
 for i in file_index_to_file_name.keys():

   # Assigns master file_name, image feature vectors
  # and product id values
  master_file_name = file_index_to_file_name[i]
  master_vector = file_index_to_file_vector[i]
  master_product_id = file_index_to_product_id[i]# Calculates the nearest neighbors of
the master item
  nearest_neighbors = t.get_nns_by_item(i, n_nearest_neighbors)

# Loops through the nearest neighbors of the master item
  for j in nearest_neighbors:
```

```
    print(j)# Assigns file_name, image feature vectors and
   # product id values of the similar item
   neighbor_file_name = file_index_to_file_name[j]
   neighbor_file_vector = file_index_to_file_vector[j]
   neighbor_product_id = file_index_to_product_id[j]# Calculates the similarity score of the similar item
   similarity = 1 - spatial.distance.cosine(master_vector,
   neighbor_file_vector)rounded_similarity = int((similarity * 10000)) / 10000.0# Appends master product id with the similarity score
   # and the product id of the similar items
   named_nearest_neighbors.append({
     'similarity': rounded_similarity,
     'master_pi': master_product_id,
     'similar_pi': neighbor_product_id})print("---------------------------------")
 print("Similarity index       : %s" %i)
 print("Master Image file name : %s" %file_index_to_file_name[i])
 print("Nearest Neighbors.     : %s" %nearest_neighbors)
 print("--- %.2f minutes passed ---------" % ((time.time() -
 start_time)/60))print ("Step.2 - Similarity score calculation - Finished ")# Writes the 'named_nearest_neighbors' to a json file
 with open('nearest_neighbors.json', 'w') as out:
 json.dump(named_nearest_neighbors, out)print ("Step.3 - Data stored in 'nearest_neighbors.json' file ")
 print("--- Prosess completed in %.2f minutes ---------" %
 ((time.time() - start_time)/60))cluster()
```

As you can see, I saved the highest 20 similarity scores for each product image in a JSON file with matching product id information. This is because I don't what to do the similarity calculation on the client-side to eliminate the effort required.

With the similarity scores stored in the JSON file, I can easily populate Elasticsearch clusters, or populate a database to enable near real-time visual search experience on the browser on my price comparison web application.

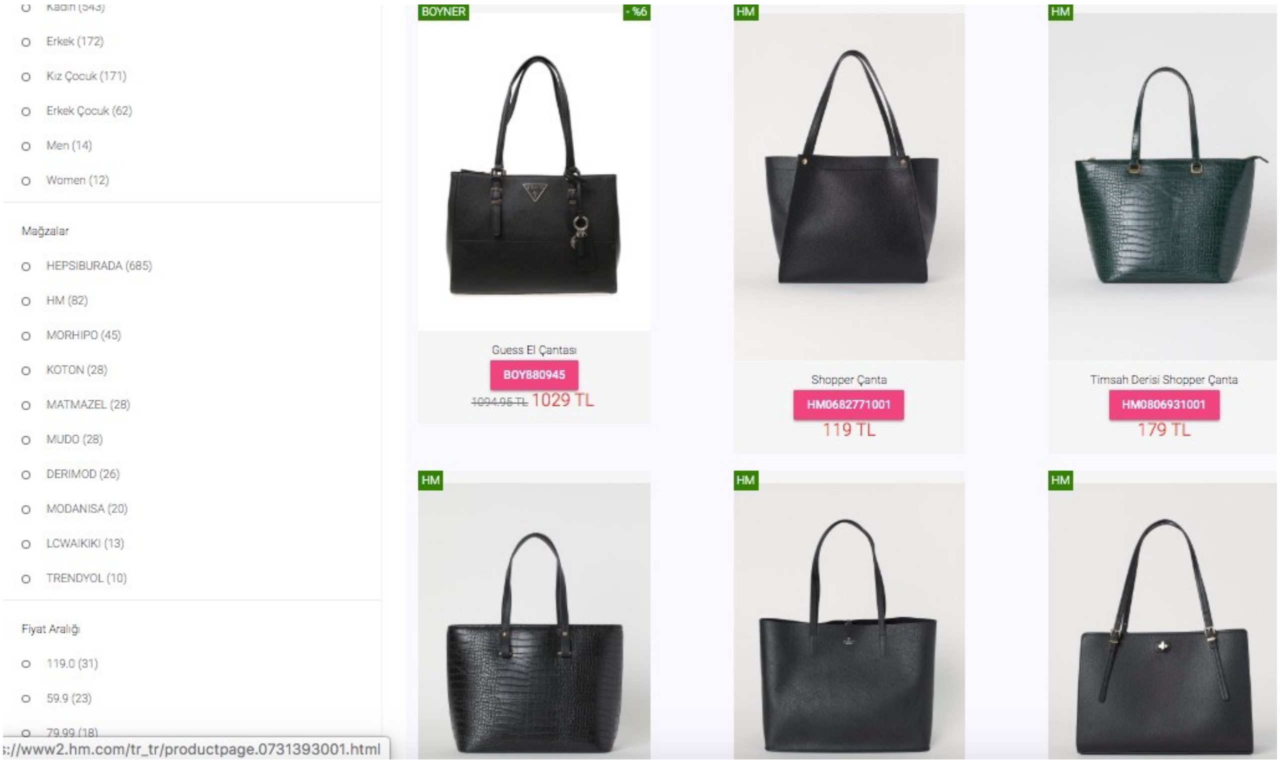
To develop such an application, web scraping plays an important role to develop and maintain product datasets daily basis. If you are interested in this subject, check my related article below.

Web Scraping of 10 Online Shops in 30 Minutes with Python and Scrapy

Get the source data you need to kick-start your App project

towardsdatascience.com

Conclusion



Fashion Search Web Application by Erdem Isbilen — Visual Search Results

As you can see above, MobileNetV2 and Annoy together do a pretty good job of finding visual similar products.

One disadvantage of this implementation is that it only works at the whole image level. It does not provide good results if the backgrounds of the images are different, even if the objects are similar.

This application can be further improved to achieve an object-level similarity search like the one on Pinterest or Houzz.

By Towards Data Science

More From Medium