Republic of Tunisia
Ministry of Higher Education,
Scientific Research and Information
and Communication Technologies

SUP'COM
Higher School of Communication of Tunis

CBIR Project

by

DHAOUI Alaeddine
Chiboub Mohamed

Supervised by

Mr. TEBOURBI Riadh

Contents

List of Figures

Introduction

Content-based image retrieval websites (CBIR) are one of the most important technologies on the web. More and more users are able to get to the website that they need using websites like Google, Yahoo and Bing. The ability to store, index and search millions and millions of images, documents and rich media in a matter of seconds or even milliseconds has become essential in bring the best user experience. The aim of this project is then to build a CBIR search engine that will allow us to search through a database containing one million image. We will explore the different steps that we did in order to run the system and obtain good results.

Chapter 1

Project Process

1.1 Content-Based Image Retrieval

1.1.1 Definition

”Content-based” means that the search analyzes the contents of the image rather than the metadata such as keywords, tags, or descriptions associated with the image. The term ”content” in this context might refer to colors, shapes, textures, or any other information that can be derived from the image itself. CBIR is desirable because searches that rely purely on metadata are dependent on annotation quality and completeness.

1.1.2 Process

Any CBIR search-engine goes through different phases in order to predict similar images to a search image. The process is as follows:

- Downloading Image Database

  We were able to get a database of over 1 million images from Open Images Dataset V6+.

- Creating Feature Database

  In our cases, to have the best results, we decide to Use VGG-16 Deep Learning Model in order to extract the features. This will give us a very high accuracy.

- Index the Feature Database

  The created index in this step must be organized in a way that makes searching for nearest feature vectors the fastest possible. Many ways can be applied but one of new ways is the Annoy Indexing Model, used by Spotify in their music prediction model.

- Image Request Feature Extraction

  Once we read the request image, we should repeat the same process by extracting the VGG features using the same model used in the feature database creation step.

- Similarity Measure

  In this step, we feed the request image feature vector to the index in order to calculate its nearest neighbors.

- Similarity Measure

  In the final step, we need to map the nearest vectors to the real images, that way we can return them to the user.

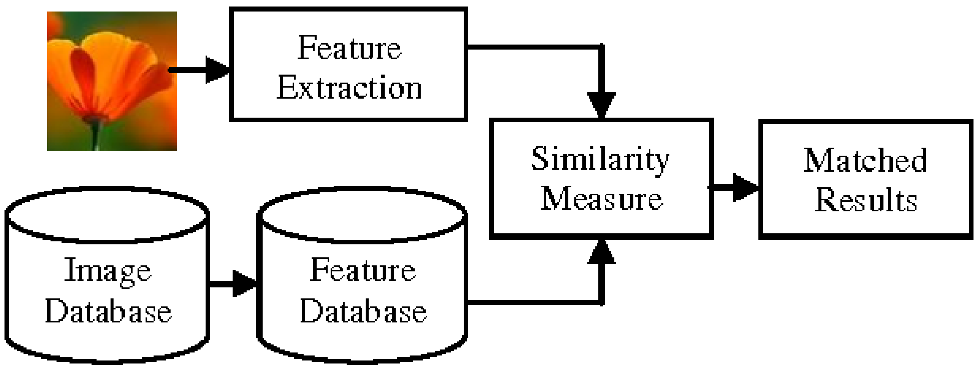


Figure 1. The Process in a CBIR project

1.2 VGG-16

1.2.1 Definition

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. Therefore we will be using the 4096 features that are generated by this model, which are highly distinctive and will make our CBIR system have the best performance possible.

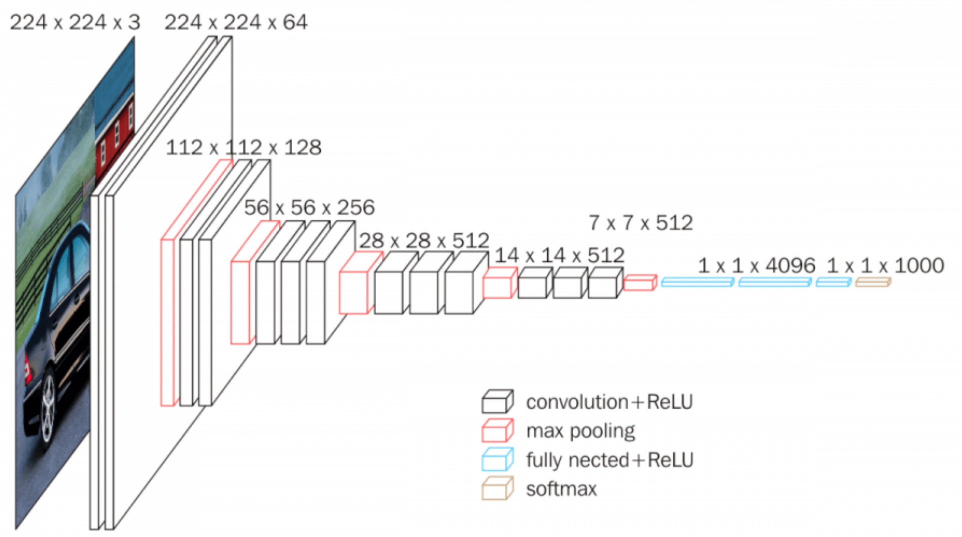
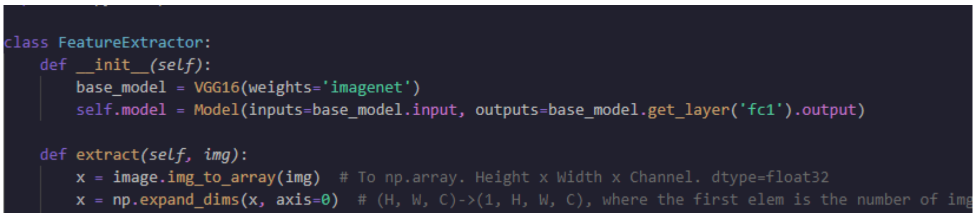


Figure 2. The Layers othe VGG-16 Deep Neural Network

1.2.2 Feature Extraction

Since we do not need the classification capabilities of the VGG-16 model, we will only need the feature that are being generated within the model. This is done by removing the last layer of the model. This is done by defining a FeatureExtractor class, which used a VGG16 Model but provides an output layer as the 'fct1' layer.



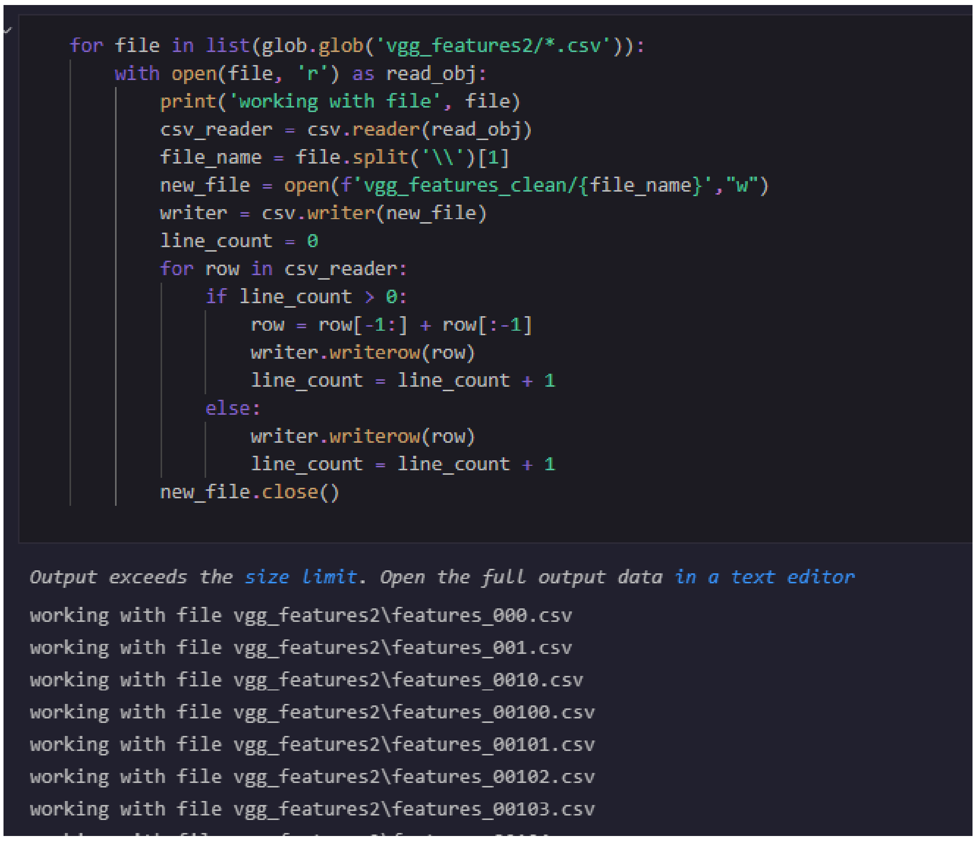
```
class FeatureExtractor:
    def __init__(self):
        base_model = VGG16(weights='imagenet')
        self.model = Model(inputs=base_model.input, outputs=base_model.get_layer('fc1').output)

    def extract(self, img):
        x = image.img_to_array(img)  # To np.array. Height x Width x Channel. dtype=float32
        x = np.expand_dims(x, axis=0)  # (H, W, C)->(1, H, W, C), where the first elem is the number of img
```

Figure 3. Dropping the last classification layer of the VGG-16 Model in order to get the VGG Features

1.2.3 Fixing Feature Dataframe

During our feature extraction, we made an error: we didnt save any reference to the real images and the feature vectors. There was no way of mapping any feature vector to the image that we took it from. That's why we had to implement a python script that was responsible to add an id column in the dataframe and re-saving the generated CSV file.



```
for file in list(glob.glob('vgg_features2/*.csv')):
    with open(file, 'r') as read_obj:
        print('working with file', file)
        csv_reader = csv.reader(read_obj)
        file_name = file.split('\\')[1]
        new_file = open(f'vgg_features_clean/{file_name}',"w")
        writer = csv.writer(new_file)
        line_count = 0
        for row in csv_reader:
            if line_count > 0:
                row = row[-1:] + row[:-1]
                writer.writerow(row)
                line_count = line_count + 1
            else:
                writer.writerow(row)
                line_count = line_count + 1
        new_file.close()
```

```
Output exceeds the size limit. Open the full output data in a text editor
working with file vgg_features2\features_000.csv
working with file vgg_features2\features_001.csv
working with file vgg_features2\features_0010.csv
working with file vgg_features2\features_00100.csv
working with file vgg_features2\features_00101.csv
working with file vgg_features2\features_00102.csv
working with file vgg_features2\features_00103.csv
```

Figure 4. Adding the image ID to the feature data frame

1.2.4 Potential Improvements

We can always improve the feature extraction by removing irrelevant features. Since 4096 value per feature vector is considered a lot, we can reduce this number to less than 1000 values per faeture vector. This will make the indexing much faster. One of the techniques that we should use is called PCA which stands for Principal Components Analysis. This techniques revolves in leaving only the features that have the most impact in the dataset.

1.3 Annoy

1.3.1 Definition

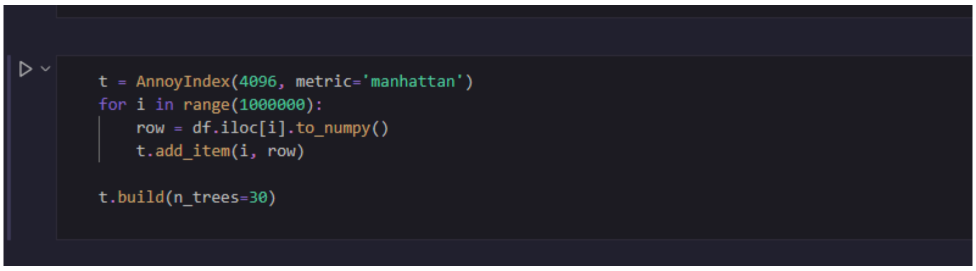
Annoy (Approximate Nearest Neighbors Oh Yeah) is a C++ library with Python bindings to search for points in space that are close to a given query point. It also creates large read-only file-based data structures that are mmapped into memory so that many processes may share the same data.
There are some other libraries to do nearest neighbor search. Annoy is almost as fast as the fastest libraries, but there is actually another feature that really sets Annoy apart: it has the ability to use static files as indexes. In particular, this means you can share index across processes. Annoy also decouples creating indexes from loading them, so we can pass around indexes as files and map them into memory quickly. Another nice thing of Annoy is that it tries to minimize memory footprint so the indexes are quite small.
The annoy index works by using random projections and by building up a tree. At every intermediate node in the tree, a random hyperplane is chosen, which divides the space into two subspaces. This hyperplane is chosen by sampling two points from the subset and taking the hyperplane equidistant from them. This is done k times so that we get a forest of trees.

1.3.2 Distances

Annoy supports multiple types of distances such as: Euclidean distance, Manhattan distance, cosine distance, Hamming distance, or Dot (Inner) Product distance.
By testing the different distances, we found out that the Manhattan distance gives the best results. Therefore our indexer was built using this type of distance.

1.3.3 Saving the index model and performance

Using our 1 million images features dataframe (which contains VGG-16 features), and our manhattan distance, we defined the Annoy Indexer and created the index.

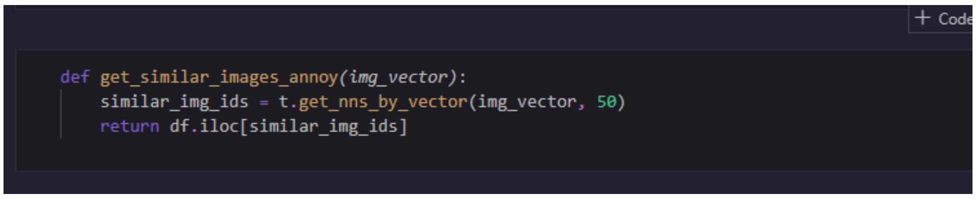


```python
t = AnnoyIndex(4096, metric='manhattan')
for i in range(1000000):
    row = df.iloc[i].to_numpy()
    t.add_item(i, row)

t.build(n_trees=30)
```

Figure 5. Creating the Annoy Index

We then defined a similarity function which takes the request feature vector and find the K nearest vectors from the Annoy index.



```python
def get_similar_images_annoy(img_vector):
    similar_img_ids = t.get_nns_by_vector(img_vector, 50)
    return df.iloc[similar_img_ids]
```

Figure 6. Calculate similar vectors function

We can then save the index into our disk and load it when we need it.

For the performance, the indexer is slow at the start, averaging 23 seconds to find the results, but then it drops to between 0.2 and 1.5 seconds for future searches, which is what we are looking for.

Chapter 2

Web Development

2.1 Frontend

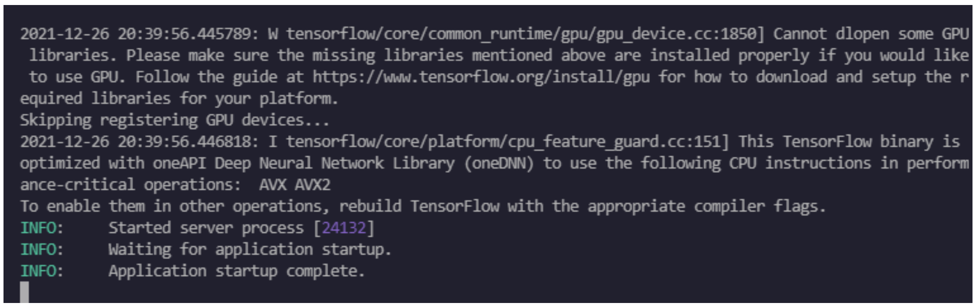
2.1.1 Streamlit

Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science.
We used is to build our frontend part of the CBIR search-engine. This gives us great flexibility to control most of the parts of our website. The frontend consists of an input, from which we select an image from the system. Once selected, we can press the 'Search' button which will send a request to the backend in order to get the most relevant images. Once the data is returned from the backend in the form of image URLs, streamlit will then add the necessary <img> tags and show the images.

2.1.2 Execution

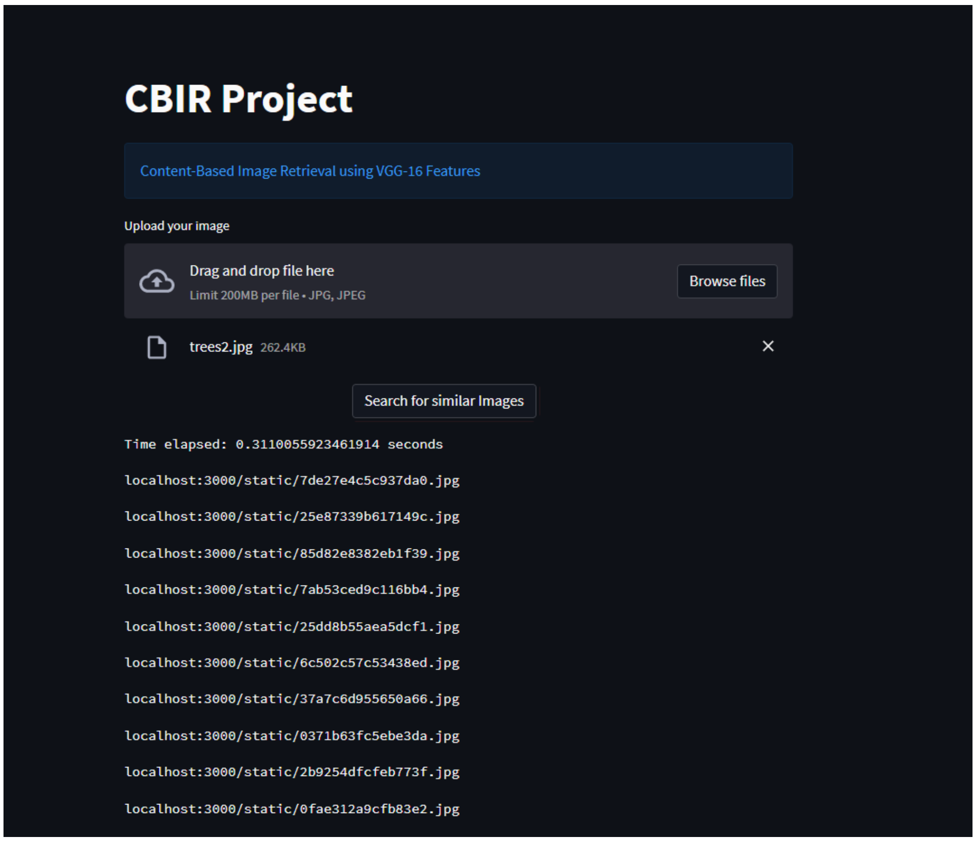
To run the frontend part, we just execute the streamlit run front_end.py command.

Since our images are stored in the computer that is being kept at the university, we couldn't show the images since the computer was closed. Therefore for demonstration purposes, we showed the URLs of the images directly.



```
2021-12-26 20:39:56.445789: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU
 libraries. Please make sure the missing libraries mentioned above are installed properly if you would like
 to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the r
equired libraries for your platform.
Skipping registering GPU devices...
2021-12-26 20:39:56.446818: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is
optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in perform
ance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
INFO:     Started server process [24132]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Figure 7. Running the streamlit command



CBIR Project

Content-Based Image Retrieval using VGG-16 Features

Upload your image

Drag and drop file here
Limit 200MB per file • JPG, JPEG

trees2.jpg 262.4KB

```
Time elapsed: 0.3110055923461914 seconds

localhost:3000/static/7de27e4c5c937da0.jpg

localhost:3000/static/25e87339b617149c.jpg

localhost:3000/static/85d82e8382eb1f39.jpg

localhost:3000/static/7ab53ced9c116bb4.jpg

localhost:3000/static/25dd8b55aea5dcf1.jpg

localhost:3000/static/6c502c57c53438ed.jpg

localhost:3000/static/37a7c6d955650a66.jpg

localhost:3000/static/0371b63fc5ebe3da.jpg

localhost:3000/static/2b9254dfcfeb773f.jpg

localhost:3000/static/0fae312a9cfb83e2.jpg
```

Figure 8. Frontend results

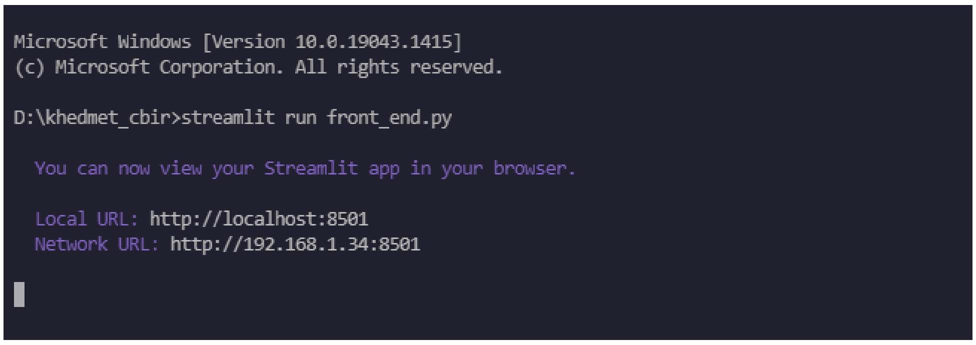
2.2 Backend

2.2.1 FastAPI

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints. We used it to create the link between the frontend and the annoy indexer. The fastapi defines a single API, which is used by the frontend the get the URLs of the images that are similar to the request image. The API loads the Annoy Index model from the start and used an global variable which represent the base URL of the images server. Our images are all located on the faculty's computer, therefore, we used the public IP address of that computer to point to the real images in order to show them.

2.2.2 Execution

To run the frontend part, we just execute the uvicorn main:back_end –reload command.



```
Microsoft Windows [Version 10.0.19043.1415]
(c) Microsoft Corporation. All rights reserved.

D:\khedmet_cbir>streamlit run front_end.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.34:8501

```

Figure 9. Running the uvicorn command

Conclusion

In conclusion, we were able to create a CBIR system using 1 million images from open google images. The prediction used VGG-16 features and Annoy Indexing to store them. We used the manhattan distance to measure the closest vectors and map them to the real images. We deployed the model in a FastAPI backend and connected it to a Streamlit Frontend where we were able to select an image and show its closest image by results This project allowed us to work as a team and to get more knowledge regarding deep learning models, indexing and web development.

Webography

[1] Annoy. url: https://github.com/spotify/annoy.

[2] Step by step VGG16 implementation in Keras for beginners. url: https : / / towardsdatascience . com / step - by - step - vgg16 - implementation - in - keras - for - beginners-a833c686ae6c.

[3] Open Images Dataset. url: https://storage.googleapis.com/openimages/web/index.html.

[4] FastAPI. url: https://fastapi.tiangolo.com/.

[5] Streamlit. url: https://streamlit.io/.