Appendix A

Project Structure

```
GPS-Image-Verification
├── Datasets
│   ├── Caltech
│   │   └── "labels.txt"
│   └── WikiCommons
│       └── "labels.txt"
├── Test
│   ├── images
│   │   ├── "image1Q.jpg"
│   │   ├── "image1R.jpg"
│   │   ├── "image2Q.jpg"
│   │   ├── "image2R.jpg"
│   │   └── "README.pdf"
│   ├── plot
│   │   └── "values.txt"
│   └── threshold
│       ├── "negbestpm.txt"
│       ├── "negbesttm.txt"
│       ├── "posbestpm.txt"
│       ├── "posbesttm.txt"
│       └── "README.pdf"
├── "README.pdf"
├── "requirements.txt"
├── "thresholdTool.py"
└── "verificationTool.py"
```

Appendix B

Test Samples

B.1 Template Matching

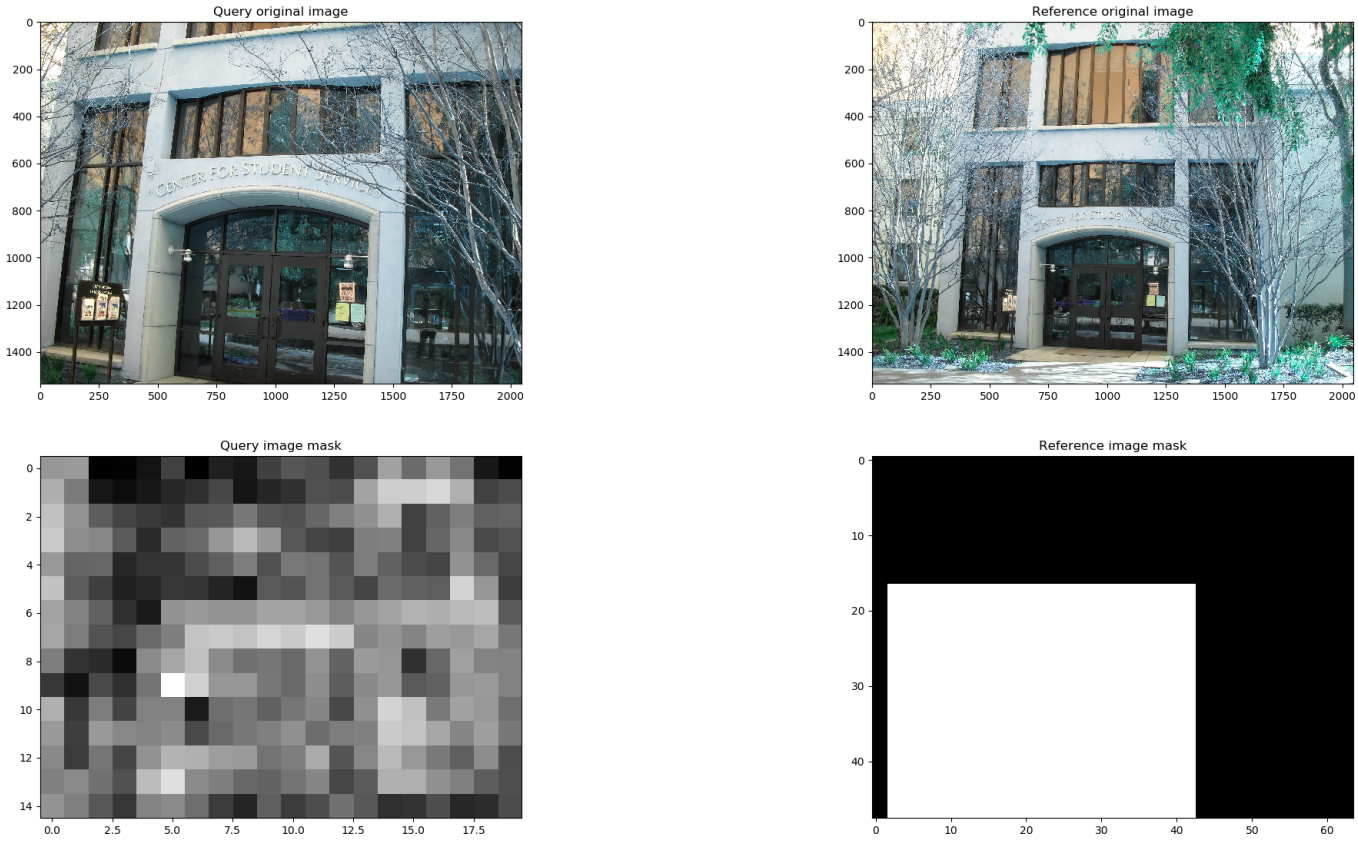

Figure B.1: The results of performing Template Matching with Correlation Coefficient. The recorded similarity was 0.778.

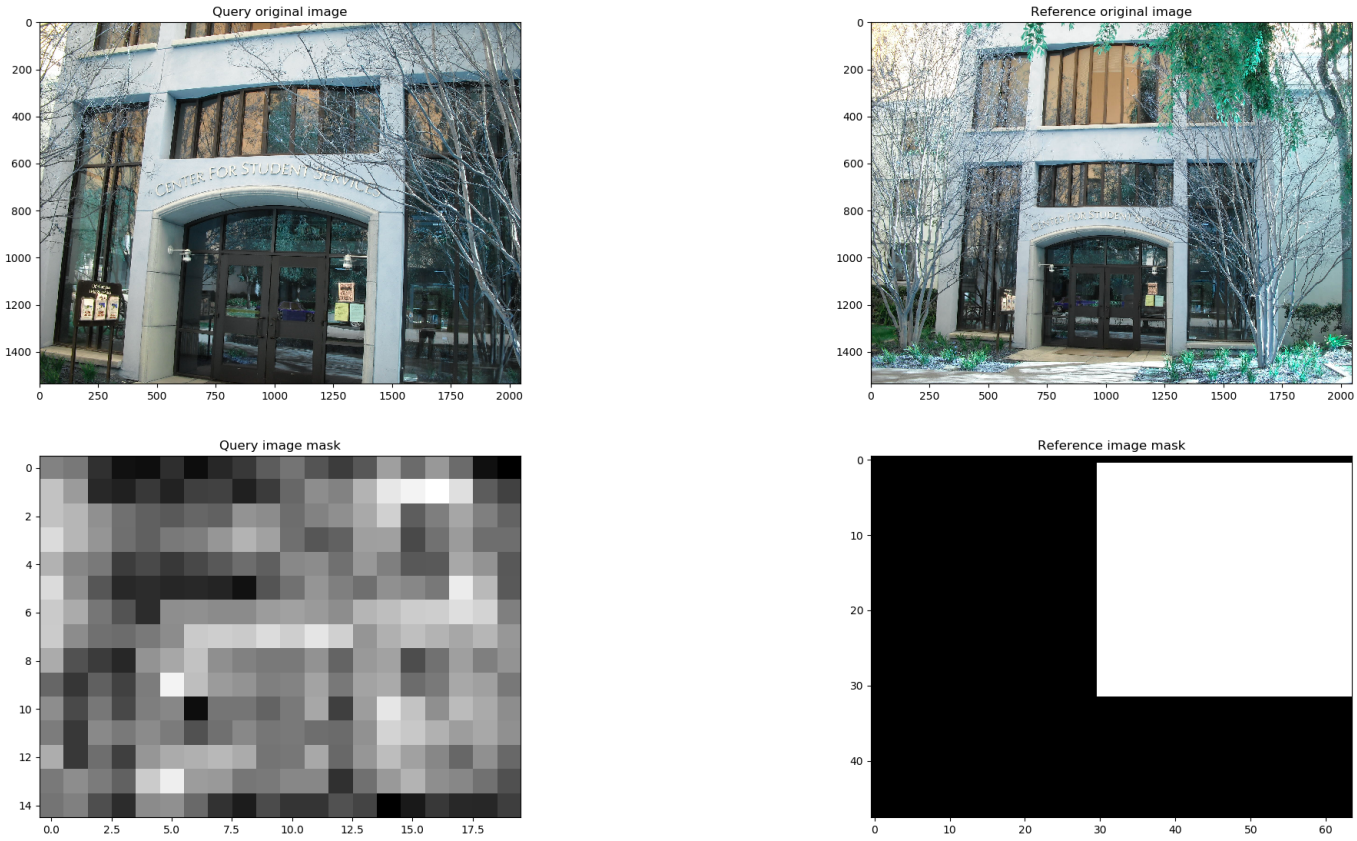

Figure B.2: The results of performing Template Matching with Normalized Cross-Correlation. The recorded similarity was 0.811.

B.2 Patch Matching

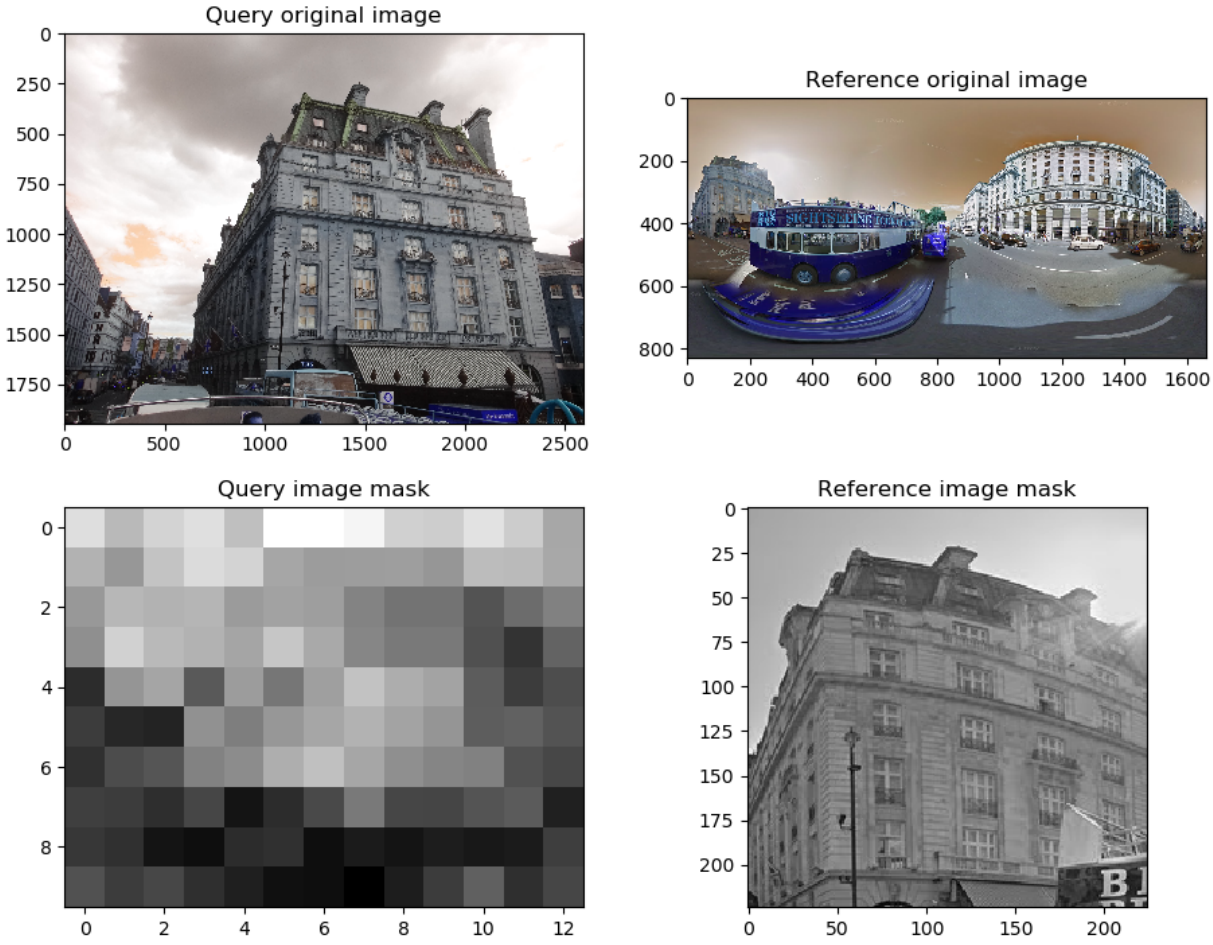

Figure B.3: The results of performing Patch Matching with Correlation Coefficient. The recorded similarity was 0.775.

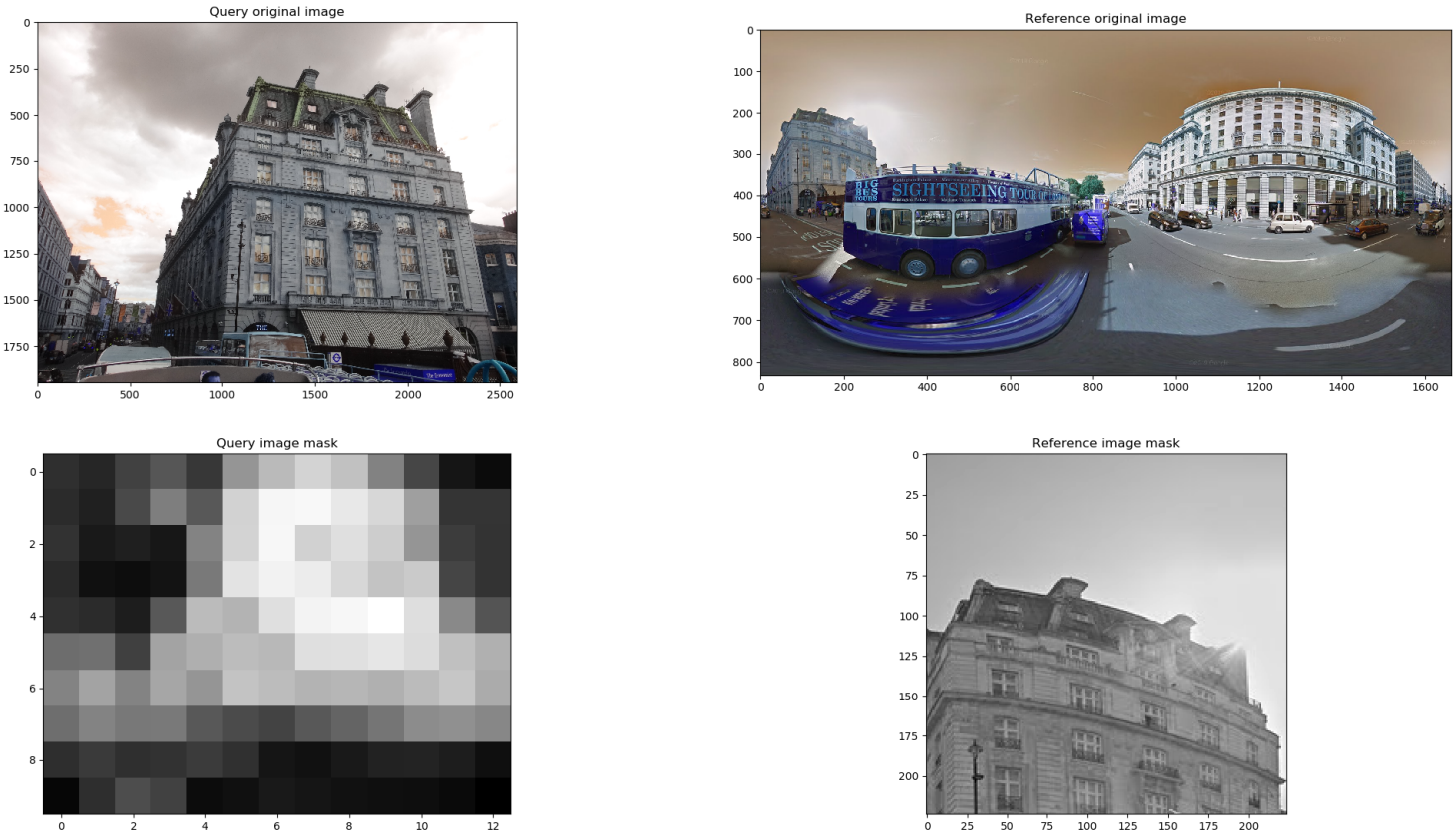

Figure B.4: The results of performing Patch Matching with Normalized Cross-Correlation. The recorded similarity was 0.759.

Appendix C

User Guide

C.1 Installation Guide

This section provides detailed installation instructions for the project described in this report. It includes two python files. The *verificationTool.py* one represents the proposed model and can be used to find the similarity between two images. *thresholdTool.py* is the script used to find the best threshold for the test runs shown in the report. Therefore, *Python 3* must be installed before running this project. *Python 3* can be downloaded from `https://www.python.org/downloads/`. The version used in this project is *3.6.8*. To install the project:

1. Python 3 virtual environment - creating a virtual environment can help preserve the system from installing and uninstalling different requirements, as well as it will provide a clean installation process. To create that environment, the following command can be executed in a terminal:

   - python3 -m venv verification-env

   The name of the environment is specified after the "venv" command. In this case it is "verification-env". The environment will create a directory at the current location that contains a copy of the python interpreter. After successfully creating the environment, it needs to be activated. This can be done with the following commands:

   - Windows - verification-env\Scripts\activate.bat
   - Unix or MacOS - source verification-env/bin/activate

After running the command, according to the system, the environment should show in the terminal. Now, the requirements for the project must be installed.

2. Installing requirements - the requirements can be installed with the help of pip, or more specifically pip3. If pip is not already installed with python 3, which it should be by default, the original guide can be followed. It can be found on `https://pip.pypa.io/en/stable/installing/`. The *pip* setup file must be downloaded and then executed with python. After it is made sure that *pip* is installed on the system, the following command can be run to install the requirements necessary in the directory of the project:

    - *pip install -r requirements.txt*

Besides, there are a lot of external libraries that have been used in the implementation of the model, which are necessary for it to operate as described. The installation script provided installs all prerequisites automatically. Nevertheless, a list of all libraries used is provided below:

- Keras - 2.3.1
- Tensorflow - 2.0.0
- NumPy - 1.16.0
- OpenCV - 3.4.2.16
- Sklearn - 0.22.1
- Argparse - 1.1
- Matplotlib - 3.1.0
- Time - built into python's interpreter (python 3.6.8)
- Warnings - built into python's interpreter (python 3.6.8)
- OS - built into python's interpreter (python 3.6.8)
- Gast - 0.3.3

Possible interference between the Keras and Tensorflow libraries can be experienced, where the former must be installed first. Additionally, after installing Tensorflow it is possible to get an error suggesting that the "gast" module is missing. A workaround solution to that is to remove that module and install it again. This is because newer versions of "gast" do not

work with specific versions of Tensorflow. This error occurs especially when a newer version of Tensorflow was already present on the machine. However, these problems should not occur if installing in a virtual environment, as suggested above.

C.2 User Instructions

After the successful installation of the requirements for the project, the user can navigate and use either the model itself or the threshold helper script through the terminal. Because the Argparse library has been used, this project can be entirely used by performing different commands in the terminal. Also, the command "--help" will show all the options available to the user. This command can be executed for both python files:

- python verificationTool.py --help
- python thresholdTool.py --help

Furthermore, a list of all commands for the *verificationTool* can be found below.

- Predicting on a pair of images - the command is (-p). An example usage would be (python verificationTool.py -p "path to query image" "path to reference image").
- Extract similarity values and save them to a file - three commands must be used in conjunction (-e; -ep; -ed). (-e) toggles the extraction mode ON, (-ep) provides a file path where the values will be saved, and (-ed) provides the location for the query and reference images as well as the labels text file. An example of using these commands would be (python verificationTool.py -e -ed "path to query images" "path to reference images" "path to labels" -ep "path to new text file").
- Plot data from a file - here the (-ep) command is used with the (--plot) command to plot data and visualise it. Additionally, the *Test/plot* folder provided in the directory of the project includes a *"values.txt"* file which can be used. It has the values for the Caltech Buildings dataset [38]. The command can look like - (python verificationTool.py --plot -ep "path to text file"). Or, if the provided file is being used - (python verificationTool.py --plot -ep ./Test/plot/values.txt).
- Test on a custom dataset - the command for that is (-test). Three paths must be provided in order to test on an image dataset. An example of that would be (python verification-Tool.py -test "path to query images" "path to reference images" "path to labels").

- Specifying the threshold - the command for that is (-thr). It can be used in conjunction with all other commands. For example, (python verificationTool.py -p ”path to query image” ”path to reference image” -thr ”some number”).

- Change matching method - to change the matching method used between Template Matching and Patch Matching the (-mm) command must be used. Similarly to the (-thr), this command can be used with other commands as well. An example of that would look like (python verificationTool.py -test ”path to query images” ”path to reference images” ”path to labels” -thr ”some number” -mm ”either tm for Template Matching or pm for Patch Matching”).

- Print likelihood masks - this is done by the (-pm) command. It is simply a toggle that enables drawing mode. This means that if there is a match, whether while testing or predicting, then the likelihood masks will be printed out. For instance, (python verificationTool.py -pm -p ”path to query image” ”path to reference image”).

- Enable the SURF comparison mode - the (-s) command disables the model and uses the SURF [4] method to either test or predict. This is done for the user to be able to verify the results in Chapter 4. An example usage of that command would be (python verificationTool.py -test ”path to query images” ”path to reference images” ”path to labels” -thr ”some number” -s).

- Change the CNN used for Feature Extraction - the (--model) command can be used in conjunction with other commands to change the default CNN model from ResNet50 [13] to any of the following - ResNet101, ResNet152 [13], VGG-19 [31] and Inception-ResNet [34]. The actual options are ”resnet50”, ”resnet101”, ”resnet152”, ”vgg19” and ”inception” respectively. For instance, the command can be used with either the predict or test commands - *python verificationTool.py -p ”path to query image” ”path to reference image” -mm pm -thr 0.65 --model vgg19*

- Change the similarity measure - the (--measure) command can be used to specify different similarity measures. The options for the command are - ”cc”(Correlation Coefficient [14]), ”ncc”(Normalized Cross-Correlation). It is recommended that the paper is being followed when picking threshold or any other parameter. Nonetheless, an example of using the command is as follows - *python verificationTool.py -p ”path to query image” ”path to reference image” -mm tm -thr 0.75 --model resnet152 --measure ncc*

Moreover, when providing single images for the (-p) command, the image extension must be specified, for instance, ".jpg". However, other image extensions are supported as well. On the other hand, when testing with the (-test) command, the full paths to the different folders must be provided. The query and reference images paths need to end with a closing slash "/". For the labels file, only the path to the actual ".txt" file must be provided. Additionally, all of the commands provided above are examples and do not represent how to run in order to replicate the actual evaluations recorded in Chapter 4.

When using the script attached that finds the best threshold, two files must be provided. These files need to be of ".txt" format. They represent the similarity values for both negative or positive samples respectively. Furthermore, by using this script, a plot of the data will be saved to an image. The plotted data image should look similar to the ones provided in Chapter 4. The negative file with similarity values must be passed over before the positive one. The commands that the script (thresholdTool.py) supports are:

- Finding the best threshold - the (-a) is the default command that will show the best accuracy and threshold for a provided set of files. The command used to do that is *python thresholdToold.py -a "path to negative sample values file" "path to positive sample values file"*. The exact order of passing the negative and positive text files must be followed. Additionally, an extra argument can be added that specifies the location of where the newly produce graph will be saved ("path to the new .jpg image"). If not specified, the figure will be saved to a *default.jpg* image.

- Finding the best threshold in reverse order - this command is very similar to the previous one, (-r). However, it is used to find the best threshold when lower values are better. This depends on the similarity measure used to generate these files. For instance, the SSD measure shows perfect alignment when the value is zero. The command is *python thresholdTool.py -a "path to negative sample values" "path to positive sample values" [optional new image path] -r*. The -r command reverses the order that the tool follows, resulting in picking the threshold accordingly. Therefore, if a positive sample has a lower value than the threshold it will be marked as a correct prediction, rather than incorrect, as it would be in general. This command has been implemented to support future expansion and modification of the project.

- Adjusting the intensity of the threshold - this command can be used to change the intensity with which the threshold is being found by the script. For example, if the intensity is

0.0001, then the script will loop through every value incrementally using that intensity. It would start from 0, then 0.0001, then 0.0002, etc. It must be noted that lower values can find better threshold, however the computation time will increase as well. Additionally, if the intensity is too low for values like SSD or SURF [4], then a memory issue might occur. Therefore, this parameter needs to be adjusted with caution. The default value is 0.000001. The command is (-ints "someNumber"). An example would be - *python thresholdToold.py -a "path to negative sample values file" "path to positive sample values file" -ints 0.0001*

Additionally, test files populated with values are provided with the project. An example of using these files is provided in the next section. It must be noted that the *verificationTool.py* has not been adjusted to accommodate similarity measures that implement that reverse order. Therefore, further modifications need to be put in place before using the model with such similarity measures. For such values, as well as SURF [4], the threshold needs to be changed accordingly. Additionally, the SURF [4] method implemented in this model does not support the (-pm) command due to its different implementation.

C.2.1 Additional Information

There are two labels files that this project uses. The first one is provided to the *extraction* mode and when plotting. The format of that file is (Similarity Value Label). Both are separated by a space. For instance, 0.50 1.

On the other hand, the labels file provided for testing adopts the format of (Query Image,Reference Image,Label). It must be noted that there is no space between the commas. An example would look like - queryImage.jpg,referenceImage.jpg,0.

As stated before, the threshold script requires two files to be passed over. A negative and a positive list. These lists should have one similarity value per row:

0.58

0.76

0.65

There are two files required because, as it can be observed, no labels are provided next to the values. Therefore, the two files must be distinguishable and represent the negative and positive predicted sample values. Additionally, all of the above is described in the *README* file provided with the code of the project.

C.3 Replicating Test Results

C.3.1 Predicting

With the actual code, there will be two different pairs of images that can be found under the *Test/images* folder. These two pairs are provided in order to test the model. It can be tested by combining differently the commands described in the previous section. Moreover, the images with names *image1Q.jpg* and *image1R.jpg* are more picked to test the Template Matching method. For instance, to test the best configuration recorded for that method, the following command can be executed - *python verificationTool.py -p ./Test/images/image1Q.jpg ./Test/images/image1R.jpg -mm tm -thr 0.679* . The other pair of images, with names *image2Q.jpg* and *image2R.jpg* respectively, are picked to test the Patch Matching method. For instance, the following command can be used to test the method itself on the best threshold - *python verificationTool.py -p ./Test/images/image2Q.jpg ./Test/images/image2R.jpg -mm pm -thr 0.652* .

The thresholdTool.py can be tested as well. There are two pairs of negative and positive text files provided under the *Test/threshold/* folder. They represent the values obtained from the best configurations for both Datasets that were tested. The files with names *negbesttm.txt* and *posbesttm.txt* can be used to test the Template Matching method on the Caltech Buildings dataset [38]. The command that can be used to do so is as follows - *python thresholdTool.py -a ./Test/threshold/negbesttm.txt ./Test/threshold/posbesttm.txt* . The other two files can be used to test the threshold value and accuracy of the Patch Matching method on the Wiki_Commons [10] dataset. The command is - *python thresholdTool.py -a ./Test/threshold/negbestpm.txt ./Test/threshold/posbestpm.txt* .

C.3.2 Testing and Downloading Datasets

The following URL can be followed to download both datasets used immediately:

- https://emckclac-my.sharepoint.com/:f:/g/personal/k1763856_kcl_ac_uk/EiSS6CNIVuRFudp28yFeRfwBUgyjEnLCA_8EnWeGMwo94g?e=ctSrOB

It contains two folders called "caltech-buildings" and "wiki_commons" respectively. They contain the images used for testing this model. The whole folders must be downloaded, so the commands can be executed without errors. The naming of the folders is already set to the one used in the commands. A labels text file will be provided in the *Datasets/Caltech* folder of the project. The file is called labels.txt. To test on that dataset (for the best configuration) the

following command can be executed:

python verificationTool.py -test ./caltech-buildings/ ./caltech-buildings/ ./Datasets/Caltech/labels.txt -mm tm -thr 0.679

It must be noted that the *caltech-buildings* folder contains another folder called *images*. Nevertheless, there is no need to specify the path to that specific folder as the images in the *labels.txt* file already have that in their path. For the Wiki_Commons dataset [10] the *labels.txt* file can be found under the *WikiCommons* in the *Datasets* folder. To run the best recorded configuration, the following command can be executed:

python verificationTool.py -test ./wiki_commons/queries/ ./wiki_commons/references/ ./Datasets/WikiCommons/labels.txt -mm pm -thr 0.652

However, there is an alternative method that can be used to download both datasets.

To download the Caltech Buildings dataset [38], the following URL can be used as it was found to contain the dataset - `http://www.mohamedaly.info/datasets/caltech-buildings` [44]. There, a download link could be found. The dataset is 195MB.

To download the Wiki_Commons [10] dataset is more difficult. The authors of the BUPM [10] paper has provided the required files as well as the download guide for their dataset. It can be found at this URL `https://gitlab.vista.isi.edu/chengjia/image-GPS`. Some of the images might be corrupted or removed from the database, hence they need to be removed from the respective folders. The query images must be separated from the reference images as they have the same names. They must be put in two separate folders. Because the Google API is a paid service, the following tool can be used to download the reference images - `https://svd360.istreetview.com/`.

The commands that execute testing for both datasets can be altered to match the actual file paths, as they are only given as an example. If the datasets are downloaded not in the main directory of the project, then the paths to them must be changed accordingly. If the latter download methods are chosen, then it must be noted that not all images from the Wiki_Commons [10] dataset have been used, as also stated in Chapter 4. Besides, some of the images might not be able to download as they have either been corrupted or removed. Therefore, the former method is preferred, as all the images are provided in a single One Drive location.

Additional Information

The paths files provided in both commands can vary depending on where the datasets have been downloaded and how they have been named. Moreover, the paper can be followed in order to change the configurations to match the runs tested in the report. Every figure in Chapter 4 (Evaluation) provides information about the configuration.

Nevertheless, there are certain limitations to the arguments that the model accepts. In order to fully replicate the test runs provided in the report, some of the code must be changed. For instance, if the configuration is about the Patch Matching method, the following lines of code can be modified:

```
imgQ = cv2.resize(queryImg, (int(queryImg.shape[1] * float(0.15)), int(queryImg.shape[0] * float(0.15))), interpolation=cv2.INTER_AREA)
query = self.process_image(imgQ)
# Get reference patches
patches = image.extract_patches_2d(referImg, (224, 224), max_patches=250)
...
```

All lines can be found under the *patch_matching* function. The first line defines the size by which the query image is resized. For instance, if the configuration says that the query image has been resized by 20% then the float number 0.15 must be changed to 0.20. The fourth line defines the number of patches used (*max_patches*). If the configuration uses 150 patches then the 250 must be changed respectively. On the other hand, if the Template Matching is being used, then several other things must be changed. The lines that must be change can be found under the *template_matching* function:

```
def template_matching(self, queryImg, referImg):
    if sum(queryImg.shape) > sum(referImg.shape):
        scales = [10, 12, 14, 16, 18, 20]
    else: scales = [22, 24, 26, 28, 30]
    ...
```

For instance, if the configuration uses only one scale range then all lines must be removed and replaced with only - (*e.g. scales = [13,14,15,16,17]*), depending on the configuration selected. In conjunction with the commands described before, the figures in Chapter 4 can be followed to obtain all the configurations used to test the model.

To test the *thresholdTool.py* file on a dataset, as stated before, two files are required. These files can be obtained by using a specific function in the *verificationTool.py* file. The function is initially commented out to prevent unwanted behaviour. To produce these files, the model must be tested on a dataset using the commands provided in this section, as well as in the

previous one. Before running the model, this function needs to be enabled. It can be found under the *test* method. The # must be removed from the front of the function:

```
# Uncomment to write samples to file
# self.write_samples_to_file(neg, pos)
```

After enabling the function, the saving directory of the files, as well as their names, can be changed. This is done in the following method:

```
# This function writes the predicted values to two files:
# negative and positive.
# This is needed for the thresholdTool script
# to find the best threshold given these two files.
#
# @param neg The list of negative predictions
# @param pos The list of positive predictions
def write_samples_to_file(self, neg, pos):
    with open('./negative.txt', 'w') as f:
        for item in neg:
            f.write(str(item)+"\n")
        f.close()
    with open('./positive.txt', 'w') as f:
        for item in pos:
            f.write(str(item)+"\n")
        f.close()
```

C.4 Commands for verificationTool.py

This section extends the information provided in the previous one by providing additional images of the commands for the verificationTool.py file. The output and visual appearance should be the same on any machine, although there could be differences. Such could be because of the way the data has been sorted. The commands use the paths from the previous section. Therefore, they can be used by the user as well, provided the required files are present and in the correct location. It must be noted that a part of the images has been blurred because it contains personal information.

C.4.1 Predicting

```
[redacted]GPS-Image-Verification>python verificationTool.py -p ./Test/images/image1Q.jpg ./Test/images/image1R.jpg -mm tm -thr 0.679 -pm
Using TensorFlow backend.
Keras: 2.3.1 TF: 2.0.0 NumPy: 1.16.0 OpenCV: 3.4.2 Sklearn: 0.22.1 Argparse: 1.1 Matplotlib: 3.1.0 Time: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [
MSC v.1916 64 bit (AMD64)] Warnings: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] OS: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:1
6:47) [MSC v.1916 64 bit (AMD64)]
Initializing tool with: threshold = 0.679 | predict_method = threshold | match_method = tm | SURF = False | model = resnet50 | measure = cc
Predicting...
BEST SIMILARITY: 0.778
The query image is present on the reference image with similarity: 0.778!

======================
Predict Time: 59s
```

Figure C.1: The command for predicting using Template Matching with a custom threshold. The command for printing has been executed as well.

```
[redacted]GPS-Image-Verification>python verificationTool.py -p ./Test/images/image2Q.jpg ./Test/images/image2R.jpg -mm pm -thr 0.652 -pm
Using TensorFlow backend.
Keras: 2.3.1 TF: 2.0.0 NumPy: 1.16.0 OpenCV: 3.4.2 Sklearn: 0.22.1 Argparse: 1.1 Matplotlib: 3.1.0 Time: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [
MSC v.1916 64 bit (AMD64)] Warnings: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] OS: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:1
6:47) [MSC v.1916 64 bit (AMD64)]
Initializing tool with: threshold = 0.652 | predict_method = threshold | match_method = pm | SURF = False | model = resnet50 | measure = cc
Predicting...
BEST SIMILARITY: 0.775
The query image is present on the reference image with similarity: 0.775!

======================
Predict Time: 74s
```

Figure C.2: The command for predicting using Patch Matching with a custom threshold. The command for printing has been invoked as well.

C.4.2 Testing

```
[redacted]\GPS-Image-Verification>python verificationTool.py -test ./caltech-buildings/ ./caltech-buildings/ ./Datasets/Caltech/labels.txt -mm tm

Using TensorFlow backend.
Keras:  2.3.1  TF:  2.0.0  NumPy:  1.16.0  OpenCV:  3.4.2  Sklearn:  0.22.1  Argparse:  1.1  Matplotlib:  3.1.0  Time:  3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [
MSC v.1916 64 bit (AMD64)]  Warnings:  3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)]  OS:  3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:1
6:47) [MSC v.1916 64 bit (AMD64)]
Initializing tool with: threshold = 0.585 | predict_method = threshold | match_method = tm | SURF = False | model = resnet50 | measure = cc
Testing using threshold...
BEST SIMILARITY:  0.663
The query image is present on the reference image with similarity: 0.663!

======================
Predict Time: 36s
REF:  ./caltech-buildings/images/DSCN0209.JPG  QUERY:  ./caltech-buildings/images/DSCN0208.JPG
PRED:  1  AND LABEL:  1

[--------------------------------------------------] 0.3%
Current Accuracy:  1.0

BEST SIMILARITY:  0.681
```

Figure C.3: The command for testing using Template Matching. The threshold command has not been invoked, although it could be, as seen in the previous section. Therefore, the results will not be the same as the best configuration recorded.

```
[redacted]GPS-Image-Verification>python verificationTool.py -test ./wiki_commons/queries/ ./wiki_commons/references/ ./Datasets/WikiCommons/labe
ls.txt -mm pm
Using TensorFlow backend.
Keras:  2.3.1  TF:  2.0.0  NumPy:  1.16.0  OpenCV:  3.4.2  Sklearn:  0.22.1  Argparse:  1.1  Matplotlib:  3.1.0  Time:  3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [
MSC v.1916 64 bit (AMD64)]  Warnings:  3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)]  OS:  3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:1
6:47) [MSC v.1916 64 bit (AMD64)]
Initializing tool with: threshold = 0.585 | predict_method = threshold | match_method = pm | SURF = False | model = resnet50 | measure = cc
Testing using threshold...
BEST SIMILARITY:  0.463
The query image is NOT present on the reference image with similarity: 0.463!

======================
Predict Time: 129s
REF:  ./wiki_commons/references/bc47207d69530b6f9bff62d5bde7410b.jpg  QUERY:  ./wiki_commons/queries/c332e89b6d9028ab595bc1ad3ad587f0.jpg
PRED:  0  AND LABEL:  0

[--------------------------------------------------] 0.2%
Current Accuracy:  1.0
```

Figure C.4: The command for testing using Patch Matching. The threshold command has not been invoked, although it could be, as seen in the previous section. Therefore, the results will not be the same as the best configuration recorded.

C.4.3 Extracting Values

```
GPS-Image-Verification>python verificationTool.py -e -ed ./wiki_commons/queries/ ./wiki_commons/references/ ./Datasets/WikiCommons/labels.txt -mm pm -ep ./Test/plot/values.txt
Using TensorFlow backend.
Keras: 2.3.1 TF: 2.0.0 NumPy: 1.16.0 OpenCV: 3.4.2 Sklearn: 0.22.1 Argparse: 1.1 Matplotlib: 3.1.0 Time: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] Warnings: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:
16:47) [MSC v.1916 64 bit (AMD64)] OS: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)]
Initializing tool with: threshold = 0.585 | predict_method = threshold | match_method = pm | SURF = False | model = resnet50 | measure = cc
Trying to create a new extraction file...
New extraction file -> ./Test/plot/values.txt!
Extracting...
BEST SIMILARITY: 0.471
[--------------------------------------------------] 0.2%
Label: 0 ...(1/662)
```

Figure C.5: The commands for extracting values to a specific text file. Here the threshold does not play a role because these commands extract the best similarity for every sample.

C.4.4 Plotting

```
GPS-Image-Verification>python verificationTool.py --plot -ep ./Test/plot/newtext.txt
Using TensorFlow backend.
Keras: 2.3.1 TF: 2.0.0 NumPy: 1.16.0 OpenCV: 3.4.2 Sklearn: 0.22.1 Argparse: 1.1 Matplotlib: 3.1.0 Time: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [
MSC v.1916 64 bit (AMD64)] Warnings: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] OS: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:1
6:47) [MSC v.1916 64 bit (AMD64)]
Initializing tool with: threshold = 0.585 | predict_method = threshold | match_method = tm | SURF = False | model = resnet50 | measure = cc
```

Figure C.6: The commands used for plotting provided the text file from the previous figure C.5. The data would be presented to the user after executing the command. Additionally, the file used can be found in the directory of the project, as stated in the previous section.

C.4.5 Using Models and Measures

```
GPS-Image-Verification>python verificationTool.py -p ./Test/images/image2Q.jpg ./Test/images/image2R.jpg -mm pm --measure ncc --model
inception -thr 0.54 -pm
Using TensorFlow backend.
Keras: 2.3.1 TF: 2.0.0 NumPy: 1.16.0 OpenCV: 3.4.2 Sklearn: 0.22.1 Argparse: 1.1 Matplotlib: 3.1.0 Time: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [
MSC v.1916 64 bit (AMD64)] Warnings: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] OS: 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:1
6:47) [MSC v.1916 64 bit (AMD64)]
Initializing tool with: threshold = 0.54 | predict_method = threshold | match_method = pm | SURF = False | model = inception | measure = ncc
Predicting...
BEST SIMILARITY: 0.977
The query image is present on the reference image with similarity: 0.977!

======================
Predict Time: 98s
```

Figure C.7: The commands for using different models and measures. The figure shows the commands for predicting, using a custom threshold, printing the likelihood masks as well as using different model and similarity measure. The previous section can be followed in order to change to other different models or measures.

C.4.6 Using SURF

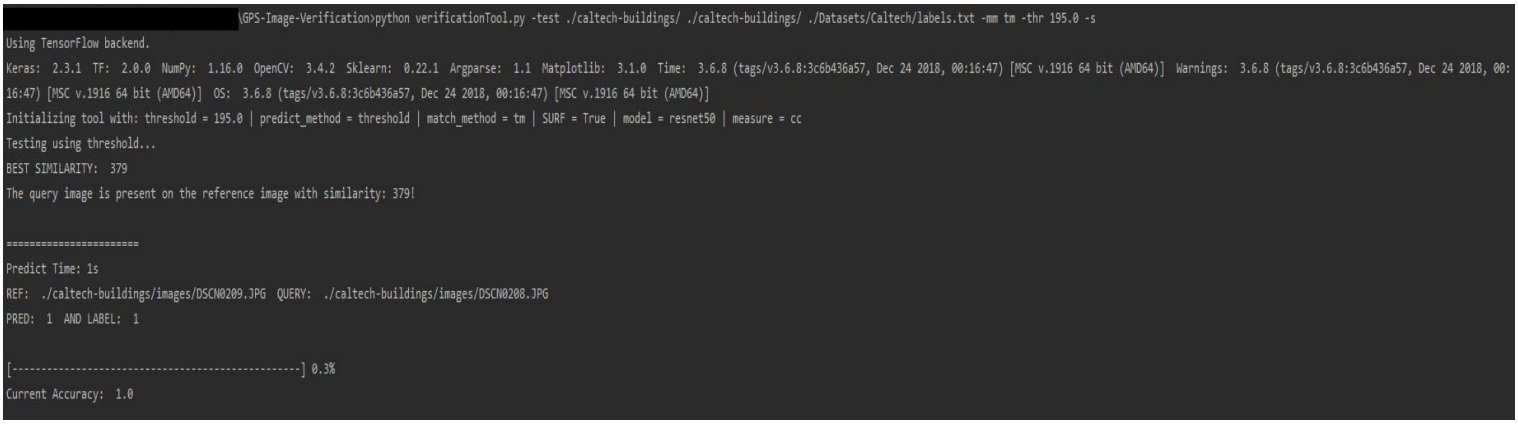

Figure C.8: This command toggles the SURF [4] method on the testing command. It results in testing on the specified dataset with that method, rather than with the ones proposed in the report. Additionally, other commands can be used in conjunction. For example, the threshold command has been used with its value adjusted accordingly. Also, this method can be used for prediction as well, not only for testing. It simply specifies the method being used by the model. However, the print command is not supported by the SURF [4] method due to its different implementation.

C.5 Commands for thresholdTool.py

The following section explores and extends the commands presented in the "User Guide" section for the thresholdTool.py file. The paths used are the same as previously shown. Thus, they can be used by the user as well, given the required files are present and in the correct location. It must be also noted that a part of the images has been blurred because it contains personal information.

C.5.1 Finding the Best Threshold

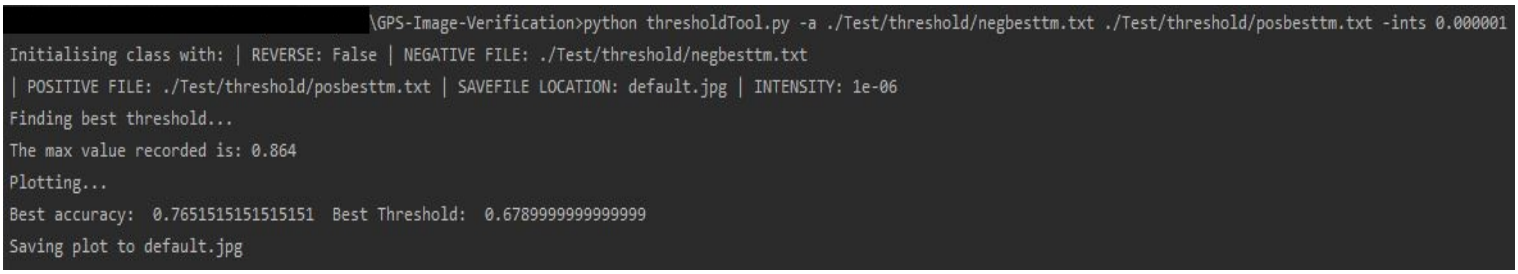

Figure C.9: This command finds the best threshold for the two files provided. Both the negative and positive files can be found in the directory of the project. The files represent the best configuration for the Template Matching method. The intensity command has been given its default value, just as an example. The path to the newly created path has been omitted to show the default usage of the command.

```
[redacted]\GPS-Image-Verification>python thresholdTool.py -a ./Test/threshold/negbestpm.txt ./Test/threshold/posbestpm.txt ./new_img_path.jpg
Initialising class with: | REVERSE: False | NEGATIVE FILE: ./Test/threshold/negbestpm.txt
| POSITIVE FILE: ./Test/threshold/posbestpm.txt | SAVEFILE LOCATION: ./new_img_path.jpg | INTENSITY: 1e-06
Finding best threshold...
The max value recorded is: 0.835
Plotting...
Best accuracy:  0.7129909365558912  Best Threshold:  0.652
Saving plot to ./new_img_path.jpg
```

Figure C.10: This command finds the best threshold for the best Patch Matching configuration. The files can be found in the directory of the project, as stated in previous sections. The intensity command has been omitted to show its default interaction. Additionally, the third path provided is for the path of the newly created graph.

C.5.2 Finding the Best Threshold - Reversed Order

```
[redacted]\GPS-Image-Verification>python thresholdTool.py -a ./Test/threshold/negbestpm.txt ./Test/threshold/posbestpm.txt ./new_img_path.jpg -ints 0.000001 -r
Initialising class with: | REVERSE: True | NEGATIVE FILE: ./Test/threshold/negbestpm.txt
| POSITIVE FILE: ./Test/threshold/posbestpm.txt | SAVEFILE LOCATION: ./new_img_path.jpg | INTENSITY: 1e-06
Finding best threshold...
The max value recorded is: 0.835
Plotting...
Best accuracy:  0.5030211480362538  Best Threshold:  0.44
Saving plot to ./new_img_path.jpg
```

Figure C.11: This command shows the best Patch Matching configuration values with the reverse mode toggled on. It finds the best minimum threshold. This, as stated before, is a future-proof option for different similarity measures. The intensity is also specified to its default value. It can be observed that the threshold and accuracy are significantly lower and different to the figures in the previous subsections. That is, again, because of the reverse order used by the script.

Appendix D

Source Code Listings

D.1 Originality Avowal

"I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary."

Preslav Kisyov

April 23, 2020

D.2 verificationTool.py

This is the file that contains the proposed model. It performs Image Verification by utilising different Computer Vision methods.

```
1  import argparse
2  import cv2
3  import numpy as np
4  import matplotlib
5  import matplotlib.pyplot as plt
6  import time
7  import warnings
8  from sklearn.feature_extraction import image
9  import os.path
10 import keras
11 import tensorflow
12 import sklearn
13 import sys
14
15 # This is the Image Verification Tool class that is used
16 # to verify whether two images are similar enough. It uses different
17 # Computer Vision algorithms and methods to perform Feature Extraction, Image
   ↪  Verification
18 # and Image Segmentation.
19 #
20 # @author Preslav Kisyov
21 # @version 1.1
22 class ImageVerificationTool:
23     # Print library versions
24     print("Keras: ", str(keras.__version__), " TF: ",
       ↪  str(tensorflow.__version__), " NumPy: ", str(np.__version__), "
       ↪  OpenCV: ", str(cv2.__version__),
```

```
25            " Sklearn: ", str(sklearn.__version__), " Argparse: ",
              ↪  str(argparse.__version__),  " Matplotlib: ",
              ↪  str(matplotlib.__version__),
26            " Time: ", str(sys.version), " Warnings: ", str(sys.version),  " OS:
              ↪  ", str(sys.version))
27
28        # Ignore CPU Warnings
29        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
30        warnings.simplefilter(action='ignore', category=FutureWarning)
31
32        # This function picks the CNN to be used
33        # for feature extraction depending on the argument
34        # passed through to the model
35        #
36        # @param model The model to be used
37        def pick_model(self, model):
38            # Initialise the model depending on the provided arg
39            # Note: Default is always ResNet50, so it cannot be None
40            if model == "resnet50":
41                from keras.applications.resnet import ResNet50, preprocess_input
42                self.resnet = ResNet50(include_top=False, weights="imagenet")
43            elif model == "resnet101":
44                from keras.applications.resnet import ResNet101, preprocess_input
45                self.resnet = ResNet101(include_top=False, weights="imagenet")
46            elif model == "resnet152":
47                from keras.applications.resnet import ResNet152, preprocess_input
48                self.resnet = ResNet152(include_top=False, weights="imagenet")
49            elif model == "vgg19":
50                from keras.applications.vgg19 import VGG19, preprocess_input
51                self.resnet = VGG19(include_top=False, weights="imagenet")
52            elif model == "inception":
53                from keras.applications.inception_resnet_v2 import
                  ↪  InceptionResNetV2, preprocess_input
```

```
54              self.resnet = InceptionResNetV2(include_top=False,
                ↪  weights="imagenet")
55          self.preprocess_input = preprocess_input
56
57      # Import the query and reference images as an array
58      #
59      # @param query_path The path to the query image
60      # @param refer_path The path to the reference image
61      # @return The query image array
62      # @return The reference image array
63      def read_images(self, query_path, refer_path): return
        ↪  cv2.imread(query_path), cv2.imread(refer_path)
64
65      # Pre-process an image so it can
66      # be passed over to a Residual Network.
67      # This includes zero-centering the pixels as
68      # well as increasing the dimension
69      #
70      # @param image The image to be pre-processed
71      # @return The pre-processed image of shape (1, h, w, 3)
72      def process_image(self, image):
73          img = np.expand_dims(image, axis=0)
74          return self.preprocess_input(img.astype(np.float32))
75
76      # Extract feature vectors from two images using a ResNet
77      #
78      # @param query_image The query image array
79      # @param refer_image The reference image array
80      # @return The query feature vector
81      # @return The reference feature vector
82      def get_features(self, query_image, refer_image):
83          query_features = self.resnet.predict(query_image)[0]
84          refer_features = self.resnet.predict(refer_image)[0]
```

```
85
86          sizes = [[query_features.shape[0], query_features.shape[1]],
87                   [refer_features.shape[0], refer_features.shape[1]]]
88
89          query_reshaped = query_features.reshape(query_features.shape[0] *
90                          query_features.shape[1], query_features.shape[2])
91          refer_reshaped = refer_features.reshape(refer_features.shape[0] *
92                          refer_features.shape[1], refer_features.shape[2])
93          return query_reshaped, refer_reshaped, sizes
94
95      # Perform Correlation Coefficient on two feature vectors
96      # in order to get a similarity vector with best matching features
97      #
98      # @param query_f The query feature vector
99      # @param refer_f The reference feature vector
100     # @return cc The Correlation Coefficient vector
101     def get_cc(self, query_f, refer_f):
102         # Local Center the pixels
103         query = query_f - np.mean(query_f, axis=0, keepdims=True)
104         refer = refer_f - np.mean(refer_f, axis=0, keepdims=True)
105
106         # Normalize query vector
107         query = np.transpose(query)
108         query_v = np.sum(np.square(query), axis=0, keepdims=True)
109
110         # Normalize refer vector
111         refer_v = np.sum(np.square(refer), axis=1, keepdims=True)
112
113         # Get denominator
114         denominator = np.multiply(np.sqrt(refer_v), np.sqrt(query_v))
115
116         # Get Dot Product
117         product = np.dot(refer, query)
```

```
118
119             # Handle division by 0 or Nan (will return 0)
120             np.seterr(divide='ignore', invalid='ignore')
121
122             # Getting Correlation coefficient
123             cc = np.divide(product, denominator)
124             return cc
125
126         # Perform The Normalized Cross-Correlation on two feature vectors
127         # in order to get a similarity vector with best matching features
128         #
129         # @param query_f The query feature vector
130         # @param refer_f The reference feature vector
131         # @return ncc The Normalized Cross-Correlation/Cosine Similarity vector
132         def get_ncc(self, query_f, refer_f):
133             # Normalize query vector
134             query = np.transpose(query_f)
135             query_v = np.sum(np.square(query), axis=0, keepdims=True)
136
137             # Normalize refer vector
138             refer_v = np.sum(np.square(refer_f), axis=1, keepdims=True)
139
140             # Get denominator
141             denominator = np.multiply(np.sqrt(refer_v), np.sqrt(query_v))
142
143             # Get Dot Product
144             product = np.dot(refer_f, query)
145
146             # Handle division by 0 or Nan (will return 0)
147             np.seterr(divide='ignore', invalid='ignore')
148
149             # Getting The Normalized Cross-Correlation/Cosine Similarity
150             ncc = np.divide(product, denominator)
```

```
151
152             return ncc
153
154         # Extract the likelihood maps for two images
155         # from a similarity vector with shape (h*w, h1*w1)
156         #
157         # @param similarity_v The similarity vector
158         # @param sizes The sizes of the query and reference maps
159         # @return query_mask The query feature mask
160         # @return refer_mask The reference feature mask
161         def get_masks(self, similarity_v, sizes):
162             # Extract best features from the two axes of the similarity vector
163             # Get the biggest values
164             query_mask = np.max(similarity_v, axis=0)
165             refer_mask = np.max(similarity_v, axis=1)
166
167             # Reshape the masks to be of two dimensions so they can be shown as
                ↪  images
168             query_mask = query_mask.reshape(sizes[0][0], sizes[0][1])
169             refer_mask = refer_mask.reshape(sizes[1][0], sizes[1][1])
170             return query_mask, refer_mask
171
172         # This function extracts the starting and ending
173         # points of the query image for it to be presented
174         # on the reference image
175         #
176         # @param loc The location of the object
177         # @param w The width of the query image
178         # @param h The height of the query image
179         # @param scale The scale to have the query mask dimensions reduced
180         # @return first_point, second_point The starting and ending points
181         def get_points(self, loc, w, h, scale=1):
182             first_point = (int(loc[0] - w / scale), int(loc[1] - h / scale))
```

```
183             second_point = (int(loc[0] + w / scale), int(loc[1] + h / scale))
184             return first_point, second_point
185
186         # Show the query and reference masks using matplotlib,
187         # as well as their original images
188         #
189         # @param images The two original images
190         # @param masks The two mask vectors
191         # @param patch If available, a patch reference image
192         def print_masks(self, images, masks, patch):
193             q_mask, r_mask = masks
194
195             # If a patch is not available, draw a likelihood object rectangle
196             # on the reference image, showing the best matched location
197             if patch is None:
198                 w, h = q_mask.shape[::-1]  # Query image
199                 _, _, _, loc = cv2.minMaxLoc(r_mask)
200                 first_point, second_point = self.get_points(loc, w, h)
201                 cv2.rectangle(r_mask, first_point, second_point, 255, -1)  # Draw
                    ↪  the object
202             else: r_mask = cv2.cvtColor(patch, cv2.COLOR_BGR2GRAY)  # Show the
                ↪  patch image instead
203
204             # Plot the query and reference masks as well as their original images
205             fig, axes = plt.figure(figsize=(15, 15)), (2, 2)
206
207             info_map = {0: ["Query original image", images[0]], 1: ["Reference
                ↪  original image", images[1]],
208                    2: ["Query image mask", q_mask], 3: ["Reference image mask",
                       ↪  r_mask]}
209
210             # Creating a subplot for every image
211             for index_ax in range(sum(axes)):
```

```
212             subPlot = fig.add_subplot(axes[0], axes[1], index_ax+1)  # rows;
                ↪  cols; index
213             subPlot.title.set_text(info_map[index_ax][0])
214             plt.imshow(info_map[index_ax][1], 'gray')
215         plt.show()
216
217     # Resize the query image with given certain percentage/scale.
218     #
219     # @param query The query image to be resized
220     # @param scale The percentage to have the image shape rescaled to
221     # @return query The newly resized query image
222     def get_query_image(self, query, scale):
223         width = query.shape[1]*scale/100
224         height = query.shape[0]*scale/100
225         query = cv2.resize(query, (int(width), int(height)),
            ↪  interpolation=cv2.INTER_AREA)
226         return query
227
228     # Perform Patch Matching technique using a query and reference image.
229     # This method extracts patches of certain size from the reference image
230     # and then extracts features and similarity vector given every patch.
231     #
232     # @param queryImg The query image array
233     # @param referImg The reference image array
234     # @return maxSimilarity The best similarity value
235     # @return bestCC The best Correlation Coefficient vector
236     # @return bestSizes The sizes of the best feature vectors
237     # @return patch The best matched patch image
238     def patch_matching(self, queryImg, referImg):
239         maxSimilarity, bestSizes, bestCC = 0, [], None
240         best_patch = None
241         # Resize/Pre-process query image
```

```
242             imgQ = cv2.resize(queryImg, (int(queryImg.shape[1] * float(0.15)),
                ↪  int(queryImg.shape[0] * float(0.15))),
                ↪  interpolation=cv2.INTER_AREA)
243             query = self.process_image(imgQ)
244             # Get reference patches
245             patches = image.extract_patches_2d(referImg, (224, 224),
                ↪  max_patches=250)
246             # Iterate over every patch image
247             for patch in patches:
248                 patch_img = self.process_image(patch)
249                 query_f, refer_f, sizes = self.get_features(query, patch_img)
250                 # Perform Correlation Coefficient and extract similarity value
251                 maxSim, cc = self.get_max_value(query_f, refer_f)
252                 # Update values to the best ones so far
253                 if maxSim >= maxSimilarity:
254                     maxSimilarity, bestCC, bestSizes, best_patch = maxSim, cc,
                        ↪  sizes, patch
255
256                 # Stop performing the method if a match has been found
257                 if self.check_max_similarity(maxSim): break
258             return maxSimilarity, bestCC, bestSizes, best_patch
259
260         # Perform the picked Similarity Measure method
261         # and extract the maximum/minimum value from the similarity vector
262         #
263         # @param query_f The query feature vector
264         # @param refer_f The reference feature vector
265         # @return A rounded similarity value
266         # @return The Similarity Measure vector
267         def get_max_value(self, query_f, refer_f):
268             # Pick the Similarity Measure to be used given the passed
269             # argument. The measure will never be None as it has default value
270             measure = None
```

```
271             if self.measure == "cc": measure = self.get_cc(query_f, refer_f)
272             elif self.measure == "ncc": measure = self.get_ncc(query_f, refer_f)
273             _, sim, _, _ = cv2.minMaxLoc(measure)
274 
275             return round(sim, 3), measure
276 
277         # Perform the Template Matching technique variation on a query and
            ↪  reference image.
278         # This method rescales the query image to different sizes, using it as a
            ↪  template in order
279         # to match it to the reference image.
280         #
281         # @param queryImg The query image array
282         # @param referImg The reference image array
283         # @return maxSimilarity The best similarity value
284         # @return bestCC The best Correlation Coefficient vector
285         # @return bestSizes The sizes of the best feature vectors
286         def template_matching(self, queryImg, referImg):
287             if sum(queryImg.shape) > sum(referImg.shape):
288                 scales = [10, 12, 14, 16, 18, 20]
289             else: scales = [22, 24, 26, 28, 30]
290             maxSimilarity, bestSizes, bestCC = 0, [], None
291             imgR = self.process_image(referImg)
292             for scale in scales:
293                 query = self.get_query_image(queryImg, scale)
294                 query = self.process_image(query)
295                 query_f, refer_f, sizes = self.get_features(query, imgR)
296                 # Perform Correlation Coefficient and extract similarity value
297                 maxSim, cc = self.get_max_value(query_f, refer_f)
298 
299                 # Update values to the best ones so far
300                 if maxSim >= maxSimilarity:
301                     maxSimilarity, bestCC, bestSizes = maxSim, cc, sizes
```

```
302
303
304             # Stop performing the method if a match has been found
305             if self.check_max_similarity(maxSim): break
306
307         return maxSimilarity, bestCC, bestSizes
308
309     # Perform a check whether further iteration of either
310     # the image scales or patches is required.
311     #
312     # @param maxSim The current best similarity value
313     # @return False If further iterations are required
314     # @return True If the iteration process should stop
315     def check_max_similarity(self, maxSim):
316         if self.extract or self.print_mask or self.isTest: return False  # Do
            ↪  whole iterations when extracting values
317         # Predict against threshold
318         return True if maxSim >= self.threshold else False
319
320     # The following code is patented, thus it must not be
321     # used for commercial use. Similar implementation of the
322     # code can also be found at -
        ↪  https://docs.opencv.org/master/d5/d6f/tutorial_feature_flann_matcher.html
        ↪  (OpenCV)
323     # The paper of the method is Speeded-Up Robust Features (SURF), with
        ↪  authors - Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool
324     #
325     # This function gets the number of matches produced by the method.
326     #
327     # @params gray_imgQ, gray_imgR The Query and Reference images in gray mode
328     # @return The number of matches above a threshold
329     def get_surf_value(self, gray_imgQ, gray_imgR):
```

```
330             surf = cv2.xfeatures2d.SURF_create(hessianThreshold=400) # Initialise
                ↪  the method
331             # Extract features
332             _, q_features = surf.detectAndCompute(gray_imgQ, None)
333             _, r_features = surf.detectAndCompute(gray_imgR, None)
334             feature_matcher =
                ↪  cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)
335             feature_matches = feature_matcher.knnMatch(q_features, r_features, 2)
336
337             return len([match_pair[0] for match_pair in feature_matches if
                ↪  match_pair[0].distance < 0.75 * match_pair[1].distance])
338
339         # Get the best maximum similarity value by performing different
340         # matching methods, depending on the choice
341         #
342         # @param path_q The path to the query image
343         # @param path_r The path to the reference image
344         # @param patch Default None, if Patch Matching is performed, it will be an
                ↪  image patch
345         # @return best_max_sim the best maximum similarity
346         def get_max_similarity(self, path_q, path_r, patch=None):
347             imgQ, imgR = self.read_images(path_q, path_r)
348             if self.surf:
349                 gray_imgQ = cv2.cvtColor(imgQ, cv2.COLOR_BGR2GRAY)
350                 gray_imgR = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)
351
352                 # Resize to 80% of the images to avoid memory overflow
353                 gray_imgQ = cv2.resize(gray_imgQ, (int(gray_imgQ.shape[1] *
                    ↪  float(0.80)), int(gray_imgQ.shape[0] * float(0.80))),
                    ↪  interpolation=cv2.INTER_AREA)
354                 gray_imgR = cv2.resize(gray_imgR,
355                                        (int(gray_imgR.shape[1] * float(0.80)),
                                        ↪  int(gray_imgR.shape[0] * float(0.80))),
```

```
356                                     interpolation=cv2.INTER_AREA)
357
358             # Get the number of matches (the similarity value)
359             best_max_sim = self.get_surf_value(gray_imgQ, gray_imgR)
360             if self.print_mask: print("Printing has been disabled for SURF
                ↪  because of its different implementation!")
361         else:
362             # Check matching method
363             if self.match_method == "tm": best_max_sim, bestCC, bestSizes =
                ↪  self.template_matching(imgQ, imgR)
364             else: best_max_sim, bestCC, bestSizes, patch =
                ↪  self.patch_matching(imgQ, imgR)
365
366             # Extract and draw image masks
367             if self.print_mask is True and best_max_sim >= self.threshold:
368                 images = [cv2.imread(path_q), cv2.imread(path_r)]
369                 mask_q, mask_r = self.get_masks(bestCC, bestSizes)
370                 self.print_masks(images, [mask_q, mask_r], patch)
371
372         print("BEST SIMILARITY: ", best_max_sim)
373         return best_max_sim
374
375     # Generate a file that can be used to visualise data
376     # The file format is as follows:
377     # FEATURE LABEL -> Similarity Value ; Prediction
378     #
379     # @param dataset A list containing the paths to both images
380     # as well as the path to the prediction labels
381     # @param dataset_path The path for the new generated file
382     def create_extr_file(self, dataset, dataset_path):
383         self.comp_time = time.time()  # Start counting time
384         extr_features, extr_labels = [], []
385         pathsQ, pathsR, labels = dataset
```

```
386             new_data = self.load_dataset(pathsQ, pathsR, labels)
387
388             # Make features and labels lists and print the current progress
389             for path_index in range(len(new_data)):
390                 similarity = self.get_max_similarity(new_data[path_index][0],
                    ↪  new_data[path_index][1])
391                 self.show_progress(path_index+1, len(new_data))
392                 label = new_data[path_index][2]
393                 print("Label: "+str(label)+"
                    ↪  ...("+str(path_index+1)+"/"+str(len(new_data))+")")
394                 extr_features.append(similarity)
395                 extr_labels.append(label)
396
397             # Create an extraction file with values
398             try:
399                 if os.path.exists(dataset_path): print("Extraction file already
                    ↪  exists! Overwriting file...!")
400                 with open(dataset_path, 'w') as f:
401                     for item in range(len(extr_features)):
402                         feature = str(extr_features[item])
403                         label = str(extr_labels[item])
404                         f.write(feature+" "+label+" \n")
405                 print("New file created -> "+dataset_path)
406             except IOError:
407
408
409                 print("Could not write to " + dataset_path + " file")
410                 raise
411             new_time = self.get_time()
412             print("Average computation time: "+str(new_time/len(new_data))+"s")
413             print("Test total time: "+str(new_time)+"s")
414
415         # Generate two arrays that contain the features and labels
```

```
416     # from a specified extraction/test file
417     #
418     # @param file_path The path to the file
419     # @return features A list of features
420     # @return labels A list of labels
421     def load_from_file(self, file_path):
422         features, labels = [], []
423         try:
424             with open(file_path, 'r') as f:
425                 lines = f.readlines()
426                 for i in range(len(lines)):
427                     features.append([float(lines[i].split(' ')[0])])
428                     labels.append(int(lines[i].split(' ')[1].split('\n')[0]))
429             f.close()
430         except IOError:
431             print("File " + file_path + " not found!")
432             raise
433         return features, labels
434
435     # Load a file so its content can be plotted on a graph
436     #
437     # @param file_path The path to the file
438     # @return pos_x The positive samples (samples with label 1)
439     # @return neg_x The negative samples (samples with label 0)
440     def load_for_plot(self, file_path):
441         neg_x, pos_x  = [], []
442         try:
443             with open(file_path, 'r') as f:
444                 lines = f.readlines()
445                 # Depending on the label pick pos/neg samples
446                 for i in range(len(lines)):
447                     if int(lines[i].split(' ')[1].split('\n')[0]) == 0:
448                         neg_x.append(float(lines[i].split(' ')[0]))
```

```
449                     else:
450                         pos_x.append(float(lines[i].split(' ')[0]))
451             f.close()
452         except IOError:
453             print("File "+file_path+" not found!")
454             raise
455 
456         return pos_x, neg_x
457 
458     # Plot data given positive and negative samples
459     #
460     # @param pos_x Positive samples
461     # @param neg_x Negative samples
462     # @return If no samples could be found
463     def plot_data(self, pos_x, neg_x):
464         if (len(pos_x) and len(neg_x)) == 0:
465             print("Could not find any samples!")
466             return
467         fig, ax = plt.subplots(figsize=(10, 8))
468         ax.set_ylabel(np.arange(0, max(pos_x), 1.0))
469         avg_x = np.mean(pos_x)
470         avg_y = np.mean(neg_x)
471 
472         plt.bar(range(len(pos_x)), pos_x, width=0.4, color="red",
            ↪  label="Similar", edgecolor='red')  # plotting by columns
473         plt.bar(range(len(neg_x)), neg_x, width=0.4,  label="Not Similar",
            ↪  color="blue", edgecolor='blue')
474         ax.set_title("Threshold Graph", fontweight='bold')
475         ax.set_ylabel('Similarity Score', fontweight='bold')
476         plt.xlabel('Prediction', fontweight='bold')
477         ax.yaxis.grid(True)
478         plt.legend(["Similar " + '%.2f'%avg_x+" avrg", "Not Similar " +
            ↪  '%.2f'%avg_y+" avrg"], loc='upper right')
```

```
479             plt.show()
480
481         # Generate a dataset from the content of a labels file
482         # containing QUERY, REFERENCE, LABEL.
483         # The result is a list of the full paths for both images as well as the
            ↪  label
484         #
485         # @param pathQ The path to the query image
486         # @param pathR The path to the reference image
487         # @param pathL The path to a labels file of format (Q, R, L)
488         def load_dataset(self, pathQ, pathR, pathL):
489             dataset = []
490             try:
491                 with open(pathL, 'r') as f:
492                     lines = f.readlines()
493                     for line in lines:
494                         line_split = line.split(',')
495                         if len(line_split) < 3:
496                             print("Unsupported labels format!")
497                             raise IOError
498                         query_image, ref_image, label = line_split[0],
                            ↪  line_split[1], int(line_split[2].split("\n")[0])
499                         dataset.append([pathQ+query_image, pathR+ref_image,
                            ↪  label])
500                 f.close()
501             except IOError:
502                 print("File not found!")
503                 raise
504
505             return dataset
506
507         # Load a test dataset and perform testing on it
508         #
```

```
509         # @param test_dataset A list of Q, R, L paths
510         # @return The results of testing
511         def test_tool(self, test_dataset):
512             if not self.check_file([test_dataset[2]]): return
513             self.isTest = True
514             data = self.load_dataset(test_dataset[0], test_dataset[1],
                ↪  test_dataset[2])
515             return self.test(data)
516
517         # This function checks whether a file or list of files
518         # exists and will not throw an error.
519         #
520         # @param files List of files
521         # @return If any of the files do not exist - False else True
522         def check_file(self, files):
523             for file in files:
524                 if not os.path.isfile(file):
525                     print("File "+str(file)+" does NOT exist!")
526                     return False
527             return True
528
529         # Perform testing on some data,
530         # returning an accuracy value
531         #
532         # @param data The data to perform testing on ([Q, R, L])
533         # @return Accuracy value
534         def test(self, data):
535             good_predictions, predictions = 0, 1
536             neg, pos = [], []
537             test_time = 0
538             length_data = len(data)
539             print("Testing using threshold...")
540             for pair in data:
```

```
541             if not self.check_file([pair[0], pair[1]]): return
542             pred, similarity, comp_time = self.predict(pair[0], pair[1])
543             print("REF: ", pair[1], " QUERY: ", pair[0])
544             print("PRED: ", str(pred), " AND LABEL: ", str(pair[-1]), "\n")
545
546             # Separate predictions to negative and positive
547             if pair[-1] == 0: neg.append(similarity)
548             else: pos.append(similarity)
549
550             good_predictions += 1 if pred == pair[-1] else 0
551             self.show_progress(predictions, len(data))
552             print("Current Accuracy: ", good_predictions/float(predictions),
                ↪  '\n')
553             predictions, test_time = (predictions + 1), (test_time +
                ↪  comp_time)
554
555         # Uncomment to write samples to file
556         # self.write_samples_to_file(neg, pos)
557
558         new_time = test_time / float(length_data)
559         print("Average computation time: "+str(new_time)+"s")
560         print("Test total time: "+str(test_time)+"s")
561         self.isTest = False
562         return good_predictions/float(len(data))
563
564     # This function writes the predicted values to two files:
565     # negative and positive.
566     # This is needed for the thresholdTool script
567     # to find the best threshold given these two files.
568     #
569     # @param neg The list of negative predictions
570     # @param pos The list of positive predictions
571     def write_samples_to_file(self, neg, pos):
```

```
572             with open('./negative.txt', 'w') as f:
573                 for item in neg:
574                     f.write(str(item)+"\n")
575                 f.close()
576             with open('./positive.txt', 'w') as f:
577                 for item in pos:
578                     f.write(str(item)+"\n")
579                 f.close()
580 
581         # Visualize the prediction results
582         #
583         # @param prediction The prediction result label
584         # @param similarity The similarity value for that prediction
585         def print_prediction(self, prediction, similarity):
586             if prediction == 1:
587                 print("The query image is present on the reference image with
                    ↪  similarity: "+str(similarity)+"!\n")
588             else: print("The query image is NOT present on the reference image
                ↪  with similarity: "+str(similarity)+"!\n")
589 
590         # This function performs a check on the extension
591         # of a file. If the extension is not supported, then
592         # the code will exit!
593         #
594         # @param pathQ The path to the query image
595         # @param pathR The path to the reference image
596         # @return True if extensions are correct else False
597         def check_file_extension(self, pathQ="", pathR=""):
598             isExtQ, isExtR = False, False
599             for ext in ['.jpg', '.jpeg', '.png', '.JPG', '.JPEG', '.PNG']:
600                 if ext in pathQ: isExtQ = True
601                 if ext in pathR: isExtR = True
602             if not isExtQ or not isExtR:
```

```
603             print("Not supported file format!")
604             return False
605         else: return True
606 
607     # Get prediction based on the result of performing
608     # Image Verification with different methods
609     #
610     # @param pathQ The path to a query image
611     # @param pathR The path to a reference image
612     # @return prediction The prediction label result
613     # @return similarity The similarity value for the current prediction
614     # @return The time that the prediction took to complete
615     def predict(self, pathQ, pathR):
616         if not self.check_file_extension(pathQ, pathR): return
617         if not self.check_file([pathQ, pathR]): return
618         self.comp_time = time.time()  # Start counting time
619         similarity = self.get_max_similarity(pathQ, pathR)
620         prediction = 1 if similarity >= self.threshold else 0
621         self.print_prediction(prediction, similarity)
622         return prediction, similarity, self.get_time()
623 
624     # Print a progress bar in order
625     # to visualize status
626     #
627     # @param num_of_pred The current number of predictions completed
628     # @param total_pred The total number of predictions
629     def show_progress(self, num_of_pred, total_pred):
630         current_status = int(round(50 * num_of_pred / float(total_pred)))
631         percentage = round(100.0 * num_of_pred / float(total_pred), 1)
632         print("[" + '=' * current_status+ "-" * (50 - current_status) + "]
            ↪  "+str(percentage) + "%", end="\n", flush=True)
633 
634     # Get the finish time of a process
```

```
635         #
636         # @return new_time The finish time of the process
637         def get_time(self):
638             new_time = int(time.time() - self.comp_time)
639             print("======================")
640             print("Predict Time: "+str(new_time)+"s")
641             return new_time
642 
643         # This is the constructor for the class
644         # It initializes all class variables
645         #
646         # @param extr_path The path to an extraction file
647         # @param extract A boolean value specifying whether to extract values or
        ↪  not
648         # @param extr_dataset A list of [Q R L] destinations
649         # @param plot A boolean value specifying whether to plot data or not
650         # @param threshold A threshold value
651         # @param print_mask A boolean value specifying whether to show a
        ↪  likelihood map or not
652         # @param match_method A string specifying the matching method (tm or pm)
653         # @param model The CNN used to extract features
654         # @param measure The Similarity Measure that will be used
655         def __init__(self, extr_path, extract, extr_dataset, plot,
656                      threshold, print_mask, match_method, surf, model, measure):
657             self.isTest = False
658             self.measure, model = measure[0], model[0]
659             self.pick_model(model)
660             self.threshold, self.predict_method, self.match_method = threshold,
            ↪  "threshold", match_method
661             self.extract, self.knn, self.print_mask, self.surf = extract, None,
            ↪  print_mask, surf
662             print("Initializing tool with: threshold = " + str(threshold) + " |
            ↪  predict_method = " +
```

```
663                 self.predict_method + " | match_method = " + match_method + " |
                    ↪  SURF = " + str(surf) +
664                 " | model = " + str(model) + " | measure = " + str(self.measure))
665
666             # Try to generate an extraction file
667             if extract:
668                 print("Trying to create a new extraction file...")
669                 if len(extr_dataset) == 0: print("Please provide a valid
                    ↪  extraction dataset!")
670                 else:
671                     print("New extraction file -> " + extr_path + "!")
672                     print("Extracting...")
673                     self.create_extr_file(extr_dataset, extr_path)
674             # Try to plot data
675             if plot:
676                 pos_x, neg_x = self.load_for_plot(extr_path)
677                 self.plot_data(pos_x, neg_x)
678
679     # The main method of the python file
680     # It gets the received arguments from the parser
681     # and creates an Image Verification Tool object
682     if __name__ == '__main__':
683         # Convert string argument to boolean
684         #
685         # @param arg The argument from the parser
686         # @return True/False or raise exception depending on argument
687         def str_converter(arg):
688             ex, arg_l = argparse.ArgumentTypeError('Boolean value True or False
                    ↪  expected, ('+str(arg)+') given!'), arg.lower()
689             if type(arg) is bool: return arg
690             if arg_l not in ['true', 'false']: raise ex
691             else: return True if arg_l == 'true' else False
692
```

```
693     # Initialize Parser Arguments
694     parser =
        ↪  argparse.ArgumentParser(formatter_class=argparse.RawTextHelpFormatter)
695     parser.add_argument('-ep', '--extr_path', action='store',
        ↪  dest='extr_path', help="Specify extraction file path",
        ↪  default="new_extracted_dataset.txt")
696     parser.add_argument('-e', '--extract', help="Toggle extraction mode True
        ↪  or False. For False just omit command.", type=str_converter,
        ↪  nargs='?', default=False, const=True)
697     parser.add_argument('-ed', '--extr_dataset', dest='extr_dataset',
        ↪  help="Specify an extraction dataset with 3 arguements:\
698
    ↪  \n -QUERY_IMAGES DESTINATION FOLDER\
699
    ↪  \n -REFERENCE_IMAGES DESTINATION FOLDER\
700
    ↪  \n -LABEL FILE DESTINATION", default=[], nargs=3)
701     parser.add_argument('--plot', dest='plot', help="Toggle the plot option
        ↪  True or False. Omit command for False", type=str_converter, nargs='?',
        ↪  default=False, const=True)
702     parser.add_argument('-thr', '--threshold', action='store',
        ↪  dest='threshold', help="Give a threshold value i.e. 0.50", type=float,
        ↪  default=0.585)
703     parser.add_argument('-mm', '--match_method', action='store',
        ↪  dest="match_method", help="Specify a matching method\
704
    ↪  to be used from [\
705
    ↪  'tm' -> for template matching, 'pm -> patch matching']. If \
706
    ↪  None, default is template matching (tm)\
707
    ↪  method", type=str,
```

```
708
709     parser.add_argument('-pm', '--print_mask', help="Toggle print mask option
        ↪ True or False. For False just omit command.\
710                                                     \nThis option prints only
    ↪ images that are similar and only when predicting!",
711                                                     type=str_converter,
                                                        ↪ nargs='?',
                                                        ↪ default=False,
                                                        ↪ const=True)
712     parser.add_argument('-test','--test_dataset', help="Test on a specified
        ↪ dataset of 3 arguements:\
713                                             \n -QUERY_IMAGES DESTINATION
    ↪ FOLDER\
714                                             \n -REFERENCE_IMAGES DESTINATION
    ↪ FOLDER\
715                                             \n -LABEL FILE DESTINATION",
    ↪ default=[], nargs=3)
716     parser.add_argument('-p', '--predict', help="Predict on a new pair of
        ↪ Query and Reference Images. \
717                                                      Specify the path of a
    ↪ QUERY and a REFERENCE image.", default=[],
718                         nargs=2)
719     parser.add_argument('-s', '--surf', help="Toggle surf comparison mode True
        ↪ or False. For False just omit command.",
720                         type=str_converter, nargs='?', default=False,
                            ↪ const=True)
721     parser.add_argument('--model', choices=['resnet50', 'resnet101',
        ↪ 'resnet152', 'vgg19', 'inception'], nargs=1, default=['resnet50'])
722     parser.add_argument('--measure', choices=['cc', 'ncc'], nargs=1,
        ↪ default=['cc'])
723     # Get a dictionary of parser arguments
724     args = parser.parse_args()
```

```
    args_dict = vars(args)

    # Remove the exceptions
    filtered_dict = dict((k, args_dict[k]) for k in args_dict.keys() if k not
    ↪  in ["test_dataset", "predict"])

    # Create class instance with specified arguments
    class_instance = ImageVerificationTool(**filtered_dict)

    # Perform class functions given specified arguments
    if args.predict:
        print("Predicting...")
        class_instance.predict(args.predict[0], args.predict[1])
    if args.test_dataset:
        class_instance.test_tool(args.test_dataset)
```

D.3 thresholdTool.py

This is the file that contains the script which chooses the best threshold. It can be used to find the best threshold and accuracy for different methods, given two lists (negative and positive) of values.

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# This is the class that picks the
# best accuracy and threshold for specifically
# provided data from two files, containing respectively
# positive and negative similarity value samples
#
# @author Preslav Kisyov
# @version 1.1
class BestThresholdTool:

    # This function appends the lines from a file
    # to a specified list
    #
    # @param lines The lines obtained from a file
    # @param l The list that will hold the passed lines
    def append_to_list(self, lines, l):
        for line in lines:
            n_line = line.split("\n")[0]
            l.append(float(n_line))

    # This function gets the samples from two files
    # into two separate lists for both positive
    # and negative samples
    #
```

```
29      # @param neg_path The path to the file with negative samples
30      # @param pos_path The path to the file with positive samples
31      # @return neg The newly created list with negative samples
32      # @return pos The newly created list with positive samples
33      def get_samples(self, neg_path, pos_path):
34          neg, pos = [], []
35          for path in [neg_path, pos_path]:
36              with open(path, "r") as f:
37                  lines = f.readlines()
38                  if len(neg) > 0: self.append_to_list(lines, pos)
39                  else: self.append_to_list(lines, neg)
40              f.close()
41          return neg, pos
42
43      # This function loops through different thresholds
44      # over two sample lists in order to find the best
45      # accuracy and threshold for the data
46      #
47      # @return best_thr The best threshold recorded
48      # @return acc The best accuracy recorded
49      # @return neg The list with negative samples only
50      # @return pos The list with positive samples only
51      def get_best_thr(self):
52          neg, pos = self.get_samples(self.neg_path, self.pos_path)
53          best_thr, acc = 0, 0
54
55          if len(neg) == 0 or len(pos) == 0:
56              print("No values have been found!")
57              exit()
58          m_neg, m_pos = max(neg), max(pos)
59          max_val = m_neg if m_neg >= m_pos else m_pos
60          print("The max value recorded is: "+str(max_val))
61          # Get the best threshold/accuracy by trying
```

```
62          if self.reverse:
63              # Get the threshold for measures that where lower values are
                ↪ better
64              for thr in np.arange(0.0, max_val, self.ints):
65                  acc_samples = 0
66                  for n in neg:
67                      if n >= thr: acc_samples += 1
68                  for p in pos:
69                      if p < thr: acc_samples += 1
70                  if acc_samples >= acc: best_thr, acc = thr, acc_samples
71          else:
72              # Get the threshold for measures that where higher values are
                ↪ better
73              for thr in np.arange(0.0, max_val, self.ints):
74                  acc_samples = 0
75                  for n in neg:
76                      if n < thr: acc_samples += 1
77                  for p in pos:
78                      if p >= thr: acc_samples += 1
79                  if acc_samples >= acc: best_thr, acc = thr, acc_samples
80
81          return best_thr, acc, neg, pos
82
83      # This function separates the samples into
84      # True and False positive/negative so they can
85      # be visualised better
86      #
87      # @param pos The positive samples
88      # @param neg The negative samples
89      # @param best_thr The best threshold found for the data
90      # @return false_pos The False positive samples
91      # @return false_neg The False negative samples
92      # @return true_pos The True positive samples
```

```
93      # @return true_neg The True negative samples
94      def get_separated_samples(self, pos, neg, best_thr):
95          false_pos, false_neg = [], []
96          true_pos, true_neg = [], []
97          # Count false positive samples
98          for p in pos:
99              if self.reverse:
100                 if p > best_thr: false_pos.append(p)
101                 else: true_pos.append(p)
102             else:
103                 if p < best_thr: false_pos.append(p)
104                 else: true_pos.append(p)
105         # Count false negative samples
106         for n in neg:
107             if self.reverse:
108                 if n < best_thr: false_neg.append(n)
109                 else: true_neg.append(n)
110             else:
111                 if n > best_thr: false_neg.append(n)
112                 else: true_neg.append(n)
113         return false_pos, false_neg, true_pos, true_neg
114
115     # This function plots the data of two
116     # sample lists as well as the recorded
117     # accuracy and best threshold
118     def plot(self):
119         print("Finding best threshold...")
120         best_thr, acc, neg, pos = self.get_best_thr()
121         print("Plotting...")
122         sum_len = len(neg) + len(pos)
123         plt.figure(figsize=(8, 5))
124         false_pos, false_neg, true_pos, true_neg =
            ↪ self.get_separated_samples(pos, neg, best_thr)
```

```
125
126         # Get linespace for all samples
127         x0 = np.linspace(0, len(neg), len(true_neg))
128         x1 = np.linspace(len(neg), sum_len, len(true_pos))
129         x2 = np.linspace(0, len(neg), len(false_neg))
130         x3 = np.linspace(len(neg), sum_len, len(false_pos))
131
132         # plot all samples
133         plt.plot(x0, true_neg, 'o', color='blue')
134         plt.plot(x1, true_pos, 'o', color='red')
135         plt.plot(x2, false_neg, 'o', color='aqua')
136         plt.plot(x3, false_pos, 'o', color='orange')
137
138         print("Best accuracy: ", acc / sum_len, " Best Threshold: ", best_thr)
139         fontP = FontProperties()
140         fontP.set_size('small')
141         x_coordinates = [0, sum_len]
142         y_coordinates = [best_thr, best_thr]
143         print("Saving plot to "+self.save_file)
144         plt.plot(x_coordinates, y_coordinates, color="green")
145         plt.legend(['True Negative Samples:
            ↪  '+str(len(true_neg))+'/'+str(len(neg)), 'True Positive Samples:
            ↪  '+str(len(true_pos))+'/'+str(len(pos)),
146                     'False Negative Samples:
                        ↪  '+str(len(false_neg))+'/'+str(len(neg)), 'False
                        ↪  Positive Samples:
                        ↪  '+str(len(false_pos))+'/'+str(len(pos)),
147                     'Threshold: ' + str(float(round(best_thr, 3)))],
148                    borderaxespad=0., bbox_to_anchor=(1.04, 1))
149         plt.title("Acc: " + str(float(round(acc / sum_len, 3))))
150         plt.xlabel("Samples")
151         plt.ylabel("Similarity Values")
152         plt.grid(True)
```

```
153             plt.tight_layout()
154             plt.savefig(self.save_file)
155             plt.show()
156
157         # The constructor of the class
158         # It manages arguments as well as
159         # calls other class functions
160         #
161         # @parameter dataset A list of arguments
162         # @param reverse A toggle for the Reverse Mode
163         # @param ints The intensity for choosing a threshold
164         def __init__(self, dataset, reverse, ints):
165             self.reverse, self.ints = reverse, abs(ints)
166             if len(dataset) == 0: exit()
167             elif len(dataset) == 2:
168                 self.neg_path, self.pos_path = dataset
169                 self.save_file = "default.jpg"
170             else: self.neg_path, self.pos_path, self.save_file = dataset
171             print("Initialising class with: | REVERSE: " + str(self.reverse) + " |
                ↪  NEGATIVE FILE: " + self.neg_path +
172                   " \n| POSITIVE FILE: " + self.pos_path + " | SAVEFILE LOCATION:
                      ↪  " + self.save_file + " | INTENSITY: " + str(abs(ints)))
173             if '.txt' not in self.neg_path:
174                 self.neg_path = self.neg_path + ".txt"
175             if '.txt' not in self.pos_path:
176                 self.pos_path = self.pos_path + ".txt"
177             if ('.jpg' or '.png' or 'jpeg') not in self.save_file:
178                 self.save_file = self.save_file+".jpg"
179
180             self.plot()
181
182     # This is the main method of the python file
183     # that gets the received arguments from a
```

```
184 # parser and creates an object of the tool
185 if __name__ == '__main__':
186     # Convert string argument to boolean
187     # This is the same method used in the verificationTool.py
188     #
189     # @param arg The argument from the parser
190     # @return True/False or raise exception depending on argument
191     def str_converter(arg):
192         ex, arg_l = argparse.ArgumentTypeError(
193             'Boolean value True or False expected, (' + str(arg) + ')
                ↪  given!'), arg.lower()
194         if type(arg) is bool: return arg
195         if arg_l not in ['true', 'false']: raise ex
196         else: return True if arg_l == 'true' else False
197
198     parser =
        ↪  argparse.ArgumentParser(formatter_class=argparse.RawTextHelpFormatter)
199     parser.add_argument('-a', '--argument', dest='dataset', help="Specify the
        ↪  file locations in a dataset with 3 arguements:\
200
    ↪  \n -NEGATIVE SAMPLES DESTINATION FILE\
201
    ↪  \n -POSITIVE SAMPLES DESTINATION FILE\
202
    ↪  \n -NEW IMAGE SAVE FILE DESTINATION", default=[], nargs='*')
203     parser.add_argument('-r', '--reverse', help="Toggle reverse mode. This
        ↪  should be used when trying to get the best threshold where the lower a
        ↪  value is, the better.",
204                         type=str_converter, nargs='?', default=False,
                            ↪  const=True)
205     parser.add_argument('-ints', '--intensity', action='store', dest='ints',
```

```
206                         help="Specify the intensity of which the threshold
                            ↪  will be chosen by. Always above 0!", type=float,
                            ↪  default=0.000001)
207     # Get a dictionary of parser arguments
208     args = parser.parse_args()
209     args_dict = vars(args)
210     obj = BestThresholdTool(**args_dict)
```