universidade de aveiro

Imageable

Reimagine your past.

10th May 2021

Project in Informatics Curricular Unit | Group 03

FUNCTIONAL REQUIREMENTS

FUNCTIONAL REQUIREMENTS

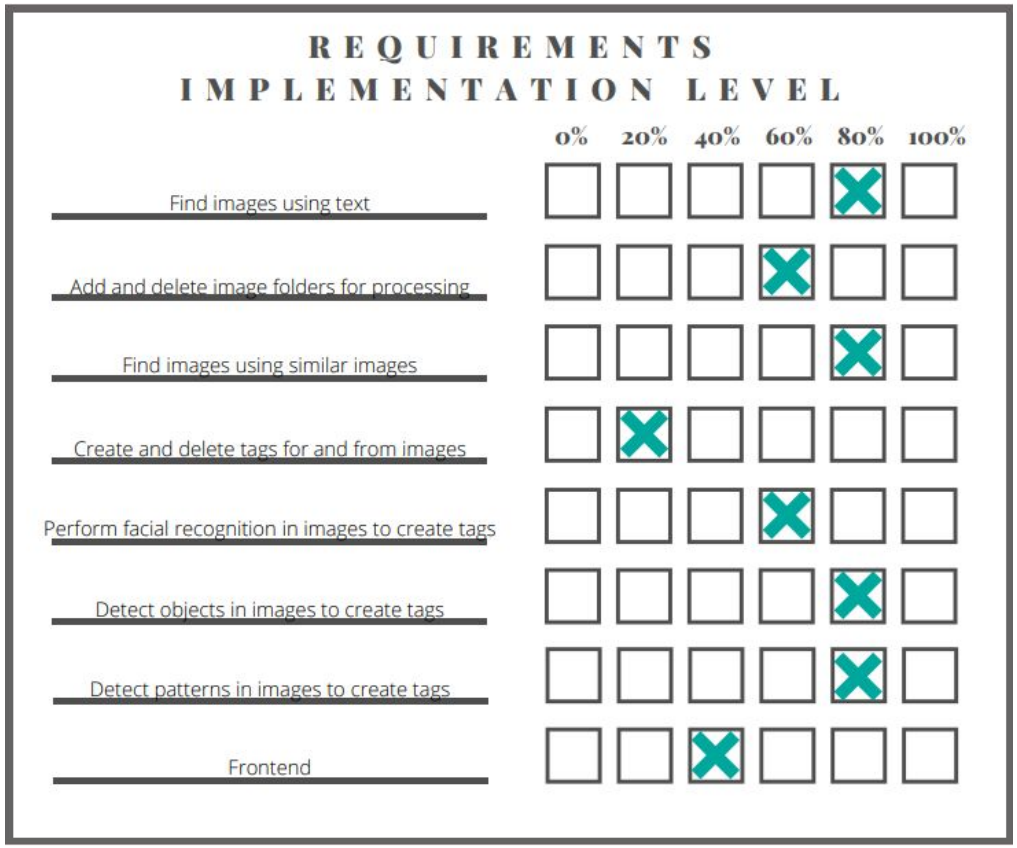

IMAGE OBJECT EXTRACTION

Extracts a list of objects present in an image as a source of its tags

IMAGE OBJECT EXTRACTION

work done
85%

- Obtainment of the YOLOv5s model model pre-trained with the COCO dataset
- Usage of the model to obtain a list of the objects and confidences
- Creation or retrieval of the entity relative to the object in the DB
- Association of that entity to the image

IMAGE OBJECT EXTRACTION

future work
21/05

- Usage of different models and combination of their accuracies
- Allow the user to define accuracy threshold for searches

FACIAL RECOGNITION

FACIAL RECOGNITION

work done
60%

- Detection of bounding boxes' coordinates of the faces
- Save of a low-resolution thumbnail of the faces insertion of its path to the DB
- Validation of whether the face is known to the system
- Creation or association of the person in the DB to the image
- If the person is unknown, assignment of a random string to the name properties

FACIAL RECOGNITION

future work
21/05

- Integration of the module in the frontend
- Allow the user to alter the name of a person
- Allow for searches using person names

SURROUNDINGS RECOGNITION

Detects the scene present in the image through its patterns to generate a tag

SURROUNDINGS EXTRACTION

work done
90%

- Load of the image and conversion to the right format for analysis
- Analysis of the image and extraction of the surrounding features
- Compararation of the image features with dataset
- Acceptance of a result, it being the first results found with score higher than 0.6

SURROUNDINGS RECOGNITION

future work 15/05

- Allow the user decide which image features score they want to accept

OPTICAL CHARACTER RECOGNITION

Detects characters present in the image through to generate tags

OPTICAL CHARACTER RECOGNITION

work done
88%

- Image processing for better OCR results
- Cut the image around what it thinks that is text
- Application of the algorithm on processed and cutted images
- Application of NLP to filter the results
- Conversion of filtered text into tags

OPTICAL CHARACTER RECOGNITION

future work 15/05

- Improvement of the image processing to achieve a better recognition of characters
- Improvement of the performance

EXIF

Extracts metadata from images to show it to a user

EXIF

work done
100%

- Verification whether an image has EXIF or not
- In images with EXIF data, extraction of datetime, longitude, latitude, width and height information
- In images with no EXIF data, extraction of width and height information

NATURAL LANGUAGE PROCESSING

Handles the tokens of a query, allowing the search for an image through a query.

NATURAL LANGUAGE PROCESSING

work done
90%

- Tokenization of a string, transforming it into an array of tokens
- Removal of punctuation and stopword tokens
- Stemming of each token, recurring to 2 different stemmers
- Obtaining of the Part-Of-Speech tag for each token
- Transformation of the Part-Of-Speech tag into the Part-Of-Speech parameter for the lemmatization function
- Lemmatization of each token

NATURAL LANGUAGE PROCESSING

future work
28/05

- Improve the performance especially on complex queries
- Transformation of the query to all lower case characters
- Use of synonyms for each token to expand the search results

ELASTIC SEARCH

Allows to search for tags faster

ELASTIC SEARCH

work done
90%

- Retrieval of images by given tags with its score
- Order results by score

FRONTEND

FRONTEND

work done
50%

- Development of the skeleton of the project's interface, with a homepage, three buttons and a search bar

- Development of modals to rename people, upload new folder, select an image to search for similar images and more info of a specific image

FRONTEND

future work 07/07

- Improvement of the overall appearance of the frontend
- Change image that appears in each image modal
- Addition of an option to delete source folders
- Change the upload folder and image method
- Modification of the appearance of the modal to rename people
- Addition of buttons to rename, add or delete tags
- Filtering of information about each type of tag (automatic, manual or name-of-folder type)
- Addition of slider to filter tags by confidence threshold
- Addition of a dashboard page with some statistics

TESTS

```
----------------------------------------------------------------------
Ran 23 tests in 59.010s

OK
Destroying test database for alias 'default'...
```

TESTS

VGG

```
class VGGTestCase(TestCase):

    def setUp(self):
        print("\n\\|/Testing VGG")

    def test_vgg_extract(self):
        vgg = VGGNet()
        result = vgg.vgg_extract_feat(dir_path + "/face.jpg")
        self.assertEquals(result is None, False)
```

TESTS

Utils

```
class UtilsTestCase(TestCase):

    def setUp(self):
        print("\n\\|/Testing utils")

    def test_random_num(self):
        self.assertTrue(1 <= getRandomNumber() <= (1 << 63))
    def test_images_in_uri(self):
        #dirsAndFiles = {}  # key - dir name, value - list of files (imgs)
        dirsAndFiles = getImagesPerUri(dir_path)
        self.assertEquals(dirsAndFiles[dir_path],["face.jpg"])
```

TESTS

Processing

```
img_path = os.path.dirname(os.path.realpath(__file__)) + "/face.jpg"
class ProcessingTestCase(TestCase):

    def setUp(self):
        print("\n\\|/Testing Processing")

    def test_filter(self):
        result = filterSentence("hello world I to kms good dead hnb lll notaword gfffffffffffffffffffff")
        expected = ['hello', 'world', 'good', 'dead', 'notaword']
        self.assertEquals(expected,result)

    def test_hash(self):
        img = cv2.imread(img_path)
        hashcode = dhash(img)
        expected = 7288338847964571648
        self.assertEquals(expected,hashcode)

    def test_exif(self):
        exif = getExif(img_path)
        expected = {'height': 1080, 'width': 2160}
        self.assertEquals(expected,exif)

def test_thumbnail(self):
    read_image = cv2.imread(img_path)
    hash = dhash(read_image)
    thumbnailPath = generateThumbnail(img_path, hash)
    self.assertTrue(cv2.imread(thumbnailPath) is not None)
    os.remove(thumbnailPath)
    self.assertTrue(cv2.imread(thumbnailPath) is None)
```

TESTS

Object

```
img_path = os.path.dirname(os.path.realpath(__file__)) + "/face.jpg"
extractor = ObjectExtract()
class OETestCase(TestCase):

    def setUp(self):
        print("\n\\|/Testing Object Detection and Extraction")

    def test_getObj(self):
        objs = extractor.get_objects(img_path)
        self.assertTrue("person" in objs["name"][0])
```

TESTS

NLP

```
class nlpTestCase(TestCase):

    def setUp(self):
        print("\n\\|/Testing NLP Filter Search")

    def test_tokenize(self):
        tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
        self.assertTrue(isinstance(tokens, list))
    def test_filterPunct(self):
        tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
        tokens = filterPunctuation(tokens)
        self.assertTrue("." not in tokens)
        self.assertTrue("," not in tokens)
        self.assertTrue("." not in tokens)
    def test_filterStopWords(self):
        tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
        tokens = filterStopWords(tokens)
        self.assertTrue("is" not in tokens)
        self.assertTrue("just" not in tokens)
        self.assertTrue("the" not in tokens)
```

TESTS

NLP

```
def test_filterDictwords(self):
    tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
    tokens = filteredDictWords(tokens)
    self.assertTrue("I" not in tokens)
    self.assertTrue("." not in tokens)
    self.assertTrue("," not in tokens)
def test_stemming(self):
    tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
    tokens = stemmingMethod(tokens)
    self.assertTrue("loving" not in tokens)
    self.assertTrue("love" in tokens)
def test_posTag(self):
    tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
    tokens = stemmingMethod(tokens)
    tokens = posTagging(tokens)
    self.assertTrue(tokens)
def test_transformTagging(self):
    tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
    tokens = stemmingMethod(tokens)
    tokens = posTagging(tokens)
    tokens = transformTagging(tokens)
    self.assertTrue(tokens)
```

TESTS

NLP

```
def test_lemmatization(self):
    tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
    tokens = stemmingMethod(tokens)
    tokens = posTagging(tokens)
    tokens = transformTagging(tokens)
    tokens = lemmatizationMethod(tokens)
    self.assertTrue("is" not in tokens)
    self.assertTrue("be" in tokens)
    self.assertTrue("beautiful" not in tokens)
    self.assertTrue("beauty" in tokens)
def test_all(self):
    tokens = tokenizeText("loving someone is something beautiful, just like the nature. I love the world.")
    tokens = filterPunctuation(tokens)
    tokens = filterStopWords(tokens)
    tokens = stemmingMethod(tokens)
    tokens = posTagging(tokens)
    tokens = transformTagging(tokens)
    tokens = lemmatizationMethod(tokens)
    self.assertTrue(tokens == processQuery("loving someone is something beautiful, just like the nature. I lov
```

TESTS

File system manager

```
dir_path = os.path.dirname(os.path.realpath(__file__))
filesistem = SimpleFileSystemManager()

class FSTestCase(TestCase):

    def setUp(self):
        print("\n\\|/Testing File System Manager")
        filesistem.addFullPathUri(dir_path, range(len(re.split("[\\\/]+", dir_path))))
    def test_exists(self):
        self.assertTrue(filesistem.exist(dir_path))
    def test_expand(self):
        filesistem.expandUri(dir_path,"expanding",8)
        self.assertTrue(filesistem.exist(dir_path+"/expanding"))
    def test_get_lastN(self):
        node = filesistem.getLastNode(dir_path)
        self.assertEquals(str(node), "tests")
    def test_get_splitgetroot(self):
        folders, root = filesistem.__splitUriAndGetRoot__(dir_path)
        self.assertTrue(folders)
        self.assertTrue(root in folders)
    def test_get_all(self):
        uris = filesistem.getAllUris()
        self.assertTrue(uris)
```

TESTS

Face recognition

```
dir_path = os.path.dirname(os.path.realpath(__file__))

class FaceRecogTestCase(TestCase):

    def setUp(self):
        print("\n\\|/Testing Face Recognition")

    def test_get_box(self):
        faceRecog = FaceRecognition()
        result = faceRecog.getFaceBoxes(dir_path + "/face.jpg")
        self.assertEqual(result[0] is None, False)
        self.assertEqual(result[1] is None, False)
```

FUNCTIONAL PROTOTYPE

ULTIMATE GOALS

ULTIMATE GOALS

- Implementation of a neural network to recognize animal breeds, developed in the TAA curricular unit
- Implementation of the results by returning a JSON file with the images' path or a folder containing those images
- Improvement of the overall application performance index
- Creation of an installation application package
- Refactor the code (using Sonarqube)
- Use of a dynamic number of threads, depending on the user's memory at the time the application is started