U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

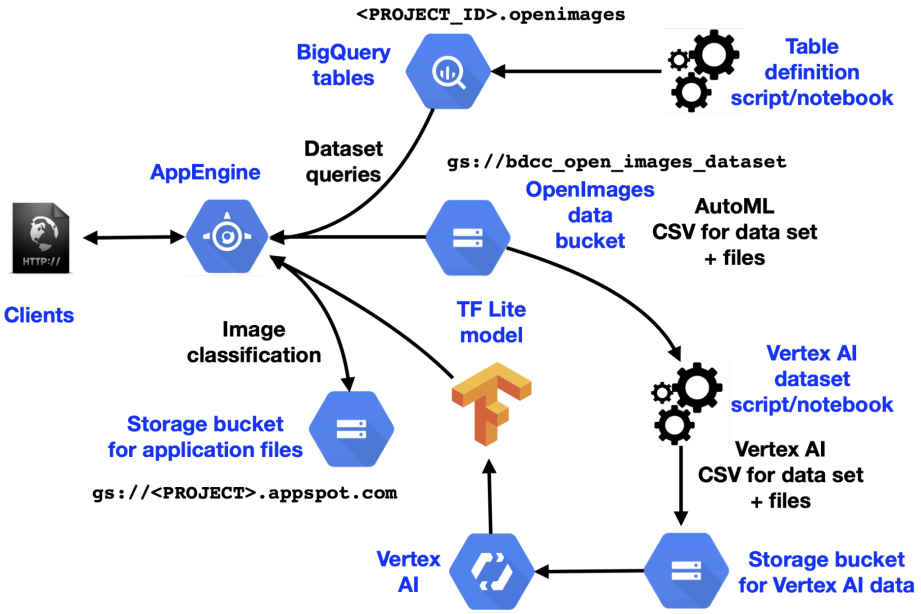
Big Data and Cloud Computing - Open Images Project

Google Cloud project - project-up202310061-bdcc / ID: 142023702067
App Engine - Application URL: https://project-up202310061-bdcc.ew.r.appspot.com/

Davi Barrel Santos (up202310061)
Porto, 2024

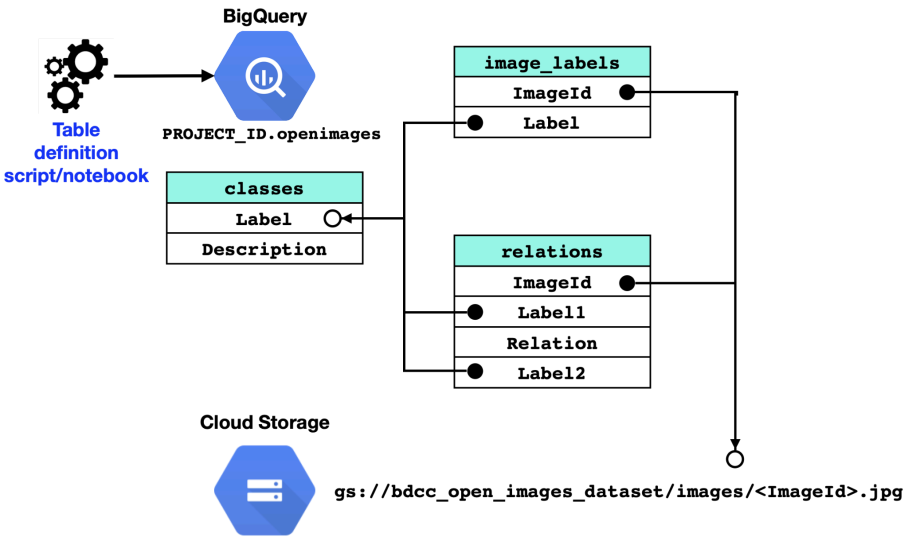
1. Application Infrastructure

This application makes use of Flask and Jinja to expose labeled images of a public database and their relation in a web interface. It also implements a TensorFlow classification Machine Learning model that classifies images according to their labels. The model was trained considering the following labels: Cat, Airplane, Wheelchair, Cart, Motorcycle, Rifle, Broccoli, Otter, Wine glass, Hamburger.



2. Reference Data Schema

The image database (https://storage.googleapis.com/bdcc_open_images_dataset/images/) is a Google Cloud public bucket that stores images by ids (*e.g. 0041c772f8b9aef6.jpg)*. The retrieval of images by the application uses a BigQuery dataset to store the schema about the images reference data (image id, image label, image relations).



3. App engine - Application endpoints

The endpoints configured for the application, their functionalities and its implementation from query reference data perspective are:

- *classes*:
    - Retrieve the count of images for all the existent classes in the database;
    - endpoint: https://project-up202310061-bdcc.ew.r.appspot.com/classes?
    - To query the reference data:

```
'''
   Select Description, COUNT(*) AS NumImages
   FROM `bdcc24project.openimages.image_labels`
   JOIN `bdcc24project.openimages.classes` USING(Label)
   GROUP BY Description
   ORDER BY Description
'''
```

This query retrieves the count of images by using Label column to join information from image_labels and classes tables from BigQuery.

- *image_search*:
    - Exposes N images of an specific class that is inputted by the end-user;
    - Endpoint: https://project-up202310061-bdcc.ew.r.appspot.com/image_search?description=Cat&image_limit=10
    - To query the reference data:

```
'''
   SELECT description, imageId from
   `project-up202310061-bdcc.openimages.classes`
   join `project-up202310061-bdcc.openimages.image_labels` using (Label)
   where description = '{img_description}'
   limit {limit}
'''
```

This query retrieves the image description (class) and its reference data (imageId) considering the inputted class and the limit of output, also defined by the end-user.

- *relations*:
    - Count and list the existent relations inside the images;;
    - Endpoint: https://project-up202310061-bdcc.ew.r.appspot.com/relations? ;
    - To query the reference data:

```
'''
   SELECT Relation, count(Relation)
   from `project-up202310061-bdcc.openimages.relations`
   group by Relation;
'''
```

Retrieves the relation and the count of images that have that relation;

- *relations_search*:
    - Looks for images that contains the inputted relations considering the image classes inputted and also the number of images requested;
    - Endpoint: https://project-up202310061-bdcc.ew.r.appspot.com/relation_search?class1=%25&relation=plays&class2=%25&image_limit=10
    - To query the reference data:

```
'''
    SELECT rlt.ImageId,
    dsc1.description as description_image1,
    rlt.relation,
    dsc2.description as description_image2,
    from `project-up202310061-bdcc.openimages.relations` rlt
    inner join `project-up202310061-bdcc.openimages.classes` dsc1
    on rlt.Label1 = dsc1.Label
    inner join `project-up202310061-bdcc.openimages.classes` dsc2
    on rlt.Label2  = dsc2.Label
    where rlt.relation like '{relation_parameter}'
    and dsc1.description like '{desc_image1}'
    and dsc2.description like '{desc_image2}'
    limit {limit};
'''
```

This query retrieves the reference data (ImageId) of images that contains the classes inputted and the relation inputted. The output size is limited by the limit parameter. To do so, it was needed to consult the *classes* table twice to get the description of images that matches the same ImageId.

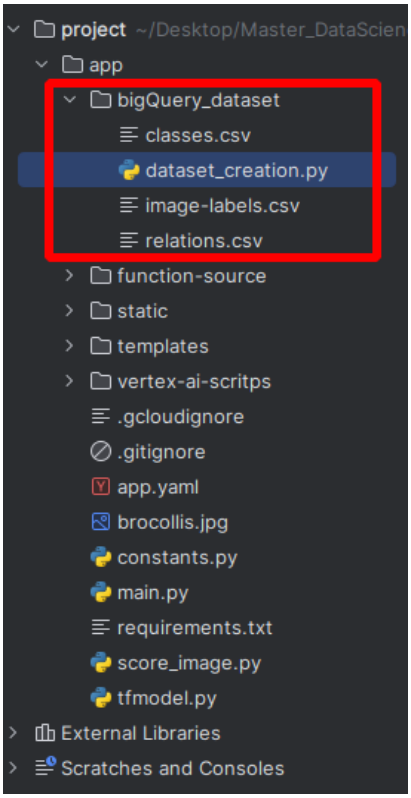
- *image_info*:
    - This endpoint exposes the image classes and the relations on it given an certain image id;
    - Endpoint:https://project-up202310061-bdcc.ew.r.appspot.com/image_info?image_id=0041c772f8b9aef6
    - To query the reference data:

```
'''
   SELECT
     rlt.ImageId,
     STRING_AGG(DISTINCT CONCAT(cls1.description,' ', rlt.relation,' ',
cls2.description)) AS relation,
     STRING_AGG(DISTINCT all_desc.all_descriptions) AS descriptions
   FROM
     `project-up202310061-bdcc.openimages.relations` rlt
   INNER JOIN
     `project-up202310061-bdcc.openimages.classes` cls1
   ON
     rlt.Label1 = cls1.Label
   INNER JOIN
     `project-up202310061-bdcc.openimages.classes` cls2
   ON
     rlt.Label2 = cls2.Label
```

```
  INNER JOIN (
    SELECT
      cls.Description AS all_descriptions,
      img.ImageId AS ImgId_1
    FROM
      `project-up202310061-bdcc.openimages.image_labels` img
    JOIN
      `project-up202310061-bdcc.openimages.classes` cls
    USING
      (Label)) AS all_desc
  ON
    rlt.ImageId = all_desc.ImgId_1
  WHERE  rlt.ImageId = '{img_id}'
  GROUP BY
  rlt.ImageId
'''
```

This query accesses the *relations* table to retrieve the relations between classes that exist in the image, then access the *classes* table twice to retrieve the images descriptions for each image in the relation. Then it goes to the *image_labels* to retrieve all the possible descriptions for that image.

4. BigQuery data set schema creation:



The BigQuery schema was defined using Python functions that make use of Google Cloud BigQuery Client APIs. The script consists in the functions:

```
def instantiate_table(bq_client: google.cloud.bigquery.Client, table_name:
str):
```

- Instantiate a table in the bigquery project by having a table name;

```
def create_classes_table(bq_client: google.cloud.bigquery.Client, df:
pd.DataFrame):
```

- creates the classes table and loads data on it by passing an Dataframe (classes.csv);

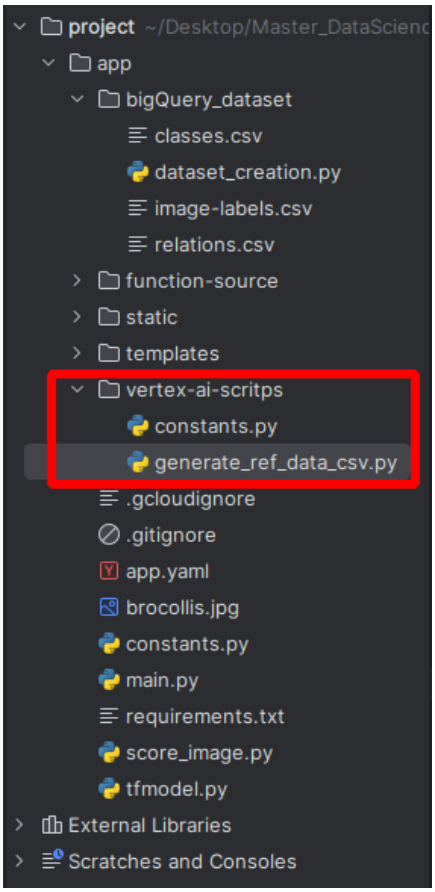
```
def create_image_labels_table(bq_client: google.cloud.bigquery.Client, df:
pd.DataFrame):
```

- creates the image_label table and loads data on it by passing an Dataframe (image_label.csv);

```
def create_relations_table(bq_client: google.cloud.bigquery.Client, df:
pd.DataFrame):
```

- creates the relations table and loads data on it by passing an Dataframe (relations.csv);

5. TensorFlow dataset preparation:



To prepare the dataset used in the TensorFlow classification model, the constants.py file stores the constants that are used in the class. The *RefDataVertexGenerator()* class was created and defined in the generate_ref_data_csv.py file.

```
class RefDataVertexGenerator():
   def __init__(self):
       self.query_result = None
       self.image_uris_df = None
       self.bigquery_client = bigquery.Client(project=PROJECT_ID)
       self.storage_client = STORAGE_CLIENT =
storage.Client(project=PROJECT_ID)
       self.vertex_bucket = STORAGE_CLIENT.bucket(VERTEX_AI_BUCKET_NAME)
       self.labels = None
```

The labels are set as attribute by the method:

```
def set_labels(self,labels_list:list):
   self.labels = labels_list
```

The private methods are:

```
def __query_image_ref(self, labels_query: list):
```

- *__query_image_ref:* Queries in the BigQuery Dataset the images reference data for each label defined in the list of labels. It makes use of a window function to retrieve only 110 images for each label. It retrieves only unique image ids:

```
-      with row_lim as (
-        select lbs.ImageId as img_id, cls.Description as label,
-        row_number() over(partition by Label) as row_num
-        from `project-up202310061-bdcc.openimages.image_labels` lbs
-        inner join `project-up202310061-bdcc.openimages.classes` cls
   using(Label)
-        where cls.Description in ({labels}))
-         select row_lim.img_id as img_id, max(row_lim.label) as label
-         from row_lim
-         where row_lim.row_num <= 110
-         group by row_lim.img_id
-   '''
```

```
def __append_image_uri(self):
```

- *__append_image_uri:* Create a column ‘image_uri’ in a dataframe which contains the image reference data retrieved from BigQuery and appends the URI for the Vertex AI bucket.

The public methods are:

```
def query_uris_from_self_labels(self):
   logging.info("quering images")
   self.__query_image_ref(self.labels)
   self.__append_image_uri()
```

- *query_uris_from_self_labels:* Queries all the reference data and store relevant info in a dataframe;

```
def load_images_to_vertex_bucket(self):
   logging.info("loading images to
{vertex_bucket_name}...".format(vertex_bucket_name=VERTEX_AI_BUCKET_NAME))
```

```
    image_url =
"https://storage.googleapis.com/bdcc_open_images_dataset/images/"
    images_df = self.image_uris_df
    images_df['image_url'] = image_url + images_df['img_id'] + '.jpg'

    load_counter = 1
    for index, row in images_df.iterrows():
        image = wget.download(row['image_url'])
        blob =
self.vertex_bucket.blob("images/{image_id}.jpg".format(image_id=row['img_id']
))
        blob.upload_from_filename(image, content_type='image/jpg')
        logging.info("loaded {load_counter} from
{total_count}...".format(load_counter=load_counter,
total_count=len(images_df)))
        load_counter += 1

        os.remove(image)
```

- *load_images_to_vertex_bucket:* Download the images from the public bucket https://storage.googleapis.com/bdcc_open_images_dataset/images/ passing a image id. Loads the image into the Vertex AI bucket and then delete the image to avoid leftovers.

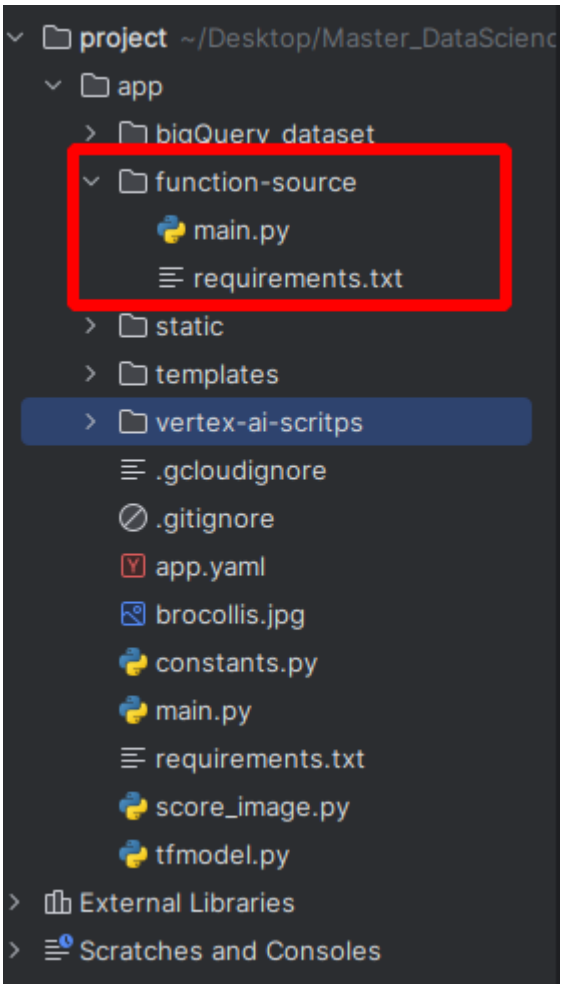
```
def load_ref_id_to_vertex_bucket(self):
    logging.info("loading reference data to
{vertex_bucket_name}...".format(vertex_bucket_name=VERTEX_AI_BUCKET_NAME))
    ref_data_gen.image_uris_df[['img_uri',
'label']].to_csv("./dataset_vertex_ai.csv", index=False, header=False)
    blob = self.vertex_bucket.blob("dataset_vertex_ai.csv")
    blob.upload_from_filename("./dataset_vertex_ai.csv")
```

- *load_ref_id_to_vertex_bucket :* Loads the reference data as csv to the Vertex AI bucket.

All this information is stored in a dataframe which is an attribute of the instances. It is accessible after performing all the actions needed. In the main we call it as:

```
if __name__ == "__main__":
    ref_data_gen = RefDataVertexGenerator()
    labels = ["Wine glass", "Motorcycle", "Hamburger", "Cart",
              "Airplane", "Cat", "Rifle", "Broccoli", "Otter", "Wheelchair"]
    ref_data_gen.set_labels(labels)
    ref_data_gen.query_uris_from_self_labels()
    ref_data_gen.load_images_to_vertex_bucket()
    ref_data_gen.load_ref_id_to_vertex_bucket()
```

6. Additional Challenge: Cloud Function



The Cloud Function (https://us-central1-project-up202310061-bdcc.cloudfunctions.net/classify_image) works as a serverless function that downloads the image passed in the URL and uses the TensorFlow model generated in the Vertex AI to classify the image in the labels Cat, Airplane, Wheelchair, Cart, Motorcycle, Rifle, Broccoli, Otter, Wine glass, Hamburger. The request structure must be:

```
{
 "image_url": "<url_for_an_image>"
}
```

The Cloud Function has as entrypoint the hello_http function in the main.py file.

```
def hello_http(request):
   print("The model is classify images of these labels:\nMake sure your image
url contain one of them.")
   request_json = request.get_json()
   image_url = request_json.get('image_url')
   image = wget.download(image_url, "image.jpg")

   download_model()
```

```
    score_response = score_image()

    os.remove(image)
    return score_response
```

- It receives the request, looks for the **image_url** key and retrieves the image_url link to download the image, which is saved as image.jpg by default.
- It calls the *score_image()* function that returns the image.jpg classification score. The response is built inside the *score_image()* function and stored in the *score_response* variable.
- The image is deleted and the score returned as response to the end-user.

```
def download_model():
    storage_client = storage.Client(project="project-up202310061-bdcc")
    vertex_bucket = storage_client.bucket("vertex-ai-project-up202310061")
    blob =
vertex_bucket.blob("model-1696837812239728640/tflite/2024-04-07T15:25:06.4039
31Z/model.tflite")

    blob.download_to_filename("model.tflite")
    print("Model downloaded to as model.tflite")

    with ZipFile("model.tflite", 'r') as zip_ref:
      zip_ref.extractall("./")
```

```
def score_image():
    model_file = os.path.join('./', 'model.tflite')
    label_file = os.path.join('./', 'dict.txt')
    image = os.path.join('./', 'image.jpg')

    tf_classifier = Model(model_file, label_file)

    results = tf_classifier.classify(image, min_confidence=0.01)
    build_response = ""
    for i,r in enumerate(results):
        build_response += ('{},{},{},{:.2f}\n'.format(image, i+1, r['label'],
float(r['confidence'])))

    return build_response
```

The Model class used in:

```
tf_classifier = Model(model_file, label_file)
```

is the same as defined in the tfmodel.py file, it was just copied to the main.py file.