Chapter 3 - Aplicatia practica

1. Art Collection News Feed

When first interacting with the app, the user will have the possibility to chose among a list of art related topics such as exhibitions, classification, cutures, places, publications, art galleries and more information about the art objects. After scanning and saving to personal collection some items, the news feed will be constantly updated with topics related to the user preferences.

The Harvard Art Museums API is a REST-style service designed for developers who wish to explore and integrate the museums’ collections in their projects. The API provides direct access to the data that powers the [museums' website](https://www.harvardartmuseums.org/) and many other aspects of the museums.

Every request must specify the resource and be accompanied by the apikey parameter and an API key.

<https://api.harvardartmuseums.org/RESOURCE_TYPE?apikey=YOUR_API_KEY>

The dataset is refreshed every day around 6 am. The user will be able to open in browser to view the official page of the Harvard Art Museum, or save his preferences to his collection (maybe share on Facebook).

At the top of the list there is a search bar for selecting the preferred categories of classification. (smart search )

2. Scan art Screen

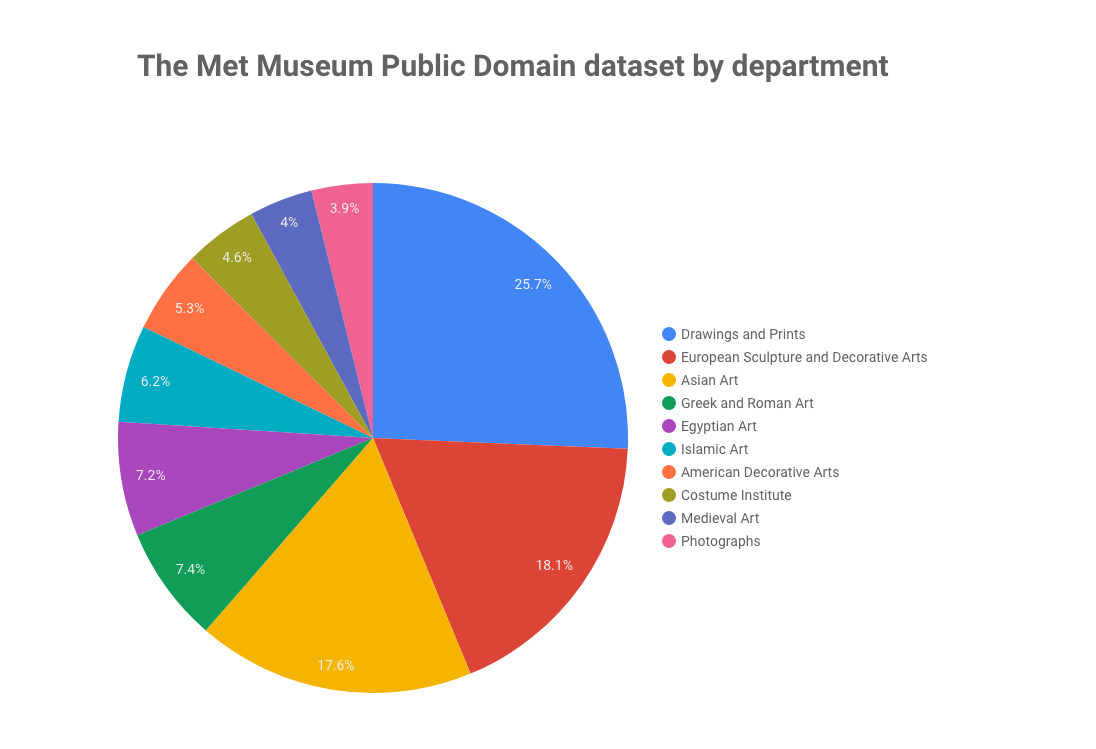
This will be the initial screen when the application is launched.

Art Classifier Flow Steps

1. Google BigQuery

Google BigQuery contains a collection of public database that are useful for a wide variety of purposes and fields of activity.

For proving training input and labels for the models processes I used the databases containing the art collection at the Metropolitan Museum. This public dataset proposes over 200,000 items, representing its domain art from a total of over 1 million of art objects.

Each entry in the set contains metadata about it, such as culture, department, region ,artist, time period and much more information, along with the official image of the item.

source=“https://cloud.google.com/blog/big-data/2017/08/when-art-meets-big-data-analyzing-200000-items-from-the-met-collection-in-bigquery#closeImage”

Example of json response:

"webDetection": {

"webEntities": [

{

"entityId": "/m/09c7b",

"score": 3.07616,

"description": "Metropolitan Museum of Art"

},

{

"entityId": "/m/02jx1",

"score": 1.0786,

"description": "England"

},

{

"entityId": "/m/04rjz",

"score": 0.75304,

"description": "Middle Ages"

},

{

"entityId": "/m/0x44",

"score": 0.74545,

"description": "Armour"

},

...

],

"fullMatchingImages": [

{

"url": "http://images.metmuseum.org/CRDImages/aa/original/DT753.jpg"

},

...

],

"pagesWithMatchingImages": [

{

"url": "http://www.metmuseum.org/art/collection/search/23939"

},

...

],

"visuallySimilarImages": [

{

"url": "https://s-media-cache-ak0.pinimg.com/736x/ce/c9/34/cec9340be5a4d789fd1156413140dd75.jpg"

},

...

]

}

2. Selecting the attributes and creating labeled dataset

For accessing and querying the BigQuery dataset the user has to wait for a job to complete.Jobs are objects that manage asynchronous tasks such as running queries, loading and exporting data. Multiple jobs can be ran concurrently in BigQuery, all the completed jobs being listed in the Jobs collection (store the project compete job history)

The jobs gets as input a Client( the project\_ID) and the query, which is a standard sql query.

The query:

SELECT

object\_name,

department,

culture,

object\_date,

artist\_display\_name ,

link\_resource

FROM `bigquery-public-data.the\_met.objects`

WHERE culture IS NOT NULL

The result of the query is accesses using the .result() method, each entry being a war in the result of the query, which will further be parsed for further computations of that data and organising it by labels.

Create a directory with the culture as name if it doesn't exist yet and remove characters that are not valid for directory name.

The URL of the artefact from bigQuery os a webpage, containing a link to download the original picture ( the picture will be shown to the user and will be added to the user collection of scanned art). This URL will be converted to the %-encoded format since it may be in other format like utf-8.

3. Downloading the data

Although the Google BigQuery holds the attributes, the photos of the art collection are actually kept at a site

from the Metropolitan Museum of Art. Therefore, to build our labeled dataset, we will need to download the photos and associate them with the labels.

If a particular culture has just a few art images, it's probably not enough to train the model.

The image source of each entry resulted from the bigQuery will be placed in a directory named with the label. From the `arts-all.list` file a selecting of the data will be copied in the file `arts-select.list`for performing the training phase.

4. Choosing a model for image classification

5. Converting the data to TFRecord format

<https://github.com/tensorflow/models.git>

The Tenserflow code will be used to process the data. The Tenserflow open source. Models library contains a collection of public models such as:

- the [official models](https://github.com/tensorflow/models/blob/master/official) are a collection of example models that use TensorFlow's high-level APIs. They are intended to be well-maintained, tested, and kept up to date with the latest stable TensorFlow API. They should also be reasonably optimized for fast performance while still being easy to read.

- the [research models](https://github.com/tensorflow/models/tree/master/research) are a large collection of models implemented in TensorFlow by researchers. They are not officially supported or available in release branches; it is up to the individual researchers to maintain the models and/or provide support on issues and pull requests.

- the [samples folder](https://github.com/tensorflow/models/blob/master/samples) contains code snippets and smaller models that demonstrate features of TensorFlow, including code presented in various blog post

The chosen model uses and extend the collection of image classification models in the directory `models/slim` . Using the TenserFlow source code from their repository, the algorithm will be able to process several different image datasets (….) and advanced models to train.

For extending the Tenserflow code to process the new dataset of images, the files dataset\_factory.py and arts.py from the TenserFlow will be copied into the project.

The raw images are converted into TFRecord format that the TenserFlow code will use.

- TFRecord=> TFRecord file format is a simple record-oriented binary format that many TensorFlow applications use for training data

- TFRecordWriter

To convert the art dataset, put the directories of downloaded pictures in a directory named `met\_art`, for instance `/your\_home\_directory/data/met\_art`.

The script convert.py will read the image files from the directory received as parameter and create two TFRecord datasets: one for training and one for validation.

Each TFRecord dataset is comprised of a set of TF-Example protocol buffers, each of which contain a single image and label. About 25% of the images are set aside for validation.

The script takes about a minute to run and the file `labels.txt` lists all the culture labels found in the images directory.

The output will be stored in data directory.

/\*\*\*\*\*\*\*\*\* example \*\*\*\*\*\*\*\*\*\*\*\*\*/

6. Creating the Tenserflow container image

To deploy the pod, you will need to create an image containing the TensorFlow code by running the command:

```

$ cd /your\_home\_directory/tensorflow-kubernetes-art-classification

$ mkdir data

$ cp /your\_home\_directory/data/\*.tfrecord data/.

$ cp /your\_home\_directory/data/labels.txt data/.

$ docker build -t your\_image\_name:v1 -f Dockerfile .

```

We include a small sample copy of the dataset in this image. The reason is twofold. First, shared filesystem is not available for the free IBM Cloud account. In normal practice, the dataset is too large to copy into the image and you would keep the dataset in a shared filesystem such as SoftLayer NFS. When a pod is started, the shared filesystem would be mounted so that the dataset is available to all the pods. Second, the computation resource provided with the free IBM Cloud account is not sufficient to run the training within a reasonable amount of time. In practice, you would use a larger dataset and allocate sufficient resources such as multiple CPU cores and GPU. Depending on the amount of computation resources, the training can run for days or over a week.

IBM Cloud Container Registry !!!!!!

7. Deploying and Running the training on Kubernetes (GPU??)

- kubectl

Along with the pod, a local volume will be created and mounted to the pod to hold the output of the training. This includes the checkpoints, which are used for resuming after a crash and saving a trained model, and the event file, which is used for visualization. Further, the restart policy for the pod is set to "Never", because once the training is complete there is no need to restart the pod again.

8. Evaluating the accuracy of the trained model

* deploy the pod
* Check evaluations status

9. Saving the trained model and logs

- Copy all the log files on the Kubernetes persistent volume to local host.

10. TensorBoard for statistics and visualisation of the training

- the event file copied from the Kubernetes persistent volume contains the log data for TensorBoard.

Start the TensorBoard and point to the local directory with the event file:

```

$ tensorboard --logdir=<path\_to\_dir>

```

Then open your browser with the link displayed from the command.

11. Loading the trained model in Kubernetes and running inference on a new input for classification

having a trained model to classify art images by culture, a new art image will be provided as input to classify the model.