Python Technical test (Machine learning team)

Context

Yubo users can upload media to their profile. To make sure this content is appropriate, we use existing machine learning models that classify the images into different categories (weapon, male shirtless, blood, …).

On Yubo, male shirtless images are not allowed, except when they're at the beach.

This test is a good example of a typical ML team project you would be facing: implement a web service that can detect the scenery of an image (beach, mountain, lagoon, …).

For this proof-of-concept, we will use the popular resnet50 model trained on places365 dataset.

Goal

Code a web service/rest API doing image classification.

The web service should return the predicted class for each input image.

Miscellaneous

This test should be done in more or less 2 hours.

What is expected

You have to deliver a project containing:

- a REST web service with a POST endpoint `/predict`
- a script to test this endpoint and measure its performance

The `/predict` endpoint:

- must accept multiple images as input
- must receive image data in the body, do not use image paths as input
- the content-type of the input request should be `multipart/form-data`
- must accept images of any size and in jpeg format

- should return, in JSON format, the top prediction for each input image.

To obtain the classification scores for an image, a second web service will be used. You do not have to code it, you only need to deploy it. See **Setup tensorflow serving below.** .

This server listens on port 8501 for HTTP requests and expects an input JSON format.

The JSON object must have a key `inputs` containing a multiple dimension array of shape:

```
(batch_size, n_colors, h, w)
```

If returns a JSON object with key `ouputs` of shape:

```
(batch_size, n_classes)
```

- `batch_size` corresponds to the number of inputs you send to the tensorflow server. Its maximum value, with the provided configuration file, is 6, but you can set it to a different value in `batching.cfg` if you like.
- `n_colors` is the number of channels in the image. The TF server expects RGB images to this value is 3
- `h` and `w` are respectively the height and width of the images. The TF server expects images of size 224x224 so both values have the same size of 224.
- `n_classes` if the number of classes supported by the provided model. The first value is the score of the first class, the second value is the score of the second class and so on. The names of the classes is provided, in order, in `categories_places365.txt`. The biggest value is the top prediction for the image.
- Each pixel for a given channel is expected to be sent as a float value normalized and standardized. See **Tips** below for more details.

Your web service must receive the images, uncompress them, resize them, send them to the TF server, process the scores to obtain the top prediction for each image and return those predictions.

Requirements

- The web service should be in **Python**.
- You can use any framework you like, as long as this framework allows your web service to process requests concurrently. **FastAPI** is **recommended** but not mandatory.
- You should not process each image sequentially, but in parallel. Using **asyncio**, along with `await/async` keywords (python tasks and coroutines) is encouraged.
- Your web server should output meaningful logs for each request, showing especially that each image is processed in parallel.

- Your test script should send multiple requests simultaneously, proving that your web service is capable of treating multiple requests at the same time.
- Your project should include a `README.md` file providing instructions on how to deploy the web service.

Bonus questions

- Support any image format as input
- Dockerise your web service.
- Implement some unit tests
- Any ideas you may have

Setup tensorflow serving

Follow these steps:

1. download this folder: https://drive.google.com/drive/folders/1RKjHivApDBAQWkkC-eSTqv0x9boo1T_h?usp=sharing
2. Build the docker image and run it

```
docker build . -t serving -f Dockerfile

docker run -p 8500:8500 -p 8501:8501 serving
```

You can test it with the following curl command:

```
curl -X POST -H 'Content-Type: application/json' -d @example_request.json  http://localhost:8501/v1/models/model:predict

result:

{
    "outputs": ...
}
```

Tips

- Example of preprocessing step using numpy to get the image in the right format

```
import json
import numpy as np
from PIL import Image

filename = "1.jpg"
shape = (224, 224)
```

```
mean, std = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]

im = Image.open(filename).convert("RGB")
im = im.resize(shape)
im = np.array(im, dtype=np.float32)
im /= 255.
im -= mean
im /= std
im = np.transpose(im, (2, 0, 1))

data = [im.tolist()]
```

Here, data is of shape (1, 3, 224, 224)