GumGum ML Engineer take-home project

Overview

GumGum is launching a new promotional website that allows anyone to upload an image and get an instant classification of their image using a pre-trained model capable of identifying 1000 classes of objects. GumGum wants to go live in a week but the front-end (which is being built by a separate team) needs an API to power it. This is exactly where you, our fancy new Machine Learning Engineer, come in to save the day!

Deadline

This project must be submitted no later than a week after the time these requirements were delivered to your email.

Classification model

Model

Download the pre-trained MobileNet classifier here

Classification vector indices

Download here

Pixel values

Pixel values for input images need to be normalized between `-1` and `1` (example: `0,255` (ints) should be scaled to `-1,1` (floats) and `0, 127` to `-1, 0`). If your input image has an alpha channel, you can safely remove/ignore it.

Resolution

Images loaded into the model need to be resized to 224px by 224px `(224, 224, 3)` regardless of original dimensions.

Encoding

We want to support input images in the JPEG, PNG and GIF formats.

API requirements

Using the language, frameworks, libraries, data stores and tooling of your choice, build a REST API that meets the following requirements:

1) Image classifier endpoint

Using TensorFlow , integrate the classification model described in the previous section with your

API. Then, create the endpoint `POST /classify-image` that accepts a single JSON parameter `image_url` which contains a link to the image to be classified. Your API should download the image, re-size and re-map it as required by the classification model and return a response that indicates the most-likely `classification`, the `confidence` (in the range of 0-1) of that classification and the `processing_time` (s) required to process and classify the image.

Example request

```
curl --request POST --header "Content-Type: application/json" --data
'{"image_url":"https://s3.amazonaws.com/gumgum-interviews/ml-engineer/cat.jpg"}'
http://localhost/classify-image
```

Example response

```
{
    "classification: "tabby",
    "confidence": "0.554",
    "processing_time": "4.731"
}
```

Note: this **is** the expected response for the example image above.

2) Classification caching

Now that your API can classify external images, improve your endpoint's processing time by introducing server-side caching based on the `image_url`. When implemented, any subsequent requests of the same URL should return the cached classification. To avoid serving stale classifications, be sure to set the TTL on the cache keys to 1 hour.

3) Reporting endpoint

Now that we can serve classifications in a performant manner, we can start improving the visibility of the service's usage via a new reporting endpoint `GET /report`. When hit, the endpoint should return a list of the top 10 most requested `image_urls` along with the following metadata for each:

```
Number of times the image was classified (including cached responses)
The minimum, maximum and average total processing times
```

Bonus

4) Dockerize your API

Write a `Dockerfile` to build a Docker image containing everything needed to run your API service. Once completed, we should be able to start your service with a simple `docker build -t classifier-api .` followed by a `docker run classifier-api`. Additional build/run arguments are allowed but should be documented in your project's `readme.md` file.

5) ElasticSearch monitoring

ElasticSearch is great for storing and querying time-series data. GumGum would like to be able to query production data for your new service in a real-time fashion using Kibana, a popular web UI that sits on top of ElasticSearch. First, spin up an AWS Elasticsearch Service cluster comprising of a single node using the AWS Console (no need to automate this). Doing so will also provision a Kibana instance that is pre-configured to connect to your cluster.

Once you have verified you can connect to your ElasticSearch service, update your API to create a new ElasticSearch document for each request to your `GET /report` endpoint. The document should include all the fields already returned by your API.

Then, create a simple visualization of your real-time data in Kibana and include a screenshot of it in your project's `readme.md`.

6) User authentication

Using the user management pattern of your choice, create a new endpoint `GET /secret` that returns `Hi <username>` for authenticated users only (otherwise return a `401` error).

Questions

Question type	Contact
Machine learning	mgreenberg@gumgum.com
API development	corey@gumgum.com

And of course feel free to contact either of us for general questions or comments :)

How to submit your project

Write a basic `readme.md` for your project that documents how to build and run your API server in a `localhost` fashion. Please include an example `curl` request and response to your `POST /classify-image` endpoint. Compress all your files into a single `.tar.gz` or `.zip` and send it to the following GumGummers:

```
bfuller@gumgum.com
cambron@gumgum.com
corey@gumgum.com
```