Harrison Kaiser
CAT 2019

Museai Model Builder Web Application

Purpose: To provide a user interface in which organizations such as museums can easily create their own image classification models so that their visitors can quickly access information about art pieces.

How it works: Organizations connect to a CAT web portal that will handle video uploads and creation of albums. Albums represent a collection of similar entities that we want an image classification model to recognize such as art pieces in an art gallery. Processing of videos involves using ffmpeg tool to get a specified number of frames per second and places those images in a separate folder on the web server. Then the user has an option to delete, edit, or train albums. Once training for an album has been initiated, it will be queued as a task on the web server. When training begins, a tensorflow python script will be executed. The script will recursively go through all of the images uploaded in that album and create a convolutional neural network when it's done training. Model files will be generated in the same folder as the uploaded images and can be used by a front-end developer.

How I built the platform: I used django web framework to create the full-stack web application. The front-end system uses jinja templating engine, bootstrap CSS framework, jQuery, and ajax to handle asynchronous requests from the web server. The back-end system uses celery for queuing tasks, mySQL for database engine, and django's model-view-template architectural pattern for communication between the front-end and the back-end system.

Setup

Requirements

OS: Ubuntu 18.04

-Django

-Python3

-FFmpeg

-MySQL Community Server

-Celery (Python Task Queue system)

-RabbitMQ (message broker for celery)

-Docker engine

-Google App Engine

Install Python3: '`sudo apt-get install python3`'

Install Pip3: '`sudo apt-get install -y python3-pip`'

Install FFmpeg: '`sudo apt-get install ffmpeg`'

Install Docker Engine: https://docs.docker.com/install/linux/docker-ce/ubuntu/

To get the project files and install required dependencies for the django web application, please run:

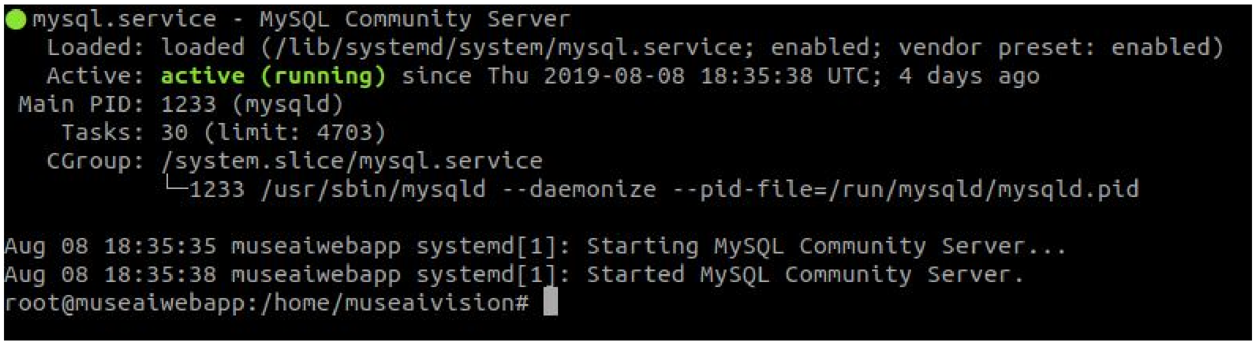
```
git clone https://github.com/NTIDCATLab/museai_django.git
cd /path/to/project
pip3 install -r requirements.txt
```

Database

Before setting up a local web server, MySQL needs to be installed and connected to Django. Please follow these steps:

1. Open the command prompt.
2. Run '`sudo apt-get install python3-dev`'
3. Run '`sudo apt-get install python3-dev libmysqlclient-dev`'
4. Run '`pip3 install mysqlclient`'
5. Run '`sudo apt-get install mysql-server`'
6. Verify that MySQL is running: '`systemctl status mysql.service`'

   You should see this output:



```
● mysql.service - MySQL Community Server
   Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2019-08-08 18:35:38 UTC; 4 days ago
 Main PID: 1233 (mysqld)
    Tasks: 30 (limit: 4703)
   CGroup: /system.slice/mysql.service
           └─1233 /usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pid

Aug 08 18:35:35 museaiwebapp systemd[1]: Starting MySQL Community Server...
Aug 08 18:35:38 museaiwebapp systemd[1]: Started MySQL Community Server.
root@museaiwebapp:/home/museaivision#
```

   If you don't see the above output, then run '`sudo systemctl start mysql`'

7. Run ‘`mysql`’
8. In the mySQL shell, run ‘CREATE DATABASE museai_data;’
9. Run ‘exit’ to return to command prompt
10. Run ‘`sudo nano /etc/mysql/my.cnf`’ to edit mysql config file
11. Add these lines to the file:

    ```
    [client]
    database = museai_data
    user = db_user
    password = db_password
    default-character-set = utf8
    ```
12. Run ‘`systemctl daemon-reload && systemctl restart mysql`’ so changes take effect.
13. Voila! You’ve set up the database!

***Important note: If the database is getting locked down for some reason, you must restart it.**

Celery Workers

If you want to start training models using the web app, you need to set up Celery and RabbitMQ server which handles execution of tasks. A python virtual environment is recommended, but not required to install dependencies for the web server. Project folder has a virtual environment you can activate with ‘`source activate model_builder`’. Follow these steps to get set up:

1. If Celery is not installed, you can with ‘`pip3 install celery`’
2. Install RabbitMQ with ‘`sudo apt-get install rabbitmq-server`’
3. Run ‘`cd /path/to/your/project`’
4. Start the workers by running ‘`celery -A museai_project  worker -l info`’

Server

If you want to set up a local development server, please follow these steps:

1. Run ‘`cd /path/to/your/project`’
2. Run ‘`python3 manage.py runserver localhost:8000`’

3. Visit localhost:8000 to view the web app.

How to deploy Django to production:

1. Install Nginx web server and Gunicorn (pre-installed on cloud servers like DigitalOcean)
2. Edit Gunicorn config file: ‘`sudo nano /etc/systemd/system/gunicorn.service`’
3. Add these sections in the file:

```
[Unit]
Description=Gunicorn daemon for Django Project
Requires=gunicorn.socket
After=network.target

[Service]
Environment=SECRET_KEY=secret
User=root
WorkingDirectory=/home/django/museai_django
ExecStart=/usr/bin/gunicorn \
          --access-logfile -  \
          --workers 3 \
          --bind unix:/run/gunicorn.sock \
          museai_project.wsgi:application

[Install]
WantedBy=multi-user.target
```

4. Enable the Gunicorn socket by running ‘`sudo systemctl start gunicorn.socket && sudo systemctl enable gunicorn.socket`’
5. Verify that Gunicorn socket was started with ‘`systemctl status gunicorn.socket`’
   If there is an error, then run ‘`journalctl -u gunicorn.socket`’ to what the error is.
6. Test Gunicorn service with ‘`systemctl status gunicorn`’
   If status is failed, then check to see why with ‘`sudo journalctl -u gunicorn`’
7. If you make any changes to gunicorn.service file, run ‘`sudo systemctl daemon-reload && sudo systemctl restart gunicorn`’ for changes to take effect

8. Now we need to configure Nginx to proxy pass to Gunicorn. Run ‘`sudo nano /etc/nginx/sites-available/museai_project`’
9. A config file would look something like this:

```
server {
      listen 80;
      server_name server_domain_or_ip;

      location /static/ {
            Root /home/django/museai_django;
      }

      location / {
            include proxy_params;
            proxy_pass http://unix:/run/gunicorn.sock;
      }

      location /viewer {
            include proxy_params;
            proxy_pass http://127.0.0.1:300/;
      }

      location /dist/ {
            root /home/museaivision
      }
}
```

   In this config file, I created two proxy passes for my django model builder web application and my viewer application which is running on a different port. Server name is the server domain or ip address.
10. Save file and close it. Run ‘`sudo ln -s /etc/nginx/sites-available/museai_project /etc/nginx/sites-enabled`’
11. Test your config file for errors with ‘`sudo nginx -t`’

12. If there are no errors, run ‘`sudo systemctl restart nginx`’
13. We can now remove rule to open port 8000 on our local development server by running ‘`sudo ufw delete allow 8000 && sudo ufw allow 'Nginx Full'`’

***Important note: If you make any changes in the Django application, you must restart the Gunicorn service.**

Test Models

If you want to test the models you created with the model builder web app, you can use a web based game called “Emoji Scavenger Hunt” that I have modified. Assuming you have installed Docker Engine on your computer, you can use a docker container that already has all of the dependencies installed! Follow these steps:

1. Run ‘`git clone https://github.com/hxk1633/museaivision.git`’
2. Run ‘`docker pull hxk1633/emoji-hunt-game`’
3. Run ‘ `docker run -it -v /path/to/project/:/model_viewer -w /model_viewer --network=host hxk1633/emoji-hunt-game bash -c “yarn prep && yarn dev”` ’
4. Visit localhost:3000?album=album_name&debug=true
   album_name corresponds to the name of the album you have created with the model builder web app and where the model files are stored. You can use a symlink to have album folders in the Django project be shared with a folder in the model_viewer application. So you may need to modify the path of where all the albums are stored in mobilenet.ts file in src/js project folder:

```
 const MODEL_FILE_URL = '/model/models/albums/' + albumName +
'/data/saved_model_web/tensorflowjs_model.pb';
const WEIGHT_MANIFEST_FILE_URL = '/model/models/albums/' + albumName +
'/data/saved_model_web/weights_manifest.json';
```