Ira A. Fulton Schools of Engineering
Arizona State University
Engineering | School of Computing, Informatics, and Decision Systems Engineering

CSE 535: Mobile Computing

Application Service for Classifying American Sign Language

Part 2

Purpose:

Students will work in teams to develop a RESTful application service for classifying American Sign Language (ASL). Students will gain hands-on experience in developing a fog-server that acts as the computing center for data sent by edge devices. This project, and its successful completion, provides an excellent opportunity to gain further exposure to relevant topics and applications in the area of mobile computing.

Technology Requirements:

- Flask Server
- TensorFlow's PoseNet
- Python 3.6.9
- OpenCV for Python
- Node.js

Project Description:

For Part 2, students will develop an online RESTful application service in Flask that accepts skeletal human pose keypoints of an ASL sign video and return the label of the sign as a JSON response. The keypoints will be generated using TensorFlow's PoseNet.

1. Using these keypoints, develop four (4) machine learning (ML) models (one by each student in the team) that can classify the listed ASL signs (some of the signs were completed in Part 1 of the project):
    - {
    - buy - https://www.signingsavvy.com/search/buy,
    - fun - https://www.signingsavvy.com/search/fun,

    - hope - https://www.signingsavvy.com/search/hope,
    - really - https://www.signingsavvy.com/search/really,
    - communicate - https://www.signingsavvy.com/search/communicate,
    - mother - https://www.signingsavvy.com/search/mother
    - }
2. For generating the key points, please follow the instructions in the provided link:
    - Generating Keypoints using TensorFlow's PoseNet - https://github.com/prashanthnetizen/posenet_nodejs_setup
3. For the dataset, you will use the videos captured in Part 1 of the project. It is recommended that your videos follow the order presented in Step 1 to receive better accuracy.
4. The JSON input given to the service follows the format of the output given by the JavaScript file provided in the GitHub repo. The JSON input would be an HTTP POST request that has “application/json” as the content-type. The body of the JSON POST request would be the contents of the keypoints.json file, but **not the file itself**.
5. The output of your service should be given as a JSON response for each JSON input of video data in the following format:
    - {
    - “1”: “predicted_label”,
    - “2”: “predicted_label”,
    - “3”: “predicted_label”,
    - “4”: “predicted_label”,
    - }
        - “1” to “4” denotes the index of your ML models
        - “Predicted_label” is the predicted sign (all in lowercase, and use “_” for space)

Make sure the URL to your service is http://localhost:5000/, where “localhost” is the domain name of your service, which is nothing but 127.0.0.1 as you are hosting it in your own local machine, and 5000 is the port number.

The grading Environment is constant and has a predefined set of dependencies you can use for your Server Program. Please use the attached “requirements.txt” file for creating your own virtual environment for this project.

Within an hour of the submission, your service will be evaluated using test data. The resulting **maximum accuracy** and **average accuracy** will be considered for grading.

- Metric explanation:
    - **Maximum accuracy** is the maximum accuracy measure obtained out of the four models during a test run.
    - **Average accuracy** is the average accuracy of all the models.

Please test your services properly before submitting the project.

Submission:

You will need to submit the following project deliverables:

- Source code of the Flask server named as “app.py”, which contains the actual server code that receives the JSON input, processes the labels using your ML models, and returns the JSON response as specified.
- ML model training notebooks or scripts under a “notebook” folder
- ML models under a “models” folder. You may use these saved models for your server.
- README file
- A report evaluating your ML models that discusses the following points:
    - An explanation of how your team approached the given problem,
    - The solution for the problem, and
    - An explanation of why the team selected the submitted ML models for this particular problem.

The five deliverables above are to be submitted in a single zip file. Please note that the zip file should only contain these deliverables and nothing else within a separate folder. The Python file that contains the Flask server code should be outside all of the other folders. Please follow the folder hierarchy listed below for your submission:

```
fog_server.zip
    ├── app.py
    ├── models/
    │   ├── model_1.pkl
    │   ├── model_2.pkl
    │   ├── model_3.pkl
    │   └── model_4.pkl
    ├── notebook/
    │   └── sample_script_training_your_ml_models.ipynb
    ├── README.md
    └── sample_report.pdf
```

The name of the zip file that is submitted should be “Project_2_(group_number).zip”. The system will automatically convert your zip filename to “fog_server.zip”.

Autograder Feedback:

There are about 60 data samples, which are the same as the training data. Each data sample is sent as a separate HTTP POST request to your application service one by one. The autograder will keep recording the response sent by your service. Finally, the accuracy metrics are calculated and the grade points are provided to you.

If your service runs as expected, then the following feedback is expected:

> The Accuracy obtained by your ML models:
>
> Model 1 : 0.16666666666666666
>
> Model 2 : 0.16666666666666666
>
> Model 3 : 0.16666666666666666
>
> Model 4 : 0.16666666666666666
>
> Average Accuracy : 0.16666666666666666
>
> Max Accuracy : 0.16666666666666666

If anything goes wrong, then the cause of the error is shown in the feedback. For example:

> Exception occurred while trying to run your service.HTTPConnectionPool(host='localhost', port=5000): Max retries exceeded with url: / (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x7f02851557f0>: Failed to establish a new connection: [Errno 111] Connection refused',))