ASL Finger Spelling

CSE 535: Mobile Computing : PROF. AYAN BANERJEE

Dheeraj Malepati, Kavya Gudipati, Madhu Priyatam Venkata, Rahul Reddy Sarikonda

I. Abstract

An application that captures videos of ASL sign language and converts it into word text. The front end involves the user uploading a video with each ASL alphabet shown separately. The backend involves Pre-processing steps including a CNN algorithm to classify the input. FrontEnd and UI-specific tasks are implemented using Android on Android Studio platform. Backend tasks are initially tested on Google Colab and later deployed on AWS Cloud Services. ASL finger spelling signs were processed in the sequence by the algorithms of Frame extraction, Posenet key points extraction, Segmentation, Palm detection and Cropping and CNN model prediction.

II. Introduction

With the exponential increase of applications, it is important to make them accessible in order to provide ease of access to disabled users. Accessibility also helps to boost the business and increase the target audience. One section of disabled users are hearing-impaired users. Finger-spelling is a method of expressing words using hand movement where ASL denotes American Sign Language. Accurate detection of ASL finger-spelling signs will be benefited integration into many applications. We came up with an architecture and prototype of detecting ASL finger-spelling signs through an android application and converting it into word text.

III. Architecture

We came up with the below architecture to ensure optimal detection of ASL signs at low latency. To achieve faster computation, we are processing the videos in the Cloud.

A. Front End

Mobile applications are faster and easier to access than web apps and are advanced in terms of features and functionality. As a result, we have implemented an android app. The app is capable for capturing a video through Camera Interface and uploading the video into the Cloud. The app has a messaging service which pushes and retrieves messages to and fro the cloud.

B. Cloud

For the cloud, we have used amazon web services(AWS) products. In order to process the input videos, we have created a virtual machine in the amazon cloud for faster computation which is Amazon Elastic Compute Cloud(EC2). The EC2 Instance is Linux Based and has backend code installed in it. Apart from the computation, Amazon Simple Storage Service(S3) is used for persistence by storing the input and the output of the computation and Amazon Simple Queue Service(SQS) is as a messaging component between EC2 and S3.

C. Process Flow

The process starts at the android application level. The application has three activities. One of the activity allows the user records a video of showing ASL signs to form a word. Another activity uploads this input video into Amazon S3. Once the video is uploaded, the application sends a new request notification to Amazon SQS. At the backend, EC2 instance polls the SQS queue for new requests. Once a new request is received, the EC2 instance downloads the input video from Amazon S3 and processes the video. The output is stored in the output bucket of Amazon S3. The third activity of the android application checks if the request is completed and displays the output retrieved from output bucket of Amazon S3. A

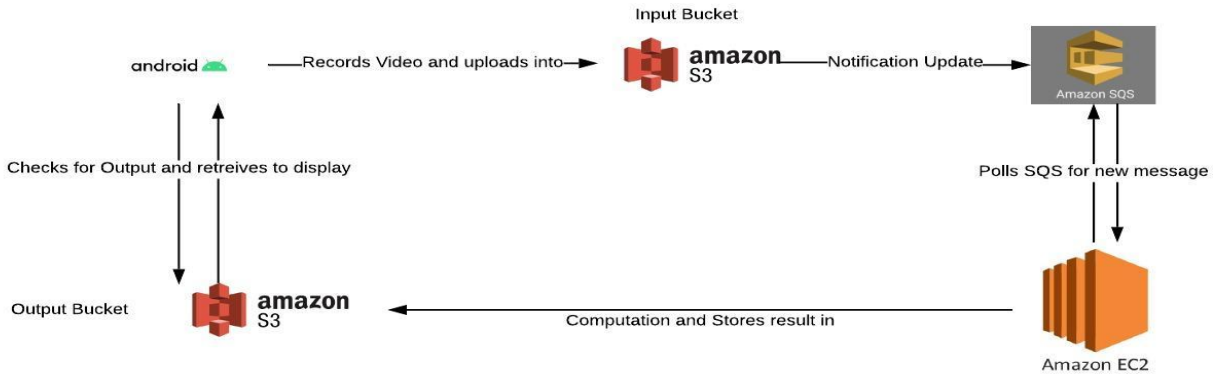

Fig. 1. Architecture diagram

pictorial description of the architecture is shown in figure 1.

IV. Project Setup and Permissions

The project setup consists of two components. The configurations are described below:

- Android device Make & Model: Motorola G5 OS: Android 8.1.0 Permissions: Camera and Storage
- Cloud Setup Amazon S3: Input Bucket Name - mcrequests - US East (N. Virginia) us-east-1(Public), Output Bucket Name - mcoutputreq - US East (N. Virginia) us-east-1(Public)
  Amazon SQS: Queue Name - mcqueue - US East (N. Virginia) us-east-1
  Amazon EC2: Ubuntu Server 18.04 LTS (HVM), SSD Volume Type (64-bit x86), 8GB RAM
- Programming Languages Android(Java), Python 3.6

V. Implementation

This section involves detailed implementation tasks in our project.

A. Record Video Activity

The user of the application is welcomed with this activity. The user has an option to record a video through a button. Figure 2 is the screenshot of this activity in the android application.

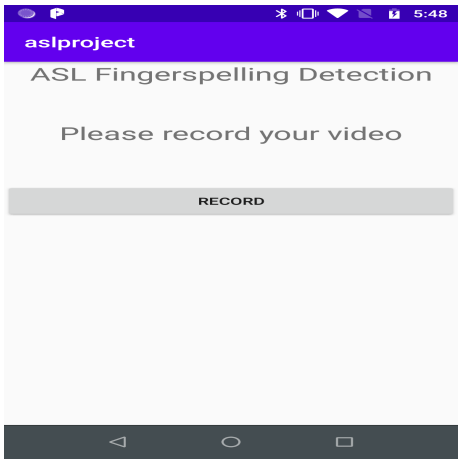

Fig. 2. Screenshot of Record Video Activity

B. Upload Video Activity

After recording the video, the user is given two options for further steps. One of the option is to go to the previous activity and another option is to process the video. Once the ”process video” is clicked, the ”request dashboard” button pops up. In the background, the input video is uploaded into Amazon S3 via asynchronous methods. Figure 3 shows the process flow of upload video activity.

C. Request Dashboard Activity

Users can view their requests and their results in this last activity. The users are given a refresh button to check their result constantly.

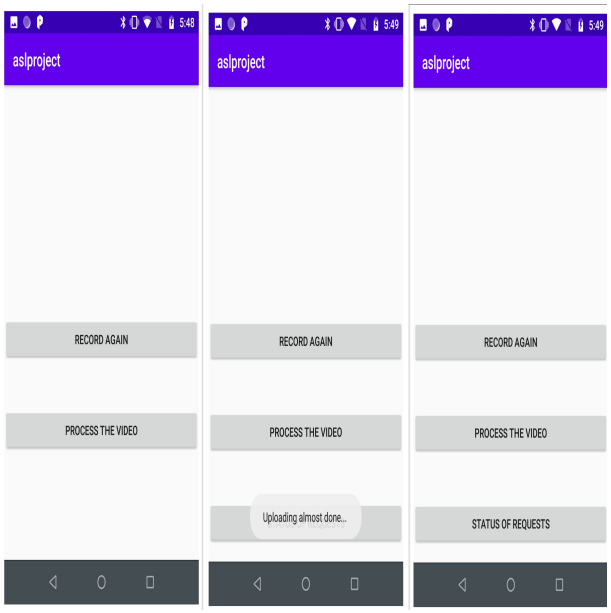

Fig. 3. Screenshot of Upload Video Activity

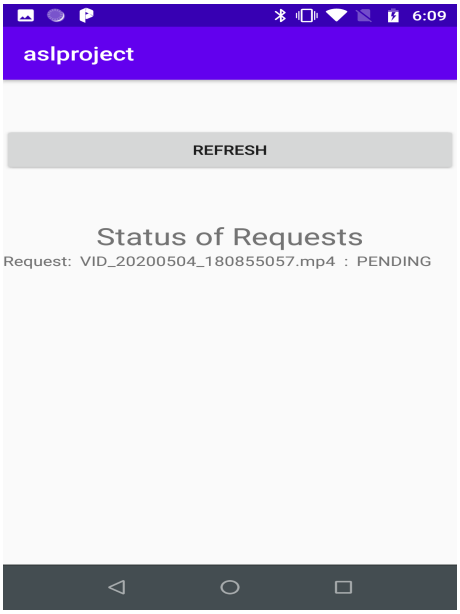

Fig. 4. Screenshot of Request Video Activity

D. Amazon S3 & SQS

Amazon AWS SDK Java is used to allow the communication between android components and Cloud Services. BasicAWSCredentials class is used to establish and authenticate the connection between android app and Amazon Cloud Service. AmazonS3Client Class and methods are used to upload and download the data from Amazon S3. We have used two buckets, Input and Output, to store videos and word text respectively.

E. Video Format

The video is shot in the sequence of initial position, ASL sign, initial position, ASL sign and so on. The initial position of the user is standing at attention posture.

F. Backend : Amazon EC2

1) Frame Extraction: We have used GitHub open source code provided by the professor in Assignment 1. We have implemented our changes to this code. Extracting and saving the video frames is done by the Open CV library. The video file is read using cv2.VideoCapture() and read() function. After extraction, the VideoCapture object is released and all the windows are destroyed.

2) Posenet: We have used GitHub open source code of posenet implementation in Python. Single person pose estimation is performed on the frames. MobileNetV1 architecture is used to support the pose estimation. The output of the posenet evaluation is an excel sheet with score, x and y coordinates of right wrist. The reason to filter the right wrist alone is to reduce the pre-processing steps and other key points were not used to judge an ASL sign.

3) Segmentation: According to the video format, the pattern in the key points was determined. Segmentation is implemented as follows:

- Mean of first 10 frames y-coordinate is calculated. In these 10 frames, the user is in initial rest position.
- Through Iteration of y-coordinate key points, an ASL sign is determined to be the user from initial position to an ASL sign to initial position. These extracted frame images are put into a folder using shutil copyfile.

For example, the mean for first 10 y-coordinate images is 1800. Iterating from frame 1, all set of frames until the next frame(whose y-coordinate is around 1800) are taken as one ASL sign. The pseudo code of segmentation algorithm is attached below.

4) Cropping Hands: PoseNet keypoints play a key role in determining the exact location of the hands. Once the ASL signs are extracted into separate

Algorithm 1 Segmentation Pseudo Code

```
Read Key Points CSV
Set initial position to 0
Set flag to 0
Set previous row to 0
for all the rows do
    if flag = 0 then
        Get integer of y coord of right wrist
        Save this value to initial position
        Set flag to 1
    else
        Get integer of y coord of right wrist
        if Check if it is equal to previous row
then
            if Δwithinitialposition > 0 then
                Add into result
            end if
        end if
    end if
end for
```

folders, the cropping algorithm is run across these folders. The cropped image follows the format of image[ycoordinate - 350 : ycoordinate, xcoordinate - 50 : xcoordinate + 250]

5) CNN Model: CNN model was trained and tested across the Kaggle dataset of ASL alphabets. Sequential type of model has been used. The input image is scaled up/down to 64 * 64. The architecture of the CNN model is as:

- First two layers are CONV2d layers. These layers will be addressing the input which are 64*64. The number of nodes are 64 and 32 respectively. A dropout layer was added later to avoid overfitting.
- Next two layers would be again CONV2d accompained with a dropout layer.
- In order to reduce the number of parameters in the model, we added a max pooling layer too.
- In order to standardize the data, we added batch normalization.
- We also flattened the model to transform the data for the next layer.
- One dense layer was added with 512 units. Later, another layer was added with 29 units to fill the gap.

The model was trained with 6 epochs with 128 batch size. 80% of the Kaggle data was used for training and the remaining 20% was used for testing.

VI. Accuracy Reports

The accuracy is measured with respect to the following metrics

- Training Accuracy: 92.73%
- Testing Accuracy: 73.4%
- F1 score: 0.9275
- Recall: 0.9029
- Precision: 0.9446
- Categorical Accuracy: 0.9230

Figure 5 shows the metrics for each epoch.

VII. Tasks Distribution

No	Task Name	Assignee
1	Architecture designing	Dheeraj, Kavya, Madhu
2	Record Video Activity	Rahul, Madhu
3	Uploading into S3 and Retrieving Messages	Dheeraj, Kavya
4	UI of other activities in Android application	Rahul, Madhu
5	Testing the UI	Kavya, Madhu
6	Configuring the Cloud Setup	Dheeraj, Rahul
7	Cropping Algorithm	Rahul, Kavya
8	Segmentation Algorithm	Dheeraj, Madhu
9	CNN Model Architecture discussion	All
10	Data pre-processing (CNN)	Kavya, Dheeraj
11	Training, Fit, evaluating results - CNN	Madhu, Rahul
12	Integration all modules into one	All

```
Train on 56380 samples, validate on 14140 samples
Epoch 1/6
62640/62640 [==============================] - 20s 315us/step - loss: 3.1221 - acc: 0.1135 - categorical_accuracy: 0.1135 - f1_m: 0.0163 - precision_m: 0.3932 - recall_m: 0.0084 - val_loss: 3.63
09 - val_acc: 0.1294 - val_categorical_accuracy: 0.1294 - val_f1_m: 0.1171 - val_precision_m: 0.3882 - val_recall_m: 0.0693
Epoch 2/6
62640/62640 [==============================] - 16s 256us/step - loss: 1.8202 - acc: 0.4350 - categorical_accuracy: 0.4350 - f1_m: 0.3610 - precision_m: 0.7604 - recall_m: 0.2513 - val_loss: 1.92
70 - val_acc: 0.4882 - val_categorical_accuracy: 0.4882 - val_f1_m: 0.5075 - val_precision_m: 0.5960 - val_recall_m: 0.4420
Epoch 3/6
62640/62640 [==============================] - 16s 252us/step - loss: 0.8338 - acc: 0.7208 - categorical_accuracy: 0.7208 - f1_m: 0.7141 - precision_m: 0.8503 - recall_m: 0.6178 - val_loss: 0.74
01 - val_acc: 0.7629 - val_categorical_accuracy: 0.7629 - val_f1_m: 0.7704 - val_precision_m: 0.8205 - val_recall_m: 0.7261
Epoch 4/6
62640/62640 [==============================] - 16s 253us/step - loss: 0.4606 - acc: 0.8423 - categorical_accuracy: 0.8423 - f1_m: 0.8396 - precision_m: 0.8997 - recall_m: 0.7874 - val_loss: 0.23
87 - val_acc: 0.9163 - val_categorical_accuracy: 0.9163 - val_f1_m: 0.9195 - val_precision_m: 0.9436 - val_recall_m: 0.8966
Epoch 5/6
62640/62640 [==============================] - 16s 254us/step - loss: 0.2925 - acc: 0.8985 - categorical_accuracy: 0.8985 - f1_m: 0.8984 - precision_m: 0.9296 - recall_m: 0.8693 - val_loss: 0.12
91 - val_acc: 0.9627 - val_categorical_accuracy: 0.9627 - val_f1_m: 0.9618 - val_precision_m: 0.9749 - val_recall_m: 0.9490
Epoch 6/6
62640/62640 [==============================] - 16s 253us/step - loss: 0.2121 - acc: 0.9273 - categorical_accuracy: 0.9273 - f1_m: 0.9275 - precision_m: 0.9460 - recall_m: 0.9097 - val_loss: 0.33
59 - val_acc: 0.8778 - val_categorical_accuracy: 0.8778 - val_f1_m: 0.8783 - val_precision_m: 0.8948 - val_recall_m: 0.8624
```

Fig. 5. Accuracy Metrics

VIII. Limitations

Our architecture uses Cloud for faster Computation. We solved this problem with the understanding of problem statement as a real world scenario. We would like to approach the problem in more realistic way by implementing the concept of Fog server. This will be an improvement to this project.

IX. Conclusion

In this project, we learnt how to develop a real-time ASL Fingerspelling detection using Android, Cloud Services, Machine Learning and Mobile Computing concepts. More particularly, in the completed product, we developed an Android application that takes input from user through the camera and processed in the cloud with least latency possible. We achieved an accuracy of 74.23%.

X. Acknowledgements

We would like to thank Professor Ayan Banerjee for giving us the opportunity to work on this project. We had a valuable experience during the course of Mobile Computing and we learnt many new concepts, technologies and gained the practical knowledge by implementing them.

References

[1] https://github.com/rwightman/posenet-pytorch

[2] https://www.kaggle.com/grassknoted/asl-alphabet