ASL Detection Using CNN

Amey Athale
Arizona State University
aathale@asu.edu

Jayesh Kudase
Arizona State University
jkudase@asu.edu

Arpit Arora
Arizona State University
apchavan@asu.edu

Aditya Chavan
Arizona State University
apchavan@asu.edu

ABSTRACT

An application that allows the user to signal out letters using American Sign Language and then using video images and palm detection algorithms predict first the letter and then combine to predict the whole word. Once the key-points are fetched from the palms they are produced to a pre-trained convolutional neural network that predicts the word.

Author Keywords

American Sign Language; Posenet; Convolutional Neural Networks; Dropout; Deep Learning

INTRODUCTION

According to World Health Organization, around 466 million people across the world have disabling hearing loss. Clearly, hearing loss is a more common problem than people deem it to be. There could be multiple types of hearing losses like Sensorineural hearing loss, conductive hearing loss or mixed hearing loss out of which the first one is the most common form. This could come into existence because of myriad of reasons. Aging, too much noise exposure, injury and existing health problems like diabetes and viral infections are few to name. Going by this pace by 2050 we would have over 900 million people with such problems.

American Sign language is a natural language is one of the key sign languages that serves the people with hearing disabilities. This is a kind of language which is visual as well as gestural at the same time. One can use hands, face expressions or torso movements to communicate effectively by these means. A slight change in visual representation like facial expression or body postures could inflect meaning into some of these signs. It is of high importance as it helps communicate to a whole together different group and at the same time makes a segment feels inclusive. Building upon this fundamental, we built this application that basically allows a person to record video using American Sign Language. We do certain amount of preprocessing on these videos before then are passed along the pipeline to achieve their end goal.

Once the videos are shot and produced, they are converted to frames using a python script [4]. The number of frames created for a video depends on the quality and the duration of the video. Usually a 5 second video produces about 150 frames. Next, they are fed to an algorithm which figures out the key points in the video image which enables data collection and further processing. Once these key points are detected we pass on these to a pretrained convolutional nueral network.

Convolutional Neural Network is a type of Deep Learning Algorithm which is mostly used for visual images. CNN basically uses a bunch of weights and biases that can be learnt, use them to assign importance to images and then use these to distinguish between them. For general purpose the filters are defined but if we have enough number of training samples, a CNN can learn these filters as well. It is of importance to note that the structure of a CNN takes inspiration from Neurons in human brain and is actually nothing but a fully connected set of neurons in one layer to another. Our convolutional neural network is trained on the American Sign Language dataset. When a video is fed, it first divides it into 2 second videos which each represent a letter. Depending on the length of the word total video size might vary from 6-8 seconds. Then these videos go through the convolutional neural network one by one and the CNN predicts the letter. These letters are then combined to form a word and that is how prediction works.

Here we can monitor the test and train loss for our initial training and change the parameters of convolution neural networks that could better suit the needs. We could add dropout layers that could be really helpful in certain scenarios. Dropout is a very essential technique often used to boost the performance of neural networks and not only that it saves a bunch of time as compared to training using other layers. If we look more closely, dropout is nothing but just ignoring few of the weights while we train the neural network. Or in other words, these neurons are not considered in either of the passes which are forward and backward pass.

Such an application would be of great use and finds a lot of application for people with hearing loss. We basically provide an end to end solution or ease and comfort of deaf people. This solution comprehends the American sign language and produces the text version of the words in acted in the videos. When implemented correctly and with high accuracy, it could help people in mitigating difficulties in their daily lives. The. Application if scaled up could potentially turn a natural language in text format easy to read and comprehend [2].

PROJECT SETUP AND REQUIREMENTS

For any project, there are always many requirements that we need to consider. One of the main necessities was generation

of alphabet data for validating our model, and then creating our testing data for the entire finger-spelled word.

As directed by professor Ayan in his lectures, we used the mobile application created by us as our first assignment to capture 5 second videos for each alphabet. Next, we randomly selected a total of 40 words, of which 20 were 3 lettered, and the rest were 4 lettered. Here, we captured videos of us finger-spelling the word, by roughly allocating 2 seconds per alphabet. These essentially conclude the data requirements.

Next, as per our original plan, we decided to train a model on our data to maximize accuracy. To accomplish this, we would need atleast need a Nvidia GTX 1080 GPU, with a minimum of 12 GB RAM enabled system. As none of our team members have such a system, we decided to use cloud computational services like Google Colab and AWS. Even if we ignore the massive time required to transfer this data to the cloud, the amount of training data is approximately 30GB, which after training, would be way more than what is freely allocated to us by these services. Still, we attempted to configure our system's GPU with Tensorflow and OpenCV on Anaconda environment. Every time we ran our algorithm, there was extensive memory leak which prevented us from performing model training on local systems.

Finally, as far as testing is concerned, we used Google Colaboratory with their GPU runtime instance to setup our image processing library(OpenCV). To ensure we are utilizing the most efficient processing unit, in our future experiments section, we have compared the execution times of our algorithm on Google Colab's GPU and TPU runtime instance.

IMPLEMENTATION

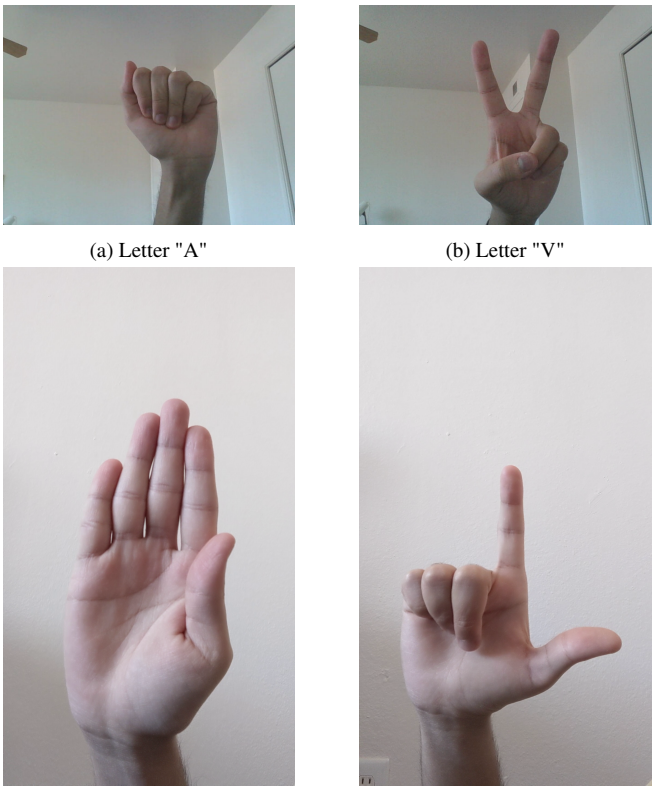
For this project, we were originally assigned quite a number of tasks that we needed to implement. The tasks were as follows:

- Collecting video data on alphabets from every member
- Developing an algorithm to directly consider data from palm and wrist
- Implement a Convolution Neural Network which has been trained on ASL alphabet data
- Algorithm to detect an entire finger-spelled word

Palm detection

The first task was to understand the input that needs to be given to a CNN for detection of a finger-spelled alphabet. Here, the only important part that we need is the palm and wrist captured in the video. As advised by Dr. Ayan Banerjee during his lectures, we considered two scenarios to accomplish this task.

First, we can capture the alphabet data such that the entire person's body is visible and the actions are performed approximately in the middle of the frame. Here, the challenge is to isolate the wrist and palms of the person. We ran a python script which used Pose-Net to generate spacial coordinates of the person captured in the frame. From those key-points, we can isolate the X and Y coordinates of the left and right wrist. We can now create a padded .png image(approximately 100 pixels in each direction) around these coordinates and consider this as our main input to our CNN.



(a) Letter "A" (b) Letter "V"

(c) Letter "B" (d) Letter "L"

Figure 1: Data captured using mobile application

Second, we took the alphabet video data in such a manner that only the palm is visible in the frame as seen in the above figure 1. This was done because PoseNet inherently poses a rather major problem. It only gives one coordinate for each wrist, and sometimes, it could be happen that another body part could be seen in the background of that point which could introduce serious noise into the image. By keeping just the hand video, we can ensure least possible noise augmentation.

Configuring CNN

For the training part of our project, we have decided to use a Convolutional Neural Network because of its advantages discussed in the introduction. Now, actually have two options here, either we can consider a 3D CNN or a 2D CNN.

A 3D CNN takes a video object as its input to its first layer. There are few challenges when trying to build a 3D CNN. First one is the lack of training data. Even the dataset that we need to train our model on contains image data on each alphabet. So, to use a 3D CNN, we would have to convert this image data into a video. Now, this architecture, ultimately breaks the video into multiple frames and trains on the individual frames. Hence, it is a redundant process to use 3D CNN when we have less training data.

Thus, a 2D CNN seems to be a good idea. Our initial plan was to develop our own 2D CNN, train the model on our generated

data and on the Kaggle dataset [1] to achieve higher accuracies. We considered using Google's Inception V3 net for this task as it has been proven to be one of the best performing model yet. We wrote a python script which will convert our video data into multiple frames as shown by figure 2, which would then be converted into a numpy array, which is a standard input format for Inception V3. We attempted to perform training this model as a whole and in parts, but every time we kept facing memory issues. Unfortunately, we lacked the computational resources required to train 2 Gigabytes worth of Kaggle image data and 7 GB worth of our own video data. Thus, we decided to make use of the pre-trained 2D CNN model provided to us by professor Ayan Banerjee.

```
./test/Amey_X.mp4 ['B', 'L', 'I', 'L', 'L', 'B', 'B']
-------

./test/Amey_Q.mp4 ['I', 'I', 'L', 'G', 'K', 'L', 'I']
-------

./test/Amey_J.mp4 ['C', 'R', 'H', 'C', 'C', 'C', 'I']
-------

./test/Amey_A.mp4 ['G', 'I', 'I', 'C', 'G', 'I', 'G']
-------

./test/Amey_E.mp4 ['G', 'G', 'G', 'I', 'I', 'I', 'I']
-------

./test/Amey_L.mp4 ['G', 'B', 'G', 'H', 'K', 'G', 'B']
-------

./test/Amey_Z.mp4 ['K', 'B', 'F', 'C', 'C', 'O', 'O']
-------

./test/Amey_S.mp4 ['D', 'L', 'L', 'L', 'L', 'L', 'L']
-------

./test/Amey_I.mp4 ['L', 'I', 'I', 'D', 'Y', 'I', 'L']
```

Figure 2: Creating multiple frames for each video file. We observe here that without pre-processing, the accuracy for each alphabet is not very good.

Preprocessing

As the video file created by each member for each alphabet was roughly the size of 10-15 MB, there was a need for some pre-processing before passing it on to the CNN. First, we performed video compression to reduce the size of the file by a scale of 5 as seen in figure 3. Next, we wrote a python code to generate 10 frames per second from our reduced video file. Finally, we used the code given to us by the prof to reduce the image size to 200 x 200 as was required by the pre-trained CNN model.

Detection of finger-spelled word

For this section, we start by gathering our own testing data. The initial task we completed was finger-spelled 40 different three or four lettered words. The final video created was approximately 6 seconds and 8 seconds long for 3 and 4 lettered word respectively. We managed to keep each alphabet for almost 2 seconds.

As explained in the above sections, we generated 10 frames per second, and 2 seconds per alphabet, hence, for a 3 lettered word, we have a total of 60 frames. Using these videos, we created a time-series frame list for each video. This frame list was passed in the order of arrival to the CNN predict function to create a prediction vector. Corresponding to each frame, we generated a prediction vector of size 26. Each element of this vector contains the probability that the gesture in this frame belongs to which corresponding alphabet. The alphabet corresponding to the element with the highest probability is returned for each frame.

```
[MoviePy] >>>> Building video movie_resized.mp4
[MoviePy] Writing audio in movie_resizedTEMP_MPY_wvf_snd.mp3
100%|██████████| 108/108 [00:00<00:00, 1105.15it/s][MoviePy] Done.
[MoviePy] Writing video movie_resized.mp4

100%|██████████| 146/146 [00:07<00:00, 19.97it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: movie_resized.mp4

----------------------------

Word you signed was :  L
```

Figure 3: Video Compression script

Word Segmentation

For our project, we have tried a basic word segmentation approach based on time spent on each word. Here, we divided the frames list into 3 or 4 approximately equal sections depending on the length of the input word as seen below in fig 4. Then, for each section, we returned the label with the maximum cardinality. Finally, we concatenated labels obtained from each section to form the entire word detected by the algorithm.

```
Ground Truth: FIRE
Segmenting word into  4 parts
['V' 'V' 'V' 'V' 'V' 'V' 'V']
['V' 'D' 'D' 'D' 'D' 'D' 'D']
['D' 'D' 'R' 'R' 'R' 'U' 'R']
['I' 'V' 'D' 'D' 'D' 'D']
--------------------

Ground Truth: VISA
Segmenting word into  4 parts
['V' 'V' 'V' 'V' 'V']
['I' 'I' 'I' 'I' 'I']
['S' 'S' 'S' 'S' 'S']
['A' 'A' 'A' 'A' 'A']
----------------------

Ground Truth: KEY
Segmenting word into  3 parts
['W' 'B' 'B' 'B' 'B' 'B' 'B' 'B']
['B' 'R' 'B' 'C' 'C' 'L' 'C' 'C']
['C' 'C' 'L' 'L' 'A' 'L' 'Y']
```

Figure 4: Word segmentation script output

EXPERIMENTS AND RESULTS

We performed various experiments to test the accuracy of the trained model. As per the first task, our group recorded 5 seconds video of each alphabets. Now, we tried training our own model on the data set link provided by Dr. Ayan Banarjee. The data set is in total of 1 GB. The model was put to train on a system with 8 GB RAM and NVIDIA GeForce 750 M processor which was configured to run deep learning models on it. However, within first epoch, the system hung up due to memory issues. As a result, we moved our focus of training on Google Colab. A subset of training data i.e. just 250 MB

took about 3.5 hours to upload on the drive. The drive was mounted to work with the script [3] on Colab and it was found that the dataset access was extremely slow. The images of just a single alphabet was taking very long (waited uptil 30 mins) to be accessed completely. Due to the above resource constraints, we shifted our efforts to work with provided 2D CNN.

The following experiments were performed on the videos of letter and word individually to drive the nature of experimentation. The experiments that we carried out were as follows.

Experiments on Videos of Alphabets

This video captured as a result of step 1 were then fed using our python script to predict the letter associated with each video, However, it was observed that the trained model failed to generalise on the nature of videos that were captured using mobile phones. The aspect ratio of the frames that were generated from the video caused problems in predicting the associated sign correctly. As a result, the video compression technique was used wherein the complete video was compressed into 200 X 200 format and as a result the frames that were generated were intuitively thought to produce better results. This resulted in 6 (A, B, I, L, R, V) out of 26 alphabets being predicted correctly.The precision, recall and f-score respectively for this result is (0.1346, 0.1538, 0.1410)

Experiments on Videos of Words

For this experiment, 20 three letter words and 20 four letter words were taken into consideration. A conclusion from the above experiment was derived that the video recorded by us resulted in poor accuracy. This gave us an idea to test the model on test images which were similar to which it was trained on. The images from the dataset that was provided by Dr. Ayan Banerjee were taken as a test set. These images are quite different from what the model is trained on. However, these images are of the same dimension i.e 200 X 200. We thought of testing our segmentation and word spelling algorithm on the video of these images. Therefore, we took the alphabet images and turned them into videos. For e.g., we took an images of C, R and Y were taken from the dataset [1] and a video was made out of them with each alphabet spanning for 2 seconds. Thus, a 6 seconds video of 'CRY' was fed to our word spelling algorithm. The words that we produced a video for along with their predictions and accuracy is as shown in table 1.

Thus, an average accuracy of 78.33% in a word prediction was obtained overall for 40 words with the above strategy.

Test Word	Prediction	Accuracy (%)
ACE	ACE	100
CRY	CRY	100
VAN	VAC	66.67
VOW	VOW	100
CAB	CAB	100
CAT	CBR	33.33
DOG	FPI	0
BAT	BAU	66.67
FAN	FAC	66.67
JOB	GOR	33.33
GUN	GUC	66.67
LIE	LIE	100
KEY	KEY	100
FIX	FIX	100
VET	VEL	66.67
MAP	PAP	66.67
ODD	ODD	100
SAW	SAW	100
CAP	CAP	100
SUN	SUC	66.67
DAMP	DAPP	75
AXIS	AXIS	100
BUNS	BUCS	100
CROW	CROW	100
DATE	DAXD	50
DRAW	LRBW	50
EGGS	EGLS	75
FILM	FILO	75
FOOD	FOOC	75
GIFT	GIEY	50
FIRE	FIRE	100
KING	KICG	75
WAGE	WAGE	100
HEAD	HEAD	100
ROSE	ROSE	100
NODE	CODE	75
EPIC	EPIC	100
SHOE	SHOE	100
MEAT	PECL	25
VISA	VISA	100

Table 1: Words and their Prediction

LIMITATIONS

- Signed languages use palms along with facial expression, movements of torso, and shoulders to express more accurately and fluently. Our project only considers palm video, and this needs to be extended further to taking the video of humans to predict the signed words better.
- Our algorithm uses Pose-Net, and the presence of another body part in the background could insert noise and reduce the prediction accuracy. Users with special needs who don't have the full range of motion of their hands could face an issue. Instead of Pose-net better algorithm to detect the finger joints or use of auto encoders to denoise the image could be done.
- A lack of large dataset to train the ML algorithm is an issue. Training data with a large number of people signing could be used to make the model generalise better on novel test data.

CONCLUSION

In our project, we have successfully developed an application that helps people with special needs to communicate with average people better. We used videos of alphabets in the American Sign Language as the input and used a palm detection algorithm to get the coordinates and trained the data on a 2D CNN. To detect a complete word, we fingerspelled 40

different words and used the word segmentation approach to identify complete words.

As we can see from the results, the application has the potential to help special people to mitigate the problems faced in their daily life. Using better algorithms and more training data, the prediction results could be much better for new test data.

ACKNOWLEDGMENTS

Our Team is grateful to Dr. Ayan Banerjee for their invaluable encouragement, suggestions, comments, and guidance throughout the course of our project. We got to learn about new concepts, technologies, and how they work while facing the real world challenges.

REFERENCES

1. Akash. 2018. Image data set for alphabets in the American Sign Language. Kaggle. (2018). https://www.kaggle.com/grassknoted/asl-alphabet.
2. Valentin Bazarevsky and Fan Zhang. 2019. On-Device, Real-Time Hand Tracking with MediaPipe. Blog. (19 July 2019). https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html.
3. Anuj Shah. 2017. CNN implementation. Github. (May 2017). https://github.com/anujshah1003/own_data_cnn_implementation_keras/blob/master/updated_custom_data_cnn.py.
4. Prasanth Sukhapalli. 2019. For Generating Key Points using Tensorflow's PoseNet. Github. (15 July 2019). https://github.com/prashanthnetizen/posenet_nodejs_setup.