Sign Language Action Detection from Video data
W251 final project

Abstract

The ability to use deep learning networks for sign language recognition has gained considerable attention over the past few years. There have been considerable advancements in using both 2D and 3D data representing letters, words and actions of the sign language to their written language translation. We explore, implement and test a variety of techniques for achieving the above and describe the challenges faced therein. We also describe the process of engineering an end-to-end solution - from data gathering and pre-processing to model development, cloud training and edge device deployment in this paper.

Introduction and motivation

There are close to 500 million hearing-impaired persons in the world today. And while sign language is the most effective form of communication for persons with hearing disabilities, only about 70 million people in the world know how to use and communicate with sign languages. This fact proves that tools which allow people with sign language skills to communicate with people without sign language skills can be immensely effective.

Such a tool would allow automatic translation from sign language to a written language like English. Two major use cases come to mind.

- One or more hearing person(s) without any sign language skills trying to interpret the meaning of a word/sentence from a hearing-impaired person's sign language indication.
- One or more hearing impaired person(s) without sign language skills trying to interpret the meaning of a word/sentence from a hearing-impaired person's sign language indication

In both of these cases, there is a language barrier between the conveyer of the message and the recipient of the message. We aim to bridge this gap by implementing a model with reasonable accuracy, trained on an appropriate sign language dataset and deploying it on a device with a camera.

Research and Related work

Over the years, work done in this area broadly falls into four categories based on the type of data being processed. The categories also reflect the nature of the problem being solved.

a. Single letter recognition using image data
b. Finger spelling sequence recognition from videos
c. Word level sign language action classification from videos
d. Sentence construction from type b and type c data described above

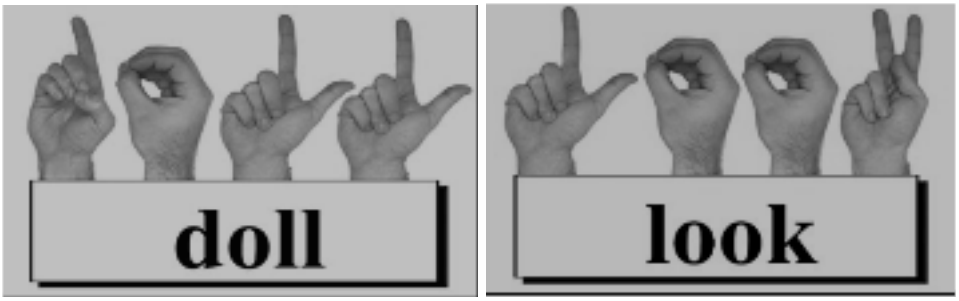
a. Single letter recognition using image data

	**L**
	**C**

The earliest work in this area involved the usage of “data gloves”. These systems mostly concentrated on finger signing in which the user spells each word with finger signs corresponding to the letter of the alphabet. In addition, there was an inconvenience of wearing the data gloves at all times.
A more “model-based approach” uses hidden markov models (HMMs) with the assumption that only coarse grained features of hand-shape, orientation and trajectory are sufficient to produce reasonably accurate results.
Thad Starner and Alex Pentland[1] is a notable reference in this area.

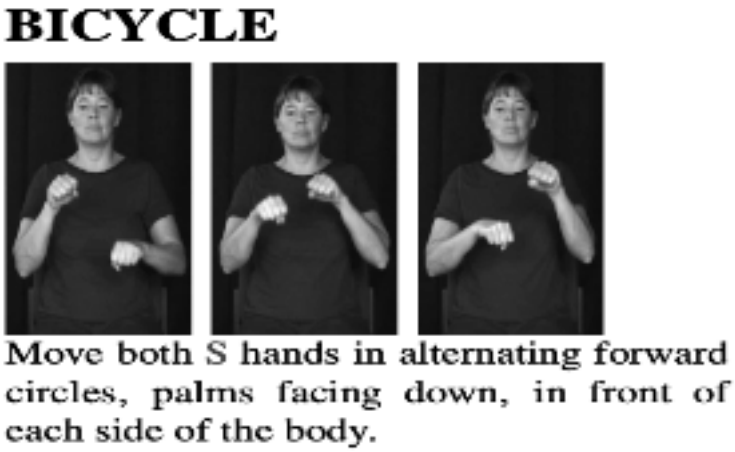
Most of the modern work in this area involves capturing images and extracting features using Convolutional Neural Networks.
For e.g. Tao, Leu and Yin[2] use a CNN model with multi view augmentation and inference fusion. The main emphasis here is on depth images which contain distance information from the camera plane to the objects in camera view. A lot of image transformation techniques are used - rotating, scaling, shifting, flipping, shearing etc. A multi view augmented strategy is then used to generate additional images from multiple perspectives. The model consists of a CNN layer for feature extraction with a fully connected layer and a softmax function for classification.

b. Finger spelling sequence recognition from videos



As per the ideas presented by Shi, Del Rio [3] and others, most of the research done in this area involves a sequence model which captures the temporal nature of the data. Models involve a LSTM with features developed from a CNN. Also, in most cases, images from a dataset like imagenet pertained with models like VGG-16 have been effectively used.

c. Word level sign language action classification from videos



This category of sign language interpretation involves using videos which indicate an action. We do not use finger spelling here, but rather an action which directly corresponds with the word being indicated. For e.g. the word “BICYCLE” (a noun) is indicated as above. Similarly words like CHAT, RUN, TALK have their corresponding actions.

The main article researched was “Spatial-Temporal Graph Convolutional Networks for Sign Language Recognition” by Amorium and others[4] in which the authors propose a propose a new approach of Spatial-Temporal Graph Convolutional Network to sign language recognition based on the human skeletal movements.

The method uses graphs to capture the signs dynamics in two dimensions, spatial and temporal, considering the complex aspects of the language. Also, the dataset mentioned and used in the paper (ASLLVD dataset) is the primary dataset used in the project.

Datasets

The dataset used in the project was the ASLLVD (American sign language lexicon video dataset) present here:
http://csr.bu.edu/asl/asllvd/annotate/index.html
The dataset has videos captured using four synchronized cameras, providing: a side view of the signer, a close-up of the head region, a half-speed high resolution front view, and a full resolution front view. We only use the full resolution front view in our analysis. In order to be able to compare our model performance with prior studies[4] on the same data, we select the 19 words (i.e., classes) that are consistent prior studies. There are 142 videos associated with those 19 words. For some words, there are multiple variations of sign languages. After removing obscure variation, we are left with 128 videos, of which 105 videos are used as training and 23 videos are used as test.

Different Modeling Approaches

The following modelling approaches were tried with the main evaluation criteria being the validation accuracy. In some cases (as shown below), training execution was preceded by a significant amount of pre-processing.

#	Model	Features	Validation accuracy
1.	3D Convolutional Neural Network for video classification using Keras	Built and tested on UCF-101 dataset No transfer learning	27%
2,	2-model architecture: detector + classifier	Uses transfer learning with ResNet	12%
3.	Open-pose with Optical-flow for pre-processing and CNN for classification	No transfer learning	28%
4.	EfficientNet	Open pose for pre-processing Optical flow for image transformation	74%

		EfficientNet for classification	

Based on the above findings, it was decided that EfficientNet with open-pose and optical flow were the best candidates for deploying the model on TX2.

Final model and inspiration

Model inspiration

Action classification is a harder problem to solve, and certainly less studied compared to image classification. Searching “image classification” in Google Scholar yielded 502,000 search results, while searching “action classification” only yielded 12,200 search results. The goal of this project is to detect sign languages via videos. But is there a way we can represent complex actions in a lower dimensional space (such as a 2D image) and leverage deep learning image classification algorithms to classify actions?

Optical flow depicts the pattern of apparent motion of object, surface, and edges[5]. The state of the art with regard to the quality of the optical flow estimate has still been defined with traditional methods[6], such as Lucas–Kanade algorithm dated back to 1981[7]. Optical flow estimation techniques take sequences of ordered images and attempt to capture the motion between two image frames. “It is widely used in computer vision to find moving objects from one frame to another, to calculate the speed of movement and the direction of motion and to determine the structure of the environment.”[5] It’s used in traffic analysis and vehicle tracking[7]. It can also be used for industrial automation purposes[8].

A handful of research papers have explored the idea of integrating optical flow with action recognition, but they all tackled this idea from different angles.

- Sevilla-Lara, Laura, et al.[9] argued that “trajectories are not the main source of information in optical flow” as they observed a small decrease in action recognition accuracy when they randomly shuffle the frames. This is not a very intuitive conclusion as one would have thought the temporal information contained in the optical flow is required to capture motion, and in turn essential for action recognition. It’s our belief that the reason removing temporal information was inconsequential in their experiments was because the actions they tried to categorize (“ApplyEyeMakeup, Fencing, Knitting, PlayingGuitar, HorseRiding”) are so distinctively different that “a single frame often contains enough information to correctly guess the category”. The authors primarily utilized optical flow as “a representation of the scene” as opposed to actions themselves. This approach will not work for our use case because, without temporal information, all the sign videos are merely people waving their hands randomly.

- Gupta, Arpan, and M. Sakthi Balan[10] fed Farneback dense optical flow from each two consecutive frames from each video into a CNN model and predicted the label by taking the majority vote. This approach produced accuracy lower than the state of the art[1] on the KTH data, and the authors contributed the lower accuracy to the use of one motion feature with consecutive frame pair optical flow.
- The method proposed by Mahbub, Upal, Hafiz Imtiaz, and Md Atiqur Rahman Ahad.[1], as referenced in the paper above, is considered the state of the art for the action classification of the KTH data. There are four steps in the proposed method. The authors (a) used Lucas Kanade optical flow estimation method to detect movement; (b) applied RAndom SAmple Consensus (RANSAC) algorithm to prune the point of interested outputted from the optical flow; (c) divided each human subject into nxn blocks and calculated the average percentage of change in the number of interest points for each block; and (d) ran Euclidean Distance classifier or Support Vector Machine (SVM)-based classifier for action classification. In the KTH data, there are six actions: boxing, handclapping, running, jogging, handwaving and walking. This method has proven to be effective in detecting full body movement. However, whether it can be equally effective in distinguishing movement of a smaller scale (such as movement of arms, hands, and fingers only) is yet to be validated.

The prior studies have shown that optical flow contains rich temporal information and can potentially be used as an input for action recognition.

Optical Flow in OpenCV

OpenCV provides a two-step process to track feature points in a video.

1. Detect some Shi-Tomasi corner points using `cv2.goodFeaturesToTrack()`

2. Track points of interests identified in Step 1 using `cv2.calcOpticalFlowPyrLK()`, which implements Lucas-Kanade optical flow estimation method

There are three key parameters in `cv2.goodFeaturesToTrack()` that we can specify to increase or decrease the number of key points detected:

- `maxCorners`
    - Definition: Maximum number of corners to return. If there are more corners than are found, the strongest of them is returned.
    - Effect: Increasing this parameter increases the number of features returned, subject to meeting the requirements specified by `qualityLevel` and `minDistance`.
- `qualityLevel`

    - Definition: Parameter characterizing the minimal accepted quality of image corners
    - Effect: Lowering the quality level increases the number of features or noises returned. The difference between features and noises is that noises are points that do not contribute to the motion detection of interest.
- `minDistance`
    - Definition: Minimum possible Euclidean distance between the returned corners
    - Effect: Decreasing this parameter increases the concentration of high-quality features in small areas.

In general, the greater the number of features returned, the more temporal information can be captured by the second step, `cv2.calcOpticalFlowPyrLK()` but also the more noises are contained in the optical flow.

There are two key parameters in `cv2.calcOpticalFlowPyrLK()` that dictates how movement is tracked:

- `winSize`
    - Definition: size of the search window at each pyramid level
    - Effect: Increasing this parameter enables larger movement between frames to be captured
- `maxLevel`
    - Definition: 0-based maximal pyramid level number; if set to 0, pyramids are not used (single level), if set to 1, two levels are used, and so on…
    - Why pyramids? Lucas-Kanade (L-K) method assumes that the displacement of the image contents between two consecutives frames is small, so L-K method itself (without pyrimads) does not work well with large motions. To circumvent this problem, OpenCV “runs optical flow at the top of pyramid first mitigates the problems caused by violating the L-K assumptions of small and coherent motion; the motion estimate from the preceding level is taken as the starting point for estimating motion at the next layer down”[11].
    - Effect: Increasing this parameter enhances the ability to detect a wider range of motions.

Below are two examples of how OpenCV draws optical flow frame by frame. Both signers below signed the word “again”. Both videos were processed using the same set of parameters. The words and figures at the top left corner were excluded from being tracked as features. (link to code)

- The good example: https://github.com/AngelaWuGitHub/w251-project/blob/master/writeup_videos_images/cv2_AGAIN_c9_optical_flow.gif

- The bad example: https://github.com/AngelaWuGitHub/w251-project/blob/master/writeup_videos_images/cv2_AGAIN_c4_optical_flow.gif

In the first (good) example, OpenCV detected key points in the signer's neck, face, knuckles, finger tips, and thighs, and was able to track the hand movement almost perfectly the first few frames. When the signer's right hand got closer to her face, OpenCV confused some of the right-hand features with neck/face features, and the estimated optical flow became erroneous.

In the second (bad) example, most key points detected by OpenCV are the patterns of the signer's tank top, which we are not interested in. Only two key points from each hand were detected. This exposed a huge flaw in using Shi-Tomasi corner detector for feature extraction. Shi-Tomasi corner detector picks up corner points that may or may not be part of the action. If we want this research to be implemented in the real world, we need to make sure the algorithm/process we design is not sensitive to cloth patterns or various backgrounds. One way to circumvent this is to prune key points that appear to be static using RANSAC algorithm[5]. But there are other ways to detect meaningful hand/arm gestures, which we explored next. Another issue with estimating optical flow using OpenCV is that L-K method doesn't work well when multiple features collide into each other. We saw inaccurate optical flow being produced when the signer's right hand moved close to her face in the first example as well as when the signer's right hand moved in front of her body in the second example. This issue was observed in almost all videos we processed.

We tried different sets of parameters in `cv2.goodFeaturesToTrack()` and `cv2.calcOpticalFlowPyrLK()` (discussed at the beginning of the section) to improve the optical flow estimation in OpenCV, but we were unable to produce satisfactory results.

Another issue we discovered is that OpenCV optical flow estimation is sensitive to photo contrast. Depending where or how sign videos are made (e.g., in the broad daylight or in a dark room, different camera settings), the resulting optical flow estimation may be different. This is a huge drawback since we need a robust representation of motions in 2D that is not impacted by the external environment.

Side note: for some videos, there are other background objects that interfere with the optical flow estimation. If the object isn't directly behind the signer, we can restrict the features detected to be within the range of the signer's upper body. We tried using ` haarcascade_upperbody.xml` to detect upper body but it was unable to detect the upper body correctly most of the time. We had better success in detecting the face using `haarcascade_frontalface_default.xml` and inferred the upper body. An example is shown below.

https://github.com/AngelaWuGitHub/w251-project/blob/master/writeup_videos_images/cv2_AGAIN_c8_upper_body.png

(link to code)

But this does not resolve the three issues we articulated above, so we decided to look to different methods for optical flow creation.

OpenPose for Feature Extraction

OpenPose is an open source package that provides “real-time multi-person keypoint detection for body, face, hands, and foot estimation”. (link)

We used OpenPose to estimate body and hand movement. OpenPose tracks 25 key points on the body and 21 key points on each hand.

OpenPose returns two types of files for each video processed: a skeleton video and a series of JSON files, one for each frame (link to an example JSON file).

Below are the skeleton videos that correspond to the two optical flow examples above:

- https://github.com/AngelaWuGitHub/w251-project/blob/master/writeup_videos_images/openpose_AGAIN_c9_skeleton.mov (have to download the .mov to view it)
- https://github.com/AngelaWuGitHub/w251-project/blob/master/writeup_videos_images/openpose_AGAIN_c4_skeleton.mov (have to download the .mov to view it)

As you can see, while OpenCV had trouble producing sensible optical flow for the second video example above, OpenPose was able to estimate the movement accurately. This demonstrates the effectiveness of OpenPose.

Out of 67 (=25+21*2) key points that OpenPose tracks, we are particularly interested in the following 15 key points (the reference to the key points numbers below can be found here: body and hand):

- Elbows (Body 3 and 6) and wrists (Body 4 and 7): All the sign languages are upper body movement, so elbows and wrists are important to keep track of.
- Nose (Body 0): When a signer expresses a certain emotion, such as sadness, he/she will move his head slightly, so we also keep track of the nose movement.
- Finger tips (Hand 4, 8, 12, 16, and 20 for each hand): The movement of most hand key points are highly correlated, and OpenPose sometimes has trouble accurately tracking each key point. So we selected five finger tips to represent the hand motion.

Manual Optical Flow

As mentioned above, OpenPose outputs one JSON file for each frame it processes. In the JSON file, OpenPose provides the estimated placement for each key point as well as the confidence score of the estimated location. We keep track of the locations for each of the 15 key features frame by frame and draw optical flow for one feature at a time.

Below are the two optical flows manually created based on OpenPose JSON files that correspond to the two OpenPose skeleton videos in the last section:

- https://github.com/AngelaWuGitHub/w251-project/blob/master/writeup_videos_images/openpose_AGAIN_c9_sim0_flow.png
- https://github.com/AngelaWuGitHub/w251-project/blob/master/writeup_videos_images/openpose_AGAIN_c4_sim0_flow.png

Although they don't look exactly the same, they do bear a resemblance to each other.

The choice of colors will be discussed in the next section.

Data Augmentation

Data augmentation is an important strategy that enables modelers to drastically increase the diversity of data going into the model.

The standard data augmentation techniques we deploy include:

- Width/height shifting
    - The signer may not be at the center of the frame, and we want to account for those scenarios in our model
- Zoom
    - The signer may be closer or further away from the camera, and we want to account for those scenarios in our model

Standard data augmentation techniques that are not used by our model:

- Rotation
    - Sign languages are conducted in a standardized position. Meaning might change if we rotate the optical flow.
- Horizontal flip
    - All the signers in our training/test videos are right-handed. Horizonal flip would not help improve the model performance.
- Vertical flip

    - We do not expect a signer to appear upside down in our videos or future application.
- Channel shift
    - The choice of colors in optical flow has temporal meaning. Channel shift will overthrow the color design.
- Brightness
    - The choice of colors in optical flow has temporal meaning. Adjustment in brightness will overthrow the color design.

We also deploy additional data augmentation techniques so that optical flow can capture as much relevant information as it possibly can from the video.

1. Optical flow simulation based on OpenPose confidence
    a. As mentioned in the previous section, the JSON file created by OpenPose includes the confidence score for estimated pixel location. From what we observed, OpenPose tends to be a lot more certain with nose, elbow, and wrist positions than fingertip positions. The confidence score for fingertip position is sometimes less than 20%.
    b. To get a more thorough representation of key point location, we decided to simulate pixel locations for each of the 15 key points and for each frame based on their confidence scores, and re-create optical flow.
    c. Here are three steps we took for simulation
        i. The first step to simulation is to make an assumption on the distribution of key point location. Without additional information, the best distribution assumption we can make is bivariate normal distribution with correlation coefficient of 0 and equal variance.
        ii. Given the confidence score for the estimated pixel location, we can estimate the variance of `x` and `y`.
        iii. Given the distribution assumption and estimated variance, we can calculate the possibilities that a given key point lands on a given pixel in a frame
2. Color gradient
    a. Based on our literature review of optical flow, prior studies only used a singular color to represent optical flow, with red or green being the most common color choice. Optical flow depicted using a singular color doesn't contain information regarding the direction of motion. For example, moving a hand from left or right or from right to left would appear exactly the same in optical flow of one color. In order to enrich the temporal information in optical flow, we used color gradient to convey the direction of motion.
    b. For primary colors (i.e., red, green, blue), color gradients can be easily defined. For example, if we want the optical flow to start with a primary color (e.g., red) and end with a darker shade of that color, we would set the starting color as (255, 0, 0) and ending color as (50, 0, 0). Similarly, color gradients can be easily

defined for secondary colors (i.e., cyan = green+blue, magenta = red+blue, yellow = green+red).

c. In total, we define seven colors and their corresponding features are listed below:
    i. White: nose (Body 0)
    ii. Red: elbows (Body 3 and 6)
    iii. Green: wrists (Body 4 and 7)
    iv. Blue: left thumb (Hand 4)
    v. Yellow: left index, middle, ring, and pinky fingers (Hand 8, 12, 16, and 20)
    vi. Magenta: right thumb (Hand 4)
    vii. Cyan: right index, middle, ring, and pinky fingers (Hand 8, 12, 16, and 20)

d. Some signers perform signs faster or slowlier than others, which results in some sign videos having less or more frames. Since signing speed is irrelevant to the meaning it communicates, so is the number of frames. To account for the difference in the number of frames between videos, we set the starting and ending colors for a given feature to be the same. If a video has more frames, color progression is smaller between frames.

3. Overlay order
    a. We draw optical flow for one feature at a time. Depending on the order we draw the features, the optical flow may turn out to be slightly different. For example, if a red feature crosses a green feature, and we happen to draw the red feature first and the green feature second, we will see the intersection of the two features to be green. If we were to draw the green feature first and red feature second, the intersection would have been red. Since the overlay order does not convey any information about the motions, we randomize the order features are drawn so that Neural Network model does not accidentally pick up irrelevant information from the optical flow.

Model: ResNet50 vs. EfficientNetB0

Since we only have 105 training videos, we started by exploring transfer learning. ResNet and EfficientNet are both well known models for image classification. There are many variations of ResNet and EfficientNet, we selected ResNet50 and EfficientNetB0 for our analysis. Below is a comparison of the number of parameters in those two models:

# of Trainable Parameters	ResNet50	EfficientNetB0
Transfer Learning	38.9K	24.3K
Retrain Weights	23.6M	4.0M

As shown in the table above, ResNet50 requires more trainable parameters compared to EfficientNetB0, especially when all weights need to be retrained.

For transfer learning, we replaced the original output layer with our output layer of 19 nodes. We also tried adding additional Dense layers before the output layers with different dropout rates. All different transfer learning attempts with ResNet50 and EfficientNetB0 yielded poor results with training accuracy less than 30%. This means all those models are under-fitting.

We started un-freeze certain layers. As expected, training accuracy increased when we un-freezed more and more layers due to the increase in trainable parameters. For ResNet50, we were able to increase training accuracy to almost 100% with less than 20 epochs but the test accuracy remained to be low. On the other hand, EfficientNetB0 performed much better with training accuracy close to 90% and test accuracy of 74%.

The final specification of our model is:

- Input = (224, 224, 3)
- Batch size = 32
- # of epochs = 15
- # of trainable parameters = 4,031,887

The model is quite stable and not much hyperparameter tuning was performed. Increasing and decreasing the number of epochs and batch size slightly does not drastically change the test accuracy. Test accuracy of 74% was better than the method proposed in Cleison Correia de Amorim et al[4] (56.82%) but was worse than the state of the art referenced in that paper (85%).

Note: The three added data augmentation techniques are extremely important in achieving the final test accuracy of 74%. When we only deployed the standard data augmentation techniques with the same final model specification, the test accuracy was less than 30%.

Cloud Configuration

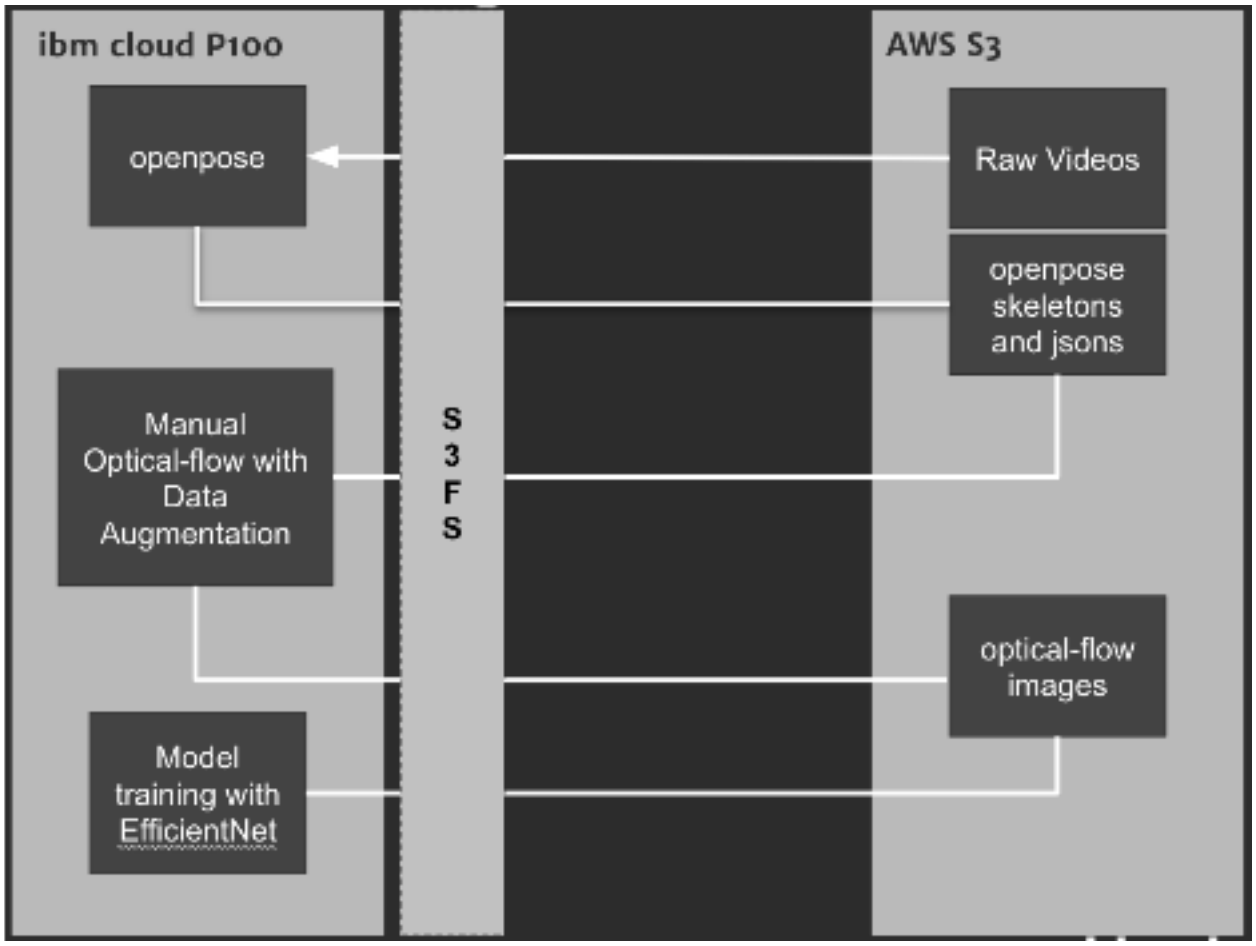
Preprocessing and model training was done on an IBM cloud based P100 instance with the following configuration.

Hardware:
NVIDIA TESLA P100
Image: Ubuntu Linux 18.04
Size: 8 vCPU
Memory: 60 GB

Software setup:

Ubuntu Linux 18.04
CUDA Release 10.2 - NVIDIA GPU Version
OpenCV 3.2
OpenPose 1.6 built from source: https://github.com/CMU-Perceptual-Computing-Lab/openpose
s3fs-fuse from https://github.com/s3fs-fuse/s3fs-fuse

The overall workflow on the P100 using AWS s3 and the primary storage layer has been shown below:



Model Inference on TX2

Jetson TX2 is the most power-efficient embedded AI computing device. This 7.5-watt supercomputer on a module brings true AI computing at the edge. It's built around an NVIDIA Pascal™-family GPU and loaded with 8GB of memory and 59.7GB/s of memory bandwidth. It opens new worlds of embedded IoT applications, including home robots, and intelligent gateways with full analytics capabilities.

- **The Plan and Installation**

We ran our inference in Jetson TX2 as it helps keeping the data close to the data sources.This enables real-time data processing at a very high speed,which is a must for complex IoT solutions with machine learning capabilities. On top of that, it mitigates network limitations, reduces energy consumption, increases security, and improves data privacy.
The plan for running inference in TX2 are as follows:

1) Download the trained EfficientNet Model from cloud
2) Run real time camera on TX2 that captures the action we do through OpenCV
3) Every frame captured by OpenCV is passed through OpenPose for key point detection.
4) Run prediction for last <n> frames
5) Display Prediction on UI

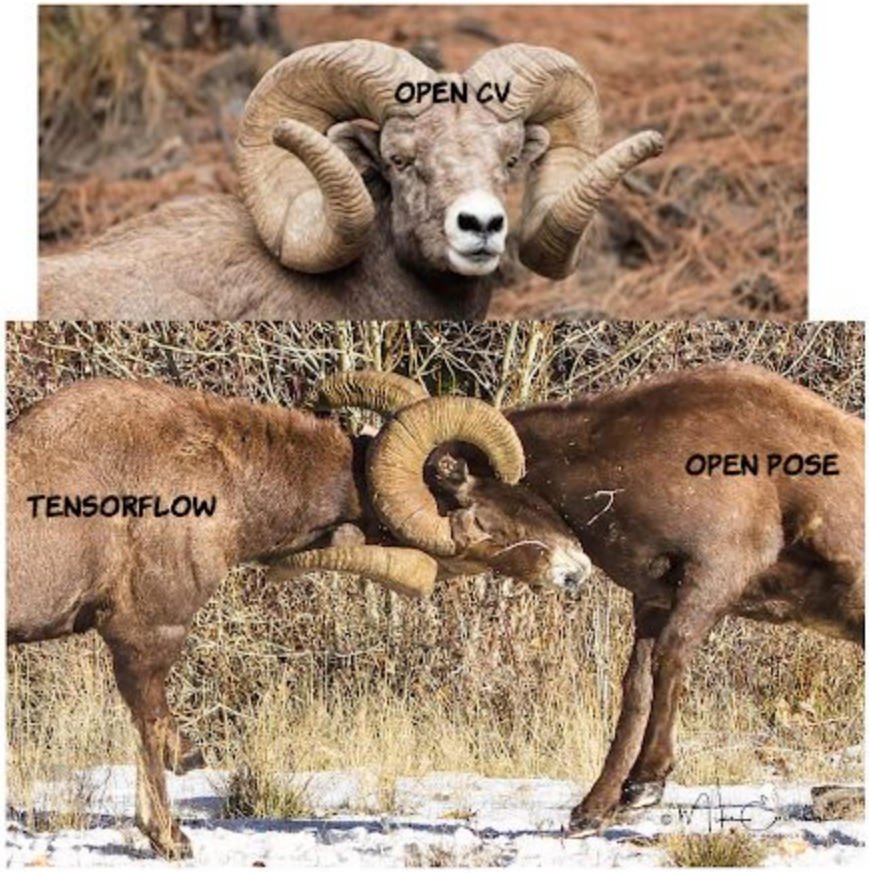
We have played around the ideal number of <n> frames to do the prediction, have found 20-25 frames is ideal to get a good prediction.

The three main components that are needed in TX2 to run inference are OpenCV,OpenPose and Tensorflow. Deploying complex deep learning models onto small embedded devices is challenging.Our case was no exception. It was quite a challenge to get the correct version of the components that are compatible with cuda 10.0 and jetpack 4.3. The below figure shows the different components that are installed in TX2 through docker containers .

OpenCV has been optimized to run on cuda enabled GPU devices. OpenPose has also been optimized for speed to use the BODY_25 caffe model and a smaller resolution. Tensorflow version is chosen such that it is compiled with cudnn 7.6.3. Instead of loading the keras model (h5) directly in TX2, we froze the model and converted to protobuf file that contains the graph definition as well as the weights of the model.Loading pb file for inference increases the memory bandwidth than loading the h5 keras model.

- **Issues on Tx2**

Jetson TX2 comes with only 8GB RAM and sometimes it's not enough . The two main memory hogging components are Tensorflow and Openpose. When tensorflow loads on a GPU machine, it allocates half of the memory to itself irrespective of its needs . We fixed this problem by explicitly setting memory of 1.5 GB for Tensorflow.
Openpose on the other hand needs a minimum of 4.5 GB to run basic body pose with BODY_25 model and with hand detection keypoints it needs further more memory.



Hence, many times during inference we faced Out Of Memory exceptions and slow rendering/processing of frames. As a future work we would like to move the inference to cloud with a better memory and CPU configuration.Each frame rendered by camera will be sent to cloud for processing with OpenPose and inference.

Conclusion and Future Work

In this paper, we proposed a novel approach of converting videos to a 2-d image of optical flow and run a neural network classifier for dynamic gesture recognition. However, there is still scope for improvements in this project. We would like to do further research on the following areas:

- We currently track 15 key points for hand and body. We convert them into three color channels. Instead we plan to use 15 different channels for each keypoint and pass it through EfficientNet as a depth parameter.

- Run our dataset through CNN-LSTM model .The videos needs to be converted into multiple frames which then goes through a CNN model to get the featureset and then run LSTM on each frame one by one to track the action. We would like to analyse if we get any performance improvement over the Optical Flow model.
- Currently the entire data set is stored in S3. We want to enhance it further and try distributed storage .
- Moving the OpenPose processing and prediction to cloud while TX2 only captures the frame might improve the inference time significantly.

References

[1] https://link.springer.com/chapter/10.1007/978-94-015-8935-2_10 | Thad Starner and Alex Pentland “Real-Time American Sign Language Recognition from Video Using Hidden Markov Models”, *The Media Laboratory Massachusetts Institute of Technology*

[2] https://www.researchgate.net/publication/328135251_American_Sign_Language_Alphabet_Recognition_Using_Convolutional_Neural_Networks_with_Multiview_Augmentation_and_Inference_Fusion | Wenjin Taoa, Ming C. Leu, Zhaozheng Yin, “American Sign Language Alphabet Recognition Using Convolutional Neural Networks with Multiview Augmentation and Inference Fusion, Department of Mechanical and Aerospace Engineering”, *Missouri University of Science*

and Technology, Department of Computer Science, Missouri University of Science and Technology

[3] https://arxiv.org/pdf/1810.11438.pdf | Bowen Shi et al., “AMERICAN SIGN LANGUAGE FINGERSPELLING RECOGNITION IN THE WILD”, *Toyota Technological Institute at Chicago, USA 2University of Chicago, USA*

[4] https://arxiv.org/pdf/1901.11164.pdf | Cleison Correia de Amorim et al, “Spatial-Temporal Graph Convolutional Networks for Sign-Language Recognition”, *Centro de Informatica Universidade Federal de Pernambuco*

[5] https://ieeexplore.ieee.org/abstract/document/6164868 | Mahbub, Upal, Hafiz Imtiaz, and Md Atiqur Rahman Ahad. "An optical flow based approach for action recognition." *14th International Conference on Computer and Information Technology (ICCIT 2011)*. IEEE, 2011.

[6] http://openaccess.thecvf.com/content_cvpr_2017/html/Ilg_FlowNet_2.html | Ilg, Eddy, et al. "Flownet 2.0: Evolution of optical flow estimation with deep networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[7] https://www.dgp.toronto.edu/~donovan/stabilization/opticalflow.pdf | O’Donovan, Peter. "Optical flow: Techniques and applications." *International Journal of Computer Vision* (2005): 1-26.

[8] https://www.researchgate.net/publication/233979332_Application_of_Optical_Flow_in_Automation | Mohamed, Mahmoud. (2012). Application of Optical Flow in Automation.

[9] https://arxiv.org/pdf/1712.08416.pdf | Sevilla-Lara, Laura, et al. "On the integration of optical flow and action recognition." *German Conference on Pattern Recognition*. Springer, Cham, 2018.

[10] https://link.springer.com/chapter/10.1007%2F978-981-10-7895-8_31 | Gupta, Arpan, and M. Sakthi Balan. "Action Recognition from Optical Flow Visualizations." *Proceedings of 2nd International Conference on Computer Vision & Image Processing*. Springer, Singapore, 2018.

[11] https://books.google.com/books?id=LPm3DQAAQBAJ&pg=PA506&lpg=PA506&dq=cv2.calcopticalflowpyrlk()+why+pyramids&source=bl&ots=2vPoWibcz9&sig=ACfU3U3KL597PabDZC_O3tYvSS404fmoLw&hl=en&sa=X&ved=2ahUKEwjr24ruy-LoAhWLQs0KHd1LB0kQ6AEwBnoECAwQKQ#v=onepage&q=cv2.calcopticalflowpyrlk()%20why%20pyramids&f=false | Kaehler, Adrian,

and Gary Bradski. *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. " O'Reilly Media, Inc.", 2016.