Abstract :

With the advent of deep learning, end to end models are being built for a wide range of problems that can work on a wide variety of image and video data. Datasets have made it possible to harness the power of the models better.In this project we have used MS-ASL [1] dataset which is one of the benchmarks for ASL labeled dataset.We propose an end to end solution that would require only video as input to translate ASL and generate a text label.We have developed an end to end MLOPs pipeline which enables the entire flow of processing input , to retraining and pushing the model for developing better models.The aim of the model is to reduce the communication barriers between community with hearing disability who use ASL as their primary means of communication and larger community of people who do not understand ASL.The aim is to use HCI to decrease dependency on human translator.

Introduction

American Sign Language (ASL) is the predominant sign language of deaf communities in the United States and most of anglophone Canada.American Sign Language (ASL) is one of the most widely used languages in the United States, and the fourth-most studied second language at American universities. At least 35 states have recognized ASL as a modern language for public schools, and hundreds of colleges and universities in the United States are offering ASL classes.There are approximately 250,000 – 500,000 ASL users in the United States and Canada, most of whom use ASL as their primary language.The most common method of communication between these two communities is through the use of human based interpretation services. Notwithstanding, this is costly as it includes human expertise. Although written communication may be an alternative solution, it is cumbersome and useless because the deaf are usually less gifted in writing spoken languages. Also, this method is very slow and impersonal, especially in an emergency such as when an accident occurs, where quick communication is required with a physician and written communication is not always possible. The aim is to help deaf people improve their social lives so they can connect easily with people who do not understand ASL as it aids as a translator with HCI.

Dataset

The dataset we have chosen is from MS-ASL [1]. It is a fairly large annotated dataset for ASL, which makes it suitable for applying deep learning methods for solving the problem. One of the key differences between this dataset and other previously annotated datasets is that it has a large variety of signers, to put things in perspective the earlier most varied dataset[2] for any sign language recognition task had 50 signers, while this dataset has annotated videos from 222 signers.

The dataset is curated from various deaf communities which use public video sharing platforms for study of ASL. So one of the advantages is that there is a lot of variety as the dataset is not specifically recorded based on the same studio like setting with proper camera angle and lighting conditions, and has a lot of variety in terms of resolution, lighting and recording conditions, which can have different backgrounds.

From a language perspective also there are variations in terms of regional, dialect and inter-signer variations for the same label. So overall the dataset closes a lot of gap in terms of variety to train a more realistic model.

Signer Distribution between train, test and validation

The dataset attempted to ensure that the signers occurring in the train, test and validation set were different, however the corpus has a very diverse frequency of videos for each signer, so it was a bit difficult to ensure the perfect division between the sets in 80%, 10% and 10% split. So the final dataset has videos from 165, 37 and 20 signers in train, test and validation set respectively from a total of 222 distinct signers.

Preprocessing

The dataset has labels, video url and start and end frame information, duration stored in json format, for train, test and validation set. So based on this we downloaded the videos, converted it into a consistent mp4 format, trimmed the video based on frame information and stored it in the respective labeled folder for all of the sets. After this the video was converted to TFRecords to feed into the TFX pipeline as input, which is described in more detail in a later section.

Model Considerations

In the beginning we considered choosing an I3D model [3], which is a 3D CNN architecture, pretrained on kinetics dataset for our task, and is state of the art for video action recognition tasks. However even though it is accurate for the task, one of the drawbacks that it has is it's a large model and has a relatively high computation and memory requirements for inference, so it's not suitable for real time quick inferencing and for use on mobile devices.

Keeping the above considerations for our requirement we switched to a different pretrained model which is MoViNets [4], the greatest advantage of which is that it has far fewer parameters and still is able to achieve fairly good performance on action recognition tasks, and is the current state of the art in terms of efficiency of inference requiring less computation and memory footprint, without sacrificing much on performance.

After deciding on our pretrained model, we wrapped the backbone model with a new classifier head, and froze all the weights except for the last layer to train on our dataset.

Experiments

To make our model meet industry standards, we wanted to produce an end-to-end platform for deploying production ML pipelines. We explored the available frameworks like Uber: Michelangelo, Google:TFX,Airbnb: Bighead, Netflix: Metaflow,Lyft: Flyte etc , also full-stack solutions for machine learning like AWS SageMaker,Azure Machine Learning, Google Cloud AutoML. At the end, we settled with TFX (since it's been introduced in one of the assignments). Since this is our first deep learning project where we try to build production ready pipelines, we decided to get our hands dirty by trying to build components instead of automating through cloud services. Thus utilized the TFX components to build the pipeline and make the steps - data loading, preprocessing, training, validation, serving and deployment to modular structure.

The architecture we settled down is to build a TFX pipeline that takes TFRecords format of the videos along with the label information as input and pass through the components like ImportExamplesGen, StatisticsGen, SchemaGen, ExampleValidator, Trainer and Pusher.Similar to CsvGen, ImportExamplesGen takes TFRecords as inputs and generate them as examples with train and eval split to pass through the further components. SchemaGen gave a clear picture of the features extracted successfully from the videos. For the frames to be extracted from videos as 5, the features blob/0,blob/1,blob/2,blob/3,blob4 are generated. We experimented with a different number of frames to check if it matches the number of blobs generated. The blob features generated are of bytes value, we explored and found that it can be processed through tensorflow to string format to capture the innate nature. SchemaGen clearly showed that we don't have any null values as expected for the dataset generated. Also, we explicitly printed the values for the dataset(height, width and length) to verify the data flow. Example validator helped in validating the absence of anomalies in the dataset.

We started our pipeline for a very small set (around 20 videos) to build a full fledged working pipeline before starting with the actual training. The Trainer component takes a custom training function as input which in turn utilizes an input\_function to parse our examples from the ImportExamplesGen component. The custom training function is written with the model we use for training and also ‘serving\_default’ signature function to serve the prediction for input videos. Along with training, we tried saving the model in two formats - .pb and tflite format. And included the same as part of the custom training function. The path to push the trained model for serving is given as part of the Pusher component. We then integrated all the components with create\_final\_pipeline function which returns the pipeline linked with the components. For orchestrating the flow, we used local\_dag\_runner from tfx for controlling the flow. With the help of the orchestrated pipeline we trained the model for detecting the signs for ASL videos. As a result of training, the pushed .pb file is used for prediction. A separate frontend simple application is built to record the videostream from the webcam and store it in a folder and apply prediction using the saved model and output the sign language demonstrated.

We experimented to include automatic CI/CD for the trained model. We used the eval configuration that checks for the model if its accuracy is improved, considering the HIGHER\_IS\_BETTER direction as the blessed model, which avoids the comparison at the first iteration and performs the comparisons for the upcoming training. We used model resolver with strategy latest\_blessed\_model\_resolver.LatestBlessedModelResolver and evaluator component with eval config to implement the functionality.

Challenges on the input dataset:



We utilized MSASL dataset to train our model detect signs from videos. We followed a series of preprocessing steps to parse all the input videos to standard mp4 format videos. Since our input dataset is videos about sign language, our challenge started at processing the input data. We explored the ways to feed video input to the TFX pipeline and found the video can be converted to TFRecords and fed to the pipeline. We built a function - convert\_videos\_to\_tfrecord2 referring to the library video2tfrecord, which is suitable for Tensorflow version 1.x and so made the changes to make it adaptable to tensorflow version 2.4.1.

The generated TFRecord has 5 frames per video along with optical flow data embedded in it. In the paper [5] we referred to, while transforming the video to TFRecords, the authors used to generate 25 frames per second. We referred to materials to decide the number of frames per second and it’s been advised to keep it as 5 to make the learning task simpler. We have the option in convert\_videos\_to\_tfrecord2 to control if we want to embed the optical flow information along with the tfrecord or not. Since the input to TFRecord conversion method is only the videos in mp4 format, while parsing the videos itself we included the sign label in the name of the video. With the help of the label name in the video, while converting them to TFRecord, a target column is added along with it to include the label value.

References

[1] Vaezi Joze, H., & Koller, O. (2019). MS-ASL: A Large-Scale Data Set and Benchmark for Understanding American Sign Language. In *The British Machine Vision Conference (BMVC)*.

[2] Jie Huang, Wengang Zhou, Qilin Zhang, Houqiang Li, and Weiping Li. Video-based sign lan-guage recognition without temporal segmentation. InProc. of the {AAAI} Conference on ArtificialIntelligence, New Orleans, Louisiana, USA, February 2018.

[3] Joao Carreira, & Andrew Zisserman. (2018). Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset.

[4] Dan Kondratyuk and Liangzhe Yuan and Yandong Li and Li Zhang and Mingxing Tan and Matthew Brown and Boqing Gong (2021). MoViNets: Mobile Video Networks for Efficient Video Recognition*. CoRR, abs/2103.11511.*

[5] Dongxu Li and Cristian Rodriguez Opazo and Xin Yu and Hongdong Li (2019). Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison*. CoRR, abs/1910.11006.*