Gesture Recognition

Abstract:

Sign languages (also known as **signed languages**) are languages that use the visual-manual modality to convey meaning. Sign languages are expressed through manual articulations in combination with non-manual elements. Sign languages are full-fledged natural languages with their own grammar and lexicon. Sign languages are not universal and are usually not mutually intelligible, although there are also similarities among different sign languages.

Wherever communities of **deaf** people exist, sign languages have developed as useful means of communication, and they form the core of local Deaf cultures. Although signing is used primarily by the deaf and hard of hearing, it is also used by hearing individuals, such as those unable to physically speak, those who have trouble with spoken language due to a disability or condition (augmentative and alternative communication), or those with deaf family members, such as children of deaf adults. Different regions have different sign languages. Indian sign language offers a huge collection of signs. It is an ideal source to use when communicating to a deaf or dumb person.

Problem Statement:

Sign language detection have always been a challenging task as sign language is not the same everywhere and the dataset have to adequate for the particular sign language detection. There have been few techniques to implement sign language detection. Machine learning requires a vast amount of data, on a regular basis to predict effectively. So we have to use a Machine learning model on small data to predict the signs for the deaf and dumb people, to understand them.

Requirements:

This project offers a visual representation to read the Indian sign language for the communication. The language used here is python, on the platform Google Colaboratory, and the module used is YOLOv5.

Python:

Python is a computer programming language often used to build websites and software automate task and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problem.

Google Colab:

Colaboratory or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

Yolov5:

YOLOv5 is a family of compound-scaled object detection models trained on the COCO dataset and includes simple functionality for Test Time Augmentation (TTA), model ensembling, hyperparameter evolution, and export to ONNX, CoreML and TFLite.

YOLO

YOLO is an abbreviation for the term ‘You Only Look Once’. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects. This means that prediction in the entire image is done in a single algorithm run.

Applications of YOLO

YOLO algorithm can be applied in the following fields:

- **Autonomous driving:** YOLO algorithm can be used in autonomous cars to detect objects around cars such as vehicles, people, and parking signals. Object detection in autonomous cars is done to avoid collision since no human driver is controlling the car.

- **Wildlife:** This algorithm is used to detect various types of animals in forests. This type of detection is used by wildlife rangers and journalists to identify animals in videos (both recorded and real-time) and images. Some of the animals that can be detected include giraffes, elephants, and bears.

- **Security:** YOLO can also be used in security systems to enforce security in an area. Let’s assume that people have been restricted from passing through a certain area for security reasons. If someone passes through the restricted area, the YOLO algorithm will detect him/her, which will require the security personnel to take further action.

- **Object detection:** Yolo can also be used in object detection system to identify the and keep the record for the useful purposes such as sign language recognition and attendance register using face detection.

Code:

```
# Cloning the YOLOv5 repository from the official github link.
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt  # install

import torch
import utils
display = utils.notebook_init()  # checks
```

```
YOLOv5 🚀 v6.1-270-g6935a54 Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla T4, 15110MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 38.7/78.2 GB disk)
```

```
#Importing and unzipping the data set in the colab.
!unzip -q ../Train_Data.zip -d ../
```

```
!python detect.py --weights runs/train/exp4/weights/last.pt --img 640 --conf 0.25 --source ../video.mp4
```

```
detect: weights=['runs/train/exp4/weights/last.pt'], source=../video.mp4, data=data/coco128.yaml, imgsz=[640, 640], conf_thres=0.25,
YOLOv5 🚀 v6.1-270-g6935a54 Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla T4, 15110MiB)

Fusing layers...
Model summary: 213 layers, 7023610 parameters, 0 gradients
video 1/1 (1/504) /content/video.mp4: 384x640 1 Hello, Done. (0.016s)
video 1/1 (2/504) /content/video.mp4: 384x640 1 Hello, Done. (0.013s)
video 1/1 (3/504) /content/video.mp4: 384x640 1 Hello, Done. (0.014s)
video 1/1 (4/504) /content/video.mp4: 384x640 1 Hello, Done. (0.014s)
video 1/1 (5/504) /content/video.mp4: 384x640 1 Hello, Done. (0.014s)
video 1/1 (6/504) /content/video.mp4: 384x640 1 Hello, Done. (0.013s)
video 1/1 (7/504) /content/video.mp4: 384x640 1 Hello, Done. (0.014s)
video 1/1 (8/504) /content/video.mp4: 384x640 1 Hello, Done. (0.022s)
video 1/1 (9/504) /content/video.mp4: 384x640 1 Hello, Done. (0.016s)
video 1/1 (10/504) /content/video.mp4: 384x640 1 Hello, Done. (0.012s)
video 1/1 (11/504) /content/video.mp4: 384x640 1 Hello, Done. (0.015s)
video 1/1 (12/504) /content/video.mp4: 384x640 1 Hello, Done. (0.013s)
video 1/1 (13/504) /content/video.mp4: 384x640 1 Hello, Done. (0.012s)
video 1/1 (14/504) /content/video.mp4: 384x640 1 Hello, Done. (0.011s)
video 1/1 (15/504) /content/video.mp4: 384x640 1 Hello, Done. (0.012s)
video 1/1 (16/504) /content/video.mp4: 384x640 1 Hello, Done. (0.012s)
video 1/1 (17/504) /content/video.mp4: 384x640 1 Hello, Done. (0.011s)
```

```
# Train YOLOv5s on custom for 90 epochs
!python train.py --img 640 --batch 16 --epochs 90 --data custom.yaml --weights yolov5s.pt --cache
```

```
train: weights=yolov5s.pt, cfg=, data=custom.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=90, batch_size=16, imgsz=640, rect=False,
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v6.1-270-g6935a54 Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla T4, 15110MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, b
Weights & Biases: run 'pip install wandb' to automatically track and visualize YOLOv5 🚀 runs (RECOMMENDED)
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Overriding model.yaml nc=80 with nc=5

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
```

Input video frame:

Output video frame: