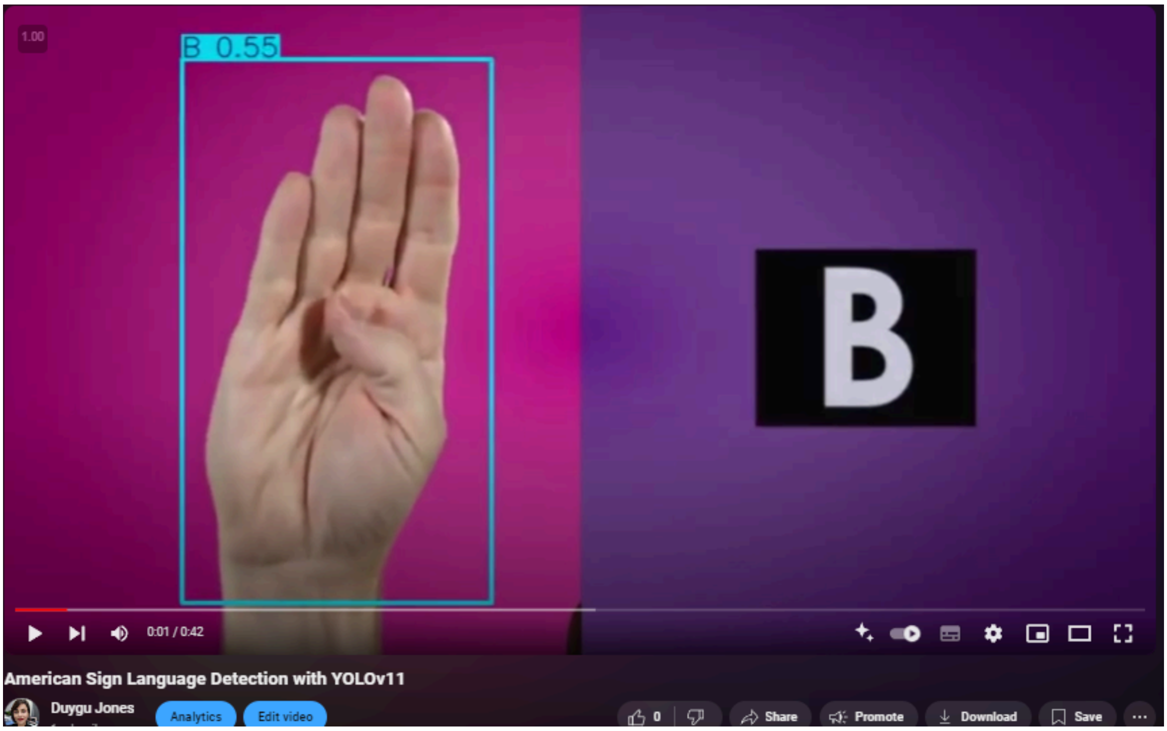
SIGN LANGUAGE DETECTION USING YOLO11

Computer Vision Project: YOLO11 - Roboflow - OpenCV

Duygu Jones | Data Scientist | Oct 2024
Follow me: duygujones.com | Linkedin | GitHub | Kaggle | Medium | Tableau



- Github Project Repository: https://github.com/Duygu-Jones/Deep-Learning-Projects/tree/main/03_Sign_Language_YOLO11

Sign Language Detection with YOLO11 🤟

💡 In this project, **Ultralytics YOLO11** model has been used Ultralytics YOLO11 GitHub Repository. This model is not only powerful for real-time object detection but also capable of a wide range of tasks such as **Object Detection**, **Classification**, **Instance Segmentation**, **Semantic Segmentation**, and **Pose Estimation**. For this project, the **small version** of YOLO11 for **object detection** was used to handle **real-time sign detection** efficiently, making it ideal for recognizing hand gestures.

About YOLO11

YOLOv11 builds upon the advancements made in previous versions (YOLOv9, YOLOv10), featuring improved architecture, better feature extraction, and optimized training techniques. The model can handle multiple tasks across different domains and is highly scalable, from mobile CPUs to powerful GPUs. YOLO11 is available in 5 different sizes, ranging from **2.6M to 56.9M parameters**, and achieves mAP scores between **39.5 and 54.7** on the **COCO dataset**, which was used for initial pre-training.

For this project, the **small version** of YOLO11 was used for **object detection** to efficiently detect hand signs in real-time.

Dataset Information

The **American Sign Language (ASL) dataset** used in this project was sourced from **Roboflow Universe/duyguj/american-sign-language-letters**. All images in the dataset were pre-labeled, ensuring accurate training data. Additionally, **data augmentation** techniques were applied within Roboflow to increase the variability of the dataset, improving the model's generalization. Techniques such as flipping, rotation, and brightness adjustments were employed.

This dataset contains a total of **1224 images**, which are split into three sets:

- **Train Set**: 1008 images (82%)
- **Validation Set**: 144 images (12%)
- **Test Set**: 72 images (6%)

Preprocessing:

- **Auto-Orient**: Applied to ensure the images are properly aligned.
- **Resize**: All images are resized to fit within **640x640 pixels**.

Data Augmentation:

Each training example has two outputs due to augmentation, which includes:

- **Rotation**: Between **-15° and +15°** to simulate different hand orientations.
- **Exposure**: Adjustments between **-10% and +10%** to account for varying lighting conditions.
- **Blur**: Up to **2px** to simulate motion or camera blur.

This setup is intended to improve the model's ability to generalize by exposing it to varied inputs.

Training Process

The YOLOv11 model was **fine-tuned** on this ASL dataset to specialize in object detection for sign language. This training process included:

- **Dataset Augmentation**: Using Roboflow to enhance the dataset with transformations.
- **Model Training**: YOLOv11 was trained using this enhanced dataset, and the performance was validated using a separate validation dataset.
- **Testing**: After training, the model was tested on a **dedicated test set** to evaluate its ability to predict unseen data.

Performance and Observations

The **final model** was tested on random **sign language images and videos** to observe its real-world performance. The results showed promising outcomes for detecting different ASL signs in real-time, demonstrating the effectiveness of the YOLO11 architecture in handling complex, gesture-based tasks.

Tools Used

- **YOLO11 (Object Detection)**: For real-time sign detection
- **OpenCV**: For image and video processing
- **Python**: For implementation and integration

Next Steps

- Improve detection accuracy for complex signs
- Expand the model to support more sign languages and gestures

Links:

- Ultralytics YOLO11 GitHub Repository
- Roboflow Universe ASL Dataset

Setup and Initialization

Access to GPU

We can use `nvidia-smi` command to do that. In case of any problems navigate to `Edit` -> `Notebook settings` -> `Hardware accelerator`, set it to `GPU`.

In [ ]:
```
# GPU Access
#!nvidia-smi
```

Install YOLO11 via Ultralytics

In [1]:
```
!pip install ultralytics supervision roboflow

from IPython import display
display.clear_output()

!pip install ultralytics --quiet
import ultralytics
ultralytics.checks()
```

```
Ultralytics 8.3.6 🚀 Python-3.10.14 torch-2.4.0 CUDA:0 (Tesla T4, 15095MiB)
Setup complete ✅ (4 CPUs, 31.4 GB RAM, 5933.9/8062.4 GB disk)
```

Load the Dataset

Configure API keys to Load the Dataset

To fine-tune YOLO11, you need to provide your Roboflow API key. Follow these steps:

- Go to your `Roboflow Settings` page. Click `Copy`. This will place your private key in the clipboard.
- In Colab, go to the left pane and click on `Secrets` (🔑). Store Roboflow API Key under the name `ROBOFLOW_API_KEY`.
- **Roboflow**: Go to your `Roboflow Dataset Download` -> Select YOLO model -> Select `Show download code` -> click `Copy`.
  - In **Colab**: go to the left pane and click on `Secrets` (🔑). Store the Roboflow API Key under a username.
  - In **Kaggle**: Go to `Add-ons` → `Secrets` → `Add Secret` (🔑) and store your Kaggle API key and username.

In [2]:
```
# Save the API key in Kaggle
from kaggle_secrets import UserSecretsClient

user_secrets = UserSecretsClient()
secret_value_0 = user_secrets.get_secret("my_api_key")
```

In [3]:
```
# Roboflow Dataset API Code
!pip install roboflow --quiet
from roboflow import Roboflow

rf = Roboflow(secret_value_0)

project = rf.workspace("duyguj").project("american-sign-language-letters-vouo0")
version = project.version(1)
dataset = version.download("yolov11")
```

```
loading Roboflow workspace...
loading Roboflow project...
```

```
Downloading Dataset Version Zip in American-Sign-Language-Letters-1 to yolov11:: 100%|██████████| 34619/34619 [00:00<00:00, 44839.41it/s]
Extracting Dataset Version Zip to American-Sign-Language-Letters-1 in yolov11:: 100%|██████████| 2424/2424 [00:00<00:00, 8137.76it/s]
```

Model Training

In [4]:
```
# Changing to the working directory in Kaggle
%cd /kaggle/working

# Training the YOLO model
!yolo task=detect mode=train model=yolo11n.pt data=/kaggle/working/American-Sign-Language-Letters-1/data

#Results saved to runs/detect/train
#Learn more at https://docs.ultralytics.com/modes/train
```

```
/kaggle/working
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt to 'yolo11n.pt'...
100%|██████████████████████████████████████| 5.35M/5.35M [00:00<00:00, 68.2MB/s]
Ultralytics 8.3.6 🚀 Python-3.10.14 torch-2.4.0 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: task=detect, mode=train, model=yolo11n.pt, data=/kaggle/working/American-Sign-Language-Le
tters-1/data.yaml, epochs=10, time=None, patience=100, batch=16, imgsz=640, save=True, save_period=-1, ca
che=False, device=None, workers=8, project=None, name=train, exist_ok=False, pretrained=True, optimizer=a
uto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False, close_mosaic=1
0, resume=False, amp=True, fraction=1.0, profile=False, freeze=None, multi_scale=False, overlap_mask=Tru
e, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False, conf=None, iou=0.
7, max_det=300, half=False, dnn=False, plots=True, source=None, vid_stride=1, stream_buffer=False, visual
ize=False, augment=False, agnostic_nms=False, classes=None, retina_masks=False, embed=None, show=False, s
ave_frames=False, save_txt=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, sho
w_boxes=True, line_width=None, format=torchscript, keras=False, optimize=False, int8=False, dynamic=Fals
e, simplify=True, opset=None, workspace=4, nms=False, 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, box=7.5, cls=0.5, dfl=1.5, pose=12.0, k
obj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scal
e=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.
0, copy_paste_mode=flip, auto_augment=randaugment, erasing=0.4, crop_fraction=1.0, cfg=None, tracker=bots
ort.yaml, save_dir=runs/detect/train
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100%|████████████████████████████████████████| 755k/755k [00:00<00:00, 13.9MB/s]
Overriding model.yaml nc=80 with nc=26

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      6640  ultralytics.nn.modules.block.C3k2            [32, 64, 1, False, 0.2
5]
  3                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
  4                  -1  1     26080  ultralytics.nn.modules.block.C3k2            [64, 128, 1, False, 0.
25]
  5                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
  6                  -1  1     87040  ultralytics.nn.modules.block.C3k2            [128, 128, 1, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    346112  ultralytics.nn.modules.block.C3k2            [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1    249728  ultralytics.nn.modules.block.C2PSA           [256, 256, 1]
 11                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 12             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 13                  -1  1    111296  ultralytics.nn.modules.block.C3k2            [384, 128, 1, False]
 14                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 15             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 16                  -1  1     32096  ultralytics.nn.modules.block.C3k2            [256, 64, 1, False]
 17                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 18            [-1, 13]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 19                  -1  1     86720  ultralytics.nn.modules.block.C3k2            [192, 128, 1, False]
 20                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 21            [-1, 10]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 22                  -1  1    378880  ultralytics.nn.modules.block.C3k2            [384, 256, 1, True]
 23        [16, 19, 22]  1    435742  ultralytics.nn.modules.head.Detect           [26, [64, 128, 256]]
YOLO11n summary: 319 layers, 2,594,910 parameters, 2,594,894 gradients, 6.5 GFLOPs

Transferred 448/499 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train', view at http://localhost:6006/
Freezing layer 'model.23.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLO11n...
AMP: checks passed ✅
train: Scanning /kaggle/working/American-Sign-Language-Letters-1/train/labels...
train: New cache created: /kaggle/working/American-Sign-Language-Letters-1/train/labels.cache
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, nu
m_output_channels=3, method='weighted_average'), CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_size=(8,
8))
/opt/conda/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork
() is incompatible with multithreaded code, and JAX is multithreaded, so this will likely lead to a deadl
ock.
  self.pid = os.fork()
val: Scanning /kaggle/working/American-Sign-Language-Letters-1/valid/labels... 1
val: New cache created: /kaggle/working/American-Sign-Language-Letters-1/valid/labels.cache
Plotting labels to runs/detect/train/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimiz
er', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.000333, momentum=0.9) with parameter groups 81 weight(decay=0.0), 88 weight(decay=
0.0005), 87 bias(decay=0.0)
TensorBoard: model graph visualization added ✅
```

```
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 10 epochs...
Closing dataloader mosaic
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, nu
m_output_channels=3, method='weighted_average'), CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_size=(8,
8))
/opt/conda/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork
() is incompatible with multithreaded code, and JAX is multithreaded, so this will likely lead to a deadl
ock.
  self.pid = os.fork()

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/10      2.47G      1.009      4.893      1.508          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141     0.0377      0.591     0.0967     0.0825

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/10      2.47G     0.8105      4.296      1.304          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.543      0.256      0.286      0.244

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/10      2.47G     0.8379      3.849      1.295          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.502       0.43       0.45      0.383

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/10      2.47G     0.7695      3.387      1.232          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.703       0.45      0.575      0.512

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       5/10      2.47G     0.6566       2.95      1.137          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.551      0.606      0.671      0.607

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       6/10      2.46G     0.5905      2.576      1.075          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.698      0.694       0.76       0.69

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       7/10      2.47G      0.555      2.289      1.049          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.746      0.711      0.815      0.762

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       8/10      2.47G      0.512      2.034      1.009          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.736       0.83       0.87      0.806

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       9/10      2.47G     0.4784      1.877     0.9843          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141       0.81      0.782      0.885       0.83

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      10/10      2.46G     0.4377      1.767     0.9515          4        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.819      0.802        0.9      0.849

10 epochs completed in 0.035 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 5.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 5.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.6 🚀 Python-3.10.14 torch-2.4.0 CUDA:0 (Tesla T4, 15095MiB)
YOLO11n summary (fused): 238 layers, 2,587,222 parameters, 0 gradients, 6.3 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141       0.82      0.802        0.9      0.849
                     A          4          4          1       0.67      0.995      0.995
                     B          9          9          1      0.741      0.984      0.883
                     C          3          3      0.757          1      0.995      0.907
```

```
                 D          6          6      0.899      0.833      0.931      0.867
                 E          4          4      0.905          1      0.995      0.995
                 F          8          8       0.86      0.875      0.907      0.894
                 G          5          5      0.919          1      0.995      0.971
                 H          8          8          1      0.954      0.995      0.929
                 I          2          2      0.282        0.5      0.695      0.695
                 J          8          8       0.95      0.875      0.962      0.681
                 K          6          6      0.996      0.667      0.894      0.853
                 L          4          4      0.764          1      0.995      0.937
                 M          8          8          1      0.745      0.967      0.888
                 N          3          3      0.129      0.333      0.191      0.187
                 O          7          7      0.918      0.857      0.978       0.94
                 P          7          7       0.88      0.571      0.757       0.71
                 Q          4          4      0.789          1      0.995       0.97
                 R          7          7      0.789      0.857      0.892      0.858
                 S          4          4      0.727          1      0.995      0.995
                 T          6          6          1      0.529      0.726      0.721
                 U          7          7      0.616      0.714        0.7      0.655
                 V          5          5          1      0.511      0.995      0.948
                 W          3          3      0.498          1      0.995      0.995
                 X          1          1      0.741          1      0.995      0.895
                 Y          8          8          1      0.613      0.879      0.744
                 Z          4          4      0.889          1      0.995      0.964
Speed: 0.2ms preprocess, 2.3ms inference, 0.0ms loss, 3.1ms postprocess per image
Results saved to runs/detect/train
💡 Learn more at https://docs.ultralytics.com/modes/train
```

In [5]:

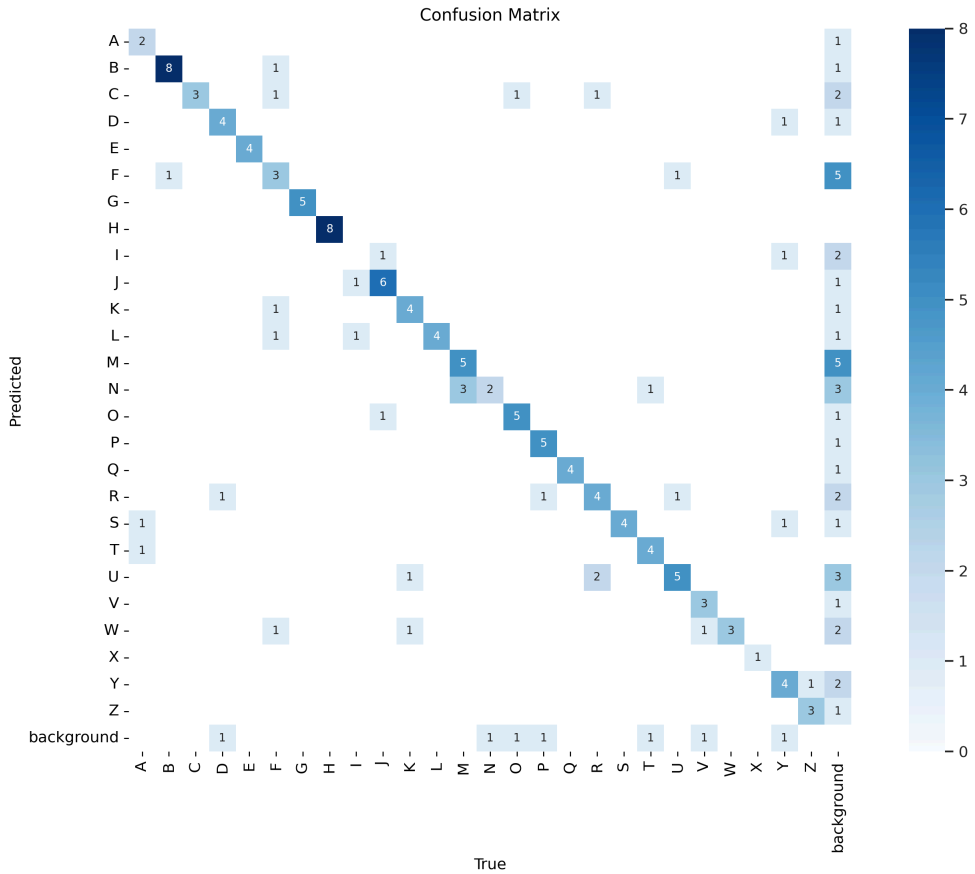
```
# The list of files from the completed training is saved;
!ls /kaggle/working/runs/detect/train/
```

```
F1_curve.png                                          results.png
PR_curve.png                                          train_batch0.jpg
P_curve.png                                           train_batch1.jpg
R_curve.png                                           train_batch2.jpg
args.yaml                                             val_batch0_labels.jpg
confusion_matrix.png                                  val_batch0_pred.jpg
confusion_matrix_normalized.png                       val_batch1_labels.jpg
events.out.tfevents.1728284614.35bd55a0240b.102.0     val_batch1_pred.jpg
labels.jpg                                            val_batch2_labels.jpg
labels_correlogram.jpg                                val_batch2_pred.jpg
results.csv                                           weights
```

In [6]:

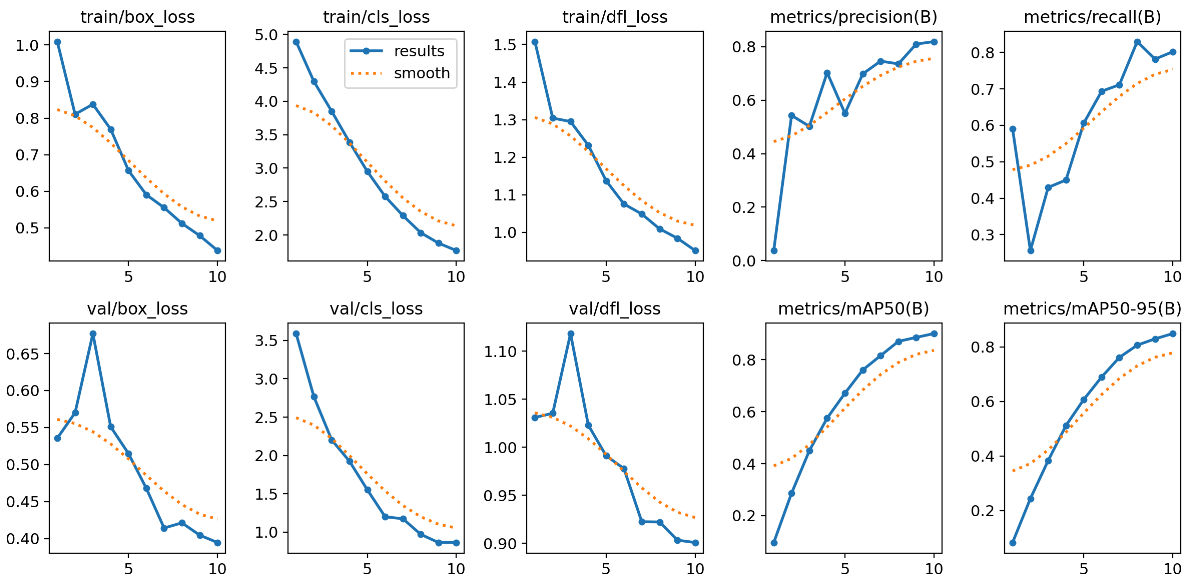
```
from IPython.display import Image as IPyImage

# Display the confusion matrix image from the specified directory in Kaggle
IPyImage(filename='/kaggle/working/runs/detect/train/confusion_matrix.png', width=1000)
```

Out[6]:



In [7]:
```
IPyImage(filename=f'/kaggle/working/runs/detect/train/results.png', width=1000)
```

Out[7]:



In [8]:
```
IPyImage(filename=f'/kaggle/working/runs/detect/train/val_batch0_pred.jpg', width=1000)
```

Out[8]:

Validation of the Model

In [9]:

```
# Run the validation task using YOLO in Kaggle
!yolo task=detect mode=val model=/kaggle/working/runs/detect/train/weights/best.pt data=/kaggle/working/.
```

```
Ultralytics 8.3.6 🚀 Python-3.10.14 torch-2.4.0 CUDA:0 (Tesla T4, 15095MiB)
YOLO11n summary (fused): 238 layers, 2,587,222 parameters, 0 gradients, 6.3 GFLOPs
val: Scanning /kaggle/working/American-Sign-Language-Letters-1/valid/labels.cach
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all        141        141      0.819      0.802        0.9      0.848
                     A          4          4          1       0.67      0.995      0.995
                     B          9          9          1      0.755      0.984      0.883
                     C          3          3      0.756          1      0.995      0.907
                     D          6          6      0.898      0.833      0.931      0.867
                     E          4          4      0.907          1      0.995      0.995
                     F          8          8      0.859      0.875      0.907      0.894
                     G          5          5      0.919          1      0.995      0.971
                     H          8          8          1      0.954      0.995      0.929
                     I          2          2      0.282        0.5      0.695      0.695
                     J          8          8       0.95      0.875      0.962      0.681
                     K          6          6      0.995      0.667      0.894      0.853
                     L          4          4      0.765          1      0.995      0.937
                     M          8          8          1      0.745      0.967      0.888
                     N          3          3      0.129      0.333      0.191      0.187
                     O          7          7      0.918      0.857      0.978       0.94
                     P          7          7       0.88      0.571      0.757       0.71
                     Q          4          4      0.787          1      0.995      0.945
                     R          7          7      0.786      0.857      0.892      0.858
                     S          4          4      0.724          1      0.995      0.995
                     T          6          6          1       0.53      0.726      0.721
                     U          7          7      0.622      0.714        0.7      0.655
                     V          5          5          1      0.511      0.995      0.948
                     W          3          3      0.498          1      0.995      0.995
                     X          1          1       0.74          1      0.995      0.895
                     Y          8          8          1      0.614      0.879      0.749
                     Z          4          4      0.889          1      0.995      0.964
Speed: 1.3ms preprocess, 3.8ms inference, 0.0ms loss, 3.2ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

Prediction

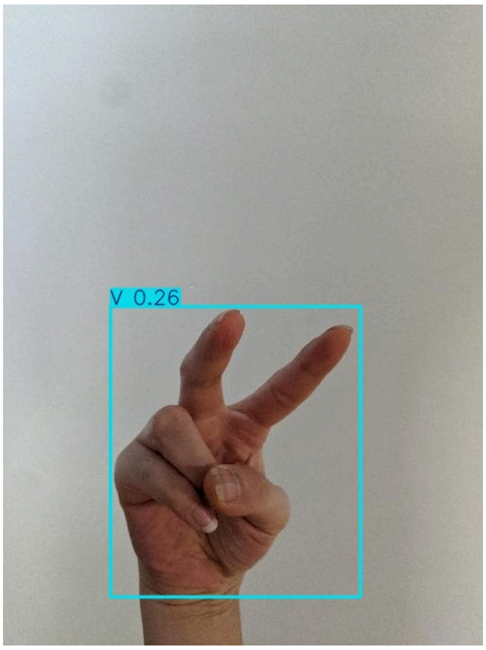
In [ ]:

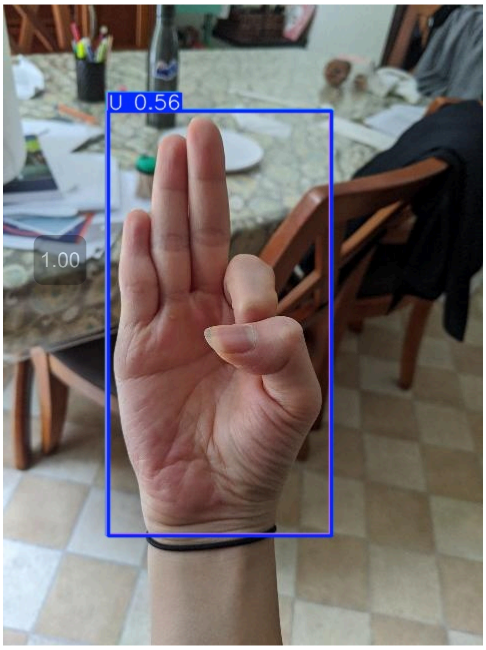
```
# Run the prediction task on Test Data
!yolo task=detect mode=predict model=/kaggle/working/runs/detect/train/weights/best.pt conf=0.25 source=

#Results saved to runs/detect/predict
# 💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Prediction with Random Images

In [16]:

```
import glob
import os
from IPython.display import Image as IPyImage, display

# Get the latest prediction folder for detection in Kaggle
latest_folder = max(glob.glob('/kaggle/working/runs/detect/predict*/'), key=os.path.getmtime)

# Display images from the prediction folder
for img in glob.glob(f'{latest_folder}/*.jpg')[15:18]:
    display(IPyImage(filename=img, width=300))
```





Predictions on Videos

In [ ]:
```
# Input video path for the first video in Kaggle
input_video_path = "/kaggle/input/asl-videos/asl_video1_40sn.mp4"  # First video path
# Output paths for saving the prediction result
output_video_path = "/kaggle/working/runs/detect/predict/asl_video1_40sn_output.avi"  # YOLO default out

# Run YOLO on the first video for object detection
!yolo task=detect mode=predict model="/kaggle/working/runs/detect/train/weights/best.pt" conf=0.25 sourc

# Results saved to runs/detect/predict2
# 💡 Learn more at https://docs.ultralytics.com/modes/predict
```

In [ ]:
```
# Convert .avi to .mp4 using FFmpeg
import os

# Path to the input .avi video
input_video = '/kaggle/working/runs/detect/predict2/asl_video1_40sn.avi'
# Path to the output .mp4 video
output_video = '/kaggle/working/runs/detect/predict2/asl_video1_40sn_output.mp4'

# FFmpeg command to convert .avi to .mp4
ffmpeg_command = f"ffmpeg -i {input_video} -vcodec libx264 {output_video}"
os.system(ffmpeg_command)
```

In [20]:
```
# Check if the .mp4 file is successfully created
!ls /kaggle/working/runs/detect/predict2/
```

```
asl_video1_40sn.avi  asl_video1_40sn_output.mp4
```

In [22]:
```
from IPython.display import HTML
import base64

# Path to the saved video (after conversion to .mp4)
save_path = '/kaggle/working/runs/detect/predict2/asl_video1_40sn_output.mp4'  # Adjusted path for first

# Load and encode the video
mp4 = open(save_path, 'rb').read()
data_url = "data:video/mp4;base64," + base64.b64encode(mp4).decode()

# Embed the video and display it in the notebook
HTML(f"""
<video width=600 controls>
      <source src="{data_url}" type="video/mp4">
</video>
""")
```

Out[22]:

In [ ]:
```
# Input video path for the second video in Kaggle
input_video_path = "/kaggle/input/asl-videos/asl_video2_30sn.mp4"  # Second video path
# Output paths for saving the prediction result
output_video_path = "/kaggle/working/runs/detect/predict/asl_video2_30sn_output.avi"  # YOLO default out

# Run YOLO on the second video for object detection
!yolo task=detect mode=predict model="/kaggle/working/runs/detect/train/weights/best.pt" conf=0.25 sourc
```

In [ ]:
```python
# Convert .avi to .mp4 using FFmpeg

import os

# Path to the input .avi video (Video 2)
input_video = '/kaggle/working/runs/detect/predict3/asl_video2_30sn.avi'

# Path to the output .mp4 video (Video 2)
output_video = '/kaggle/working/runs/detect/predict3/asl_video2_30sn_output.mp4'

# FFmpeg command to convert .avi to .mp4
ffmpeg_command = f"ffmpeg -i {input_video} -vcodec libx264 {output_video}"
os.system(ffmpeg_command)
```

In [26]:
```python
# Check if the .mp4 file is successfully created
!ls /kaggle/working/runs/detect/predict3/
```

```
asl_video2_30sn.avi  asl_video2_30sn_output.mp4
```

In [28]:
```python
from IPython.display import HTML
import base64

# Path to the saved video (after conversion to .mp4)
save_path = '/kaggle/working/runs/detect/predict3/asl_video2_30sn_output.mp4'  # Adjusted path for Video

# Load and encode the video
mp4 = open(save_path, 'rb').read()
data_url = "data:video/mp4;base64," + base64.b64encode(mp4).decode()

# Embed the video and display it in the notebook
HTML(f"""
<video width=600 controls>
      <source src="{data_url}" type="video/mp4">
</video>
""")
```

Out[28]:

If you find this work helpful, don't forget to give it an 👍 UPVOTE! and join the discussion! 💬

Thank you...

Duygu Jones | Data Scientist | 2024

Follow me: duygujones.com | Linkedin | GitHub | Kaggle | Medium | Tableau