Configuration Manual

MSc Research Project
Data Analytics

Prithiviraj Mohanraj
Student ID: X21196044

School of Computing
National College of Ireland

Supervisor: Vitor Horta.

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Prithiviraj Mohanraj
Student ID:	X21196044
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Vitor Horta
Submission Due Date	18 / 09 / 2023
Project Title:	Evaluating the Robustness of YOLOv5 and YOLOv7 in ASL Detection Across Diverse Lighting Conditions
Word Count:	1777
Page count:	10

Signature: ………………Prithiviraj Mohanraj……..………………………………………

Date: ………18/09/2023…………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Prithiviraj Mohanraj
X21196044

1 Introduction

This manual offers comprehensive guidance for replicating the research project on ASL gesture recognition. It covers the entire setup and execution process.

2 System Requirement

The research presented in this thesis was conducted using the following system configurations:

2.1 Hardware Requirements

1	Device Name	ASUS VivoBook Notebook 2
2	Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
3	RAM	16.0 GB (15.8 GB usable)
4	Type	64-bit operating system, x64-based processor

2.2 Software Requirements

Operating System: A 64-bit Windows operating system ensures compatibility with the listed software and libraries.

Anaconda Navigator 3 for Windows: Anaconda provides an environment management system which is necessary to prevent conflicts between library versions. It can be downloaded from the official website.

Jupyter Notebook (Version 6.4.12): Within Anaconda, you can launch Jupyter Notebook, the platform where Python scripts are executed interactively.

Python (Version 3.9): Our codebase is tested against Python 3.9 to ensure maximum compatibility.

CV2 (OpenCV for Python): This library aids in image processing tasks and capturing images via webcam. It can be installed using pip install opencv-python.

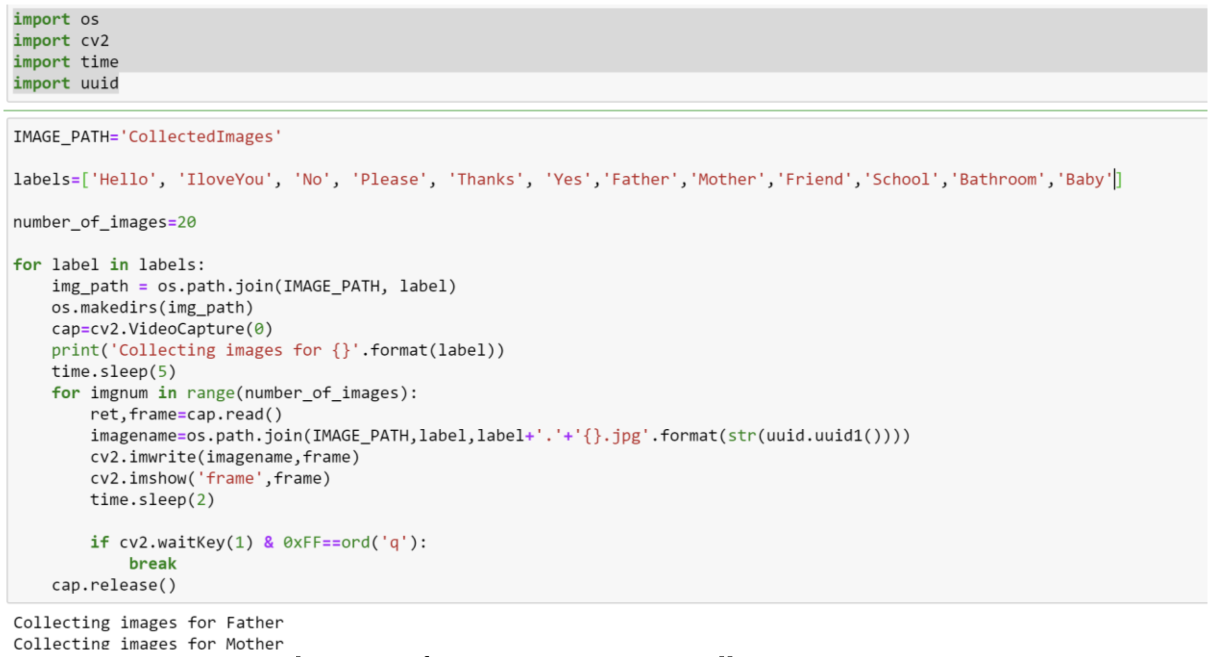
Augmentor: For data augmentation. It can be installed using pip install Augmentor.

LabelImg: An open-source graphical image annotation tool that is useful for drawing bounding boxes and labeling images for our dataset

3.0 Data Preparation and Collection

3.1 Image Capturing

Directory Preparation: First, ensure you have the CollectedImages directory in your project workspace. If it doesn't exist, manually create it or use the Python os library to generate it programmatically.



```
import os
import cv2
import time
import uuid
```

```
IMAGE_PATH='CollectedImages'

labels=['Hello', 'IloveYou', 'No', 'Please', 'Thanks', 'Yes','Father','Mother','Friend','School','Bathroom','Baby']

number_of_images=20

for label in labels:
    img_path = os.path.join(IMAGE_PATH, label)
    os.makedirs(img_path)
    cap=cv2.VideoCapture(0)
    print('Collecting images for {}'.format(label))
    time.sleep(5)
    for imgnum in range(number_of_images):
        ret,frame=cap.read()
        imagename=os.path.join(IMAGE_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        cv2.imwrite(imagename,frame)
        cv2.imshow('frame',frame)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF==ord('q'):
            break
    cap.release()
```

```
Collecting images for Father
Collecting images for Mother
```

Figure 1 Python script for ASL Gesture Data Collection

3.2 Initiating Image Capture:

The code snippet presented in Figure 1 is tailored to capture and store images of specific American Sign Language (ASL) gestures. Here's a breakdown of what each section accomplishes:

3.3 Initialization:

IMAGE_PATH='CollectedImages': This designates the directory where the captured images will be saved.

labels=['Hello', 'IloveYou', 'No', 'Please', 'Thanks', 'Yes',' Father','Mother','Friend','School','Bathroom','Baby']

These are the specific ASL gestures that the script is designed to capture. Each label represents a unique gesture.

3.4 Data Capture Process:
For each label in the 'labels' list, the script performs the following steps:

A unique folder is created for the label using os. makedirs(img_path). This ensures organized storage where images corresponding to each label are stored in separate folders.
The webcam is accessed using cap=cv2. VideoCapture(0). This allows the script to capture real-time video feed.

The user is notified about the gesture to be captured with print('Collecting images for {}'. format(label)).

A pause of 5 seconds (time. sleep(5)) gives the user time to get ready.
For each gesture, number_of_images (which is set to 20) images are captured. For every image:
The webcam captures a frame using ret, frame=cap. read().
The captured frame is saved with a unique name generated by combining the label and a universally unique identifier (UUID). This ensures that each saved image has a distinct name.

The frame is displayed in real-time to the user with cv2. imshow('frame', frame), allowing the user to adjust or change gestures as required.

A 2-second pause (time. sleep(2)) is introduced between consecutive image captures, giving the user a brief moment to reset or adjust.

The script provides an option to exit the capture process prematurely. Pressing the 'q' key will break out of the image capture loop for the current label.
Clean Up:

Once all images for a particular label are captured, the webcam is released with cap. release(), freeing up the camera resource for other applications or subsequent runs.
In essence, Figure 1 depicts a systematic process for gathering a rich dataset of various ASL gestures, ensuring quality and consistency throughout the collection phase.

3.5 Data Augmentation
Understanding Augmentation: Data augmentation artificially enlarges the dataset by applying minor transformations (like rotations, zooming, or flipping) on the original images, which helps improve model accuracy and prevents overfitting.

Using Augmentor:

Installation: If not installed, use pip install Augmentor within your Python environment.
Pipeline Creation: For each gesture label directory in CollectedImages10, initialize an Augmentor pipeline.
Augmentation Operations: Define the types of augmentations you want. For example, random rotations, flips, or zooming.
Execution: Execute the pipeline to generate augmented images.

3.6 Image Annotation

Setting Up LabelImg:
Installation: Clone the LabelImg repository and set it up according to its official guidelines.
Launching: Once installed, launch LabelImg.
Annotation Process:
Directory Selection: In LabelImg, select the CollectedImages10 directory.
Bounding Box Drawing: For each image, manually draw a rectangle around the gesture.

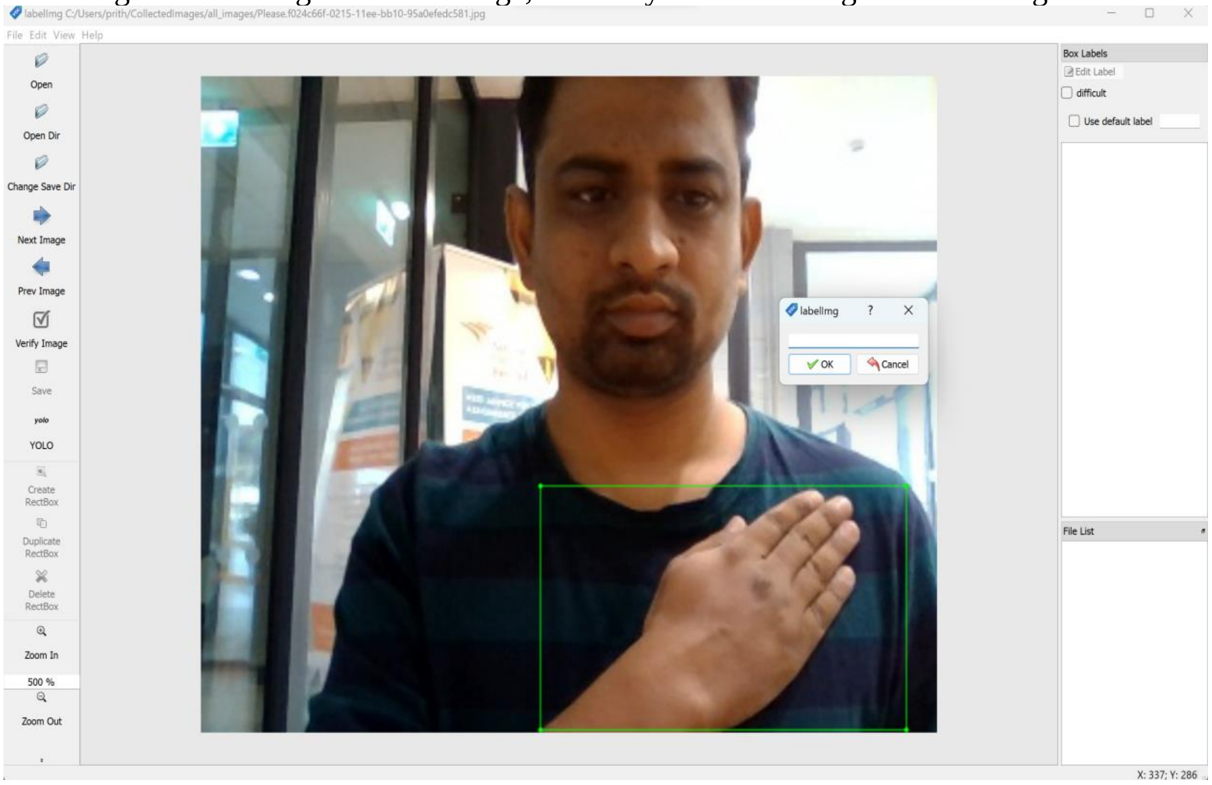


Figure 2: LabelImg Tool Interface showcasing the bounding box annotation process for ASL gesture images.

Label Assignment: After drawing the rectangle, assign an appropriate label to the bounding box.
Saving Annotations: LabelImg will create XML files by default. Ensure you save the annotations in YOLO format, which will produce a .txt file for each image.

4. Model Configuration and Training

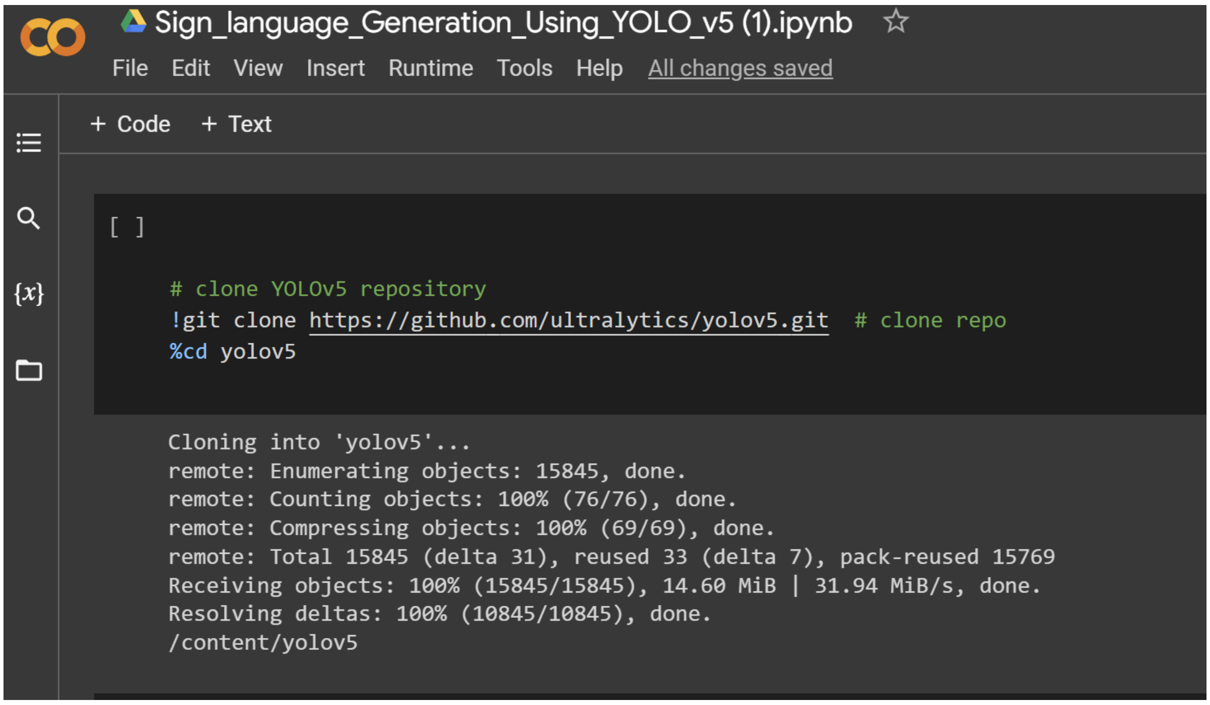
4.1 Setting Up the Google Colab Environment

Before starting the model configuration and training process, Google Colab, a cloud-based Python development environment that offers free GPU support, is utilized.

Accessing Google Colab: Visit Google Colab.
Creating a New Notebook: Click on "New Notebook" to start a fresh session.
Activating GPU: Navigate to Runtime > Change runtime type and select GPU from the hardware accelerator dropdown. This ensures that the model training will utilize GPU resources, speeding up the process considerably.

4.2 Cloning and Setting Up the YOLOv7 Repository

Repository Cloning: Clone the YOLOv7 repository by executing !git clone https://github.com/WongKinYiu/yolov7 in a Google Colab cell. This command fetches all necessary files and codebase required for YOLOv7.
Entering the Directory: Navigate to the cloned directory using %cd yolov7.
Installing Dependencies: Install all required libraries and dependencies by running !pip install -qr requirements. txt.


Sign_language_Generation_Using_YOLO_v5 (1).ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
[ ]

    # clone YOLOv5 repository
    !git clone https://github.com/ultralytics/yolov5.git  # clone repo
    %cd yolov5

    Cloning into 'yolov5'...
    remote: Enumerating objects: 15845, done.
    remote: Counting objects: 100% (76/76), done.
    remote: Compressing objects: 100% (69/69), done.
    remote: Total 15845 (delta 31), reused 33 (delta 7), pack-reused 15769
    Receiving objects: 100% (15845/15845), 14.60 MiB | 31.94 MiB/s, done.
    Resolving deltas: 100% (10845/10845), done.
    /content/yolov5
```

Figure 3: Cloning the YOLOv7 repository from GitHub for the setup and initialization of the model training environment.

4.3 Uploading and Preparing the Dataset

Dataset Upload: The dataset you've prepared, including images and annotations, should be zipped and uploaded to Google Colab. Use the Colab file upload feature or drag-and-drop the dataset zip (e.g., sdata.zip).
Unzipping the Dataset: Extract the uploaded dataset into the yolov7 directory using !unzip /content/sdata.zip -d /content/yolov7/.

4.4 YAML Configuration

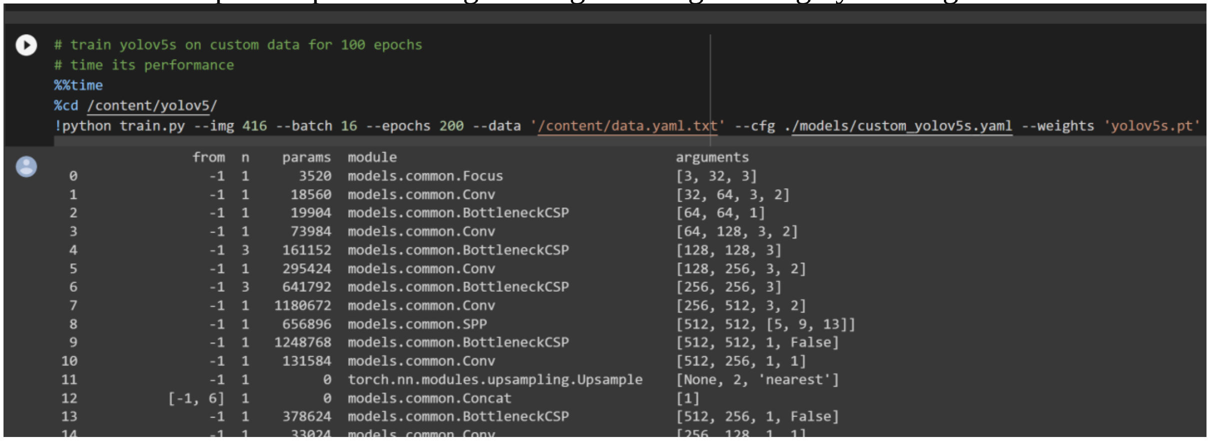
The YAML (.yaml) file is essential for YOLO's training process. It defines parameters like the number of classes, class names, and paths to training and validation datasets.

Reading the YAML File: Load your predefined YAML file (data.yaml.txt) to determine the number of classes using the provided Python script.
Customizing Model Configuration: Customize the default YOLOv5s model configuration to match your dataset. This can include defining the number of classes, adjusting model depth, width, or architecture. Use the given writetemplate function to modify and save changes.

4.5 Training the Model

Model Training: Start training by running the provided script, setting appropriate image size, batch size, epochs, data paths, model configuration, initial weights, and naming conventions.

Image Size: --img 416 specifies the size of the input images for the model.
Batch Size: --batch 16 defines how many images are processed simultaneously.
Epochs: --epochs 200 sets the number of training cycles.
Data Path: '--data /content/data.yaml.txt' points to the YAML configuration file.
Model Configuration: --cfg ./models/custom_yolov5s.yaml specifies the modified model configuration.
Weights: 'yolov5s.pt' denotes the pre-trained weights used as the starting point.
Name: --name yolov5s_results gives a name to the training run for easier identification.
Cache: --cache speeds up the loading of images during training by caching them.



```
# train yolov5s on custom data for 100 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 200 --data '/content/data.yaml.txt' --cfg ./models/custom_yolov5s.yaml --weights 'yolov5s.pt'
```

```
                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Focus                     [3, 32, 3]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     19904  models.common.BottleneckCSP             [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  3    161152  models.common.BottleneckCSP             [128, 128, 3]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    641792  models.common.BottleneckCSP             [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1    656896  models.common.SPP                       [512, 512, [5, 9, 13]]
  9                -1  1   1248768  models.common.BottleneckCSP             [512, 512, 1, False]
 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    378624  models.common.BottleneckCSP             [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
```

Figure 4: Initiating the training process of the YOLO model on the curated ASL dataset

Monitoring with TensorBoard: To visually track and monitor the training process, activate TensorBoard using the %tensorboard --logdir runs command. This will provide a dynamic dashboard showing loss curves, accuracy, and other important metrics in real-time.

Visual Verification: After training, verify the model's learning by inspecting prediction results on training data. The provided Python script renders ground truth labels alongside model predictions for visual assessment.

Once the training is complete, you will have a trained model ready for testing and deployment. Always remember to save the trained weights and configuration files for future reference or further training.

5 Model Evaluation and Insights

In the pursuit of a robust and practical deep learning solution, training a model isn't enough. It's vital to evaluate its performance and derive actionable insights, ensuring its applicability in real-world scenarios.

5.1 Model Performance Metrics

Precision and Recall: Given the binary nature of object detection (the object is present or not), precision and recall offer crucial insights. Precision measures how many detected items are relevant, while recall captures how many relevant items are detected.

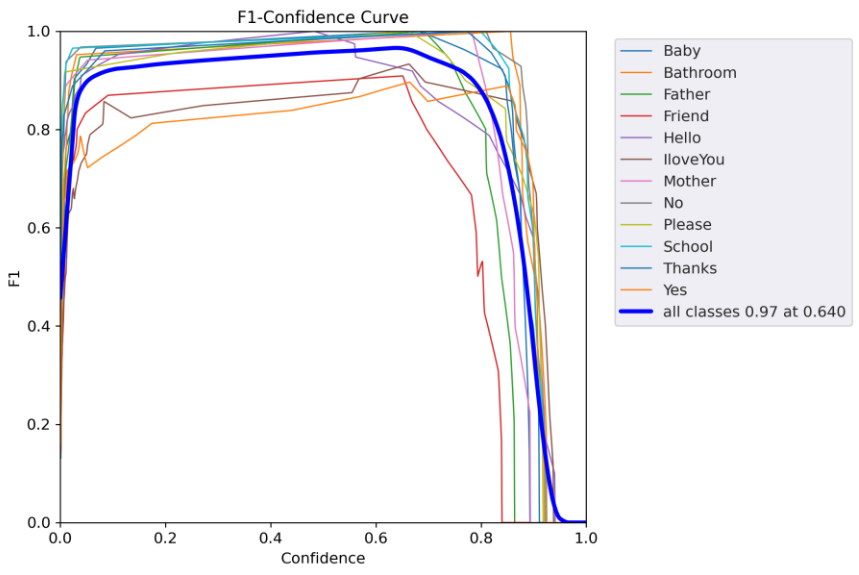


Figure 5 F1 Score Curve

F1 Score and Mean Average Precision (mAP): The F1 score harmonizes precision and recall. mAP, particularly crucial for object detection tasks, averages the maximum precision value across all recall levels.

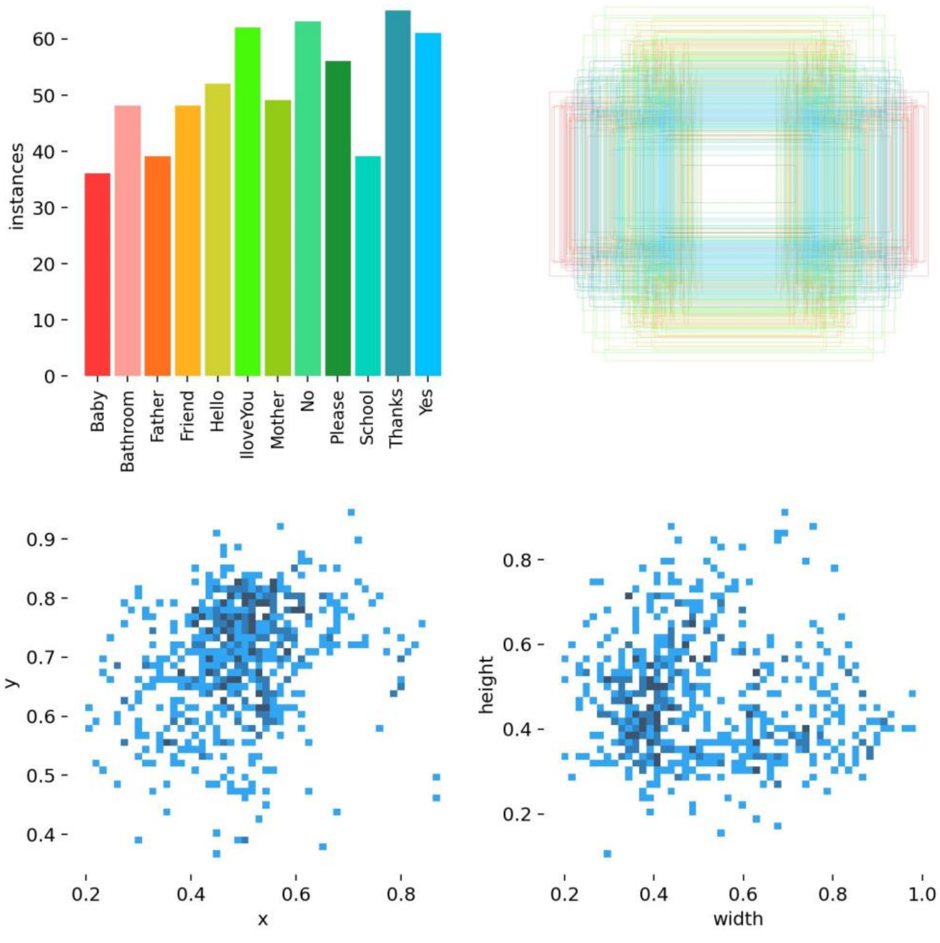


Figure 6 Labels

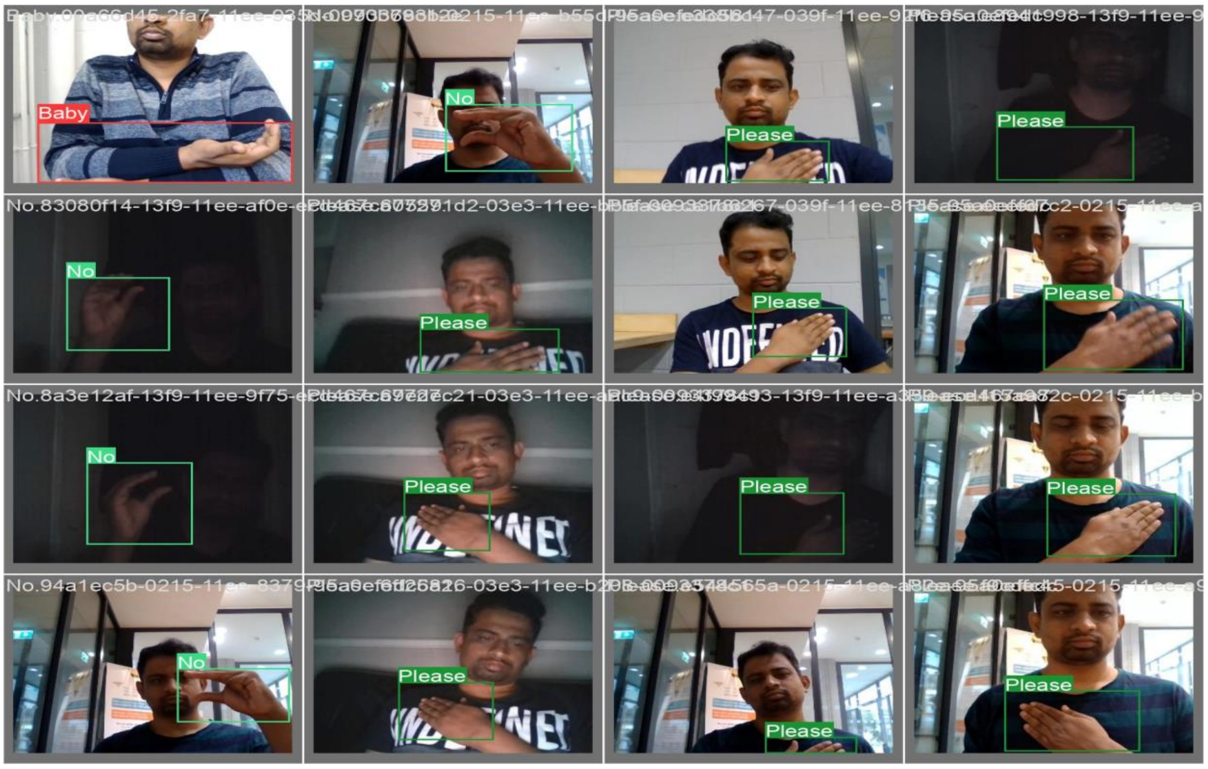


Figure 7 Train batch of YOLOv5s and YOLOv7 across Different Lighting Conditions

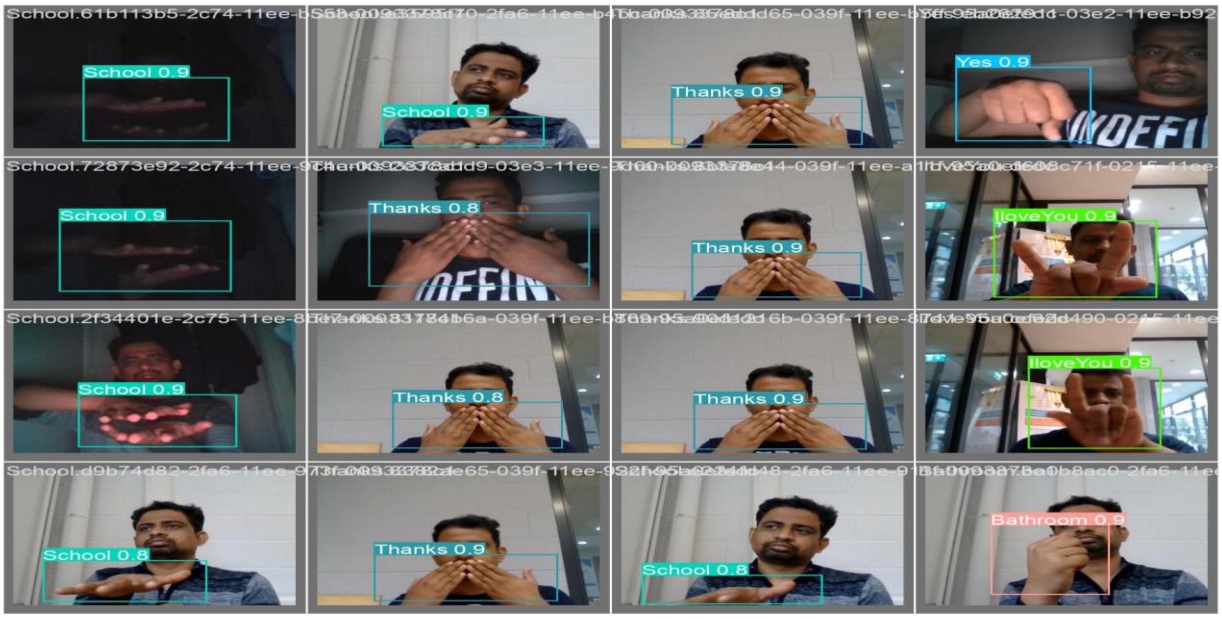
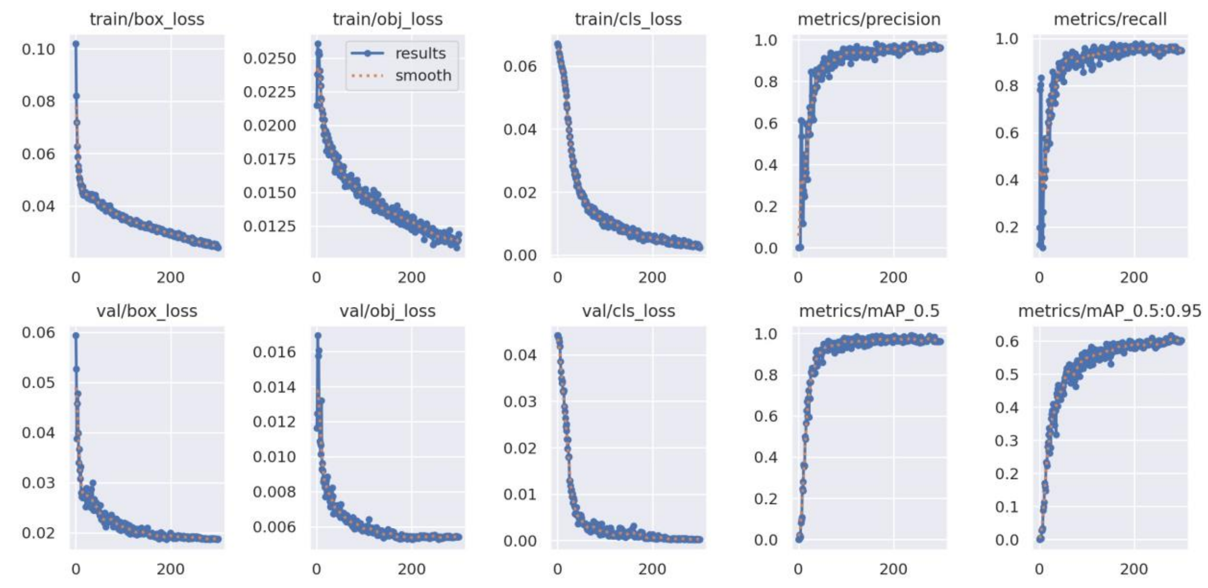


Figure 8 Validation batch of YOLOv5s and YOLOv7 across Different Lighting Conditions

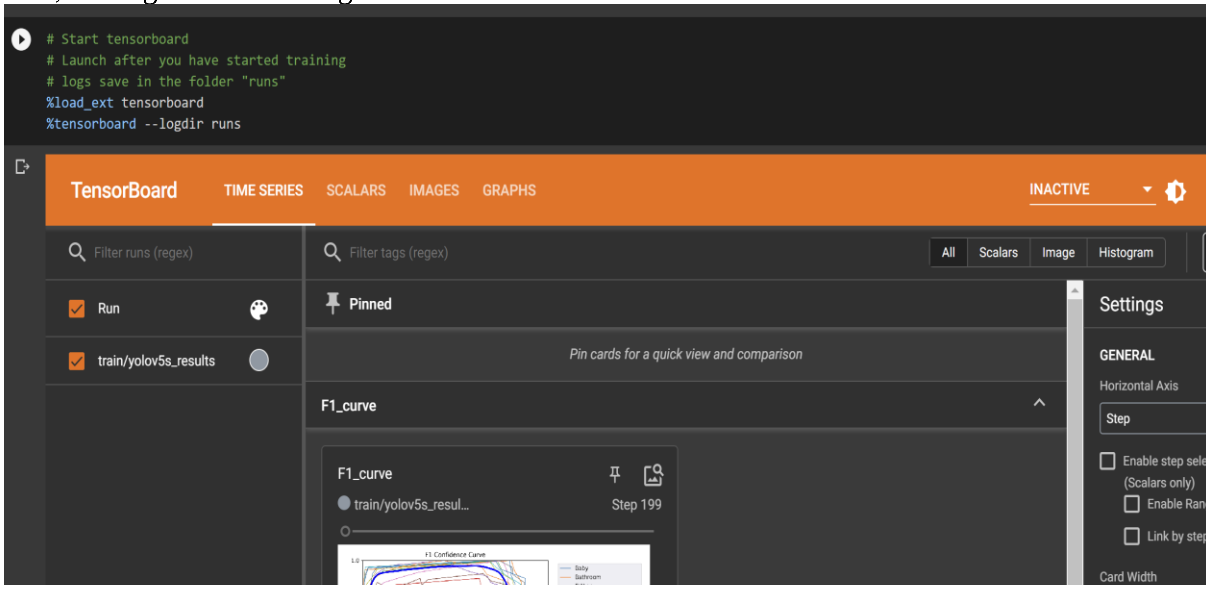
5.2 Visualization of Results

Loss Curves: These graphs give insights into the model's learning process, depicting how the error rates change over epochs. A falling curve symbolizes successful learning, while plateaus or increases may hint at issues like overfitting.



5.3 Diagnostic Tools

TensorBoard: Beyond real-time training metrics, TensorBoard can compare metrics across multiple runs, making model tweaking more data-driven.



```
# Start tensorboard
# Launch after you have started training
# logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
```

Class-specific Analysis: By breaking down performance metrics by class, one can identify if the model struggles with specific gestures or lighting conditions. This diagnostic is crucial for fine-tuning and ensuring balanced performance.

5.4 Post-Evaluation Tweaks

Class Weights Balancing: If certain classes are underrepresented in the dataset, the model might prioritize more abundant classes. Assigning class weights can mitigate this.

Augmentation Variations: If the model struggles with specific lighting conditions, consider augmenting the training dataset with more examples in such settings.

Hyperparameter Tuning: Parameters like learning rate, batch size, and optimizer choice can be revisited based on performance metrics.

References

Li, Y., Cheng, R., Zhang, C., Chen, M., Ma, J. and Shi, X., 2022, October. Sign language letters recognition model based on improved YOLOv5. In 2022 9th International Conference on Digital Home (ICDH) (pp. 188-193). IEEE.

Luo, W., 2022, July. Research on gesture recognition based on YOLOv5. In 2022 3rd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE) (pp. 447-450). IEEE.

Rastgoo, R., Kiani, K. and Escalera, S., 2021. Sign language recognition: A deep survey. Expert Systems with Applications, 164, p.113794.

Sahoo, A.K., Mishra, G.S. and Ravulakollu, K.K., 2014. Sign language recognition: State of the art. ARPN Journal of Engineering and Applied Sciences, 9(2), pp.116-134.

Sharma, P. and Anand, R.S., 2021. Deep models and optimizers for Indian sign language recognition.

Von Agris, U., Zieren, J., Canzler, U., Bauer, B. and Kraiss, K.F., 2008. Recent developments in visual sign language recognition. Universal Access

https://link.springer.com/article/10.1007/s41666-022-00114-1