

# PROGRESS REPORT 1

## SMART GLASSES DEVELOPMENT



---

## INTRODUCTION & PROBLEM STATEMENT:

Millions of people worldwide struggle with visual impairments that limit their ability to navigate safely, recognize objects or people, and access written information. Existing assistive tools often address only one aspect of the problem, such as text reading or obstacle detection, without providing unified or intelligent experience. This lack of integration reduces independence and confidence for visually impaired individuals in daily life.

The **Smart Assistive Ecosystem Project** aims to solve this challenge by developing a unified ecosystem that connects multiple assistive devices including Smart Glasses for the visually impaired and Smart Gloves for the deaf and mute through a central mobile application.

In this phase, we began with Smart Glasses, which represent the first device in our ecosystem. The purpose of developing the Smart Glasses first is to establish a reliable foundation for object detection, text reading, and environmental awareness using AI and sensor technology. These glasses will eventually integrate with other components to enable complete communication and navigation support within the ecosystem.

---

## AI ENGINEERING:

### DATASET IDENTIFICATION

**Dataset Title:** Egyptian New Currency 2023

**Source:** Developed by the graduation team from BANHA UNIVERSITY – FACULTY OF COMPUTERS AND ARTIFICIAL INTELLIGENCE.

### Overview:

The EGYPTIAN NEW CURRENCY 2023 dataset is a comprehensive and high-quality image collection designed for computer vision research in currency recognition and classification. It includes both traditional and newly introduced Egyptian banknotes, with images captured under varied real-world conditions to ensure robustness and practical usability in AI applications.

### Classes (Currency Denominations):

The dataset contains **eight classes**, each representing a distinct Egyptian banknote denomination:

1. 5 Egyptian Pound Banknote
2. 10 Egyptian Pound Banknote
3. 20 Egyptian Pound Banknote
4. New 10 Egyptian Pound Banknote
5. New 20 Egyptian Pound Banknote
6. 50 Egyptian Pound Banknote
7. 100 Egyptian Pound Banknote
8. 200 Egyptian Pound Banknote

### Image Diversity:

To improve the dataset's generalization ability, images were captured from multiple perspectives and under varied environmental conditions, including:

- **Frontal Views:** Head-on images highlighting main banknote features.
- **Angled and Rotated Shots:** Simulating different viewing positions and orientations.
- **Background Variations:** Including plain, textured, and natural backgrounds under different lighting conditions.

### Technical Details:

- **Image Resolution:** High-resolution images for detailed visual analysis.
- **Format:** Standardized image file formats compatible with common machine learning frameworks.

### Intended Use:

This dataset is intended for use in **machine learning and computer vision** tasks such as:

- Currency recognition and classification
- Object detection and image segmentation
- Model training for financial and security applications

Its real-world diversity and high quality make it a valuable benchmark for researchers developing intelligent recognition systems.

### Dataset link:

<https://www.kaggle.com/datasets/belalsafy/egyptian-new-currency-2023>

### Notebook Link:

<https://www.kaggle.com/code/rudainahaitham/final-currency>

➔ In this phase we used the Egyptian currency dataset to be trained on yolo11n model.

---

## DATA PREPARATION AND EXPLORATORY DATA ANALYSIS (EDA)

To begin, the dataset was first loaded and verified to ensure that all image files were correctly located in their respective directories. This step was essential to confirm the dataset's structure before proceeding with analysis and model training.

After confirming that all files existed in the specified paths, we analyzed the dataset's structure by visualizing the **number of images available for each currency class**. A bar chart was created to display the image count per denomination, giving a clear overview of the dataset's distribution across all eight classes.

```
[7]: data_path = '/kaggle/input/egyptian-new-currency-2023/dataset'
train_path = os.path.join(data_path, 'train')
val_path = os.path.join(data_path, 'valid')

print(f'Train path exists: {os.path.exists(train_path)}')
print(f'Val path exists: {os.path.exists(val_path)}')

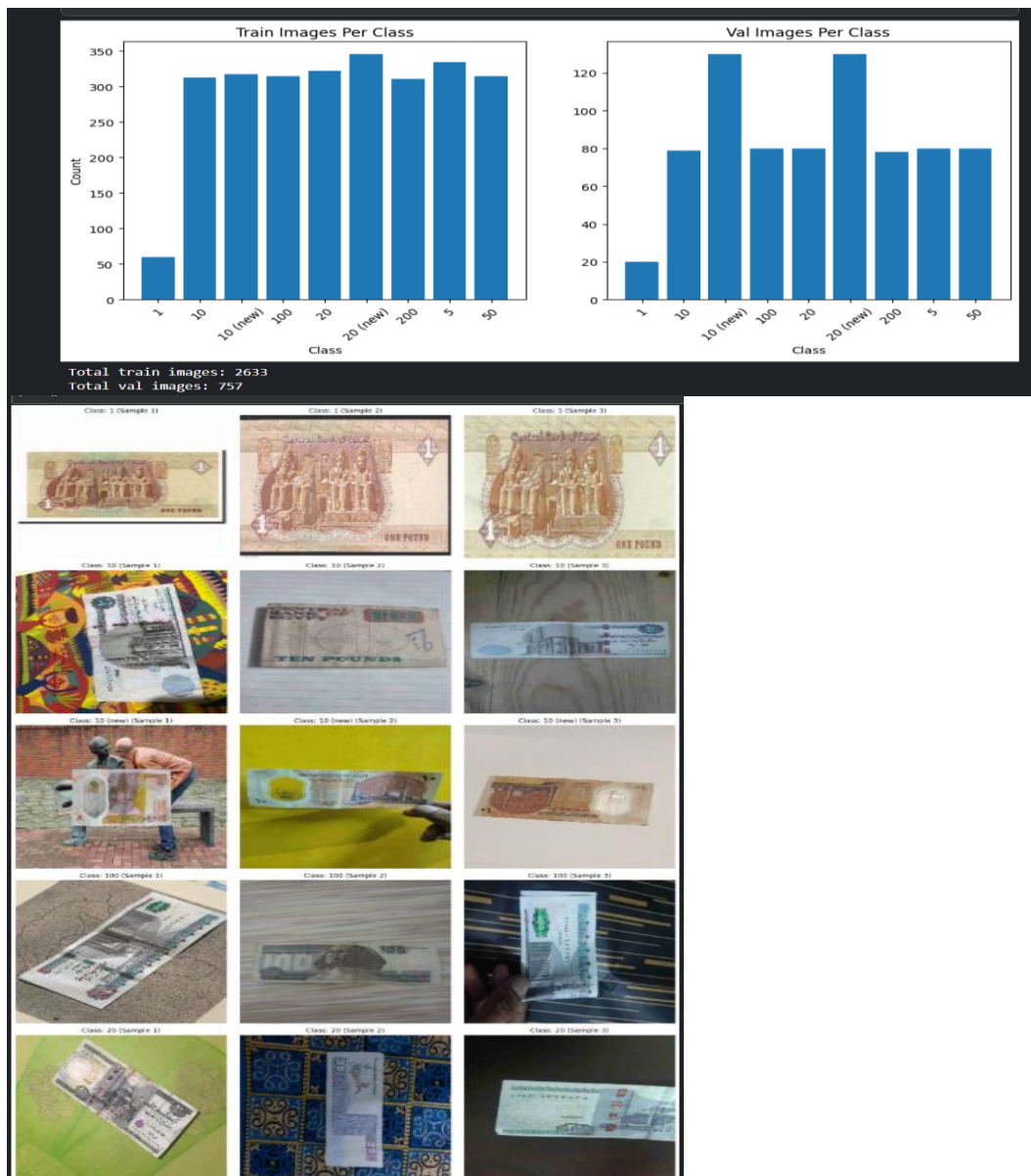
Train path exists: True
Val path exists: True

[8]: classes = sorted([d for d in os.listdir(train_path) if os.path.isdir(os.path.join(train_path, d))])
nc = len(classes)
print(f'Number of classes: {nc}')
print(f'Classes: {classes}')

Number of classes: 9
Classes: ['1', '10', '10 (new)', '100', '20', '20 (new)', '200', '5', '50']
```

## SAMPLE VISUALIZATION

To better understand the dataset, several **sample images** from different classes were displayed. This provided insight into the dataset's diversity in terms of lighting conditions, backgrounds, and orientations.



---

## IMAGE SIZE ANALYSIS

We then examined the **dimensions of the images (Height, Width, and Channels)**. The images were found to have varying sizes, which indicated the need for resizing during the data preprocessing phase to maintain consistency for model training.

```
[11]: sample_sizes = []
      for cls in classes:
          cls_path = os.path.join(train_path, cls)
          images = [f for f in os.listdir(cls_path) if f.endswith('.jpg')]
          if images:
              img_path = os.path.join(cls_path, images[0])
              img = cv2.imread(img_path)
              sample_sizes.append((cls, img.shape))

      print("Sample image sizes (class, (H, W, C)):")
      for s in sample_sizes:
          print(s)
```

```
Sample image sizes (class, (H, W, C)):
('1', (101, 201, 3))
('10', (1280, 720, 3))
('10 (new)', (427, 640, 3))
('100', (1280, 960, 3))
('20', (960, 1280, 3))
('20 (new)', (427, 640, 3))
('200', (1280, 960, 3))
('5', (360, 480, 3))
('50', (720, 1280, 3))
```

---

## YAML FILE CONFIGURATION

A **YAML configuration file** was created for the YOLO model to define the dataset paths and class names.

This file acts as a reference for the YOLO training process, specifying:

- The directories for **training**, **validation**, and **testing** images.
- The **names of the classes** (the eight currency denominations).

Although other formats such as JSON can be used, YAML is the simplest and most widely supported configuration format for YOLO models.

During initial testing, the model produced an error indicating that it required a **directory path instead of a file path**. After adjusting the configuration to point directly to the Kaggle /input dataset folder, the issue was resolved successfully.

```

names:
- '1'
- '10'
- 10 (new)
- '100'
- '20'
- 20 (new)
- '200'
- '5'
- '50'
nc: 9
path: /kaggle/input/egyptian-new-currency-2023/dataset
train: train
val: test

```

```

[12]: yaml_data = {
      'path': '/kaggle/input/egyptian-new-currency-2023/dataset',
      'train': 'train',
      'val': 'test',
      'nc': len(classes),
      'names': classes
    }

    yaml_path = '/kaggle/working/currency.yaml'
    with open(yaml_path, 'w') as f:
        yaml.dump(yaml_data, f, default_flow_style=False)

    print(f"YAML created at: {yaml_path}")
    with open(yaml_path, 'r') as f:
        print(f.read())

```

YAML created at: /kaggle/working/currency.yaml

## Training Results

The model was trained for several epochs, and the **training and validation performance metrics** were recorded.

The training results showed **100% accuracy in the Top-5 accuracy metric**, suggesting the possibility of **overfitting**.

To further investigate this, we performed evaluations on both the **validation** and **test sets**, and generated:

- Confusion matrices for training and validation sets.
- Accuracy and loss graphs across epochs.

	all	0.989	1		
Epoch	GPU_mem	loss	Instances	Size	
48/50	1.12G	0.2171	7	224: 100%	165/165 5.6it/s 29.4s0.1ss
	classes	top1_acc	top5_acc: 100%		24/24 4.8it/s 5.0s0.2s
	all	0.989	1		
Epoch	GPU_mem	loss	Instances	Size	
49/50	1.13G	0.1956	7	224: 100%	165/165 5.7it/s 29.1s0.4s
	classes	top1_acc	top5_acc: 100%		24/24 4.6it/s 5.2s0.2ss
	all	0.991	1		
Epoch	GPU_mem	loss	Instances	Size	
50/50	1.14G	0.1713	7	224: 100%	165/165 5.8it/s 28.6s0.3ss
	classes	top1_acc	top5_acc: 100%		24/24 4.6it/s 5.2s0.3ss
	all	0.989	1		

Test:

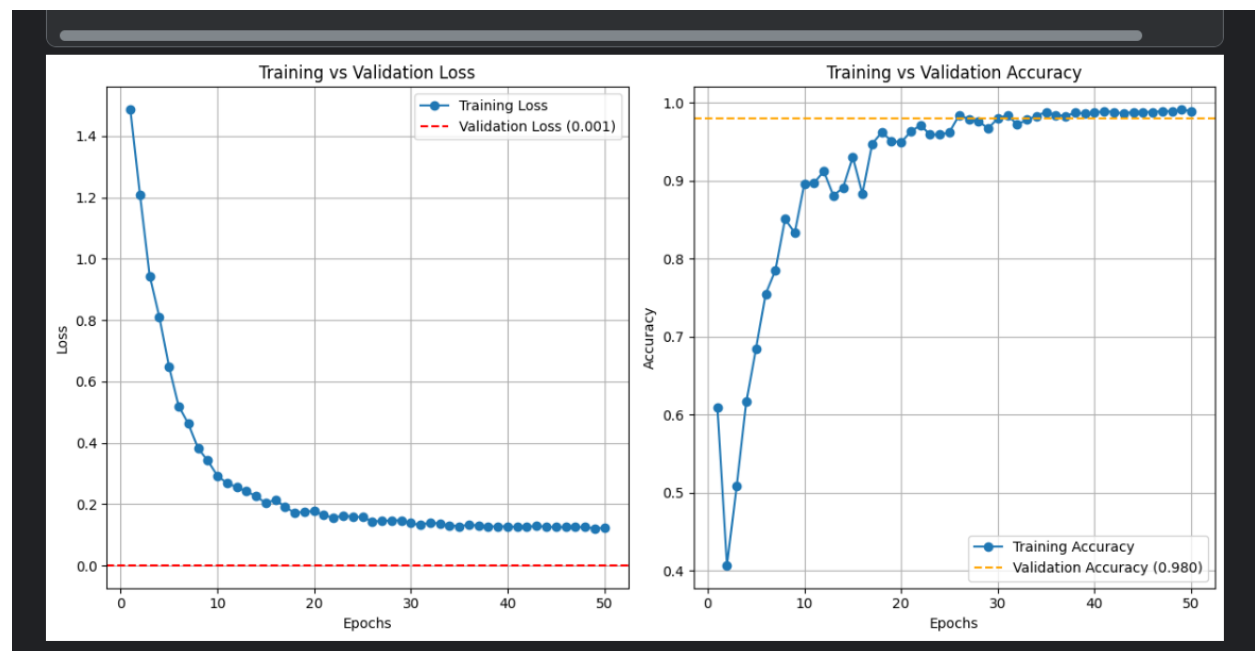
```
Read-only file system: '/kaggle/input/egyptian-new-currency-2023/dataset/test/50/50.1.jpg'
WARNING ⚠ test: Cache directory /kaggle/input/egyptian-new-currency-2023/dataset is not writeable, cache not saved.
classes top1_acc top5_acc: 100% ————— 18/18 4.8it/s 3.7s0.4s
/usr/local/lib/python3.11/dist-packages/matplotlib/colors.py:721: RuntimeWarning: invalid value encountered in less
  xa[xa < 0] = -1
/usr/local/lib/python3.11/dist-packages/matplotlib/colors.py:721: RuntimeWarning: invalid value encountered in less
  xa[xa < 0] = -1
all 0.983 1
```

Validation:

```
classes top1_acc top5_acc: 100% ————— 48/48 6.5it/s 7.4s0.1s
/usr/local/lib/python3.11/dist-packages/matplotlib/colors.py:721: RuntimeWarning: invalid value encountered in less
  xa[xa < 0] = -1
/usr/local/lib/python3.11/dist-packages/matplotlib/colors.py:721: RuntimeWarning: invalid value encountered in less
  xa[xa < 0] = -1
all 0.991 1
```

## Performance Visualization

Graphs were plotted to illustrate **training and validation accuracy and loss** over epochs. The validation curve appeared as a straight line because the validation process was executed only once, providing a single value for each metric.





Validation test:

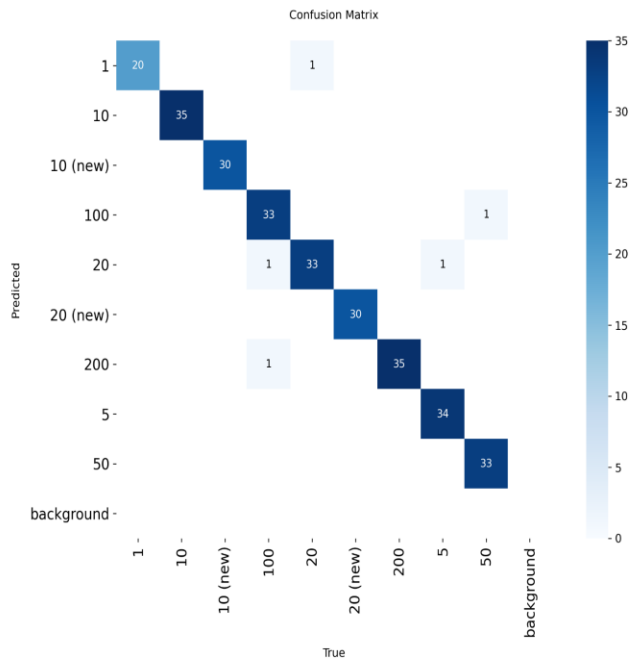




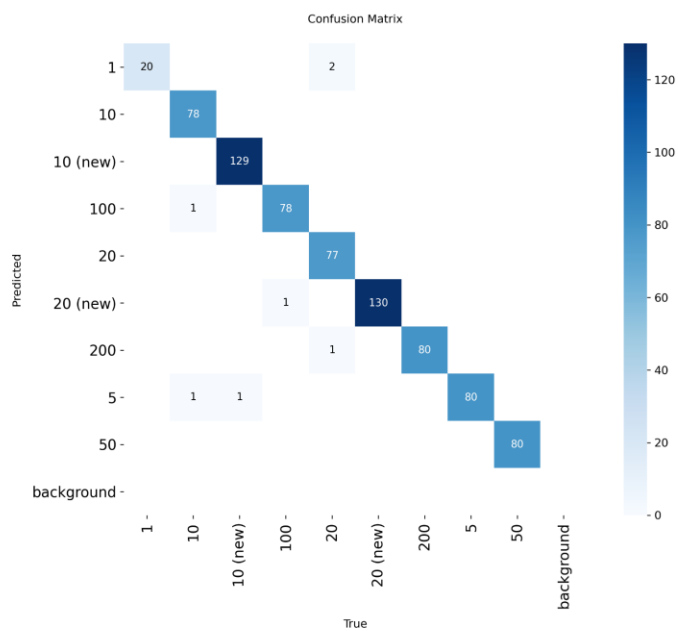
## Confusion Matrices

Confusion matrices were generated for both **training** and **validation** datasets to visualize model performance across different classes. These matrices helped identify where the model performed well and where misclassifications occurred.

-Confusion Matrix for validation:



-Confusion Matrix for train:



**Dataset Title:** COCO 2017 (Common Objects in Context)

**Source:**

Developed by Microsoft Research; introduced in “MICROSOFT COCO: COMMON OBJECTS IN CONTEXT” (Lin et al., 2014).

Research Paper: [arXiv:1405.0312](https://arxiv.org/abs/1405.0312)

**Overview:**

COCO 2017 is a large-scale dataset designed for **object detection, segmentation, and image captioning**. It contains natural, complex scenes where multiple objects appear in realistic contexts, encouraging models to understand not just objects but their relationships and environments.

**Key Characteristics:**

- Around **330K images** with rich annotations.
- Tasks include **detection, segmentation, keypoint estimation, and captioning**.
- Split into **train, validation, and test** sets (Train2017  $\approx$ 118K, Val2017  $\approx$ 5K).
- Each image includes multiple labeled objects in natural settings.

**Classes (Object Categories):**

The dataset contains **80 object classes**, covering everyday items such as:

person, bicycle, car, dog, cat, chair, cup, bottle, laptop, book, clock, ... up to toothbrush.

(A full list of 80 classes can be added in the appendix.)

**Intended Use:**

Used as a **benchmark** for developing and evaluating computer vision models, including YOLO, Faster R-CNN, and Mask R-CNN. It provides pretraining data that helps models learn general object features before fine-tuning on custom datasets.

**Relevance to Project:**

COCO’s diversity and complexity make it ideal for **pretraining** or **benchmarking** object detection models before adapting them to specialized domains—such as **Egyptian currency recognition**. Its variation in lighting, angles, and object contexts aligns well with your dataset’s design.

**Notes:**

- Focuses on **general objects**, not specific domains like currency.
- **Imbalanced classes** exist but are manageable through augmentation or weighting.
- Serves as a strong **baseline** for evaluating detection and segmentation performance.

**Dataset link:**

<https://www.kaggle.com/datasets/awsaf49/coco-2017-dataset/data>

## Notebook link:

<https://www.kaggle.com/code/rudainahaitham/yolo11n-on-coco>

➔ In this phase we used the COCO dataset to be trained on yolo11n model.

---

### DATA PREPARATION AND EXPLORATORY DATA ANALYSIS (EDA)

The COCO 2017 dataset was first **loaded and verified** to ensure that all image files were correctly located in their respective directories before beginning the analysis and model training process.

After confirming the dataset paths, we **analyzed the image distribution** across all 80 classes. A bar chart was generated to visualize the **number of images per class**, providing a clear understanding of the dataset's balance and diversity.

```
[17]: data_path = '/kaggle/input/coco-2017-dataset/coco2017'
train_img_path = os.path.join(data_path, 'train2017')
val_img_path = os.path.join(data_path, 'val2017')
ann_path = os.path.join(data_path, 'annotations')

train_ann_file = os.path.join(ann_path, 'instances_train2017.json')
val_ann_file = os.path.join(ann_path, 'instances_val2017.json')

print(f'Train path exists: {os.path.exists(train_img_path)}')
print(f'Val path exists: {os.path.exists(val_img_path)}')
print(f'Annotation files exist: {os.path.exists(train_ann_file)}, {os.path.exists(val_ann_file)}')

Train path exists: True
Val path exists: True
Annotation files exist: True, True
```

```
[18]: with open(train_ann_file, 'r') as f:
      train_data = json.load(f)
      with open(val_ann_file, 'r') as f:
          val_data = json.load(f)

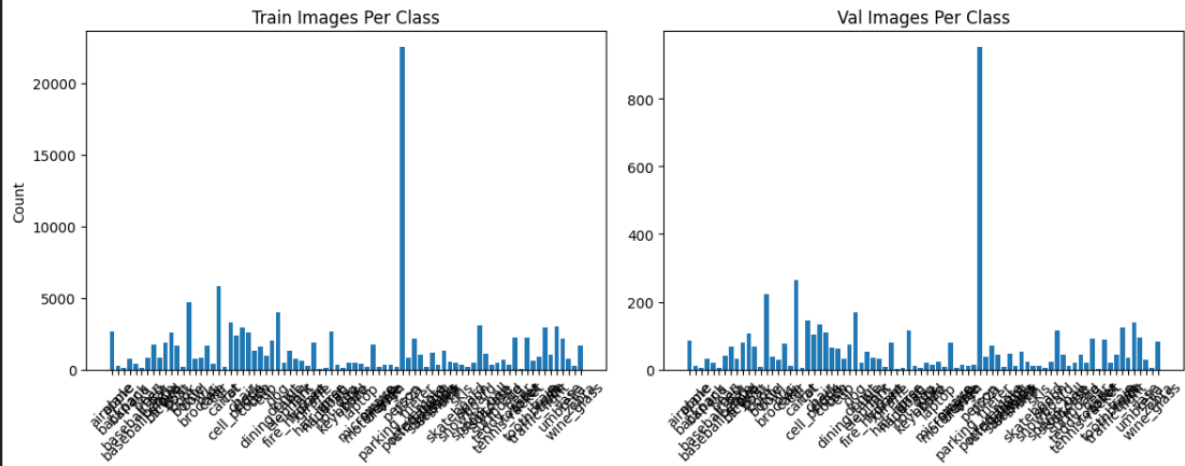
[12]: data_path = output_path
train_path = os.path.join(data_path, 'train2017')
val_path = os.path.join(data_path, 'val2017')

# Check if paths exist
print(f'Train path exists: {os.path.exists(train_path)}')
print(f'Val path exists: {os.path.exists(val_path)}')

Train path exists: True
Val path exists: True
```

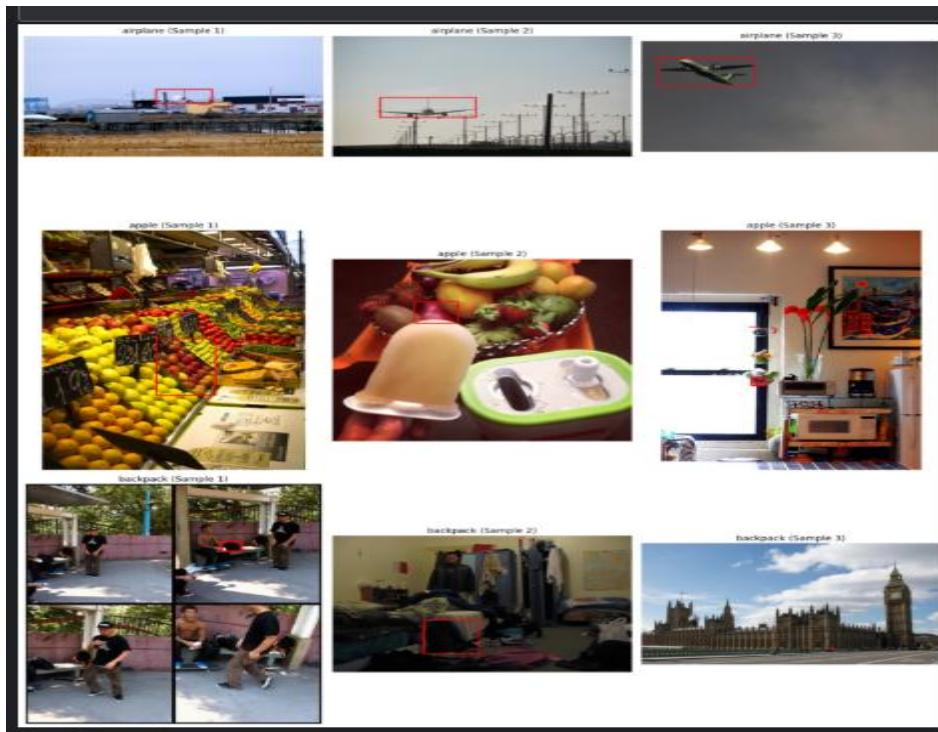
Number of classes: 80

Classes: ['airplane', 'apple', 'backpack', 'banana', 'baseball\_bat', 'baseball\_glove', 'bear', 'bed', 'bench', 'bicycle', 'bird', 'boat', 'book', 'bottle', 'bowl', 'broccoli', 'bus', 'cake', 'car', 'carrot', 'cat', 'cell\_phone', 'chair', 'clock', 'couch', 'cow', 'cup', 'dining\_table', 'dog', 'donut', 'elephant', 'fire\_hydrant', 'fork', 'frisbee', 'giraffe', 'hair\_drier', 'handbag', 'horse', 'hot\_dog', 'keyboard', 'laptop', 'knife', 'microwave', 'motorcycle', 'mouse', 'orange', 'oven', 'parking\_meter', 'person', 'pizza', 'potted\_plant', 'refrigerator', 'remote', 'sandwich', 'scissors', 'sheep', 'sink', 'skateboard', 'skis', 'snowboard', 'spoon', 'sports\_ball', 'stop\_sign', 'suitcase', 'surfboard', 'teddy\_bear', 'tennis\_racket', 'tie', 'toaster', 'toilet', 'toothbrush', 'traffic\_light', 'train', 'truck', 'tv', 'umbrella', 'vase', 'wine\_glass', 'zebra']



## SAMPLE VISUALIZATION

To gain a better understanding of the data, several **sample images** from different categories were displayed. These samples illustrate the dataset's variety in object types, backgrounds, and scene complexity.



---

## YAML FILE CREATION

A **YAML configuration file** was then created to define the dataset paths and class names required for YOLO model training. This file helps the model identify where the training, validation, and testing images are located and what object categories to recognize.

```
[49]: import yaml

yaml_data = {
    'path': '/kaggle/input/coco-2017-dataset/coco2017',
    'train': 'train2017',
    'val': 'val2017',
    'test': 'test2017',
    'nc': 80,
    'names': [
        'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light',
        'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
        'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
        'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',
        'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
        'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
        'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
        'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear',
        'hair drier', 'toothbrush'
    ]
}

yaml_path = '/kaggle/working/coco.yaml'
```

---

## MODEL LOADING AND ERRORS

During model training, several issues were encountered:

- When loading data directly from the **Kaggle /input** directory, an error occurred indicating that **no images were found**.
- When using the YAML configuration, another error appeared related to **OpenCV and NumPy compatibility**, which has not yet been fully resolved.

These issues are currently under investigation to ensure proper dataset loading and successful model initialization for the next training phase.

```
error: OpenCV(4.12.0) :-1: error: (-5:Bad argument) in function 'resize'
> Overload resolution failed:
> - src is not a numpy array, neither a scalar
> - Expected Ptr<cv::UMat> for argument 'src'
```

---

## HARDWARE IMPLEMENTATION: ESP32-CAM LIVE STREAMING SETUP

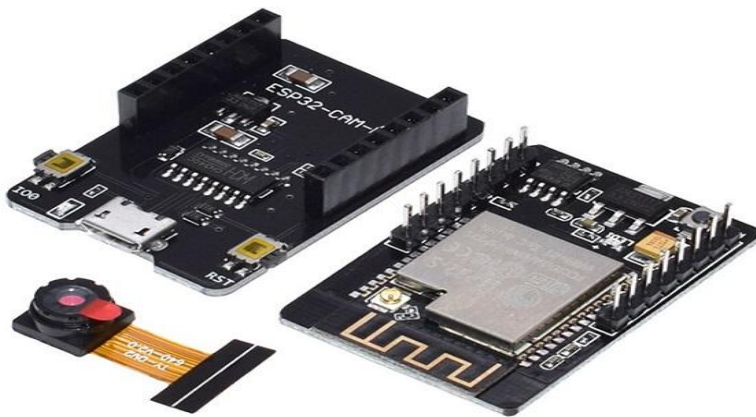
The hardware component of this project uses the **ESP32-CAM** module to capture and stream live video directly to the application. The ESP32-CAM is a low-cost camera module based on the ESP32 chip, featuring built-in Wi-Fi and Bluetooth connectivity. This makes it ideal for real-time image capture and transmission tasks in lightweight AI or IoT applications.

---

---

### REQUIRED COMPONENTS

- ESP32-CAM module (AI-Thinker version)
- ESP32-CAM MB module



---

### SOFTWARE SETUP

The ESP32-CAM module was programmed using the **Arduino IDE**. The following steps were performed to install and configure the necessary drivers and environment:

1. **Add ESP32 Board to Arduino IDE:**
  - Open **File** → **Preferences**.
  - In the “Additional Boards Manager URLs” field, add:  
[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)
  - Then navigate to **Tools** → **Board** → **Boards Manager**, search for **ESP32**, and click **Install**.
2. **Select the Correct Board:**
  - From **Tools** → **Board**, select:  
AI Thinker ESP32-CAM



### 3. Configure the Board Settings:

- **Flash Frequency:** 80 MHz
  - **Partition Scheme:** Huge App (3MB No OTA)
  - **Upload Speed:** 115200
  - **Port:** Select the COM port corresponding to the MB module
  - **Flash Mode:** QIO
  - **CPU Frequency:** 240 MHz
- 

## CODE SETUP

The **CameraWebServer** example was used for live video streaming.  
Steps:

1. Open **File** → **Examples** → **ESP32** → **Camera** → **CameraWebServer**.
2. In the code, locate and edit the following section to configure the Wi-Fi mode:
3. `const char* ssid = "ESP32-CAM-AP";`
4. `const char* password = "12345678";`

This configuration makes the ESP32-CAM act as an **Access Point (AP)** — creating its own local Wi-Fi network.

Any **mobile device or laptop** can connect directly to this network, allowing **live streaming without external Wi-Fi**.

```
const char* ssid = "ESP32-CAM-AP";  
const char* password = "12345678";
```

```
WiFi.softAP(ssid, password);  
Serial.println("Access Point Started");  
Serial.print("IP address: ");  
Serial.println(WiFi.softAPIP());
```

---

## UPLOADING THE CODE

1. Click **Upload** in Arduino IDE.
  2. Once the upload is complete **press the reset (RST)** button to run the program.
-

## TESTING THE LIVE STREAM

After the upload:

1. Open the **Serial Monitor** at **115200 baud rate**.
2. The ESP32-CAM will display an **IP address**, typically 192.168.4.1.
3. Connect your device (mobile or laptop) to the **ESP32-CAM Wi-Fi network**.
4. Open the displayed IP address in any web browser to start **live video streaming** directly.



## INTEGRATION WITH THE APPLICATION

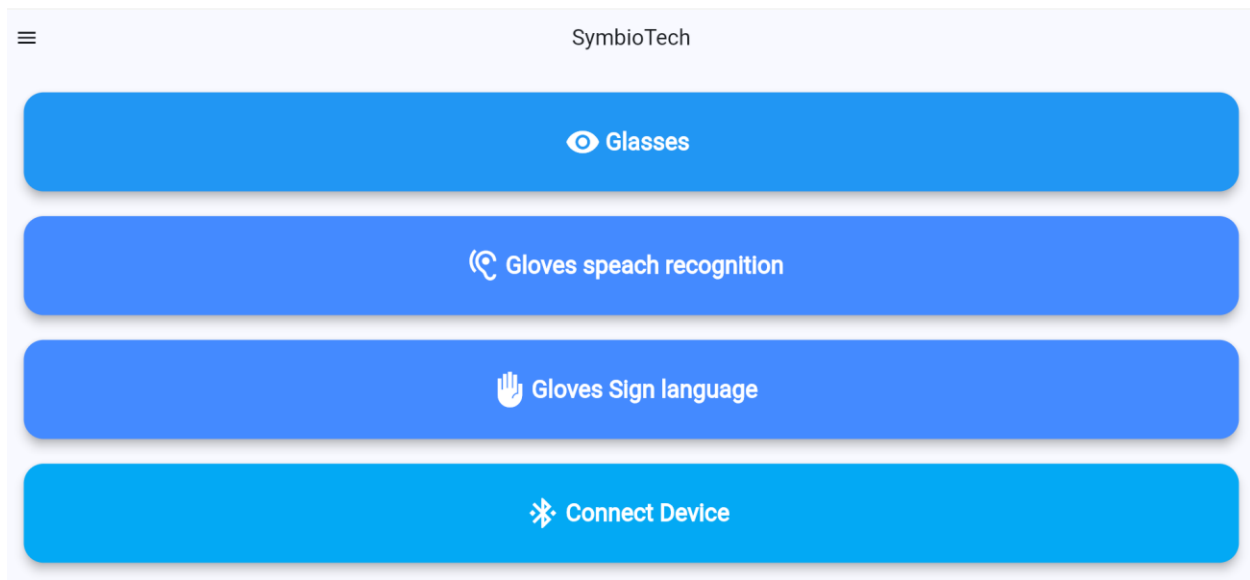
Once the ESP32-CAM live stream was verified, it was integrated with the developed **application**. The live feed was streamed directly to the interface, enabling **real-time image acquisition** for further analysis and processing.

## Software Engineering:

### MOBILE APPLICATION

---

#### 1. MAIN HOME SCREEN PRIMARY NAVIGATION



#### Description:

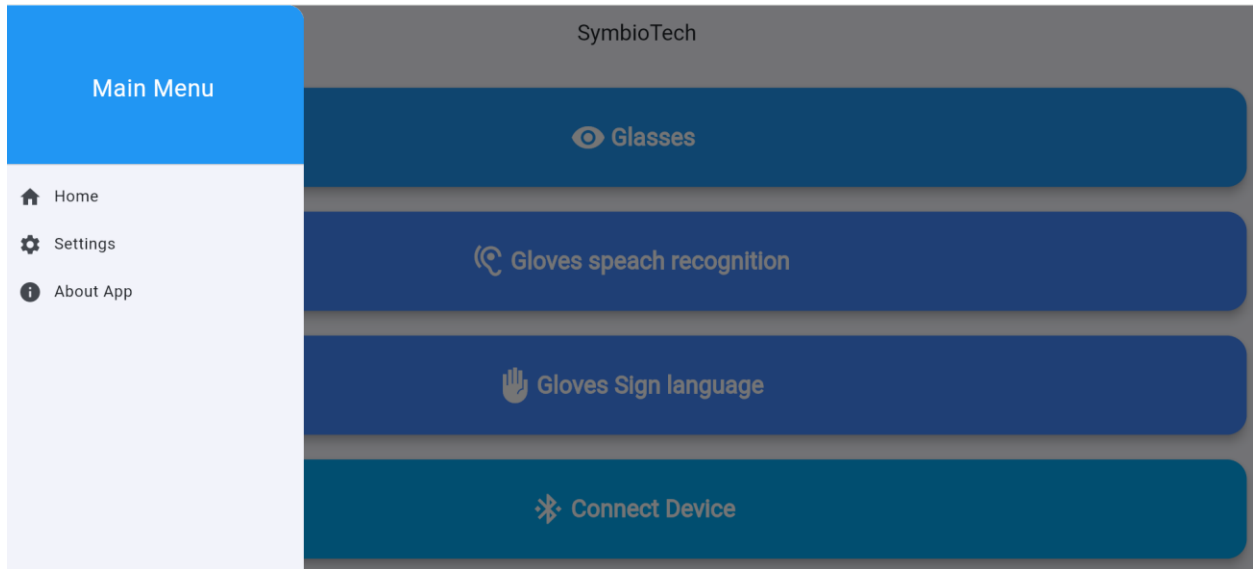
The home screen features four large, color-coded buttons for easy navigation:

- Glasses → Live camera stream for text/scene recognition.
- Gloves Speech Recognition → Converts speech to on-screen captions.
- Gloves Sign Language → Translates hand gestures into text/speech (placeholder).
- Connect Device → Pairs with external assistive hardware.

#### Accessibility Features:

- High-contrast blue buttons with bold white text.
- Icons + labels for visual and cognitive clarity.
- Spacious layout for users with motor impairments.

## 2. Main Menu Drawer Quick Access Panel



Description:

The side drawer provides quick access to core sections:

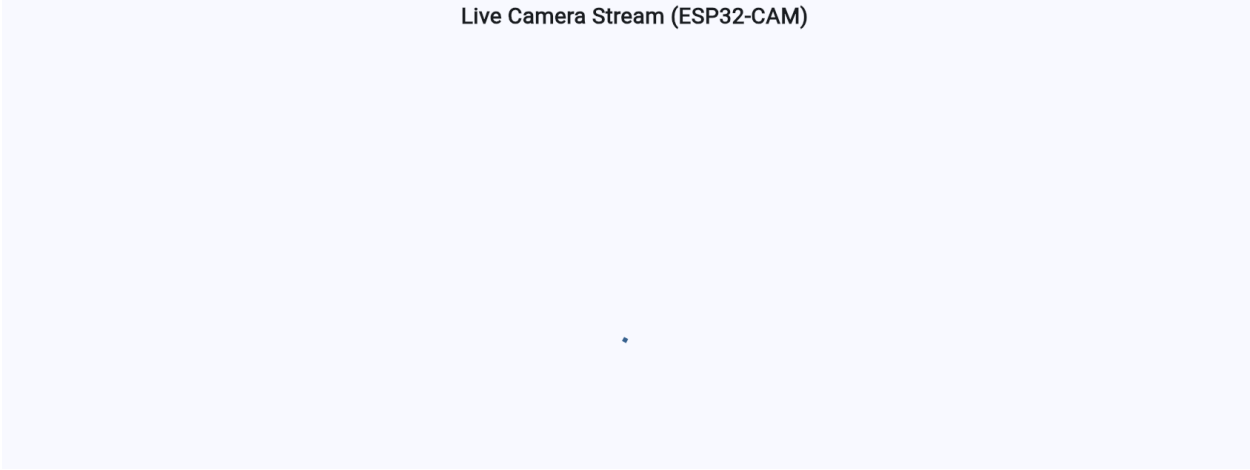
- Home → Return to main dashboard.
- Settings → Customize app preferences (future feature).
- About App → Learn about SymbioTech's mission.

Accessibility Features:

- Clear iconography and consistent typography.
  - Dark background with white text for low-vision users.
  - Simple tap-to-navigate design.
-

### 3. GLASSES FEATURE LIVE CAMERA STREAM (ESP32-CAM)

#### Live Camera Stream (ESP32-CAM)



#### Description:

This screen displays a live MJPEG video feed from an ESP32-CAM device (typically mounted on assistive glasses).

- Title: “Live Camera Stream (ESP32-CAM)”
- The video container has rounded corners and fills the available space.
- If the stream fails, an error message appears: “Error loading camera stream. Please connect to ESP32-CAM Wi-Fi.”

#### Accessibility Features:

- Real-time visual input for users with low vision.
  - Future integration will include OCR/text-to-speech for reading signs, menus, or documents.
  - Error handling guides users to fix connectivity issues.
-

**DESIGN PHILOSOPHY FOR SPECIAL NEEDS**

SymbioTech was developed with inclusive design principles at its core:

Large Touch Targets	Easier interaction for users with motor impairments
High Contrast UI	Improved visibility for low-vision users
Icon + Text Labels	Supports users with cognitive or literacy challenges
Minimalist Layout	Reduces cognitive overload
Hardware Integration	Enables independence via wearable assistive tech

**TECHNICAL IMPLEMENTATION**

- Built with Flutter for cross-platform compatibility (iOS & Android).
- Uses flutter\_mjpeg package to display live streams from ESP32-CAM devices.
- Modular architecture allows future expansion (e.g., AI-based OCR, sign language detection).
- All screens follow Material 3 design guidelines for consistency and accessibility.