

# Gemini Chat

## Player Det...a \_ Medium

PDF

## player\_detection

## Code folder

Attached is code and a article describing the code for a method to identify players and their jersey numbers in a football game. I want to apply the



Of course. I can help you with that. Applying computer vision techniques from one sport to another is an excellent idea, and your plan is very solid.

I will provide the rewritten Python code with the requested ball detection feature, detailed comments, and comprehensive instructions for training and use.

Given your plan to run the detection on a Raspberry Pi 5, I've made a slight adjustment for better performance. Instead of the older YOLOv3, I've used **YOLOv8n**, a much newer, faster, and more efficient version from the same family of models. It's ideal for edge devices like a Pi while still being highly accurate. The Ultralytics library also makes it incredibly easy to use and train.

Here are the two Python files you requested, followed by detailed instructions.

## 1. train models.py

ers\_classification.ipynb  
bounding box detection model and the number classification model.

## Python

1

```
# train_models.py
```

```
import os
import pickle
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten, Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout

# =====
=====
# Configuration
# =====
=====

# Set the paths to your data and where you want to save the models.
# You will need to create these pickle files from your SJN-210k dataset.
# See the instructions section for the expected format.
DRIVE_PATH = 'drive/MyDrive/Colab Notebooks/numbers_detection'
BBOX_DATA_PATH = os.path.join(DRIVE_PATH, 'data_bbox.pickle')
CLASSIFICATION_DATA_PATH = os.path.join(DRIVE_PATH, 'data.pickle')

# Paths to save the trained models
NUMBER_DETECTION_MODEL_PATH = os.path.join(DRIVE_PATH, 'number_detection.h5')
NUMBER_CLASSIFICATION_MODEL_PATH = os.path.join(DRIVE_PATH, 'numbers_classifier_aug.h5')

# =====
=====
# Model 1: Number Bounding Box Detection
# =====
=====

def train_number_bbox_detector():
    """
    Trains the model to detect the bounding box of the jersey number.
    This part is adapted from 'number_detection.ipynb'.
    """
    print("[INFO] Training the number bounding box detector...")

    # --- 1. Load Data ---
    # The 'data_bbox.pickle' file should contain the training and testing data.
    # It's expected to be a list with the following structure:
    # [trainImages, trainLabels, trainTargets, trainFilenames,
    # testImages, testLabels, testTargets, testFilenames]
    # - Images are numpy arrays of shape (224, 224, 3)
    # - Targets are the normalized bounding box coordinates [startX, startY, endX, endY]
    print("[INFO] Loading number bounding box data...")
    with open(BBOX_DATA_PATH, 'rb') as handle:
        data = pickle.load(handle)
```

```

        (trainImages, _, trainTargets, _, testImages, _, testTargets, _)
= data

        # --- 2. Define Model Architecture ---
        # We use a pre-trained VGG16 model as the base and add a new regression head
        # to predict the 4 bounding box coordinates.
        print("[INFO] Building model architecture...")
        vgg = VGG16(weights="imagenet", include_top=False,
                     input_tensor=Input(shape=(224, 224, 3)))

        # Freeze the base layers to leverage pre-trained weights
        vgg.trainable = True

        # Add the custom regression head
        flatten = Flatten()(vgg.output)
        bboxHead = Dense(128, activation="relu")(flatten)
        bboxHead = Dense(64, activation="relu")(bboxHead)
        bboxHead = Dense(32, activation="relu")(bboxHead)
        bboxHead = Dense(4, activation="sigmoid")(bboxHead) # 4 outputs for
bbox coordinates

        model = Model(inputs=vgg.input, outputs=bboxHead)

        # --- 3. Compile and Train Model ---
        # Using Adam optimizer and Mean Squared Error for loss, as this is a
regression task.
        print("[INFO] Compiling model...")
        opt = Adam(learning_rate=1e-4)
        model.compile(loss="mse", optimizer=opt)

        print(model.summary())

        print("[INFO] Training bounding box regressor...")
        H = model.fit(
            trainImages, trainTargets,
            validation_data=(testImages, testTargets),
            batch_size=32,
            epochs=10, # The original notebook used 10 epochs
            verbose=1
        )

        # --- 4. Save Model and Plot Performance ---
        print(f"[INFO] Saving number detection model to {NUMBER_DETECTION_MODEL_PATH}...")
        model.save(NUMBER_DETECTION_MODEL_PATH)

        # Plot the training loss

```

```

plt.style.use("ggplot")
plt.figure()
plt.plot(H.history["loss"], label="train_loss")
plt.plot(H.history["val_loss"], label="val_loss")
plt.title("Bounding Box Regression Loss")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="lower left")
plt.savefig(os.path.join(DRIVE_PATH, 'bbox_loss_plot.png'))
print("[INFO] Bounding box detector training complete.")

# =====
=====

# Model 2: Number Classification
# =====
=====

def train_number_classifier():
    """
    Trains the model to classify the digit on the jersey.
    This part is adapted from 'numbers_classification.ipynb'.
    """
    print("\n[INFO] Training the number classifier...")

    # --- 1. Load Data ---
    # The 'data.pickle' file should contain the training and testing images and labels.
    # It's expected to be a list with the following structure:
    # [X_train, y_train, X_test, y_test]
    # - X arrays are numpy arrays of images (shape (224, 224, 3))
    # - y arrays are one-hot encoded labels for digits 0-9
    print("[INFO] Loading number classification data...")
    with open(CLASSIFICATION_DATA_PATH, 'rb') as handle:
        (X_train, y_train, X_test, y_test) = pickle.load(handle)

    # --- 2. Define Model Architecture ---
    # A custom Convolutional Neural Network (CNN) for image classification.
    print("[INFO] Building model architecture...")
    classifier = Sequential()
    classifier.add(Conv2D(128, (3, 3), input_shape=(224, 224, 3), activation='relu'))
    classifier.add(MaxPooling2D(pool_size=(2, 2)))
    classifier.add(Dropout(0.2))

    classifier.add(Conv2D(64, (3, 3), activation='relu'))
    classifier.add(MaxPooling2D(pool_size=(2, 2)))

```

```
classifier.add(Dropout(0.2))

# Additional layers from the notebook
classifier.add(Conv2D(64, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Dropout(0.2))

classifier.add(Conv2D(32, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Dropout(0.2))

classifier.add(Flatten())

classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dense(units=64, activation='relu'))
# Final layer has 10 units (for digits 0-9) with softmax for probability distribution
classifier.add(Dense(units=10, activation='softmax'))

# --- 3. Compile and Train Model ---
# Using Adam optimizer and categorical crossentropy for multi-class classification.
print("[INFO] Compiling model...")
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

print(classifier.summary())

# Use ImageDataGenerator for data augmentation to improve model robustness
datagen = ImageDataGenerator(
    rotation_range=30,
    shear_range=0.5,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=False,
    vertical_flip=False
)
datagen.fit(X_train)

epochs = 80 # The original notebook used 80 epochs
batch_size = 32

print("[INFO] Training classifier with data augmentation...")
H = classifier.fit(  
    X_train, y_train, validation_data=(X_val, y_val),  
    epochs=epochs, batch_size=batch_size, verbose=1)
```

```

        datagen.flow(x_train, y_train, batch_size=batch_size),
        steps_per_epoch=len(X_train) // batch_size,
        validation_data=(X_test, y_test),
        epochs=epochs
    )

    # --- 4. Save Model and Plot Performance ---
    print(f"[INFO] Saving number classification model to {NUMBER_CLASSIFICATION_MODEL_PATH}...")
    classifier.save(NUMBER_CLASSIFICATION_MODEL_PATH)

    # Plot the training loss
    plt.style.use("ggplot")
    plt.figure()
    plt.plot(np.arange(0, epochs), H.history["loss"], label="train_loss")
    plt.plot(np.arange(0, epochs), H.history["val_loss"], label="val_loss")
    plt.title("Classification Loss")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss")
    plt.legend(loc="lower left")
    plt.savefig(os.path.join(DRIVE_PATH, 'classification_loss_plot.png'))
    print("[INFO] Number classifier training complete.")

# =====
# Main Execution
# =====
=====

if __name__ == '__main__':
    # You can choose to run one or both training functions
    # by uncommenting them.
    train_number_bbox_detector()
    print("\n" + "="*80 + "\n")
    train_number_classifier()

```

## 2. detect\_volleyball.py

This script is the main application for running inference on a video. It integrates player detection, ball detection (new), number bounding box detection, and number classification.

Python



```
# detect_volleyball.py
```

```
import os
import cv2
import numpy as np
import tensorflow as tf
from ultralytics import YOLO
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# =====
=====

# Configuration
# =====
=====

# --- Paths ---
# Path to the directory where you saved your trained models
MODEL_DIR = 'drive/MyDrive/Colab Notebooks/numbers_detection'
INPUT_VIDEO_PATH = os.path.join(MODEL_DIR, 'video_samples/input/example.mp4')
OUTPUT_VIDEO_PATH = os.path.join(MODEL_DIR, 'video_samples/output/volleyball_output.mp4')

# --- Model Confidence Thresholds ---
# Confidence threshold for YOLOv8 object detection (players and balls)
# Only detections with a score higher than this will be considered.
YOLO_CONF_THRESHOLD = 0.5

# Confidence threshold for the number classification model
# Only classifications with a score higher than this will be accepted.
NUMBER_CONF_THRESHOLD = 0.4

# =====
=====

# Helper Functions
# =====
=====

=====

def detect_number_bbox(player_image, bbox_model):
    """
    Detects the bounding box of the number within a player's image.

    Args:
        player_image: A crop of the frame containing a single player.
        bbox_model: The trained number bounding box detection model.

    Returns:
        A tuple of (startX, startY, endX, endY) for the number's bounding box.
    """

    # Preprocess the image for the model
    image = cv2.resize(player_image, (224, 224))
    image = image / 255.0
```

```

image = image / 255.0
image = np.expand_dims(image, axis=0)

# Predict the bounding box
preds_bbox = bbox_model.predict(image, verbose=0)[0]
return tuple(preds_bbox)

def identify_number(number_image, classifier_model):
    """
    Classifies the digit within the number's bounding box.

    Args:
        number_image: A crop of the player image containing the number.
        classifier_model: The trained number classification model.

    Returns:
        The classified digit (int) or -1 if confidence is too low.
    """
    # --- Preprocessing ---
    # The original notebook converted the image to grayscale and then back
    # to 3 channels, which can sometimes help the model focus on shape over color.
    gray = cv2.cvtColor(number_image, cv2.COLOR_RGB2GRAY)
    image = np.zeros_like(number_image)
    image[:, :, 0] = gray
    image[:, :, 1] = gray
    image[:, :, 2] = gray

    # Resize, apply Gaussian blur, and normalize
    image = cv2.GaussianBlur(cv2.resize(image, (224, 224)), (5, 5), 0)
    image = image / 255.0
    image = np.expand_dims(image, axis=0)

    # --- Prediction ---
    preds = classifier_model.predict(image, verbose=0)[0]

    # Get the class with the highest probability
    i = np.argmax(preds)

    # Return the digit only if the confidence is high enough
    if preds[i] > NUMBER_CONF_THRESHOLD:
        return i
    else:
        return -1

# =====
# Main Inference Pipeline
# =====

```

```
=====

def main():
    """
    Main function to process the video and perform detections.
    """

    print("[INFO] Loading models...")
    # --- Load YOLOv8 Model ---
    # 'yolov8n.pt' is a small, fast model pre-trained on the COCO dataset.
    # The COCO dataset includes classes for 'person' and 'sports ball'.
    # This is great for performance on a Raspberry Pi.
    yolo_model = YOLO('yolov8n.pt')

    # --- Load Custom Models ---
    # Load the models you trained for number detection and classification.
    try:
        classifier_bbox_numbers = tf.keras.models.load_model(os.path.join(MODEL_DIR, 'number_detection.h5'))
        classifier_numbers = tf.keras.models.load_model(os.path.join(MODEL_DIR, 'numbers_classifier_aug.h5'))
    except IOError:
        print("[ERROR] Trained models not found. Please run train_models.py first.")
        return

    print("[INFO] Opening video file...")
    video = cv2.VideoCapture(INPUT_VIDEO_PATH)
    if not video.isOpened():
        print(f"[ERROR] Could not open video file: {INPUT_VIDEO_PATH}")
        return

    # --- Video Writer Setup ---
    # Get video properties to create a writer with the same settings.
    frame_width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(video.get(cv2.CAP_PROP_FPS))

    # Define the codec and create VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    writer = cv2.VideoWriter(OUTPUT_VIDEO_PATH, fourcc, fps, (frame_width, frame_height))
    print(f"[INFO] Output will be saved to {OUTPUT_VIDEO_PATH}")

    # --- Main Processing Loop ---
    while True:
        ret, frame = video.read()
```

```

    if not ret:
        break

    # --- YOLOv8 Detection (Players and Balls) ---
    # The model.predict() function handles all the preprocessing and
    detection.

        # We specify the classes we're interested in: 0 for 'person' and
        32 for 'sports ball'.
        results = yolo_model.predict(frame, conf=YOLO_CONF_THRESHOLD, cl
        asses=[0, 32], verbose=False)

    # results[0].boxes contains all detected bounding boxes
    boxes = results[0].boxes

    # Draw bounding boxes for all detected objects (players and ball
    s)
    for box in boxes:
        # Get bounding box coordinates
        x1, y1, x2, y2 = [int(i) for i in box.xyxy[0]]

        # Get the class ID
        cls_id = int(box.cls[0])

        # --- If a PERSON is detected, run the number detection pipe
        line ---
        if cls_id == 0: # Class ID for 'person' in COCO
            player_image = frame[y1:y2, x1:x2]

            # Check if the cropped player image is valid
            if player_image.shape[0] == 0 or player_image.shape[1] =
            = 0:
                continue

            # 1. Detect the number's bounding box on the player
            (startX_rel, startY_rel, endX_rel, endY_rel) = detect_nu
            mber_bbox(player_image, classifier_bbox_numbers)

            h, w = player_image.shape[:2]

            # Convert relative coordinates to absolute coordinates o
            n the player crop
            startX_abs = int(startX_rel * w)
            startY_abs = int(startY_rel * h)
            endX_abs = int(endX_rel * w)
            endY_abs = int(endY_rel * h)

            # Crop the number image from the player image
            # Note: The y-coordinates might seem reversed because of

```

```

# Note: The y coordinates might seem reversed because of
how they were labeled,
    # a common occurrence. The logic from your notebook is p
reserved.

    number_crop = player_image[endY_abs:startY_abs, startX_a
bs:endX_abs]

    if number_crop.shape[0] == 0 or number_crop.shape[1] ==
0:
        continue

    # 2. Identify the number in the cropped image
    number_pred = identify_number(number_crop, classifier_nu
mbers)

    # --- Draw on Frame ---
    # Draw the main player bounding box
    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 2)

    if number_pred != -1:
        # If a number was identified, draw its specific boun
ding box
        num_x1 = x1 + startX_abs
        num_y1 = y1 + endY_abs
        num_x2 = x1 + endX_abs
        num_y2 = y1 + startY_abs
        cv2.rectangle(frame, (num_x1, num_y1), (num_x2, num_
y2), (255, 0, 0), 2)

        # Add the player number as a label
        label = f"Player {number_pred}"
        cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HE
RSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)

        # --- If a SPORTS BALL is detected ---
        elif cls_id == 32: # Class ID for 'sports ball' in COCO
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            cv2.putText(frame, "Ball", (x1, y1 - 10), cv2.FONT_HERSH
EY_SIMPLEX, 0.7, (0, 255, 0), 2)

    # Write the processed frame to the output video file
    writer.write(frame)

    # --- Cleanup ---
    print("[INFO] Processing complete.")
    video.release()
    writer.release()

```

```
cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

### 3. Detailed Instructions (README)

Here are the detailed instructions on how to set up your environment, prepare the data, and run the code.

## Volleyball Player, Number, and Ball Detection

This project provides a complete pipeline to detect players, their jersey numbers, and the volleyball in a video feed. It uses a combination of deep learning models:

1. **YOLOv8n**: For detecting players and the ball. This is a state-of-the-art, lightweight object detector perfect for real-time applications.
2. **Number Bounding Box Detector**: A custom model (VGG16-based) to locate the number on a detected player's jersey.
3. **Number Classifier**: A custom CNN to identify the digit on the jersey.

### I. Setup and Installation

1. **Clone/Download Code**: Place `train_models.py` and `detect_volleyball.py` in your project directory.
2. **Create a Python Environment**: It's highly recommended to use a virtual environment.

Bash

```
python -m venv venv
source venv/bin/activate # On Windows, use 'venv\Scripts\activate'
```

3. **Install Required Libraries**:

Bash

```
pip install tensorflow opencv-python ultralytics matplotlib scikit-learn
```

4. **Directory Structure**: Organize your project folder like this. The `DRIVE_PATH` variable in the scripts points to the `numbers_detection` folder.

```
your_project_folder/
|   train_models.py
```

```

└── train_models.py
└── detect_volleyball.py
└── drive/MyDrive/Colab Notebooks/
    └── numbers_detection/
        ├── video_samples/
        │   ├── input/
        │   │   └── example.mp4      <-- Your input video
        │   └── output/           <-- Output videos will be saved here
    e
    └── data_bbox.pickle      <-- Your prepared bbox data
    └── data.pickle          <-- Your prepared classification data

```

## II. Data Preparation (Crucial Step)

The training script `train_models.py` expects your data to be in specific `.pickle` files. You will need to create these files from your **SJN-210k dataset**.

1. `data_bbox.pickle` (for Number Bounding Box Detection):

- This file contains the images and their corresponding number bounding box coordinates.
- You need to write a script to load your dataset, preprocess it, and save it in this format.
- **Format:** A single list pickled to a file.

Python



```

# Expected structure of the list to be pickled
data_to_pickle = [
    trainImages,    # NumPy array of training images, shape (num_samples, 224, 224, 3)
    trainLabels,   # Not used by this model, can be empty list
    trainTargets,  # NumPy array of training bbox coordinates, shape (num_samples, 4)
    trainFilenames,# Not used by this model, can be empty list
    testImages,    # NumPy array of test images
    testLabels,    # Not used by this model, can be empty list
    testTargets,   # NumPy array of test bbox coordinates
    testFilenames # Not used by this model, can be empty list
]
# Bbox coordinates must be normalized (0 to 1) as [startX, startY, endX, endY]

```

2. `data.pickle` (for Number Classification):

- This file contains the cropped images of the numbers and their corresponding

labels.

- You will need to use the bounding box information from your dataset to crop the number images.
- **Format:** A single list pickled to a file.

Python

```
# Expected structure of the list to be pickled
data_to_pickle = [
    X_train, # NumPy array of training number images, shape (num_samples, 224, 224, 3)
    y_train, # NumPy array of one-hot encoded labels, shape (num_samples, 10)
    X_test, # NumPy array of test number images
    y_test # NumPy array of one-hot encoded test labels
]
# Example for one-hot encoding: digit '5' would be [0, 0, 0, 0, 0,
1, 0, 0, 0, 0]
```

### III. Model Training

Once your data is prepared, you can train the custom models.

- **Run the script from your terminal:**

Bash

```
python train_models.py
```

- **What it does:**

1. It first calls `train_number_bbox_detector()`, which loads `data_bbox.pickle`, `number_detection.h`  
5 `bbox_loss_plot.png`
  2. It then calls `train_number_classifier()`, which loads `data.pickle`, trains `numbers_classifier_aug.h5` `classification_loss_plot.png` also saved.
- **Training Time:** This will take a significant amount of time, especially on a CPU. It is highly recommended to use a machine with a dedicated GPU for training.

### IV. Running the Detection

After training, you can run the main detection pipeline on a video.

1. **Place your video in the `video_samples/input/` directory.**

Place your video in the `video_samples/input/` directory.

2. **Update the path** in `detect_volleyball.py` if your video has a different name.
3. **Run the script:**

Bash

```
python detect_volleyball.py
```

4. **Output:** The script will process the video frame by frame, draw bounding boxes  
`video_samples/output/volleyball_output.mp4`

## V. How the Models Work (Quick Summary)

1. **YOLOv8n (Player and Ball Detection):**
  - **Source:** Loaded automatically from the Ultralytics library, pre-trained on the COCO dataset.
  - **How it Works:** YOLO (You Only Look Once) is a single-shot detector. It divides an image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously. It's extremely fast because it processes the entire image in one pass. We use it to find all instances of "person" and "sports ball".
2. **Number Bounding Box Detector (Number Localization):**
  - **Source:** Trained by you using `train_models.py`.
  - **How it Works:** This is a regression model. It takes an image of a *single player* (cropped out by YOLO) and, instead of classifying it, predicts four continuous values: the `(x1, y1, x2, y2)` coordinates of the number on that player's jersey. It uses the powerful feature-extraction capabilities of VGG16 to understand the visual patterns that indicate where a number is likely to be.
3. **Number Classifier (Number Identification):**
  - **Source:** Trained by you using `train_models.py`.
  - **How it Works:** This is a standard Convolutional Neural Network (CNN) for classification. It takes the small image of the number (cropped using the coordinates from the previous model) and predicts which digit (0-9) it is.

## VI. Considerations for Raspberry Pi 5

- **Performance:** The Raspberry Pi 5 is powerful, but this is a demanding task. The use of **YOLOv8n** is a key optimization.

- **Model Simplification:** If you still face performance issues, you could:
    - Train a smaller number detection/classification model. For example, use MobileNet as a base instead of VGG16, or create a simpler custom CNN.
    - Reduce the video resolution during processing.
  - **Dependencies:** Ensure all required libraries (especially TensorFlow and OpenCV) are compiled or installed correctly for the ARM architecture of the Raspberry Pi. Using a lightweight version like `tf-lite` for the custom models would be an excellent next step for optimization.
-