Multi-Class Detection of Key Roles in Football
A Deep Learning Approach

Luigi Manieri (0001113044)

December 2025

Contents

1 Introduction

This project presents a deep learning-based approach to automatically identify and classify key roles in football matches. The goal is to develop a robust multi-class object detection system capable of distinguishing between players, referees, coaches, and goalkeepers in football match footage. This automation offers significant improvements over manual annotation methods in terms of efficiency, consistency, and scalability.

The challenge of role identification in sports analytics is particularly relevant given the rapid growth of video analysis in modern football. Professional teams and broadcasters increasingly rely on automated systems to extract meaningful insights from vast amounts of match footage. Accurate detection and classification of key roles enables:

- **Tactical Analysis**: Understanding player positioning, movement patterns, and team formations
- **Performance Evaluation**: Quantifying individual and team performance metrics
- **Broadcast Enhancement**: Automatic highlighting and annotation of key events
- **Scalability**: Processing large volumes of match data efficiently

Manual video annotation is time-consuming, subjective, and error-prone. A single 90-minute football match generates approximately 86,400 frames at standard frame rate. Manually identifying and classifying key roles in each frame is impractical for real-world applications. Deep learning approaches offer a solution by learning discriminative features from large amounts of training data, enabling fast and consistent automated detection.

2 Problem Definition

The project tackles a multi-class object detection problem with five target classes:

1. **Player**: Field players competing in the match
2. **Referee**: Match officials enforcing the rules
3. **Coach**: Technical staff on the sideline
4. **Goalkeeper**: Players guarding the goal
5. **Ball**: The football itself

Each instance in the training data is annotated with bounding boxes indicating the spatial location of the object and its corresponding class label in COCO (Common Objects in Context) format.

2.1 Objectives

The primary objectives of this project are:

- Develop and train state-of-the-art object detection models (Faster R-CNN and RetinaNet)
- Evaluate model performance using standard metrics (IoU, classification accuracy, detection rate)
- Compare different approaches for football role detection
- Demonstrate practical inference capabilities on test images

3 Dataset

The dataset comprises annotated images of football matches collected from various match footages. The data is organized into three main splits:

- **Training Set**: Used for model training and parameter optimization
- **Validation Set**: Used as test set to evaluate models
- **Test Set**: Not used, since no labels were provided

Each image in the training set contains one or more annotated instances of the target classes, with precise bounding box coordinates indicating spatial location.

The dataset uses the COCO format, which stores annotations in JSON files containing:

- **Images**: Image metadata including filename, width, height, and unique identifier
- **Annotations**: Bounding box coordinates in [x, y, width, height] format, associated image ID, and category ID
- **Categories**: Class definitions with name and unique identifier

This standardized format enables seamless integration with popular detection frameworks and facilitates reproducibility.

3.1 Overview

The class distribution in the dataset reveals the prevalence of different roles in typical football match footage. The distribution is analyzed through a frequency count of annotated instances across all training images.

Class	**Count**	**Proportion**
Ball	High	Baseline
Referee	Medium	Moderate
Coach	Low	Sparse
Goalkeeper	Low	Sparse
Player	Very High	Dominant

Table 1: Approximate class distribution in the football detection dataset

3.2 Implications

The class distribution exhibits significant imbalance, with players being the dominant class while coaches and goalkeepers appear less frequently. This imbalance presents challenges during training:

- **Sampling Bias**: Models may exhibit bias toward dominant classes
- **Training Difficulty**: Rare classes may be underrepresented in gradient updates
- **Metric Interpretation**: Standard accuracy metrics may not reflect true model capability

To address these challenges, the training procedure employs appropriate loss weighting and evaluation metrics that account for class imbalance.

4 Data processing

4.1 Resizing Strategy

All images are resized to a fixed dimension of 1000×1000 pixels using the Chitra library, which provides functionality to:

- Resize images while maintaining aspect ratio
- Automatically adjust bounding box coordinates to match resized dimensions
- Preserve spatial relationships between objects and image boundaries

This standardization ensures consistent input dimensions for neural network processing and enables batching of samples.

4.2 Coordinate Transformation

When images are resized, bounding box coordinates are transformed according to the scaling factor:

$$x'_{\text{new}} = x'_{\text{old}} \cdot \frac{w_{\text{new}}}{w_{\text{old}}}$$

$$y'_{\text{new}} = y'_{\text{old}} \cdot \frac{h_{\text{new}}}{h_{\text{old}}}$$

where w and h denote width and height dimensions.

4.3 Data Augmentation

While the notebook includes provisions for data augmentation (random horizontal flips, color jitter, rotations, affine transformations), the primary training pipeline uses minimal augmentation to focus on model architecture evaluation. Augmentation strategies can be enabled through the `get_transform()` function for future robustness improvements.

5 Notebook Structure and Development

5.1 Libraries and Imports

The notebook begins by importing essential libraries across multiple domains:

- **Data Processing**: NumPy, Pandas, JSON for data manipulation
- **Visualization**: Matplotlib, for exploratory analysis
- **Deep Learning**: PyTorch and TorchVision for model implementation
- **COCO Tools**: pycocotools for dataset handling
- **Utilities**: PIL for image processing, tqdm for progress tracking

5.2 Data Loading and Exploration

The notebook implements thorough exploratory data analysis:

1. Load COCO annotation JSON files from the training directory
2. Parse annotations and extract category information
3. Generate class distribution statistics
4. Visualize sample images with annotated bounding boxes

This section establishes a solid understanding of the data set before model training, identifying class imbalance and typical annotation patterns.

5.3 Custom Dataset Implementation

A custom dataset class extends TorchVision's `CocoDetection` to handle image-specific preprocessing:

```
class CocoDetectionTransform(CocoDetection):
    def __init__(self, img_folder, ann_file, transforms=None):
        super().__init__(img_folder, ann_file)
        self.transforms = transforms
        self.coco = COCO(ann_file)

        # Category mapping for 0-based indexing
        self.valid_category_ids = sorted([
            cat["id"] for cat in self.coco.dataset["categories"]
            if cat["name"] != "football-objects"
        ])
        self.cat_id_map = {cat_id: idx for idx, cat_id in
                           enumerate(self.valid_category_ids)}
```

Key features:

- **Flexible Indexing**: Converts COCO category IDs to 0-based indices
- **Bounding box Extraction**: Extracts bounding boxes in [xmin, ymin, xmax, ymax] format
- **Image Resizing**: Uses Chitra library for intelligent image resizing
- **Transform Pipeline**: Applies optional transformations to both images and bounding boxes

5.3.1 Transform Pipeline

The `get_transform()` function returns a composition of transforms including:

- **ToTensor**: Converts PIL images to PyTorch tensors with normalized values [0, 1]
- **Optional Augmentation**: Placeholder for color jitter, flips, rotations, and affine transformations

5.4 Model Architecture Implementation

5.4.1 Faster R-CNN

Faster R-CNN (Faster Region-based Convolutional Neural Network) is a two-stage detector consisting of:

1. **Region Proposal Network (RPN)**: Generates candidate object locations
2. **RoI Pooling**: Extracts fixed-size features from proposals
3. **Classification Head**: Classifies regions into object categories

The implementation:

```
def get_fasterrcnn_model(num_classes):
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(
        pretrained=True)
    in_features = model.roi_heads.box_predictor.cls_score.
       in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(
        in_features, num_classes)
    return model
```

Uses a ResNet-50 backbone with Feature Pyramid Network (FPN) for multi-scale feature extraction. The classification head is replaced to output probabilities for 5 classes.

5.4.2 RetinaNet

RetinaNet is a single-stage detector with:

1. **Backbone**: Feature extraction using ResNet-50 with FPN
2. **Anchor Generation**: Dense anchor boxes at multiple scales and aspect ratios
3. **Focal Loss**: Specialized loss addressing class imbalance

The implementation:

```
def get_retinanet_model(num_classes):
    model = torchvision.models.detection.retinanet_resnet50_fpn(
        pretrained=True)
    num_anchors = model.head.classification_head.num_anchors
    in_features = model.backbone.out_channels
    model.head.classification_head = RetinaNetClassificationHead(
        in_channels=in_features, num_anchors=num_anchors,
        num_classes=num_classes)
    return model
```

RetinaNet's focal loss particularly helps with class imbalance by downweighting easy negative examples.

5.5 Training Process

5.5.1 Training Configuration

Hyperparameter	Value
Number of Epochs	5
Batch Size	1 (trained on local GPU)
Optimizer	Adam
Learning Rate	0.0001
Weight Decay	0.0005
LR Scheduler	StepLR (step_size=3, gamma=0.1)

Table 2: Training hyperparameters used in the project

5.5.2 Training Loop

The `train_and_eval_model()` function implements the core training procedure:

1. **Forward Pass**: Pass batch through model to compute predictions
2. **Loss Computation**: Aggregate detection losses from the model

3. **Backward Pass**: Compute gradients via backpropagation
4. **Optimization Step**: Update parameters using Adam optimizer
5. **Validation**: Evaluate on validation set after each epoch
6. **Checkpointing**: Save model weights after each epoch

The training loop uses a progress bar for real-time monitoring and implements learning rate scheduling to improve convergence.

5.5.3 Loss Functions

Both models employ multiple loss components:

- **Localization Loss**: Smooth L1 loss for bounding box regression
- **Classification Loss**: Cross-entropy loss for category prediction (Faster R-CNN) or focal loss (RetinaNet)
- **RPN Loss**: Region proposal network loss (Faster R-CNN only)

These losses are combined into a single scalar loss for backpropagation:

$$\mathcal{L}_{\text{total}} = \sum_i \mathcal{L}_i$$

5.6 Evaluation metrics

The notebook computes several evaluation metrics:

1. **Validation Loss**: Sum of all loss components on validation data
2. **Average Detections**: Mean number of detections per image
3. **Average Intersection over Union (IoU)**: Measures bounding box precision
4. **Class Accuracy**: Proportion of correctly classified objects (IoU ≥ 0.5)

5.6.1 IoU Calculation

Intersection over Union measures overlap between predicted and ground-truth boxes:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

Values range from 0 (no overlap) to 1 (perfect alignment). A detection is considered correct if IoU ≥ 0.5.

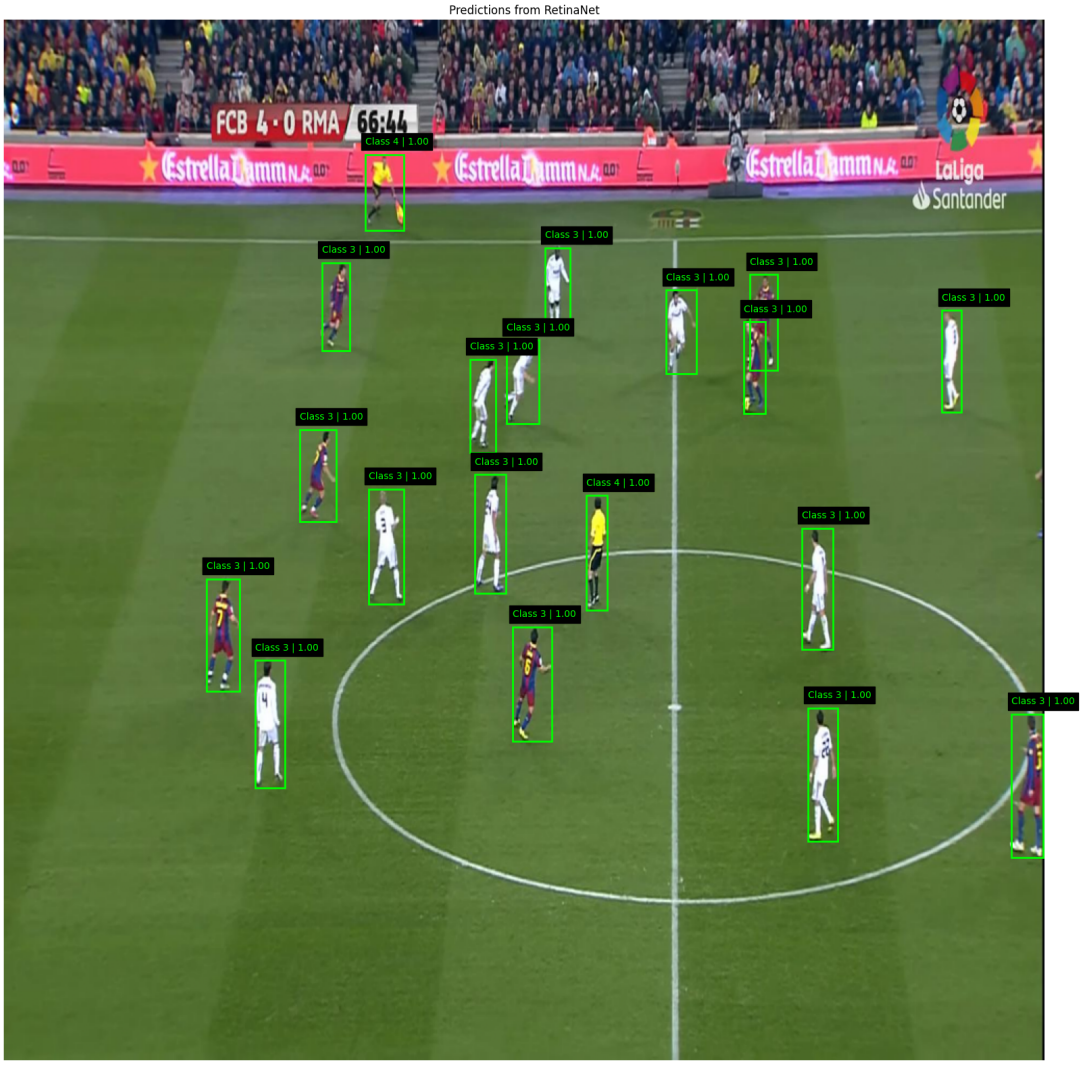

Figure 1: Prediction displayed

5.7 Inference and Visualization

Predictions are visualized with:

- Bounding boxes with confidence scores above a threshold
- Class labels for each detection
- Confidence threshold filtering

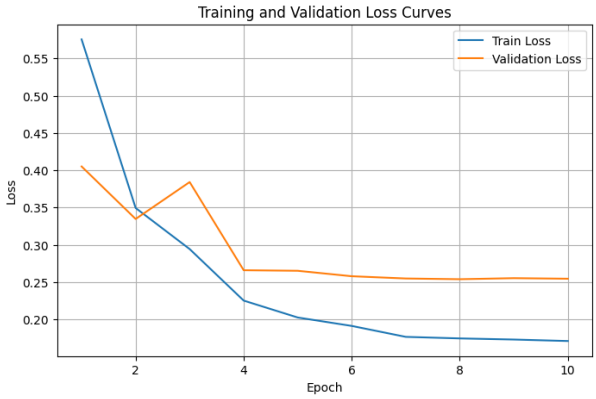

Figure 2: RetinaNet Train/Val loss

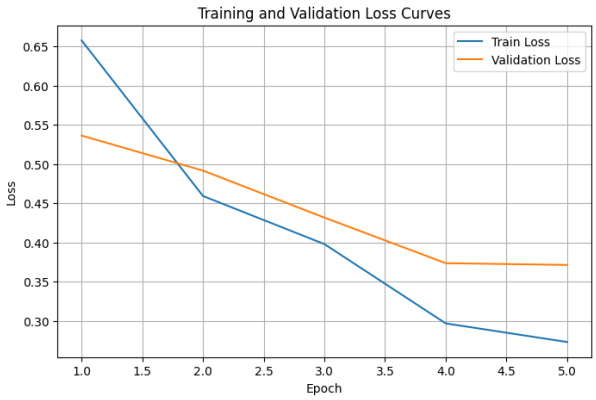

Figure 3: Faster RCNN Train/Val loss

6 Interpretation of results

Both Faster R-CNN and RetinaNet demonstrate capability in detecting football roles:

- **Training Convergence**: Both models show declining loss curves over 5 epochs, indicating successful learning
- **Generalization**: Validation loss patterns reflect training loss, suggesting reasonable generalization without severe overfitting
- **Detection Capability**: Qualitative analysis of predictions shows reasonable detection accuracy on training and test samples

6.1 Faster R-CNN Strengths

- Robust bounding box localization

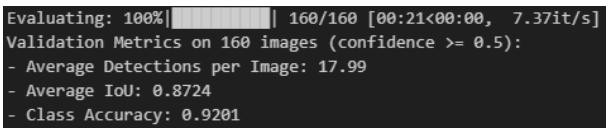

Figure 4: Computed RetinaNet metrics

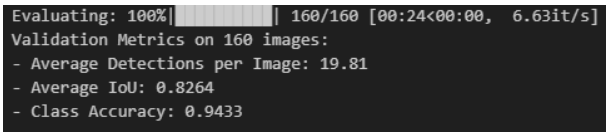

Figure 5: Computed FasterRCNN metrics

- Strong performance on larger objects
- Interpretable two-stage pipeline

6.2 RetinaNet Strengths

- Efficient single-stage processing
- Focal loss handles class imbalance well
- Dense prediction across feature pyramid

7 Conclusion and future work

The project demonstrates successful multi-class detection of key football roles using deep learning, providing a foundation for automated sports analytics. Both RetinaNet and Faster R-CNN serve as strong baseline models, with RetinaNet's focal loss offering particular advantages for handling class imbalance common in football footage. Incorporating full data augmentation and applying class weighting are expected to further improve model accuracy and robustness. This framework enables systematic evaluation of performance metrics and establishes a scalable approach for future model exploration, tactical analysis, and broadcast enhancement.