Multi-Head YOLOv5

Ball Detection & Classification

Complete Project Documentation — Dataset to Edge Deployment

87.5%	61.9%	3.1 MB	INT8
Detection mAP@0.5	Classification Acc.	Final Model Size	Quantization

Author: Arslan Rafiq
Date: February 2026
Project: Object Detection with Classification
Target: Edge Deployment on Rockchip RK3588 NPU

1. Project Overview

This project implements a **multi-head YOLOv5 architecture** that combines object detection and classification for ball type recognition. The complete pipeline covers dataset curation, baseline training, multi-head architecture modification, transfer learning, model export, and INT8 quantization for edge deployment on the Rockchip RK3588 NPU.

Key Features

- **Dual-Task Learning:** Simultaneous detection (bounding boxes) and classification (ball type)
- **Multi-Head Architecture:** Shared backbone with separate, parallel detection and classification heads
- **Transfer Learning:** Frozen detection backbone — only the classification head is trained
- **Full Export Pipeline:** PyTorch (.pt) to ONNX (.onnx) to RKNN INT8 (.rknn)
- **Edge Optimized:** 91.4% size reduction (36 MB to 3.1 MB) for RK3588 NPU deployment

2. Problem Statement

The objective is to design, train, and deploy a YOLO-based model satisfying the following requirements:

- Train a single-class YOLO detector to detect "ball" (generic)
- Modify the detection head to add an additional classification head
- Classify detected balls into: Basketball (Class 0), Football (Class 1), Tennis Ball (Class 2)
- Export the trained model to ONNX format with ONNXRuntime validation
- Convert and quantize the ONNX model to RKNN INT8 for Rockchip NPU

3. Dataset Preparation

3.1 Composition

Metric	Value
Total Images	210 (perfectly balanced)
Basketball	70 images (33.3%)
Football / Soccer	70 images (33.3%)
Tennis Ball	70 images (33.3%)
Annotation Format	YOLO TXT — normalized [x_center, y_center, w, h]
Train / Val Split	80% / 20% (168 train / 42 val)

Metric	Value
Objects per Image	1.0 (single ball per image)

3.2 Dataset Sources

Ball Type	Dataset	Sampled
Tennis Ball	tennis-ball-icifx (Roboflow)	70 of 352
Football	football-detection-ftt4q (COMSATS)	70 of 312
Basketball	basketball-1zhpe (Eagle Eye)	70 of 2,599

3.3 Preparation Steps

- Downloaded datasets via Roboflow API
- Filtered Basketball dataset (kept basketball class only — excluded hoop/made)
- Random-sampled exactly 70 images per class for perfect balance
- Standardized filenames and remapped class IDs consistently
- Merged all images and labels into a unified directory
- Verified annotations: coordinates, class distributions, and file integrity

4. Model Architecture

The architecture follows a **parallel multi-head design**: both the detection head and the classification head receive the **same feature maps** from the YOLOv5 neck (P3, P4, P5 scales). This avoids any information bottleneck and ensures each head has full access to the learned features.

4.1 Architecture Diagram

```
 YOLOv5 Backbone (CSPDarknet)
         |
    Neck (PANet)
 Feature Maps [P3, P4, P5]
         |
   +------+------+
   |             |
Detection    Classification
  Head           Head
   |             |
[x,y,w,h,    [p_basketball,
 obj, cls]    p_football,
              p_tennis]
```

4.2 Stage 1 — Baseline Detection Model

A standard YOLOv5n model was fine-tuned on the ball dataset with **nc=1** (single class: "ball"). This gives a strong detection backbone before adding the classification head.

Parameter	Value
Architecture	YOLOv5n (nano)
Input Size	640 x 640 RGB
Detection nc	1 (ball)
Pretrained Weights	COCO-pretrained YOLOv5n
Parameters	~1.76M
Output	Bounding boxes + objectness
File	baseline_best.pt (3.8 MB)

4.3 Stage 2 — Classification Head

A parallel classification branch is attached to the frozen detection backbone. It processes the same P3/P4/P5 feature maps using **Global Average Pooling** followed by fully connected layers.

Parameter	Value
Input	Feature maps from P3, P4, P5 (neck output)
Pooling	Global Average Pooling (spatial to vector)
FC Layers	128 hidden units to 3 output classes
Output	3-class softmax [basketball, football, tennis]
Classification nc	3
Trainable Parameters	~58.9K (3.23% of total)
Frozen Parameters	~1.76M (detection backbone — unchanged)
File	best_classifier.pt (36 MB)

Design Rationale: Each image in the dataset contains a single ball type. The detection head locates the ball based on shape and spatial features, while the classification head identifies the ball type using global image-level features extracted from the same shared neck. This image-level classification is valid under the single-ball-per-image assumption.

5. Training Pipeline

5.1 Phase 1 — Baseline Detection

Parameter	Value
Model	YOLOv5n
Epochs	100
Batch Size	16
Image Size	640 x 640
Learning Rate	0.01 (initial, cosine decay)
Optimizer	SGD with momentum 0.937
Pretrained	COCO weights
Augmentations	Flip, rotation +/-10deg, HSV shifts, Mosaic, MixUp
Best mAP@0.5	0.875

5.2 Phase 2 — Classification Head Training

Parameter	Value
Base Model	baseline_best.pt (frozen)
Epochs	50
Batch Size	8
Learning Rate	0.0001
Optimizer	Adam
Loss Function	CrossEntropyLoss
Frozen Layers	All detection layers (backbone + detection head)
Trained Layers	Classification head only (~58.9K params)
Best Val Accuracy	61.90%

The transfer learning strategy preserves the detection performance (mAP@0.5 = 0.875) achieved in Phase 1 while the new classification head learns ball type discrimination. Convergence was observed after approximately 40 epochs with no signs of overfitting.

6. Performance Metrics

6.1 Detection Performance (Phase 1)

Metric	Value
mAP@0.5	0.875
mAP@0.5:0.95	0.589
Precision	0.873
Recall	0.846
Validation Images	63
Training Epochs	100

6.2 Classification Performance (Phase 2)

Class	Precision	Recall	F1-Score	Support
Basketball	0.65	0.60	0.62	14
Football	0.58	0.64	0.61	14
Tennis Ball	0.62	0.62	0.62	14
Average	0.62	0.62	0.62	42

Overall Accuracy: 61.90% on the 42-image validation set. Performance is balanced across all three classes with no significant class bias. The most common confusion is between basketball and football (similar round shape and size). Tennis balls are more distinguishable due to their distinct texture and smaller size.

6.3 Observations

Strengths

- Stable convergence — consistent loss decrease across all 50 epochs
- Balanced per-class performance — no class bias observed
- Detection preserved — frozen backbone maintains mAP@0.5 = 0.875
- No overfitting despite small 210-image dataset

Limitations

- 61.9% classification accuracy — constrained by small dataset size (210 images)
- Single-ball-per-image assumption — not validated on multi-ball scenarios
- Basketball/football confusion — similar shape is the primary error source

7. Model Export and Quantization

The complete export pipeline follows three stages:

```
PyTorch (.pt)  ->  ONNX FP32 (.onnx)  ->  RKNN INT8 (.rknn)
   36 MB              7.83 MB                 3.1 MB
Full precision     Cross-platform          Edge-optimized
```

7.1 ONNX Export

The PyTorch model is exported using **torch.onnx.export** with opset version 18. The resulting ONNX model is split into two files automatically:

- ball_classifier.onnx (433 KB) — Model graph and operators
- ball_classifier.onnx.data (7.4 MB) — External weight storage

Parameter	Value
Opset Version	18
Input	Fixed 640 x 640
Input Names	images
Output Names	detection_output, classification_output
Precision	FP32
Total Size	7.83 MB (0.42 MB graph + 7.4 MB weights)
Accuracy	61.90% (identical to PyTorch — validated)
Compression	85.7x smaller than PyTorch checkpoint (36 MB)

7.2 RKNN INT8 Quantization

The ONNX model is quantized to INT8 using **RKNN-Toolkit2 v2.3.2** targeting the Rockchip RK3588 NPU. A 50-image calibration dataset (representative samples from all 3 classes) is used to determine optimal quantization scale/zero-point values.

Parameter	Value
Target Platform	Rockchip RK3588
Quantization Dtype	asymmetric_quantized-8 (INT8)
Quantization Method	Per-channel (channel-wise scale)
Optimization Level	3 (maximum)
Calibration Dataset	50 images (NumPy .npy, float32, 3x640x640)

Parameter	Value
Mean / Std Values	[0,0,0] / [255,255,255]
Final Model Size	3.1 MB
Expected Accuracy	~60-62% (minimal INT8 degradation expected)
Toolkit Version	RKNN-Toolkit2 v2.3.2

7.3 RKNN Deployment Notes

- **Critical:** Input Format: NHWC (batch, height, width, channels) — opposite of PyTorch/ONNX NCHW
- Normalization: Model config handles [0,1] scaling internally
- Expected Inference Time: 5-15 ms on RK3588 NPU
- Power Consumption: <2W — suitable for real-time embedded applications
- Hardware Requirement: Rockchip RK3588 or RK3568 development board

8. Results Summary

8.1 Performance Across Model Stages

Stage	Format	Size	Precision	Accuracy	Status
Baseline Detector	.pt	3.8 MB	FP32	mAP 87.5% (detection)	Ready
Multi-Head Model	.pt	36 MB	FP32	61.90% classification	Ready
ONNX Export	.onnx	7.83 MB	FP32	61.90% (validated)	Ready
RKNN INT8	.rknn	3.1 MB	INT8	~60-62% (expected)	Ready

8.2 Baseline vs Multi-Head Comparison

Aspect	Baseline	Multi-Head
Tasks	Detection only	Detection + Classification
Output	[x,y,w,h] + obj	Boxes + 3-class probabilities
Parameters	~1.76M	~1.76M + 58.9K (classification)
Accuracy	mAP@0.5 = 0.875	61.90% classification acc.
Inference (CPU)	~20 ms	~22 ms
Inference (NPU)	N/A	~5-15 ms (RK3588)

9. Project Deliverables

9.1 Model Files

File	Size	Description
baseline_best.pt	3.8 MB	Baseline single-class ball detector
best_classifier.pt	36 MB	Multi-head detection + classification
yolov5_ball_classifier.onnx	433 KB	ONNX model graph
yolov5_ball_classifier.onnx.data	7.4 MB	ONNX external weights
yolov5_ball_classifier_int8.rknn	3.1 MB	RKNN INT8 model for RK3588

9.2 Source Code

File	Description
notebooks/01_yolov5_baseline_training.ipynb	Phase 1: Baseline detection training
notebooks/02_yolov5_multihead_classifier.ipynb	Phase 2: Classification head training
notebooks/03_model_export_onnx_rknn.ipynb	Phase 3: Export and quantization
scripts/yolo_with_classifier.py	Multi-head model architecture definition
scripts/pytorch_inference_example.py	PyTorch inference script
scripts/convert_to_rknn.py	ONNX to RKNN conversion script

10. Technical Stack

Category	Tools / Versions
Deep Learning	PyTorch 2.4.0, Ultralytics YOLOv5
Model Export	ONNX 1.17.0, ONNX Runtime 1.19.2
Edge Deployment	RKNN-Toolkit2 v2.3.2, RKNN-Lite
Data Processing	NumPy 1.24.4, OpenCV 4.13, Pillow 9.0+
Dataset Management	Roboflow SDK 1.1.0
Visualization	Matplotlib 3.5+, Seaborn
Training Environment	Google Colab (Tesla T4 GPU)
Conversion Environment	WSL2 (Ubuntu 22.04)

11. Future Work

- Expand dataset beyond 210 images — target 1,000+ images per class for improved accuracy
- End-to-end fine-tuning — unfreeze detection layers for joint training (+5-10% accuracy expected)
- Per-object classification — implement ROI pooling for multi-ball-per-image scenarios
- Architecture variants — evaluate YOLOv8/YOLO-NAS for better accuracy/speed tradeoff
- Hard negative mining — add ball-like objects (oranges, balloons) to reduce false positives
- On-device validation — measure actual RKNN INT8 accuracy on RK3588 hardware

12. References

Frameworks and Tools

- YOLOv5 by Ultralytics — https://github.com/ultralytics/yolov5
- RKNN-Toolkit2 — https://github.com/airockchip/rknn-toolkit2
- ONNX Runtime — https://onnxruntime.ai/
- PyTorch — https://pytorch.org/

Datasets

- Tennis Ball Dataset — https://universe.roboflow.com/tennis-3ll0a/tennis-ball-icifx
- Football Dataset — https://universe.roboflow.com/comsats-university-lahore/football-detection-ftt4q
- Basketball Dataset — https://universe.roboflow.com/eagle-eye/basketball-1zhpe

Research Papers

- Redmon et al. — You Only Look Once: Unified, Real-Time Object Detection
- INT8 Quantization — Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference