Detection of Ball and Player Location from Frame-by-Frame Soccer Match Images

Timothy Majidzadeh, Etienne Ndedi

Description

We use You Only Look Once ("YOLO") – a particular family of Convolutional Neural Networks ("CNNs") designed for object detection[1] – to detect the locations of a soccer ball and soccer players in images extracted from the 'SoccerTrack' image dataset.[2] We begin with a summary of our problem, research approach, methodology, data and data exploration. We discuss the problems of small objects and unbalanced classes in image detection, and demonstrate how we overcome these obstacles through image augmentation and over-sampling on the presence of an object (the ball). We show that YOLO is a better fit for our use case than simpler classifiers such as logistic regression and a small CNN, then improve our model through experiments on machine learning hyperparameters and compare different versions of YOLO. We conclude with a discussion of our study's limitations and avenues for future work.

Objective

To develop a generalizable machine learning model using YOLO which can identify ball location and player location from images of a soccer match with high accuracy. This could streamline the labor-intensive process of collecting data.

Problem Statement

Soccer is the most popular sport in the world as it is also the most accessible. Demand for data and analysis in the sport has significantly increased in the past 10 years as teams chase insights which determine victory or defeat. Thousands of hours of manual labor are required to annotate soccer matches using historical methods, but machine learning tools using object detection could speed up this process by identifying details such as ball and player location.

Approach / Methodology

We follow the following steps for this research.

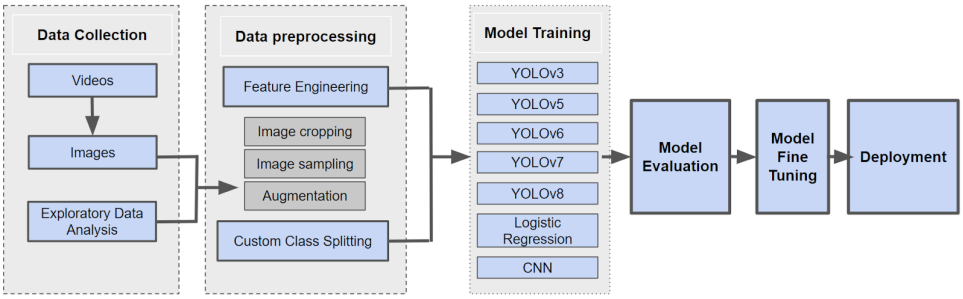
- **Data Collection:** We downloaded the data from Kaggle and extract every frame as an image.

[1] Redmon, Joseph et al., "You Only Look Once: Unified, Real-Time Object Detection," ("YOLO Object Detection"), *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
[2] Uchida, Ikuma, et al, "SoccerTrack: A Dataset and Tracking Algorithm for Soccer with Fish-eye and Drone Videos," ("SoccerTrack"), kaggle.com, available at https://www.kaggle.com/datasets/atomscott/soccertrack/data, 2022.

- **Exploratory Data Analysis (EDA):** We examined the bounding box labels and demonstrated three issues to address: the small-object problem, class imbalance, and asymmetry.
- **Data Preprocessing/Data Engineering:** We stack the labels data and match every label to the correct image. We randomly crop every image to an 800p x 800p square, and update all of the bounding boxes to match the new image. We create two random samples from the cropped images: one simple random sample of 10,400 images, and one stratified random sample of the same size that includes the ball in 50% of the images.[3] We use tools in each package's version of YOLO to apply further image augmentations.
- **Model Development:** We train YOLOv3, YOLOv5, YOLOv6, YOLOv7, and YOLOv8 on an 80% split of the processed data.
- **Model Evaluation:** We show the evaluation metrics and confusions of our models on a 10% validation split.
- **Model Fine-Tuning:** Through experiments, we identify our preferred hyperparameters, augmentations, and other model improvements.
- **Deployment:** We create a final model with our preferred hyperparameters, augmentations, and training dataset. The weights from this model are saved and can be used for predictions.

Block Diagram

Our approach can be summarized visually as follows:



Datasets

SoccerTrack

SoccerTrack is an open-source dataset hosted on Kaggle, which its authors created for research at the University of Tsukuba in Japan.[4] The data is organized as follows:

- **Top View Data:** Sixty 30-second .mp4 videos, running at 30 FPS at 2160x3840p, which show a soccer match as viewed by a drone flying directly above the center circle of the pitch.
    - A set of CSV files which give coordinates for the bounding boxes of each of the 22 players and the ball for each frame of each video.
    - A set of .mp4 videos which appear the same as the original videos, but with the bounding boxes visualized.

[3] The process of cropping and over-sampling the ball is necessary to address the small-object and class imbalance problems. See the 'Datasets' section for more discussion.
[4] Uchida, Ikuma, et al, "SoccerTrack," 2022.

- **Wide View Data:** Sixty-six 30-second .mp4 videos, running at 30 FPS at 1000x6500p, which show a soccer match from a wide angle.
    - A set of CSV files which give coordinates for the bounding boxes of each of the 22 players and the ball for each frame of each video.
    - A set of .mp4 videos which appear the same as the original videos, but with the bounding boxes visualized.

We extract every frame from every video as an image in its native resolution. We stacked all of the CSV files which contain the bounding box locations as labels, matched them to the corresponding frames, and conducted exploratory data analysis (EDA). We learned about three key issues through EDA:

1) **Objects (especially the ball) are small relative to the size of the images.**

    Consider this table of average bounding box sizes (measured in pixels) for the top-view images:

EDA: Bounding Box Size Summary Statistics

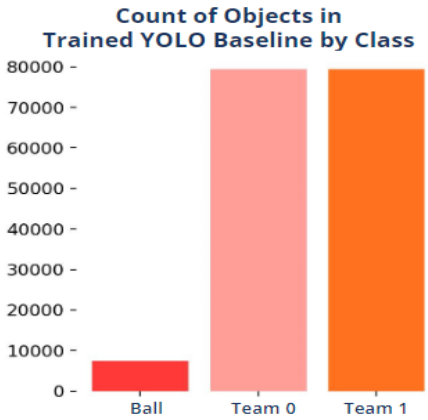
Image Size: 3840x2160 (w x h)	**Count (Frames)**	**Null Values (Count)**	**Mean**	**Standard Deviation**	**Minimum**	**Maximum**
X-Coordinate (Box Top Left Corner)	53,220	1,256	1,724.53	776.32	274.0	3,482
Y-Coordinate (Box Top Left Corner)	53,220	1,256	1,010.91	571.69	-3.0	2,157
Box Width (Pixels)	53,220	1,256	14.99	2.81	8.0	24

Box Height (Pixels)	53,220	1,256	14.10	2.70	3.0	24

The top-view images have a width of 2160p and a height of 3840p. This means that the objects are quite small within the image: the average ball width of 14.99 is only 0.3% of the image width, and the average ball height of 14.10 is only 0.6% of the image height![5] Small objects are hard to detect for most models.[6] To account for this, we take a random-cropping approach.[7]

2) Classes are imbalanced.

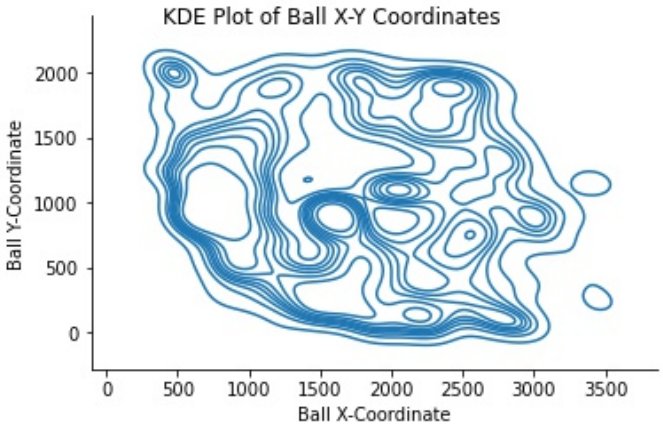
Intuitively, we know that there are 11 players on each team and only one ball. We can also observe this imbalance with a bar graph (below). The entire pitch is visible in the images, so objects rarely leave the frame.



This imbalance can lead the model to over-train on the majority classes (the players) and not learn enough from the minority class (the ball).[8] We mitigate this problem by randomly cropping images, which reduces the number of objects per image, then over-sampling images which contain the ball to reduce the imbalance.[9]

3) The location distribution is asymmetrical.

Consider the KDE plot of the ball's location in top-view images, below:



We can see that there is a peak on one side of the field which is not mirrored on the other side. It is likely that one team enjoyed more possession during one half of the game which the videos

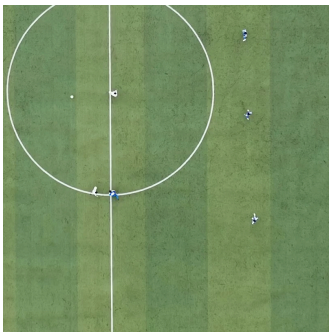
[5] We find a similar issue for the wide-view boxes, where the ball is on average 0.2% of the image width and 1.2% of the image height.
[6] See, e.g., Jacob Solawetz, "How to Detect Small Objects: A Guide," robloflow blog, available at https://blog.roboflow.com/detect-small-objects/.
[7] This is described in the 'SoccerTrack Square' dataset section, below.
[8] See, e.g., "Imbalanced Data," Machine Learning Foundational Courses, Google for Developers, available at developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data.
[9] This is described in the 'SoccerTrack Square' dataset section, below.

reflect, leading to an imbalance. This shows that we should include a randomized left-right flip as an image augmentation.

Our Augmented Dataset: 'SoccerTrack Square'

Of the three above problems, problem 3 (asymmetric locations) is easiest to address - we can apply a pre-written randomized left-right flip augmentation before training.[10] For problems 1 and 2, we generate the 'SoccerTrack Square' dataset using three steps:

1) **Randomly crop and zoom each image in both the Top View and Wide View sets to an 800px800p square.** When objects appear in this square, they are a larger fraction of the image. Additionally, we can train at the original resolution of the image without needing to compress to a lower pixel count and lose detail.[11] The effect of this is apparent: you likely can't see the ball in the original image (left, below), but you might be able to see it as a small dot left of the center stripe in the cropped and zoomed image (right, below).

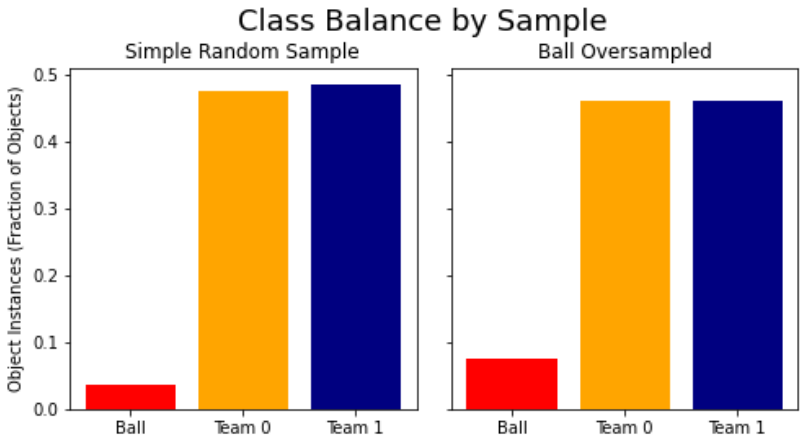


2) **Recalculate the bounding box coordinates** to match the cropped images.[12]

3) **Over-sample the ball** by conducting a stratified random sample (n=10,400) of the cropped images. We ensure that the ball appears in 50% of the sample, players appear *without* the ball in 30%, and no objects appear in 20%, all while maintaining a 50-50 top view-wide view split.

   As a benchmark, we also create a simple random sample (n=10,400) of all the cropped images. With a simple random sample, the ball only appears in about 11.7% of the cropped images. Over-sampling improves the class imbalance ratio from approximately 1:12:12 to 1:6:6.

[10] The packages we use to execute YOLO include image augmentations as tunable parameters. See, e.g., "Model Training with Ultralytics YOLO," Ultralytics, available at https://docs.ultralytics.com/modes/train/#augmentation-settings-and-hyperparameters.
[11] Model training times increase as the input resolution of images increases. Training times for the full-size images at their native resolution are prohibitively long.
[12] We wrote our own function for cropping & re-calculating; built-in augmentation functions don't accommodate bounding box labels.



What is Considered Success/Failure?

There are three key characteristics which define the success of the model:

- Can the model correctly locate each object and identify its class: that is, the ten outfield players for Team A and B, and the ball?
- Does the model accurately predict the location/bounding boxes of each object?
- Does the model succeed for all three classes, including the ball?

The model needs to be highly accurate, or else it is not useful as a replacement for manual review of soccer videos when encoding match data. We consider “success” to be precision and recall rates of at least 90%, with an Intersection over Union (IoU) classification threshold value of 50%.

Evaluation Parameters

We will evaluate primarily using common parameters for object detection models:

- Loss Values:
    - Box loss, which is Complete Intersection over Union (cIoU), for each bounding box in both the predictions and the labels, overlap of predicted and actual bounding boxes.
    - Objectness loss, which is based on the difference between the predicted probability of an object being present and the true label (similar to cross-entropy in a classification model).
    - Classification loss, based on correct or incorrect predictions.
- Precision - the proportion of true positives among positive predictions.
- Recall - the proportion of positive labels which are correctly predicted.
- Mean Average Precision (mAP), which is the precision for each image averaged over all images. Evaluated with prediction thresholds of 0.5 (mAP0.5), and a range (mAP0.5-0.95).

We will also consider confusion matrices to understand which classes the model performs better or worse on. In one instance, we sacrifice some overall accuracy & player accuracy for the sake of accuracy on the ball, which is difficult to locate but arguably the most important for our objective.

Experiments

We perform the following experiments, the results of which we report in the next section:

- We consider logistic regression and simplified CNN classifiers which only check for the presence of the ball, Team 0, and Team 1 somewhere in the image.
- We try five YOLO models (v3, v5, v6, v7 and v8) under baseline conditions (a sample of the SoccerTrack data, without cropping or over-sampling).
- We test the difference between the cropped data with a simple random sample and the cropped data when over-sampled on the ball, using YOLOv8.
- We test the effect of adding additional augmentations on top of our baseline:[13] random rotation up to 20 degrees, random shear up to 10 degrees, and perspective shift up to 0.0003.
- We optimize the base learning rate and final learning rate over all combinations of base learning rate: (0.001, 0.01, 0.1) and final learning rate (0.01, 0.1).[14]
- We optimize the batch size over (16, 32, 64).
- We check the optimizers Stochastic Gradient Descent ('SGD') and Adam.

Unless otherwise stated, experiments use the YOLOv7-tiny model, the SoccerTrack Square data with cropping & over-sampling, baseline image augmentation only, learning rates (0.01, 0.01), batch size 64, Adam optimizer. We test one parameter at a time.

Tests, Results, Graphs, and Discussions

We begin with simple classifiers on the SoccerTrack Square dataset which only test for the presence of each class (the ball, team 0, and team 1) somewhere in the image. We try logistic regression using the greyscale value of every pixel, and we try a 3-layer Convolutional Neural Network ("CNN") using color images. The results are below:

Machine Learning Classifiers – Log. Regression and CNN

Model	Output Type	Loss	Accuracy	Precision	Recall
Log. Regression	Binary (Ball)	1.665	.560	0.500	.179
Log. Regression	Binary (Team 0)	0.745	.719	.816	.701
Log. Regression	Binary (Team 1)	3.671	.712	.733	.989
Log. Regression	Multiclass Predictor	4.561	.197	.392	.138
CNN	Binary (Ball)	0.693	0.481	N/A	.000
CNN	Binary (Team 0)	0.586	0.700	0.716	1.000
CNN	Binary (Team 1)	0.595	0.719	0.722	1.000

[13] We apply an initial set of augmentations for all YOLO baselines and YOLO experiments: color hue, saturation, and value shifting, image translation, image zoom/scaling, a randomized left-right flip, and appending multiple images into mosaics. These are adjusted using a config file in the YOLO packages.

[14] In this case, the final learning rate is multiplied to the base learning rate. So, for a base learning rate of 0.1 and a final learning rate of 0.01, the final learning rate in absolute terms is 0.1 * 0.01 = 0.001.

CNN	Multiclass Predictor	1.549	0.468	0.202	0.450

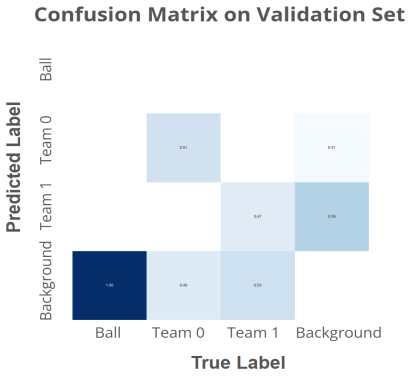
We can see that these models offer no improvement over simply predicting the most common class in the stratified sample.[15] Furthermore, these models only report which classes are present, but not how many are present or where they are located within each image. For an object detection task, we require a larger and more complex neural network; this is why we deploy YOLO for object detection.

Our baseline YOLO implementation uses a simple random sample of 10,400 images from the non-cropped SoccerTrack dataset. The results are as follows:

Baseline YOLO Models

Model	Author	Box Loss	Obj. Loss	Class. Loss	Precision	Recall	mAP 0.5	mAP 0.5-0.95
YOLOv3	Ultralytics	0.116	.247	.0063	.586	.264	.206	.059
YOLOv5	Ultralytics	0.041	.024	.0010	.343	.202	.178	.054
YOLOv6	Meituan	3.397	2.560	.9375	.177	.044	.035	.013
YOLOv7	Wong Kin-Yiu	0.019	.019	.0008	.447	.313	.264	.082
YOLOv8	Ultralytics	2.156	.821	1.061	.401	.267	.298	.116

We can see that these baselines fall well short of our goal of 90% precision and recall. Furthermore, the baselines fall victim to the small-object and imbalanced-class problems when trying to detect the ball. The baseline models almost never predict that the ball is present at all – this is demonstrated by the completely blank top-left square of the YOLOv7 confusion matrix (right). Our objective is to maximize precision and recall *for all classes* – clearly, something must be done to locate the ball. Therefore, we apply the cropping and oversampling in our SoccerTrack Square dataset,[16] and test each of the YOLO models again:



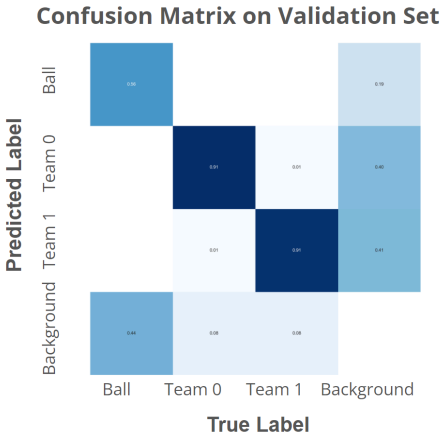
Baseline and Improved YOLO Models

Model	Baseline or Improved	Box Loss	Obj. Loss	Class. Loss	Precision	Recall	mAP 0.5	mAP 0.5-0.95
YOLOv3	**Improved**	**.057**	**.095**	**.0067**	**.772**	**.682**	**.673**	**.255**

[15] Recall that the Logistic Regression model and CNN model which do not rely on YOLO are classifiers, not object detectors. Accuracy values of 0.56 and 0.71 for binary models might *seem* passable - but this is just the rate at which of the ball and each team in our stratified sample.

[16] See the description in the 'Datasets' section.

YOLOv3	Baseline	.116	.247	.0063	.586	.264	.206	.059
YOLOv5	**Improved**	**.051**	**.032**	**.0054**	**.806**	**.678**	**.691**	**.264**
YOLOv5	Baseline	.041	.024	.0010	.343	.202	.178	.054
YOLOv6	**Improved**	**1.966**	**1.275**	**1.124**	**.760**	**.648**	**.648**	**.243**
YOLOv6	Baseline	3.397	2.560	0.9375	.177	.044	.035	.013
YOLOv7	**Improved**	**.0465**	**.0281**	**.0040**	**.841**	**.742**	**.745**	**.281**
YOLOv7	Baseline	.019	.019	.0008	.447	.313	.264	.082
YOLOv8	**Improved**	**1.858**	**1.267**	**1.016**	**0.512**	**0.299**	**0.255**	**0.074**
YOLOv8	Baseline	2.156	.821	1.061	.401	.267	.298	.116



Improvements are apparent: the smallest improvement in precision is about 11% (for YOLOv8) and the largest improvement in precision is about 58% for (for YOLOv6). These models also have the smallest and largest improvements in recall (3% and 60%, respectively) and mAP-0.5 (-4% and 61%). Finally, we can see that the models can detect the ball, as demonstrated by the new confusion matrix for the best model, YOLOv7 (left).

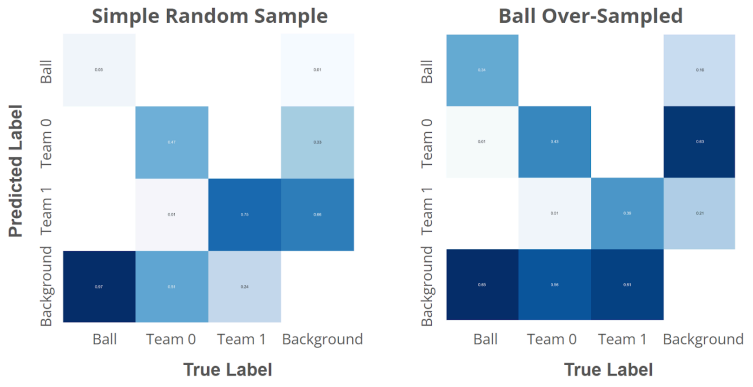
Our next step is to test the hyperparameters and augmentations described in the 'Experiments' section for further improvements. The results are listed below:

Parameter Tuning Results

Model	Tested Parameter	Box Loss	Obj. Loss	Class. Loss	Precision	Recall	mAP 0.5	mAP 0.5-0.95
YOLOv8	SRS Sample	1.931	1.304	1.159	.721	.479	.383	.149
YOLOv8	**Stratified Sample**	**1.858**	**1.267**	**1.016**	**0.512**	**0.297**	**0.255**	**0.074**

YOLOv7	LRs 0.1, 0.01	.050	.029	.0045	.807	.692	.706	.247
YOLOv7	**LRs 0.01, 0.01**	**.0465**	**.0281**	**.0040**	**.841**	**.742**	**.745**	**.281**
YOLOv7	LRs 0.001, 0.01	.0484	.0287	.0053	.676	.713	.681	.246
YOLOv7	LRs 0.1, 0.1	.0516	.0297	.0047	.724	.587	.607	.186
YOLOv7	LRs 0.01, 0.1	.0474	.0284	.0041	.831	.733	.738	.269
YOLOv7	LRs 0.001, 0.1	.0485	.0287	.0051	.778	.7105	.7042	.264
YOLOv7	Batch, 16	.0491	.0289	.0043	.816	.728	.728	.279
YOLOv7	Batch, 32	.0495	.0288	.0043	.770	.689	.683	.236
YOLOv7	**Batch, 64**	**.0465**	**.0281**	**.0040**	**.841**	**.742**	**.745**	**.281**
YOLOv7	**Opt., Adam**	**.0465**	**.0281**	**.0040**	**.841**	**.742**	**.745**	**.281**
YOLOv7	Opt., SGD	.0472	.0282	.0052	.808	.714	.717	.273
YOLOv7	Basic Aug.	.0465	.0281	.004	.841	.742	.745	.281
YOLOv7	**Extra Aug.**	**.0452**	**.0321**	**.005**	**.832**	**.737**	**.734**	**.282**

The stratified sample reduces precision and recall for YOLOv8 relative to cropping & simple random sampling – however, we choose to continue using the stratified sample because of class-level accuracy, demonstrated below, which shows that the ball is ignored under a simple random sample:



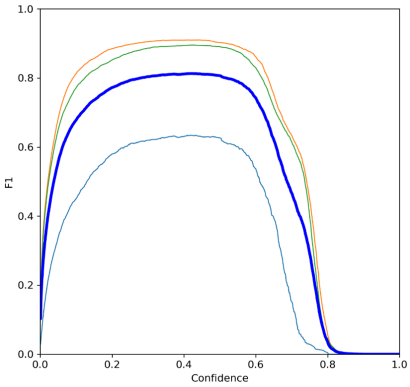
One of our objectives is to develop a model which succeeds for *each class*. There may be a small to moderate penalty to accuracy when using the stratified sample, but we consider it worthwhile because of the dramatic improvement in accuracy on the ball. A soccer tracking model which can't see the ball would have very little use for coaches and analysts!

Aside from this, we identify the best model and parameters, which are summarized in the "Comparisons" section, and run a tuned model for an extended period of 20 training epochs:

Tuned Model Training Results

Model	Epochs	Box Loss	Obj. Loss	Class. Loss	Precision	Recall	mAP 0.5	mAP 0.5-0.95
YOLOv7	10	0.048	0.033	0.0053	0.796	0.703	0.700	0.250
YOLOv7	**20**	**0.042**	**0.031**	**0.0041**	**0.868**	**0.771**	**0.763**	**0.303**

This is our best model so far, for all metrics. An F1 curve on the validation set (right) shows that the precision-recall balance is maximized at around 0.4-0.6 confidence, and we conduct validation and testing at a threshold of 0.5. When deployed on the test set, this model achieves overall rates of 87% precision, 80% recall, and 78% mAP.5,[17] which is even better than the training & validation sets. The model's recall rate on the ball in particular is 53% – which we did not define as a success at the outset of our research, but it is a significant improvement over the starting rate of 0%.



Constraints

Even the smallest 'nano' or 'tiny' YOLO models are complex neural networks with millions of parameters. They must be trained for several hours or days on large image datasets. One must purchase hardware with sufficient data storage, memory, and CPU or GPU for the task, or lease virtual machines. For the SoccerTrack data in particular, the images are almost 700 GB in size when saved to storage in native resolution. Programs for extracting every frame, then creating cropped versions, take about 5 to 7 days. A 5-epoch YOLO training run, using a 'nano' model with between 2-10m parameters, for the SoccerTrack Square sample, takes around 6 hours. We devoted about 28 days of computing time (split between two people) to data extraction, data processing, and model training. In parallel, we needed 6-7 calendar weeks of development time to write programs, conduct EDA, and prepare deliverables.

Standards

We executed YOLOv3, YOLOv5, and YOLOv7 by cloning the GitHub repositories of their implementers (Ultralytics and Wong Kin-Yiu – see Works Cited). We executed YOLOv8 in Python by importing the

[17] The model has precision of 93%, recall of 90%, and mAP.5 of 91% for the Team 0 and Team 1 classes in the test set.

Ultralytics Python package. YOLOv6 is implemented by Meituan Vision AI, and supported by the same Ultralytics Python package which implements YOLOv8.

We managed local Python 3.9 and 3.10 environments using Anaconda. We used Pandas 1.3.4, NumPy 1.23.5, Pillow 8.4.0, PathLib 16.0.0, and MoviePy 1.0.3 for data processing. We use TensorFlow 2.15 and Keras 2.15 to implement the logistic regression and basic CNN. We use matplotlib 3.4.3 and seaborn 0.11.2 for EDA plotting, in addition to automatic output from the YOLO packages. We use scikit-learn 0.24.2 to calculate certain evaluation metrics.

Comparison

Based on the tests and comparisons in the results tables, we find that the most accurate model is YOLOv7, the best learning rate combination is (0.01, 0.01), the best batch size is 64, and the best optimizer is Adam. We observe a loss in overall performance when oversampling on the ball for YOLOv8, but we prefer to use over-sampling anyway because confusion matrices show that the "increased performance" from simple random sampling is only because the ball is ignored. Similarly, we observe a small loss when applying extra image augmentations, but we choose to apply them anyway; the extra loss is small and improving generalization is critically important due to our dataset limitations.

Limitations of the Study

The biggest limitation is that although there are over 100,000 original frames extracted from the SoccerTrack data, all of them are from a single soccer match.The model may have over-trained on the players' gender, race, appearance, jersey colors, the field quality, the surrounding background, and the two available viewing angles, or other details specific to this one match. We work on this by including augmentations such as a randomized left-right flip, randomized rotation, HSV color augmentation, perspective and shear adjustments, mosaic combinations of images, multiple viewing angles, etc. However, future work ***must*** address this issue of generalization and ethics using more matches.

Additionally, although we tested several different models and hyperparameters in the 'Experiments' section, due to computing time restrictions[18] we did not test every *combination* of models and hyperparameters. It is best to test all combinations when possible, because there could be an untested combination which is better than the hyperparameters optimized individually.

Future Work

The clearest way to improve this study is to address the 'Limitations' section - that is, by sourcing data from a large number of games and exhaustively testing the combinations of hyperparameters.

Additionally, it is possible to improve results by scaling up training time and complexity. After 20 training epochs, there is not yet evidence of over-training within our sample, and the model accuracy has not fully plateaued. With additional time, we could run more training epochs or use larger models to improve accuracy.[19] Finally, it may be worthwhile to test non-YOLO object detection models, such as Mask R-CNN, Faster R-CNN, RetinaNet and others.

---

[18] A 5-epoch training run on the SoccerTrack Square dataset takes about 6 hours - see 'Constraints'.

[19] In fact, the Ultralytics FAQ recommends ***300*** epochs as a starting point! See "Tips for Best Training Results," Glenn Jocher, docs.ultralytics.com, available at https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/#training-settings.

Works Cited

Uchida, I., et al. (2022) “SoccerTrack: A Dataset and Tracking Algorithm for Soccer with Fish-eye and Drone Videos,” (“SoccerTrack”), kaggle.com, available at https://www.kaggle.com/datasets/atomscott/soccertrack/data.

Redmon, J., et al. (2016). “You Only Look Once: Unified, Real-Time Object Detection,” (“YOLO Object Detection”), *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Redmon, J., and Farhadi, A., (2018). “YOLOv3: An incremental improvement,” arXiv 1804.02767.

Jocher, G. (2020). YOLOv5 by Ultralytics (Version 7.0). https://doi.org/10.5281/zenodo.3908559.

Li, C., et al. (2023). YOLOv6 by Meituan Vision AI Department (Version 3.0), https://doi.org/10.48550/arXiv.2301.05586.

Wang, C., et al. (2022). “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.” arXiv 2207.02696.

Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLO (Version 8.0.0). https://github.com/ultralytics/ultralytics.

T. Majidzadeh and E. Ndedi, Project Proposal, “Detection of Ball and Player Location from Frame-by-Frame Soccer Match Videos,” MIDS W207 Spring 2024.

Homework Submissions of Etienne Ndedi, MIDS W207 Spring 2024.

Homework Submissions of Timothy Majidzadeh, MIDS W207 Spring 2024.

“How to Detect Small Objects: A Guide,” Jacob Solawetz, robloflow blog, available at https://blog.roboflow.com/detect-small-objects/.

“Imbalanced Data,” Machine Learning Foundational Courses, Google for Developers, available at developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data.

“Model Training with Ultralytics YOLO,” Ultralytics, available at https://docs.ultralytics.com/modes/train/#augmentation-settings-and-hyperparameters.

“Tips for Best Training Results,” Glenn Jocher, docs.ultralytics.com, available at https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/#training-settings.