# **YOLOv5 Training Report: Microplates Dataset**

## **Introduction**

This report provides a comprehensive analysis of the training process and performance metrics for the YOLOv5 object detection model on the microplates dataset. The dataset consists of three classes: buffer, positive, and negative. The training metrics are recorded and saved in a CSV file by the model, which contains all necessary information about the model’s performance during the training process.

## **Dataset Overview**

The microplates dataset is a collection of images containing microplates, which are specialized laboratory equipment used in various scientific experiments. The dataset is composed of three distinct classes:

1. **Buffer**: This class represents the background or negative control wells in the microplate.
2. **Positive**: This class represents the wells containing positive samples or reactions.
3. **Negative**: This class represents the wells containing negative samples or reactions.

The dataset was used to train the YOLOv5 model, a state-of-the-art object detection algorithm developed by Ultralytics. YOLOv5 is known for its high performance and efficiency in detecting and localizing objects in images.

## **Training Configuration**

The training process for the YOLOv5 model was carried out using the following configuration:

* **Model**: YOLOv5
* **Dataset**: Microplates dataset
* **Classes**: Buffer, Positive, Negative
* **Epochs**: 35
* Batch size: 16

## **Training Metrics**

There were various training metrics recorded during the training process. These metrics provide valuable insights into the model's performance and convergence. The following sections analyze the key metrics recorded:

### **Loss Functions**

The training process utilizes three different loss functions:

1. **Box Loss**: This loss function measures the model's accuracy in predicting the bounding box coordinates for the detected objects.
2. **Object Loss**: This loss function measures the model's ability to distinguish between objects and non-objects.
3. **Class Loss**: This loss function measures the model's accuracy in classifying the detected objects into the correct classes (buffer, positive, or negative).

The loss functions are calculated for both the training and validation datasets.

### **Precision, Recall, and mAP**

The CSV file also includes several performance metrics:

1. **Precision**: This metric measures the model's ability to correctly identify positive samples (true positives) from the total detected samples (true positives + false positives).
2. **Recall**: This metric measures the model's ability to detect all positive samples (true positives) from the total number of actual positive samples (true positives + false negatives).
3. **mAP@0.5**: This metric represents the mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5, which is a common evaluation metric for object detection models.
4. **mAP@0.5:0.95**: This metric represents the mAP calculated over various IoU thresholds from 0.5 to 0.95, providing a more comprehensive evaluation of the model's performance.

These metrics are calculated for both the training and validation datasets, allowing for an assessment of the model's generalization performance.

### **Learning Rate Schedulers**

The training process includes information about the learning rate schedulers used during the training process. The learning rate is a hyperparameter that controls the step size during the optimization process. The three columns x/lr0, x/lr1, and x/lr2 represent the learning rates used for different components of the model (e.g., backbone, classifier, etc.).

## **Performance Analysis**

Based on the provided metrics, we can analyze the model's performance during the training process.

### **Loss Functions**

The loss functions for both the training and validation datasets generally decrease as the training progresses, indicating that the model is learning and improving its performance over time.

### **Precision, Recall, and mAP**

The precision, recall, and mAP metrics provide insights into the model's object detection capabilities. The training precision and recall values show a steady increase as the training progresses, indicating that the model is becoming better at detecting and classifying the objects correctly.

The mAP@0.5 and mAP@0.5:0.95 metrics, which are calculated on the validation dataset, provide a more reliable assessment of the model's performance on unseen data. The mAP@0.5 metric reaches a maximum value of around 0.358 (35.8%) at epoch 32, while the mAP@0.5:0.95 metric reaches a maximum value of around 0.192 (19.2%) at epoch 34. These values suggest that the model has achieved reasonable performance in detecting and localizing the microplate objects, but there is still room for improvement.

### **Learning Rate Schedulers**

The learning rate values provided in the CSV file can be used to analyze the learning rate schedules employed during the training process. The learning rates generally decrease over time, which is a common practice in deep learning to ensure convergence and prevent oscillations or divergence during the later stages of training.

## **Conclusion**

This report provides a detailed analysis of the training process and performance metrics for the YOLOv5 model on the microplates dataset. The provided metrics indicate that the model has achieved reasonable performance in detecting and classifying the microplate objects, with room for further improvement. Future work could involve exploring different hyperparameter configurations, data augmentation techniques, or transfer learning approaches to potentially enhance the model's performance. Additionally, qualitative analysis of the model's predictions on a subset of the dataset could provide valuable insights into its strengths and weaknesses, guiding further refinements or adjustments.

## **Guidance about tflite weights**

The tensorflow version is 2.15.0

You can test the model in tflite format by running these command, assuming you are in the YOLOv5 directory:

* Detect: python detect.py --weights runs/train/exp/weights/best-fp16.tflite
* Validate: python val.py --weights runs/train/exp/weights/best-fp16.tflite
* PyTorch Hub: model = torch.hub.load('ultralytics/yolov5', 'custom', 'runs/train/exp/weights/best-fp16.tflite')

Visualize the model architecture by clicking on this link and just drag and drop the tflite weights in the upload area, and visualize the full architecture of the model:  [https://netron.app](https://netron.app/)