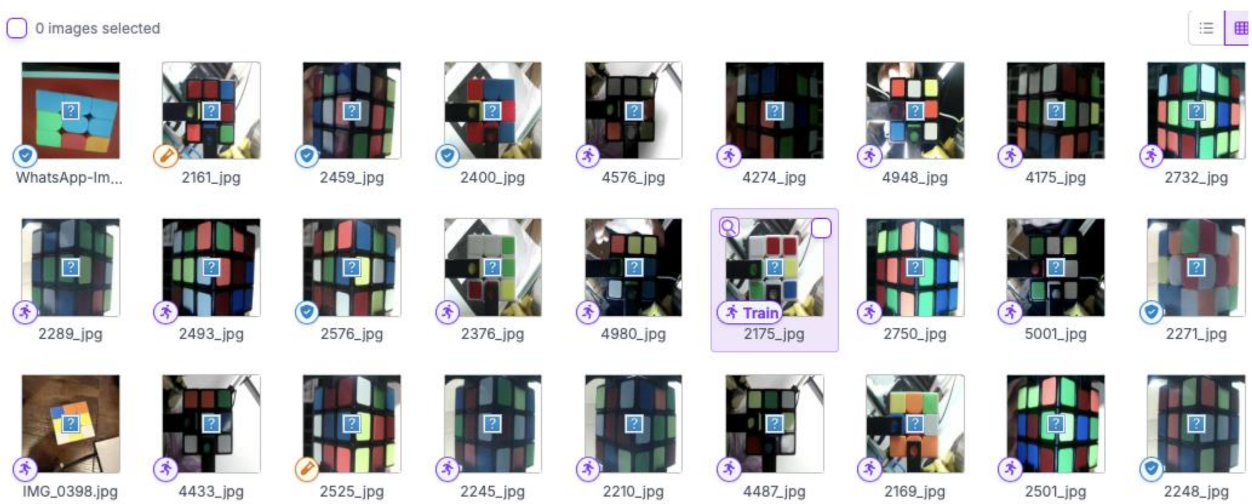
Cube Surface Color Detection Model Training Documentation

This document provides a detailed overview of training a YOLO-based object detection model to identify colors on individual squares of a Rubik's Cube. It covers dataset creation, preprocessing, training procedures, integration with a Raspberry Pi-powered DOFBOT robotic arm, evaluation methods, and next steps.

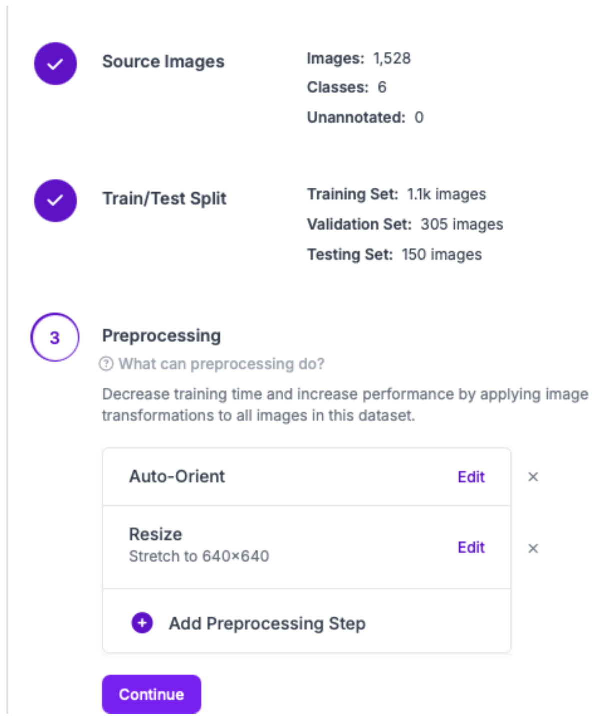
1. Working with Sample Datasets on Roboflow



Dataset Preparation

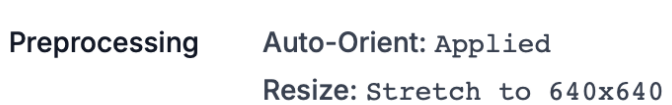
- Dataset Collection: Gather images of Rubik's Cubes in various orientations and lighting conditions. Ensure that all six colors (red, blue, yellow, green, orange, white) are represented.
- Annotation: Use Roboflow's annotation tool to label each square with its corresponding color. YOLO annotations require bounding boxes around objects and labels in the format:
  <class-id> <x_center> <y_center> <width> <height> (normalized to image dimensions).

- Exporting Annotations: Export the dataset in YOLO format for compatibility with YOLO training frameworks.

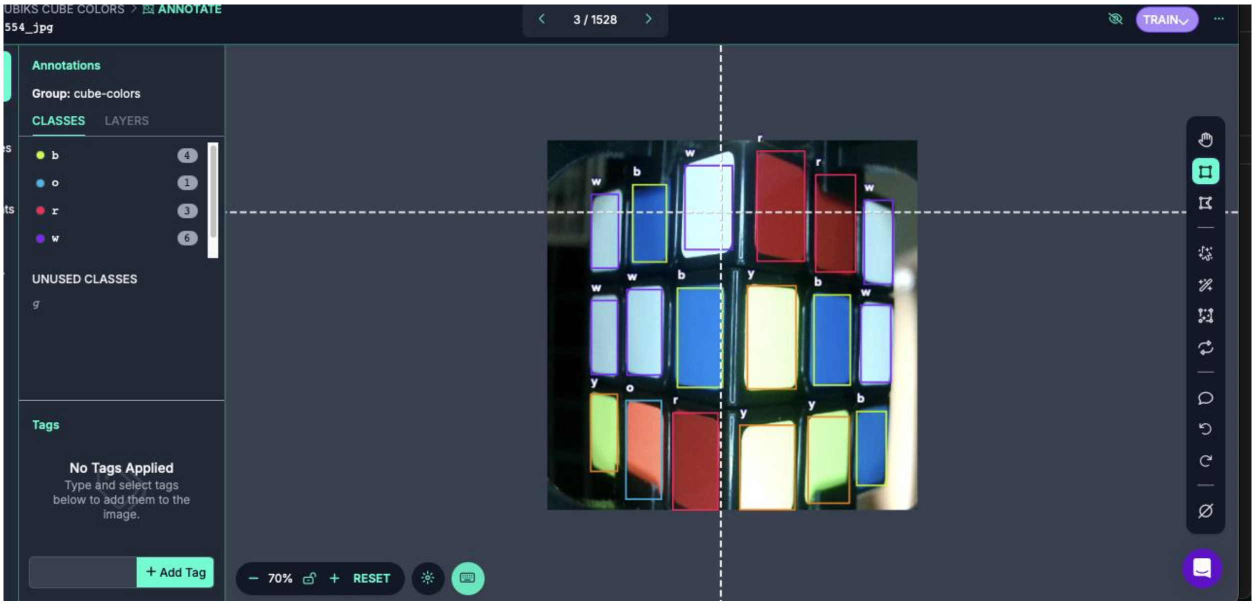


Preprocessing

- Auto-Orient: Automatically adjusts image orientation based on metadata to ensure consistency during training.
- Resize: Standardize image dimensions to 640×640 pixels for YOLO input requirements. This ensures uniformity without distortion.



2. Incorporating Base Sample into DOFBOT Robotic Arm with Camera



Integration Steps

- Capture new images directly from the robotic arm camera to create a realistic base sample dataset.
- Annotate these images using Roboflow or LabelImg software.
- Generate YOLO-compatible annotations and split your dataset into:
    - Training Set (70%)
    - Validation Set (20%)
    - Testing Set (10%)

Data Transformations Simplified

For simplicity and practicality in real-time applications, focus primarily on two transformations:

- Resizing Images: Standardize all images to 640×640 pixels.
- Auto-Orient: Correct orientation automatically based on metadata.

3. Creating Model Training Code and Integration

Training Code

The following Python code snippet demonstrates how to train a YOLOv4 model using the annotated dataset, please refer to attachments for full code.

python

```
!python train.py \
--data data.yaml \
--cfg yolov4.cfg \
--weights yolov4.conv.137 \
--epochs 100 \
--img-size 640
```

- data.yaml: Specifies paths to training and validation datasets.
- cfg: Configuration file for YOLOv4.
- weights: Pretrained weights for transfer learning.

(Full detailed code provided separately as an attachment.)

Training Code

The following Python code snippet demonstrates how to train a YOLOv4 model using the annotated dataset:

```
!python train.py \
--data data.yaml \
--cfg yolov4.cfg \
--weights yolov4.conv.137 \
--epochs 100 \
--img-size 640
```

- data.yaml: Specifies paths to training and validation datasets.
- cfg: Configuration file for YOLOv4.

- weights: Pretrained weights for transfer learning.

Integration with DOFBOT Camera

Integration with DOFBOT Camera System

To implement real-time detection using the DOFBOT robotic arm camera:

1. Install necessary Python libraries:

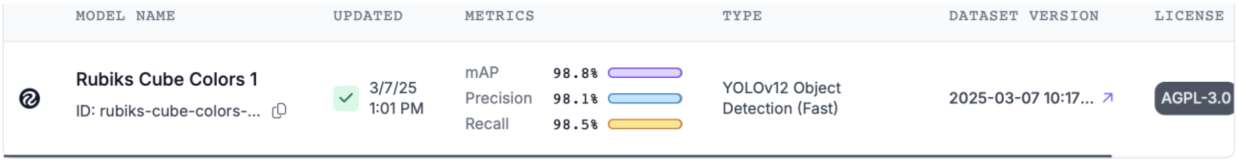
```
pip install opencv-python roboflow
```

1. Modify the DOFBOT camera script to feed live video frames into the trained YOLO model:

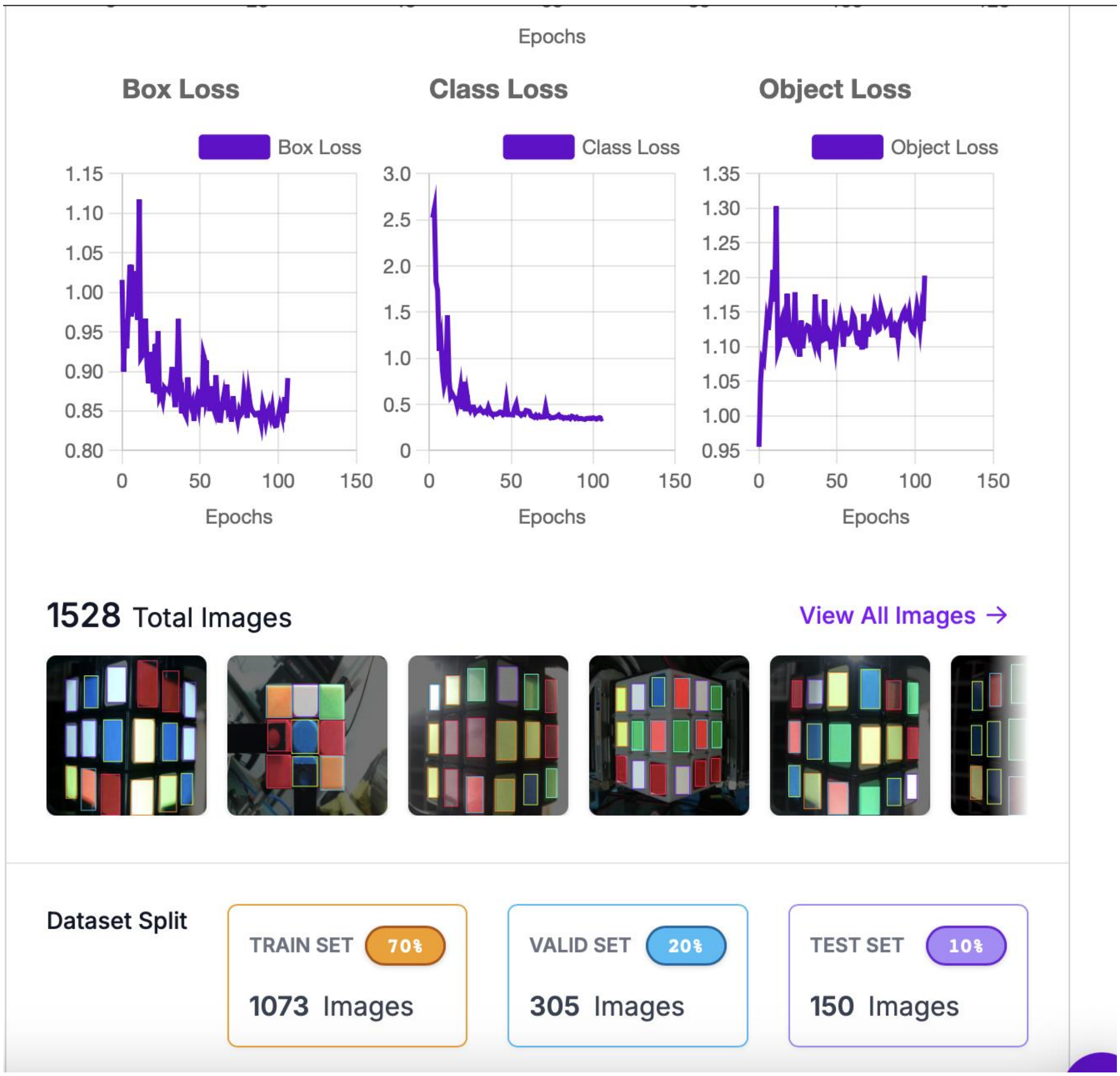
```
import cv2
from yolov4 import Detector

detector = Detector(weights="best.pt", config="yolov4.cfg", classes="classes.txt")

cap = cv2.VideoCapture(0)
while True:
  ret, frame = cap.read()
  detections = detector.detect(frame)
  # Display detections on frame
  cv2.imshow("Rubik's Cube Detection", frame)
  if cv2.waitKey(1) & 0xFF == ord('q'):
    break
cap.release()
cv2.destroyAllWindows()
```

Please refer to the full github for all files: https://github.com/Robjects-ML/SmartVision.git

4. Evaluating the Model Performance



Evaluation was done on roboflow with the sample dataset tested for metrics as shown in the visual below:



5. Next Steps

To further enhance the project and achieve a robust solution, consider these next steps:

Dataset Expansion

- Capture additional images covering all six Rubik's Cube colors under varying lighting conditions and angles.
- Regularly update annotations and retrain models with expanded datasets.

Model Optimization

- Experiment with lightweight models such as YOLOv4-Tiny for improved inference speed suitable for Raspberry Pi hardware constraints.
- Fine-tune hyperparameters (learning rate, epochs, batch size) to improve accuracy without compromising inference speed.

Deployment

- Deploy optimized trained models onto Raspberry Pi hardware integrated within the DOFBOT robotic arm using OpenCV or Roboflow Inference API.

Real-Time Applications

- Integrate detection outputs directly into robotic control algorithms for real-time manipulation tasks based on detected cube colors.

Advanced Features

- Investigate adding depth-sensing or stereo vision capabilities for more advanced spatial understanding and precise manipulation of Rubik's Cubes.