Project 1: Gym Equipment Classification

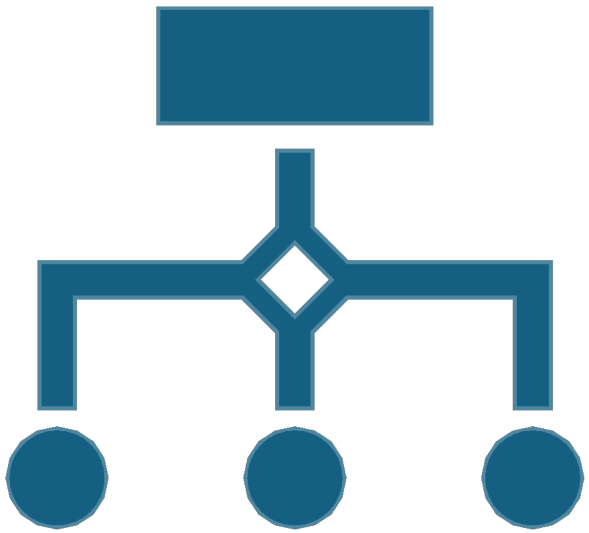
Project 2 : Automatic Number/License Plate Detection and Recognition System

Sudheer kumar Tatavalu

Hima sai Kuruba

Harika Suravarapu

Project 1: Gym Equipment Classificatio n



Dataset

- Task : image classification
- Dataset Link : https://drive.google.com/drive/folders/1pyTg4sFcYURECiE8vEt3KurKmDxeDkyb?usp=drive_link
- Number of samples and image size : 1113 and 256*256
- Number of classes : 5
- Partition: train- 80% test- 10% validation-10%
- Mean= tensor([0.7282, 0.7209, 0.7178]) and
- Standered Deviation= ([0.2826, 0.2832, 0.2816])

Data Augumentation techniques :

- Resizing : All images are resized to a fixed size of 256×256
- Conversion to Tensor : Images are converted from PIL format to PyTorch tensors.
- Dataset Partitioning : The dataset is split into training (80%) and validation (20%) sets
- A base transformation is temporarily applied to calculate the mean and standard deviation of the dataset for normalization

Sample images from 2 classes

Dumbells

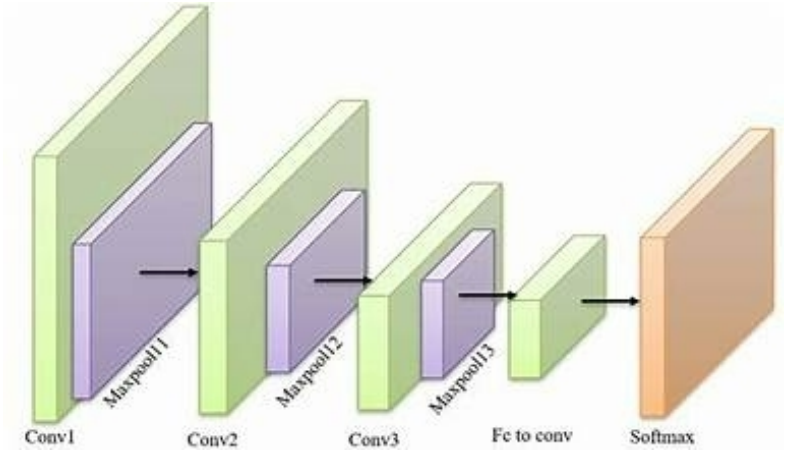
Rowing machine

Architechure of NN and Loss function

$$\mathcal{L} = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$$

Convolutional Neural Network (CNN) architecture. A CNN with 3 convolutional layers, followed by pooling, fully connected layers, and dropout.

Since this is a multi-class classification problem, use here we used Cross-Entropy Loss function.



Optimization Details

Mini-Batch Size:

- 32 (used for training, validation, and testing).

Optimization Algorithm:

- Stochastic Gradient Descent (SGD).

Hyperparameters for SGD:

- Learning Rate: 0.01
- Momentum: 0.9
- Weight Decay: 1e-4

Other Tuned Hyperparameters:

- Number of Filters: 32
- Number of Convolutional Layers: 3
- Step Scheduler: StepLR (Step Size: 2, Gamma: 0)

Hyperparameter Tuning Strategy

Tuning Strategy:

- Manual search based on experimental results.
- Overfitting strategy on two images to validate model flexibility.

Hyperparameter Ranges/Values:

Learning Rate: 0.01 to 0.001

- **Batch Size:** 32
- **Number of Layers:** 2 to 7
- **Number of Filters:** 16, 32, 64
- **Weight Decay:** 1e-4 to 5e-4
- **StepLR Gamma:** 0 to 0.1

Hyperparameter Tuning without Learning Rate Decay

Optimization Process Overview

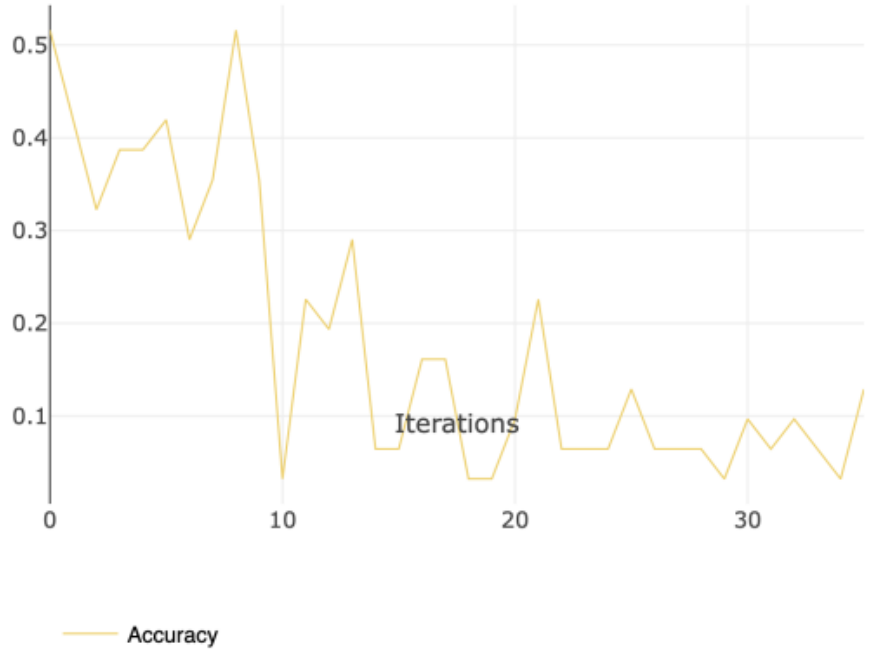
- **Optimization Method:** HyperParameterOptimizer
- **Search Strategy:** Random Search (with fallback to Bayesian or HP-Bandster if available)
- **Parameters Tuned:**
  - num_layers (Range: 3 to 7)
  - num_filters (Range: 5 to 32)
  - learning_rate (LogUniform from 1e-3 to 1e-1)
  - momentum (Range: 0.7 to 0.9)
  - weight_decay (Discrete: 1e-3, 1e-2, 1e-1)

Hyperparameter Tuning with Learning Rate Decay

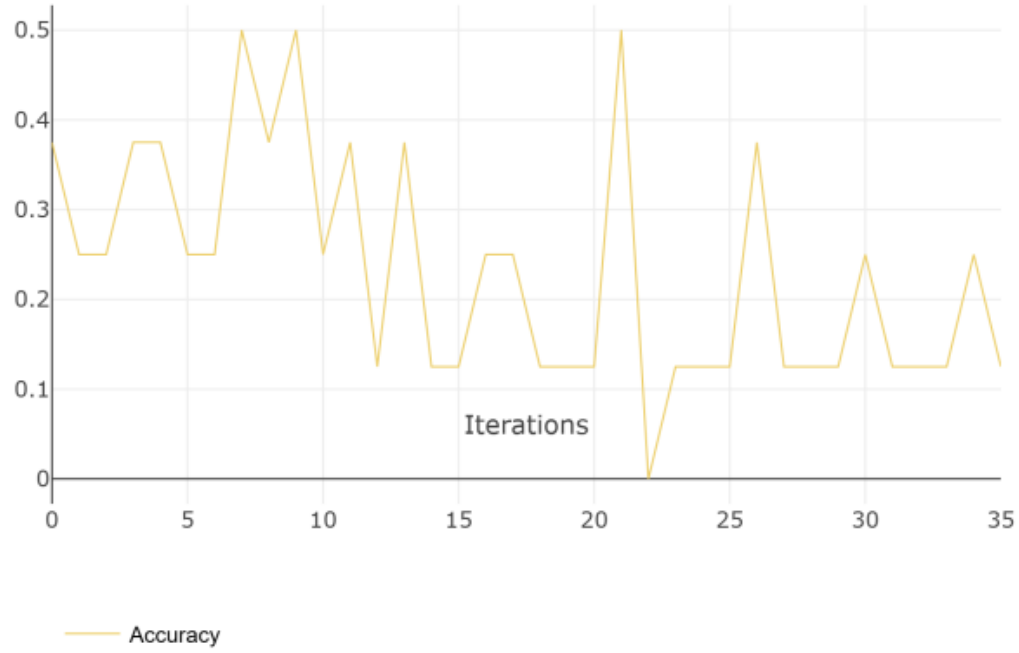
Optimization Process Overview (with Learning Rate Decay)

- **Optimization Method:** HyperParameterOptimizer
- **Search Strategy:** Random Search (with fallback to Bayesian or HP-Bandster if available)
- **Parameters Tuned:**
  - num_layers (Fixed: 5)
  - num_filters (Fixed: 17)
  - learning_rate (Fixed: 0.01)
  - momentum (Fixed: 0.74)
  - weight_decay (Fixed: 0.1)
  - gamma (LogUniform from 1e-4 to 1e-1 for learning rate decay

Accuracy of HPO Tuning without LR Decay



Accuracy of HPO Tuning with LR Decay



Transfer Learning

What is Transfer Learning?

Leverages pre-trained models (e.g., ResNet-18) on large datasets (e.g., ImageNet) for fine-tuning on specific tasks.

Why Use Transfer Learning?

- **Faster Training**: Pre-learned features like edges and textures.
- **Improved Accuracy**: Particularly useful for small datasets.

Experiment Setup

Model Architecture:

- **Base Model**: Pre-trained **ResNet-18** on ImageNet.
- **Modifications**: Fine-tuned only the final fully connected (FC) layer for the new task (6 classes: dumbbell, cycle, etc.).

Dataset:

- **Classes**: Dumbbell and Cycle.
- **Split**: 80% for training, 20% for testing.
- **Image Preprocessing**: Resized to 224x224, normalized using ImageNet mean and std.

Training Configuration:

- **Batch Size**: 4
- **Optimizer**: Stochastic Gradient Descent (SGD) with momentum (0.9).
- **Learning Rate**: 0.001 with a learning rate scheduler (StepLR, step size=7, gamma=0.1).
- **Loss Function**: Cross-Entropy Loss.
- **Epochs**: 25 epochs.

Device: Model trained on **GPU** (if available) or **CPU**.

Results & Analysis

Training Accuracy and Loss:

- **Accuracy**: Achieved ~85% accuracy on training set after 25 epochs.
- **Loss**: Reduced to ~0.35 by the end of training.

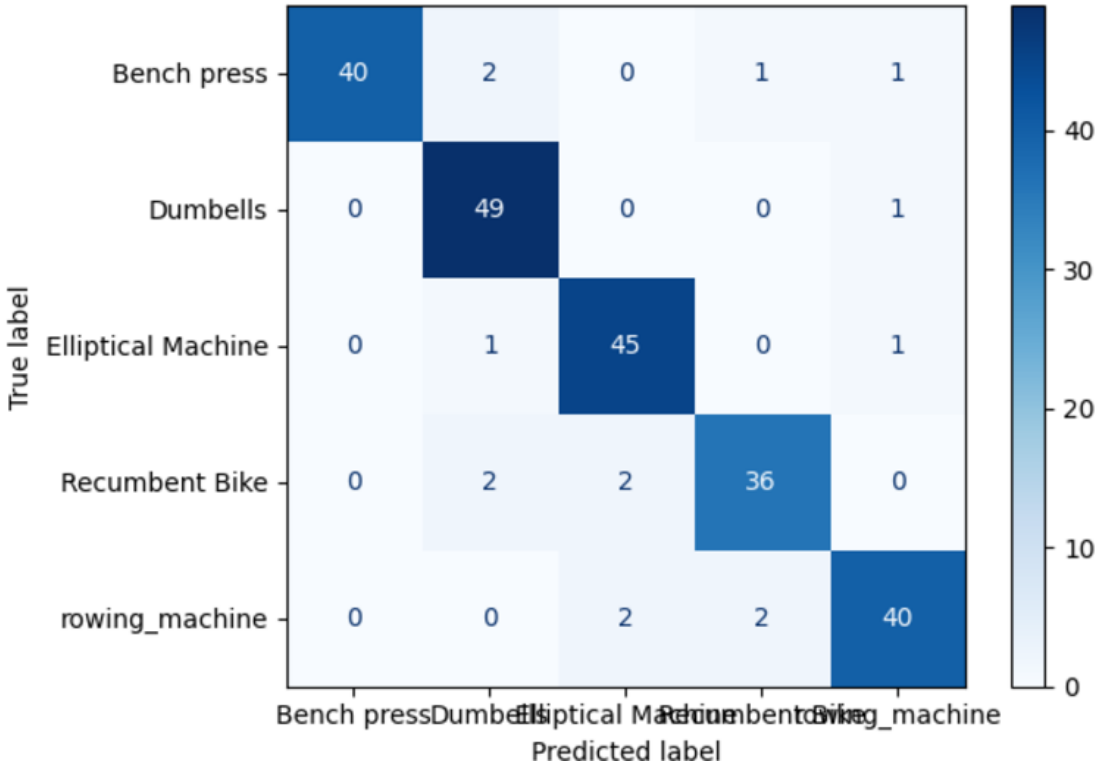
Test Accuracy:

- **Accuracy on Test Set**: ~82% accuracy on test data.
- **Confusion Matrix**: Good class separation with some misclassifications due to visual similarities.

Performance Metrics:

- **Precision**: ~0.87, indicating the model is good at predicting the correct class.
- **Recall**: ~0.80, showing some missed instances of the classes.
- **F1-Score**: Balanced scores across classes.

Model Evaluation




```
Classification Report:
                    precision    recall  f1-score   support

       Bench press     1.0000    0.9091    0.9524        44
          Dumbells     0.9074    0.9800    0.9423        50
Elliptical Machine     0.9184    0.9574    0.9375        47
    Recumbent Bike     0.9231    0.9000    0.9114        40
    rowing_machine     0.9302    0.9091    0.9195        44

          accuracy                         0.9333       225
         macro avg     0.9358    0.9311    0.9326       225
      weighted avg     0.9351    0.9333    0.9333       225

Validation Loss: 0.2496
Validation Accuracy: 0.9333
```

Transfer Learning vs Fully trained NN

Comparing the performances

Aspect	Transfer Learning (ResNet18)	Fully Trained NN (Custom CNN)
Training Time	Significantly faster due to pre-trained weights	Longer, as training is from scratch
Accuracy	Achieved high accuracy with fewer data	Higher potential accuracy with more data
Data Requirement	Effective with limited data (smaller dataset)	Needs a larger dataset to perform well
Computational Cost	Lower, efficient use of resources	Higher, more resources required
Flexibility	Less flexible, constrained by pre-trained features	More flexible, learns task-specific features

Project 2: Automatic Number/License Plate Detection and Recognition System

Data Collection and processing

Project Overview:

- **Task:** Object detection and number plate text extraction using YOLOv5 and Tesseract OCR.
- Detects three classes: **Logo**, **Number Plate**, and **Signal Lights**.

Dataset Details:

- **Link to Dataset:** Dataset stored on Google Drive for easy access in Google Colab.
- **Number of Samples:** Approximately **300** images collected and annotated.
- **Image Size:** Images are resized to **640x640** pixels for YOLOv5 input.
- **Number of Classes: Three (3) classes:**
  - Logo
  - Number Plate
  - Signal Lights

Data Partitioning

- **Data Splits:Training Set:** 80% of the dataset.
- **Validation Set:** 10% of the dataset.
- **Test Set:** 10% of the dataset.

- dataset/

```
├── train/
│   ├── car0.jpg
│   ├── car0.txt
│   ├── car1.jpg
│   ├── car1.txt
│   └── ...
├── test/
└── val/
```

Data Normalization

Normalization Process:

1. Images are normalized using a scale factor of **1/255**.
2. Pixel values are scaled to the range **[0, 1]** for efficient model training.
3. Mean and standard deviation normalization is handled internally by YOLOv5's pre-processing pipeline.

Data Augmentation Techniques:

- **Augmentations Applied:Horizontal Flips:** Randomly flip images to improve robustness.
- **Random Resizing:** Vary image sizes to prevent overfitting.
- **Color Jittering:** Adjust brightness, contrast, and saturation.
- **Rotation and Scaling:** Slight rotations and rescaling to simulate varied camera angles.
- **Gaussian Noise:** Add noise for robustness against noisy inputs.

Sample images

- Visualize two samples from different classes and their annotations
- Provide the data structure of labels for one sample

Sample1:

Sample1: labelled

Sample2:

Sample2: labelled

Sample 1: Class - Number Plate
[Number Plate Image with Bounding Box]

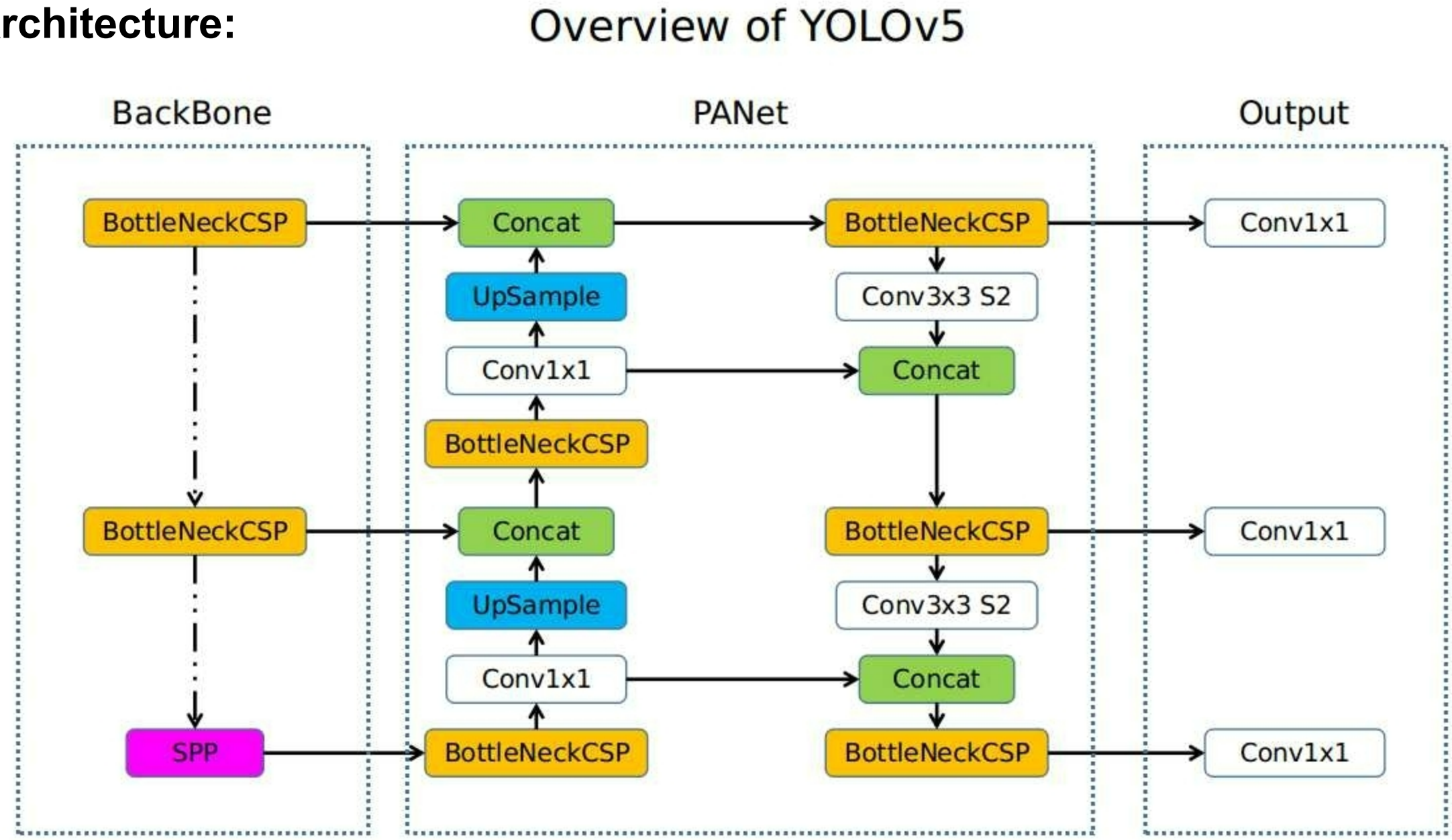
•**Annotation Details:**

•Bounding Box: [791, 1807, 1066, 1995]
xmin, ymin, xmax, ymax

•Class: **Number Plate**

```
<annotation>
    <folder>Cars</folder>
    <filename>Car10.jpg</filename>
    <path>C:\Cars\Car10.jpg</path>
    <size>
        <width>2571</width>
        <height>3056</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>number_plate</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>791</xmin>
            <ymin>1807</ymin>
            <xmax>1066</xmax>
            <ymax>1995</ymax>
        </bndbox>
    </object>
```

Architecture:



Total Loss:
Total Loss=λcoord·IoU Loss+λobj·Objectness Loss+λcls·Classification Loss

Where:
λcoord , λobj and λcls are weights for each loss component

YOLOv5 uses a combination of three loss components to optimize object detection performance:

1. **Bounding Box Loss (IoU Loss):**
    1. Measures the overlap between predicted and ground-truth boxes.
    2. Formula:

$$\text{IoU} = \text{Area of Overlap} / \text{Area of Union}$$

1. **Class Probability Loss (ClassificObjectness Loss (Confidence Loss):**
    1. Penalizes incorrect predictions of whether an object exists in a particular grid cell.
    2. Uses Binary Cross-Entropy (BCE) loss to predict the presence of an object.
2. **ation Loss):**
    1. Uses BCE or Cross-Entropy loss to classify objects into the correct categories.

NN architecture:

	from	n	params	module	arguments
0	-1	1	3520	models.common.Conv	[3, 32, 6, 2, 2]
1	-1	1	18560	models.common.Conv	[32, 64, 3, 2]
2	-1	1	18816	models.common.C3	[64, 64, 1]
3	-1	1	73984	models.common.Conv	[64, 128, 3, 2]
4	-1	2	115712	models.common.C3	[128, 128, 2]
5	-1	1	295424	models.common.Conv	[128, 256, 3, 2]
6	-1	3	625152	models.common.C3	[256, 256, 3]
7	-1	1	1180672	models.common.Conv	[256, 512, 3, 2]
8	-1	1	1182720	models.common.C3	[512, 512, 1]
9	-1	1	656896	models.common.SPPF	[512, 512, 5]
10	-1	1	131584	models.common.Conv	[512, 256, 1, 1]
11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	0	models.common.Concat	[1]
13	-1	1	361984	models.common.C3	[512, 256, 1, False]
14	-1	1	33024	models.common.Conv	[256, 128, 1, 1]
15	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
16	[-1, 4]	1	0	models.common.Concat	[1]
17	-1	1	90880	models.common.C3	[256, 128, 1, False]
18	-1	1	147712	models.common.Conv	[128, 128, 3, 2]
19	[-1, 14]	1	0	models.common.Concat	[1]
20	-1	1	296448	models.common.C3	[256, 256, 1, False]
21	-1	1	590336	models.common.Conv	[256, 256, 3, 2]
22	[-1, 10]	1	0	models.common.Concat	[1]
23	-1	1	1182720	models.common.C3	[512, 512, 1, False]
24	[17, 20, 23]	1	21576	models.yolo.Detect	[3, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]

YOLOv5s summary: 214 layers, 7027720 parameters, 7027720 gradients, 16.0 GFLOPs

Transfer Learning:

```
!python train.py --data data.yaml --weights yolov5s.pt \
--img 640 --epochs 10 --batch-size 8 --name yolo_initial2 \
--freeze 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
100% 755k/755k [00:00<00:00, 14.2MB/s]
Overriding model.yaml nc=80 with nc=3

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12          [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16          [-1, 4]  1         0  models.common.Concat                    [1]
```

Training

- Mini-batch size
- Optimization algorithm and its hyperparameter settings (e.g., learning rate, weight decay)
- Hyperparameter search

Mini-batch Size and Optimization

Mini-batch Size:

- Mini-batch size of **8** provides a balance between training efficiency and memory constraints.

Optimization Algorithm: SGD with MomentumPurpose: Helps accelerate convergence by incorporating past gradient directions.

Hyperparameter setting:

- **Momentum** (0.7 to 1.0)
- **Learning Rate (**0.0001 to 0.01)
- **Weight Decay**: (0, 1.0e-9, 1.0e-5)

Hyperparameter Search

Random Search for Hyperparameter Tuning

Method: Random Search was used to explore a wide range of hyperparameters.

Best Hyperparameters Achieved:

- **Learning Rate:** 0.01
- **Weight Decay:** 1e-09
- **Momentum:** 0.8151

- This combination of hyperparameters achieved the best model performance

Loss Comparison (Training vs. Validation)

- **Epoch 0**:**Train Box Loss**: 0.11641
- **Train Object Loss**: 0.051936
- **Train Class Loss**: 0.039596
- **Validation Box Loss**: 0.10114
- **Validation Object Loss**: 0.043514
- **Validation Class Loss**: 0.035188

Epoch 24:**Train Box Loss**: 0.036988

- **Train Object Loss**: 0.029989
- **Train Class Loss**: 0.0044069
- **Validation Box Loss**: 0.037226
- **Validation Object Loss**: 0.020346
- **Validation Class Loss**: 0.0036811

Key Observations from Data:

•As training progresses, we expect both **training loss** and **validation loss** to decrease.

Performance Metrics (Precision, Recall, mAP)

- **Epoch 0**:**Precision**: 0.073413
- **Recall**: 0.0011638
- **mAP@0.5**: 0.00083556
- **mAP@0.5:0.95**: 0.00023351

- **Epoch 24**:**Precision**: 0.92911
- **Recall**: 0.0044069
- **mAP@0.5**: 0.92868
- **mAP@0.5:0.95**: 0.56218

Key Observations:

•**Precision** and **Recall** increase significantly as training progresses, showing that the model is getting better at detecting objects.

•**mAP** (both at IoU 0.5 and 0.5:0.95) improves over time, indicating that the model is improving its detection quality across both easy and harder detection cases

Pre- and Post-Training Comparison (Summary)

Training Loss:

- Training loss has significantly decreased, indicating that the model has learned the patterns in the data. For example:
  - **Epoch 0**: **Train Box Loss** = 0.11641, **Train Object Loss** = 0.051936
  - **Epoch 24**: **Train Box Loss** = 0.036988, **Train Object Loss** = 0.029989

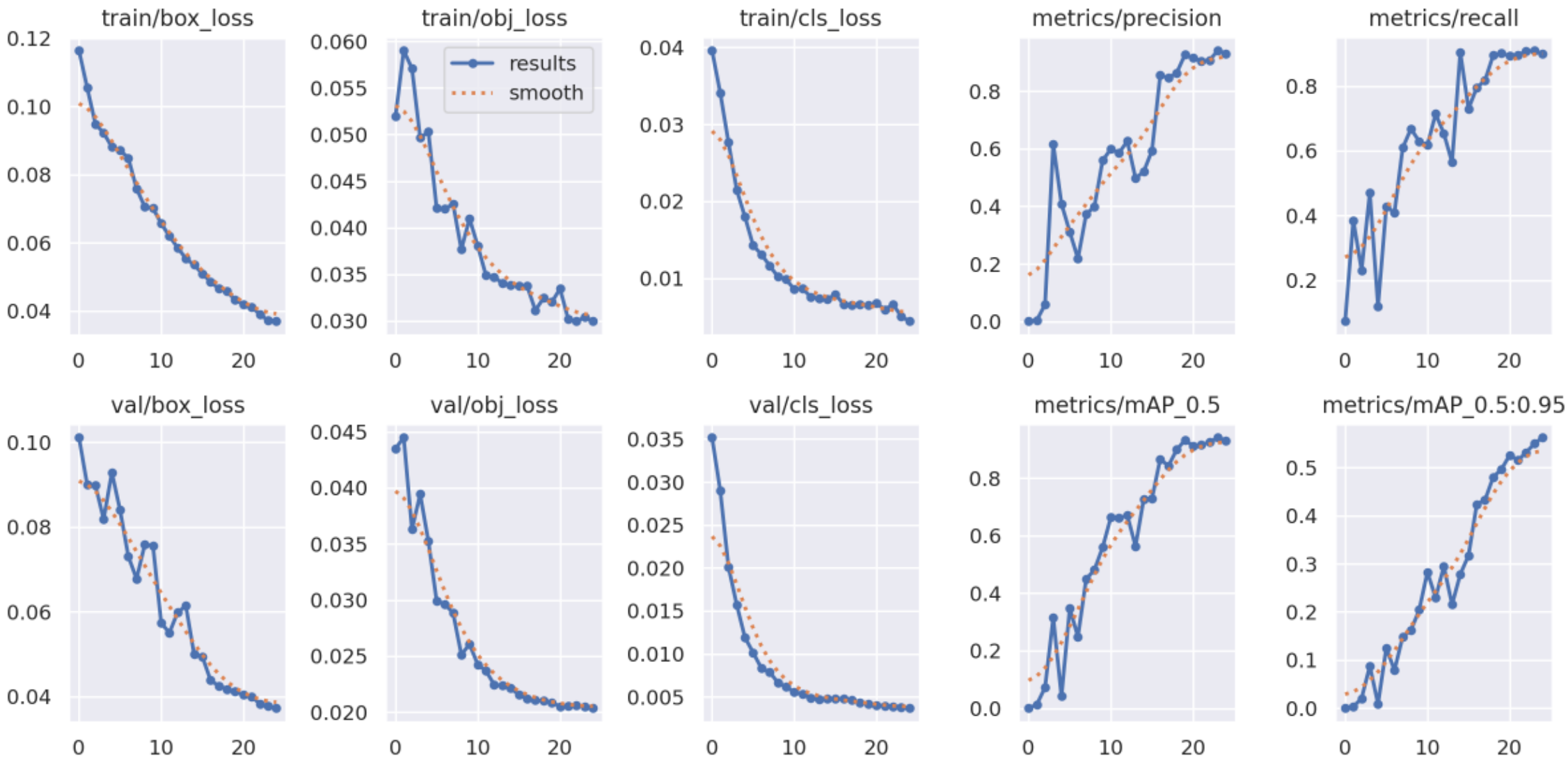
Validation Loss:

- Validation loss also decreased, though at a slightly slower rate than the training loss, which indicates that the model is generalizing well to unseen data.
  - **Epoch 0**: **Validation Box Loss** = 0.10114
  - **Epoch 24**: **Validation Box Loss** = 0.037226

Performance Metrics:

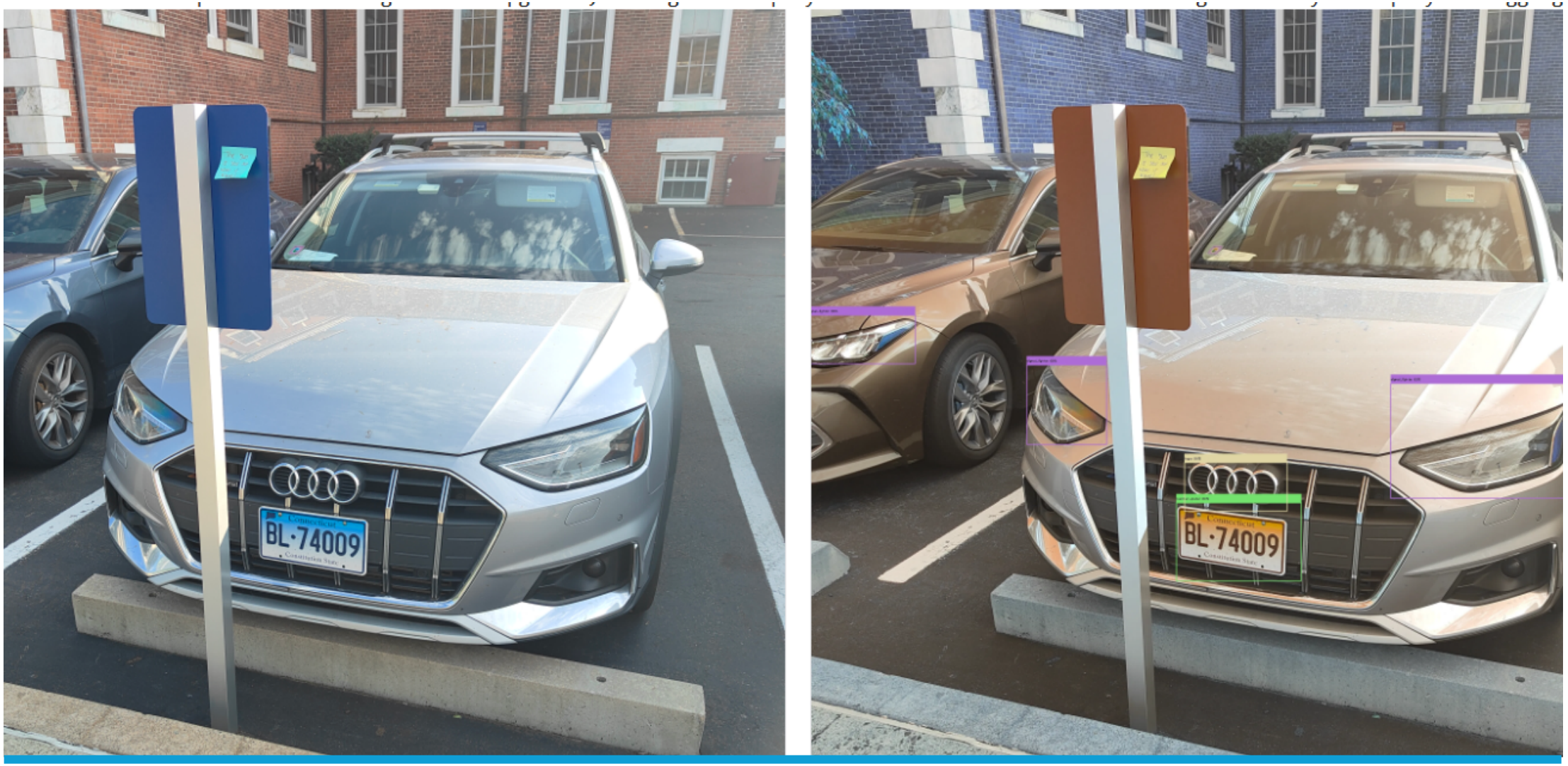
- The model shows a marked improvement in performance, as seen from the **precision**, **recall**, and **mAP** metrics:
  - **Epoch 0**: **Precision** = 0.073413, **Recall** = 0.0011638, **mAP@0.5** = 0.00083556, **mAP@0.5:0.95** = 0.00023351
  - **Epoch 24**: **Precision** = 0.92911, **Recall** = 0.0044069, **mAP@0.5** = 0.92868, **mAP@0.5:0.95** = 0.56218

Graphical Representation:



BEFORE TRAINING:

AFTER TRAINING:



CONCLSUION:

Before Training: The model struggles with object detection and classification.

After Training: The model improves its object localization, classification, and confidence levels, indicating that it has learned the relevant patterns from the training data.