Project1: Weather Classification Project2: Object Detection Using YOLOv5s

Team Members:

- Raghavendra Aakula
- Vaishnavi Kukkala
- Sai Charan Chandu Patla

Project1: Weather Classification

Dataset

- **Task:** Classify weather conditions into 5 classes: *Rainy, Sunrise, Cloudy, Foggy and Shine.*
- **Dataset Link:** https://www.kaggle.com/datasets/vijaygiitk/multiclass-weather-dataset/data
- **Number of Samples:** 1500
- **Image Size:** 256×256
- **Classes:** 5 weather categories. **(Rainy**, **Sunrise**, **Cloudy**, **Foggy**, and **Shine)**
- **Partitioning:** Training: 70%, Validation: 20%, Testing: 10%
- **Normalization:** Mean: [0.485,0.456,0.406], Std: [0.229,0.224,0.225]
- **Data Augmentation**: None

Sample Images

NN

- **Architecture:** Custom CNN with:
  - 3 Convolutional Layers
  - 32 Filters/Layer
  - Kernel Size: 3×33
  - Pooling: MaxPooling with 2×22
- **Classifier:** Fully connected layer after flattening.
- **Loss Function:** CrossEntropyLoss.

NN

```
class CustomCNN(nn.Module):
    def __init__(self, num_classes, num_filters=32,num_layers=2, input_size=(256, 256)):
        super(CustomCNN, self).__init__()

        # Create convolutional layers dynamically
        layers = []
        in_channels = 3
        for _ in range(num_layers):
            layers.append(nn.Conv2d(in_channels,num_filters, kernel_size=3, padding=1))
            layers.append(nn.ReLU())
            layers.append(nn.MaxPool2d(kernel_size=2))  # Reduce spatial dimensions by half
            in_channels = num_filters

        self.features = nn.Sequential(*layers)

        # Calculate flattened size after convolutional layers
        height, width = input_size
        for _ in range(num_layers):
            height //= 2
            width //= 2
        self.flattened_size = num_filters * height * width

        self.classifier = nn.Linear(self.flattened_size, num_classes)

    def forward(self, x):
        x = self.features(x)  # Convolutional layers
        x = torch.flatten(x, 1)  # Flatten
        x = self.classifier(x)  # Fully connected layer
        return x
```

Optimization and Hyperparameter tunings

- **Mini-Batch Size:** 32.
- **Optimization Algorithm:**
    - Optimizer: SGD
    - Learning Rate: 0.01
    - Momentum: 0.9
    - Weight Decay: $1 \times 10-4$
- **Tuned Hyperparameters:**
- Number of Layers
- Filters per Layer

Optimization and Hyperparameter tunings

Without Learning Rate Decay:

The following hyperparameters were tuned using **grid search** over a defined range:

1. **Number of Layers:** 3 to 7 (step size 1). Range: [3,4,5,6,7]
2. **Number of Filters:** 5 to 40 (step size 2). Range: [5,7,9,...,39]
3. **Learning Rate:** Log-uniform distribution from 1e-3 to 1e-1
4. **Momentum:** Uniform range from 0.7 to 1.0. Range: [0.7,0.8,0.9,1.0]
5. **Weight Decay:** Discrete values [1e-3, 1e-2,1e-1]

Optimization and Hyperparameter tunings

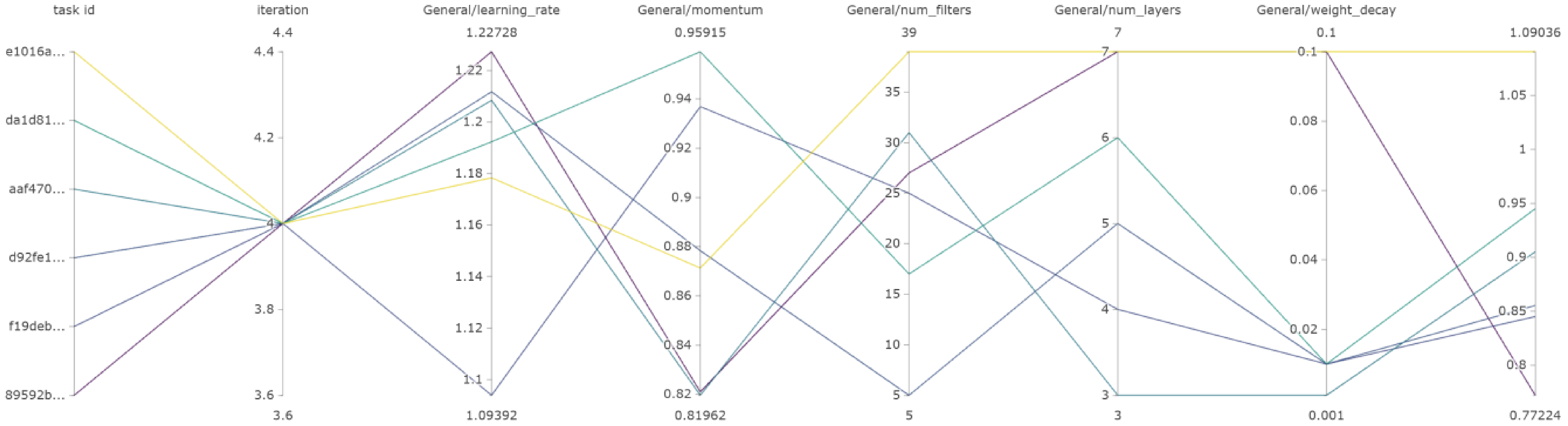
With Learning Rate Decay

We have fixed the hyperparameters based on the results that we obtained from tuning using ClearML

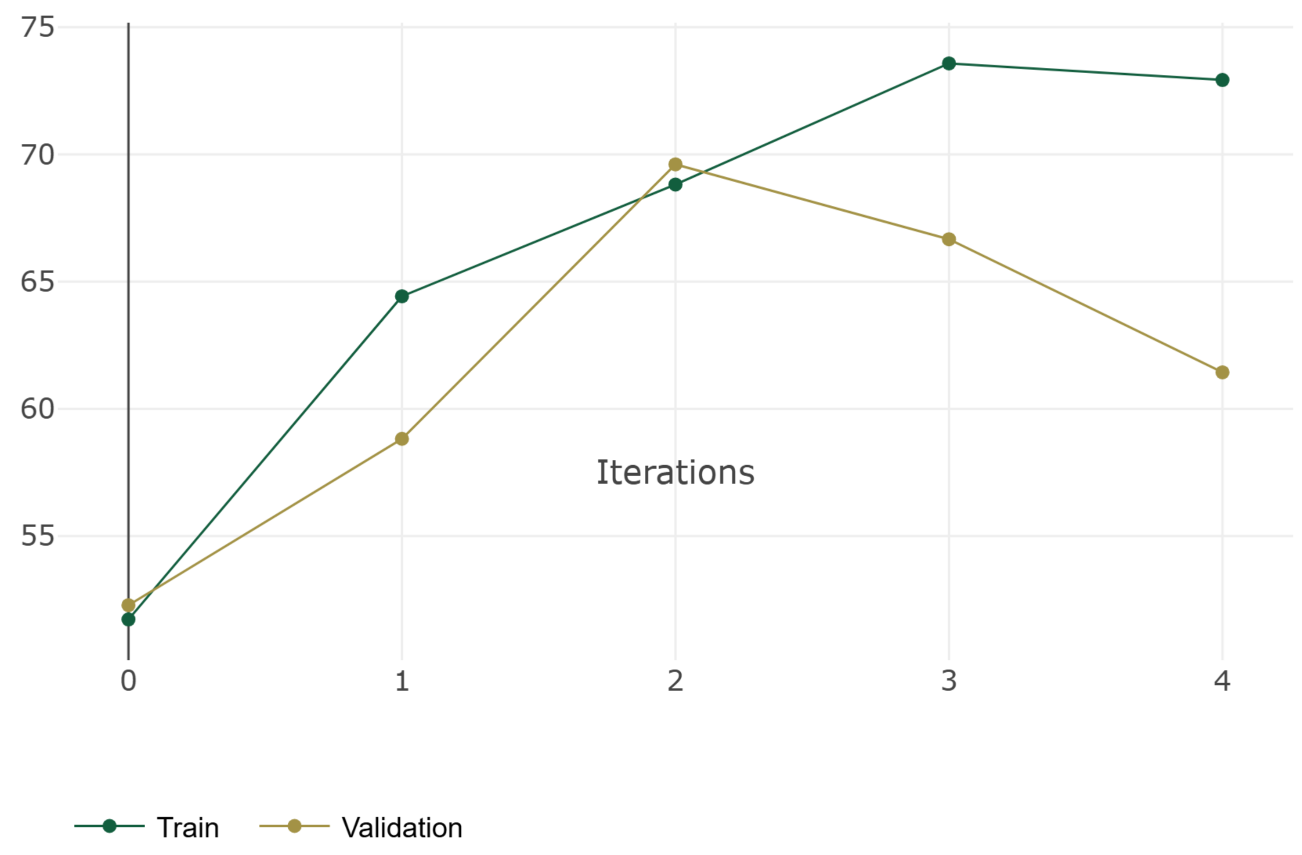
Fixed Hyperparameters (Given by ClearML After Tuning):

1. **Number of Layers:** 7
2. **Number of Filters:** 27
3. **Learning Rate:** 1.22
4. **Momentum:** 0.82
5. **Weight Decay:** 0.1
6. **Learning Rate Decay (γ\gammaγ):** Log-uniform distribution from 1e-4 to 1e-1

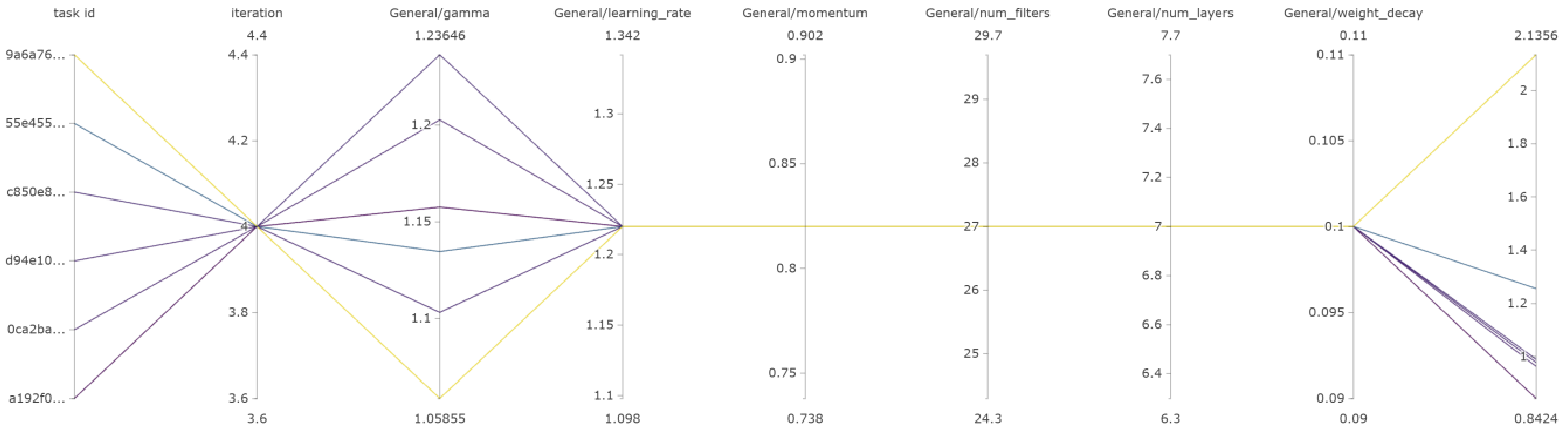
Results: Parallel Coordinates for Hyperparameter Tuning without Learning Rate decay



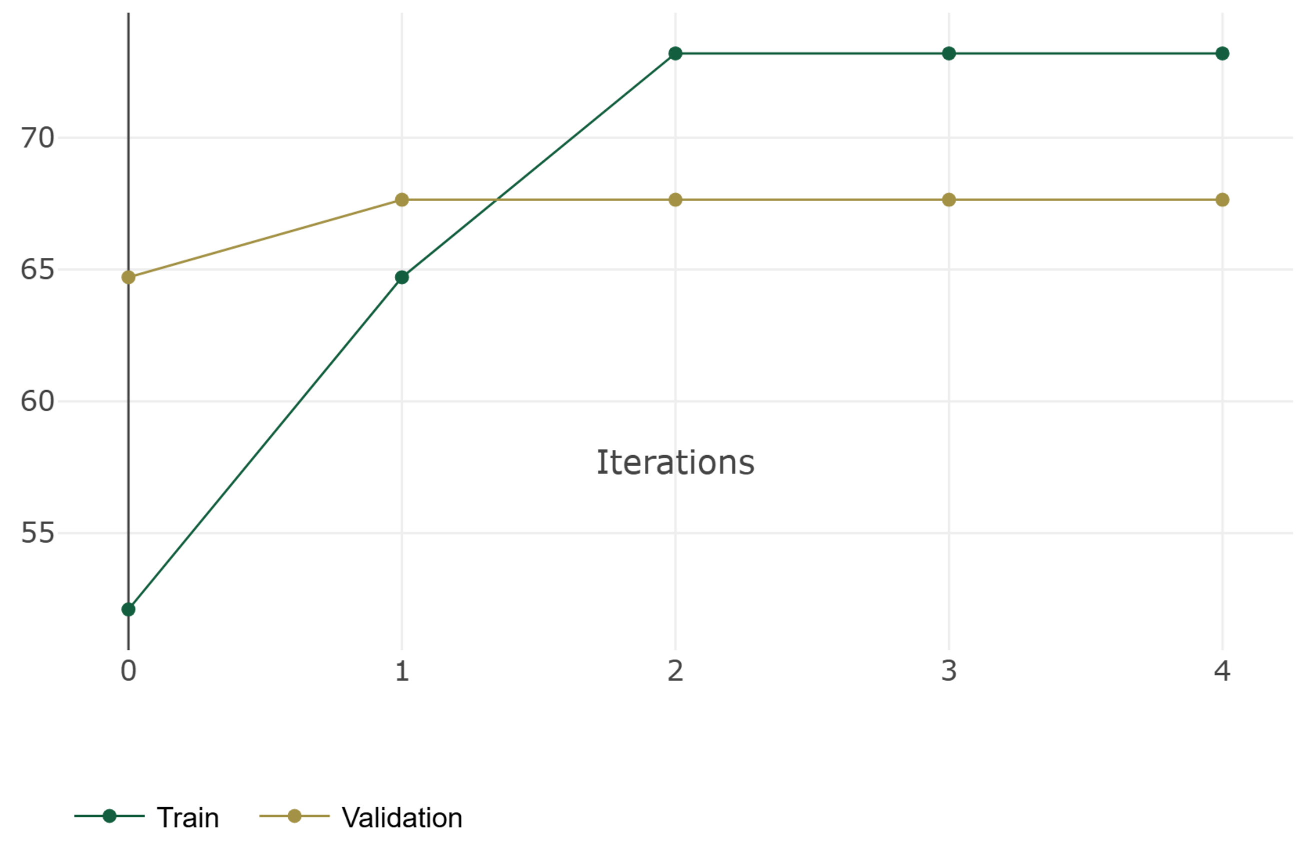
Train and Validation accuracies for the optimal Hyperparameters without Learning Rate decay



Parallel Coordinates for Hyperparameter Tuning with Learning Rate decay



Train and Validation accuracies for the optimal Hyperparameter s with Learning Rate decay



Transfer Learning

•**Model:** Pre-trained ResNet18 (ImageNet weights).
•**Task:** Weather Classification (5 classes: Rainy, Sunrise, Cloudy, Foggy, Shine).
•**Modifications:** Replaced the fully connected (fc) layer for 5 output classes.
•**Optimizer:** AdamW (lr=0.0001)
•**Loss Function:** CrossEntropyLoss.
•**Data Partitioning:**
•Training: 70%
•Validation: 20%
•Testing: 10%
•**Epochs:** 5

Resnet18 Results

- **Training Performance:**
  - Train Accuracy: 94.95%
  - Train Loss: 0.2114 (Epoch 5)
- **Validation Performance:**
  - Val Accuracy: 95.33%
  - Val Loss: 0.1396 (Epoch 5)
- **Test Performance:**
  - Test Accuracy: **96.00%**
  - Test Loss: **0.1089**
  - Precision: **0.9606**
  - Recall: **0.96**
  - F1 Score: **0.9601**

Transfer Learning vs Fully trained NN

Metric	Transfer Learning (ResNet18)	Fully Trained NN
Test Loss	0.10	Higher
Test Accuracy	96.0%	Lower (Typically ~83-85%)
Training Accuracy	80.0%	Higher (~92-94%)
Validation Accuracy	94.0%	Lower (~85-90%)
Generalization	Excellent	Decent
Training Time	Shorter	Longer (Due to training from scratch)

Key Observations

1. **Transfer Learning (ResNet18):**
   1. Achieved superior **test accuracy (96.0%)** and **validation accuracy (94.0%)**, demonstrating excellent generalization.
   2. **Training accuracy (80%)** was lower than validation/test accuracies, indicating robust regularization or better alignment with unseen data.
   3. Significantly reduced training time due to leveraging pre-trained weights.
2. **Fully Trained NN:**
   1. Higher **training accuracy (92-94%)** but lower validation and test accuracies, suggesting slight overfitting.
   2. Required longer training time due to learning features from scratch.
   3. Validation accuracy (~85-90%) was decent but fell short compared to transfer learning.

Conclusion

Transfer Learning: Outperformed the fully trained model in terms of accuracy, generalization, and training efficiency.

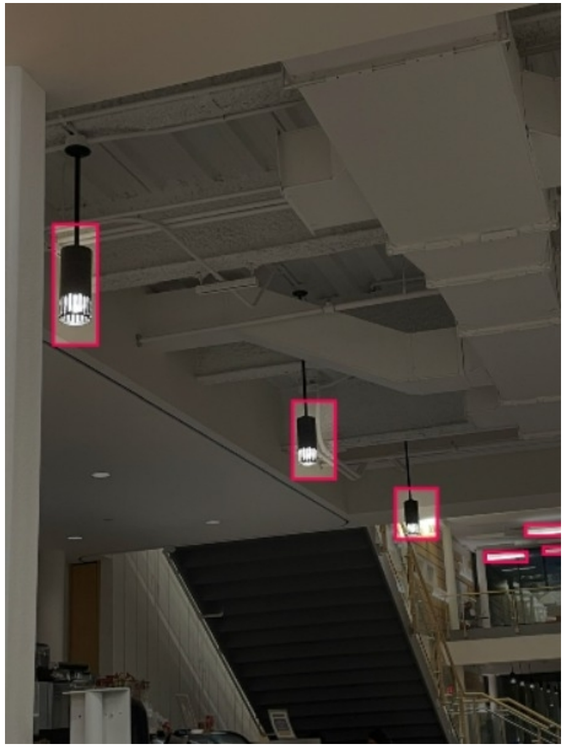
Fully trained NN showed promise but was less effective, particularly in generalization to unseen data.

Project 2: Object Detection Using YOLOv5s

Data Collection and processing

- **Task:** Object detection for **Trees** and **Lights**.
- **Dataset Link :** https://unhnewhaven-my.sharepoint.com/:u:/g/personal/vkukk2_unh_newhaven_edu/EZZFdjj-EpVBgcMQQjYsOBIBwcal04dKjyDX-qBPJGaFEg?e=T6j2Zp
- **Number of Samples:** 238 images (601 after augmentation).
- **Image Size:** 640*640*3
- **Classes:** 2 (Trees, Lights).
- **Partitioning:**
  - Training: 80% (190 images)
  - Validation: 10% (24 images)
  - Test: 10% (24 images).
- **Normalization:** ImageNet mean [0.485,0.456,0.406] and std [0.229,0.224,0.225].
- **Augmentation:** Random 15-degree flips.

Samples Images with annotations



Lights with bounding boxes

Trees with bounding boxes

Trees, Lights with bounding boxes

Data Structure

Annotation Format: Class ID: Numeric label (0 for Tree, 1 for Light).

- Bounding Box: (xcenter, ycenter, w, h), normalized between 0 and 1.
- Image Tensor: A 3D tensor of shape (640*640*3) representing the RGB image.

Image annotations:

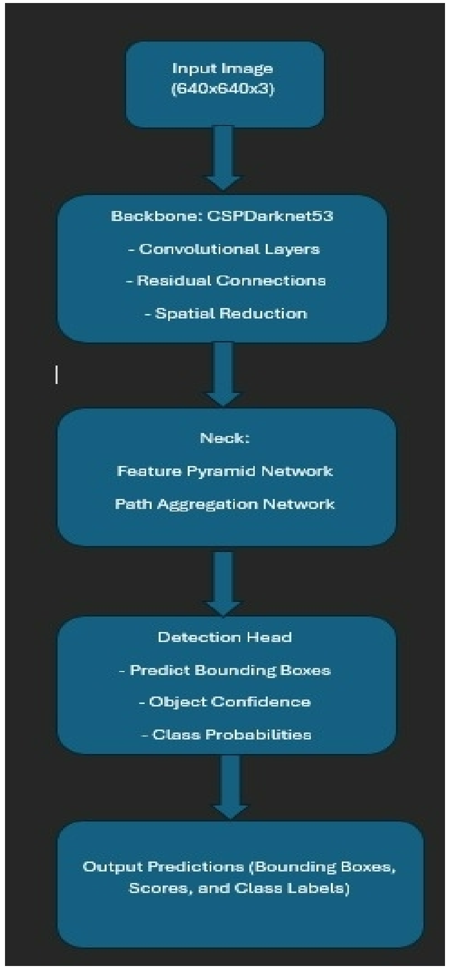
- A list of dictionaries where each dictionary contains:

class_id: The numeric label for the class ("Trees" or "Lights").

bbox: The bounding box in YOLO format (x_center, y_center, width, height), all values normalized between 0 and 1 relative to the image dimensions.

NN

- **Base Model:** YOLOv5s
  Backbone: CSPDarknet53 (Feature Extraction).
  Neck: PANet (Feature Aggregation).
  Head: Bounding Box and Class Predictions.
- **Parameters:** ~7M.
- **Loss Function:**
  Classification Loss: BCEWithLogitsLoss.
  Objectness Loss: Confidence-modulated BCE.
  Bounding Box Loss: CIoU Loss

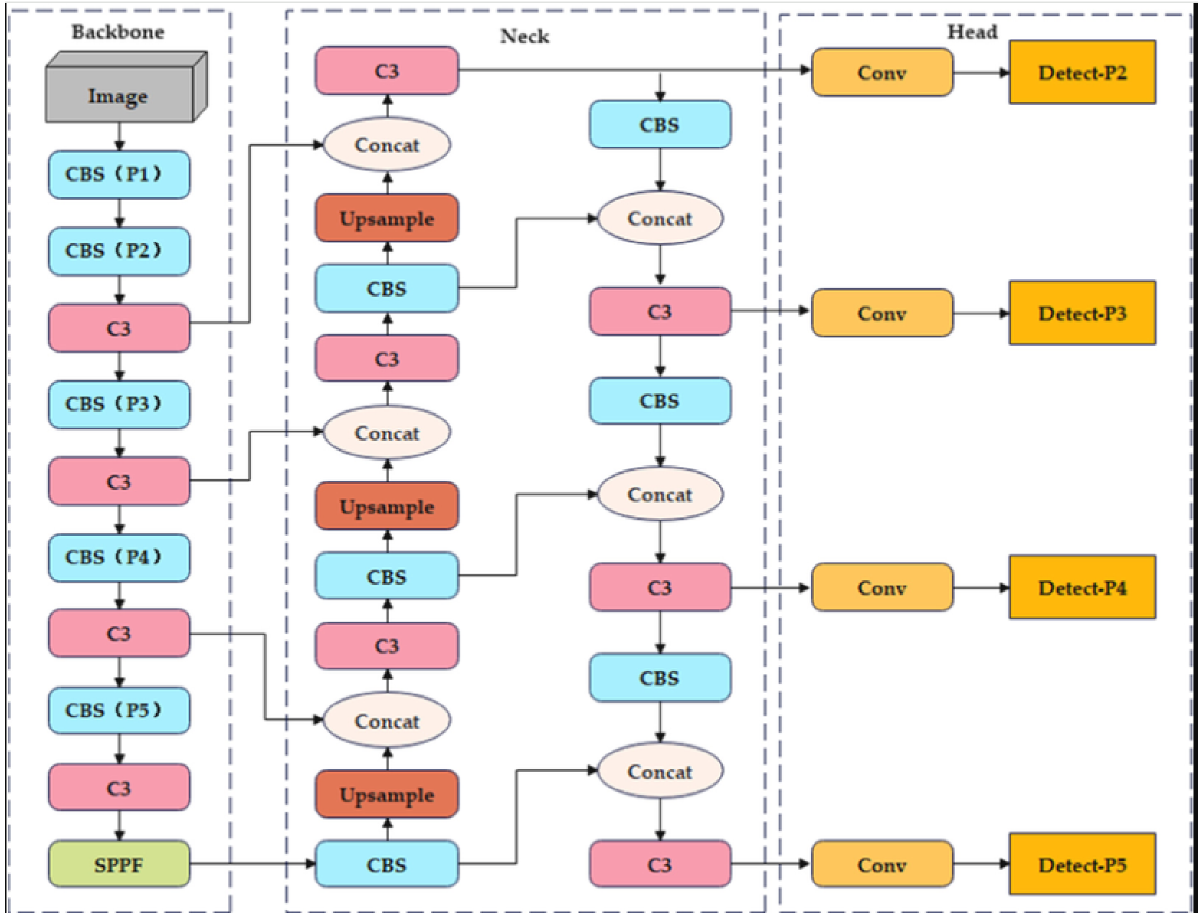


NN model from pytorch

```
                 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     18879  models.yolo.Detect                      [2, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198
YAML Configuration:
```

NN

Architecture of YOLOv5s model.



Loss Function

The model uses a combination of:

• **Classification Loss**: BCEWithLogitsLoss with optional Focal Loss for handling class imbalance.

• **Objectness Loss**: BCEWithLogitsLoss modulated by object confidence.

• **Bounding Box Loss**: CIoU loss to evaluate bounding box alignment.

The total loss is a weighted sum of these components for balanced optimization

```
# Regression
pxy = pxy.sigmoid() * 2 - 0.5
pwh = (pwh.sigmoid() * 2) ** 2 * anchors[i]
pbox = torch.cat((pxy, pwh), 1)  # predicted box
iou = bbox_iou(pbox, tbox[i], CIoU=True).squeeze()  # iou(prediction, target)
lbox += (1.0 - iou).mean()  # iou loss

# Objectness
iou = iou.detach().clamp(0).type(tobj.dtype)
if self.sort_obj_iou:
    j = iou.argsort()
    b, a, gj, gi, iou = b[j], a[j], gj[j], gi[j], iou[j]
if self.gr < 1:
    iou = (1.0 - self.gr) + self.gr * iou
tobj[b, a, gj, gi] = iou  # iou ratio

# Classification
if self.nc > 1:  # cls loss (only if multiple classes)
    t = torch.full_like(pcls, self.cn, device=self.device)  # targets
    t[range(n), tcls[i]] = self.cp
    lcls += self.BCEcls(pcls, t)  # BCE
```

Transfer Learning

This is the snapshot of the code where we froze earlier part of the network.

```
# Freeze
freeze = [f"model.{x}." for x in (freeze if len(freeze) > 1 else range(freeze[0]))]  # layers to freeze
for k, v in model.named_parameters():
    v.requires_grad = True  # train all layers
    # v.register_hook(lambda x: torch.nan_to_num(x))  # NaN to 0 (commented for erratic training results)
    if any(x in k for x in freeze):
        LOGGER.info(f"freezing {k}")
        v.requires_grad = False
```

Training

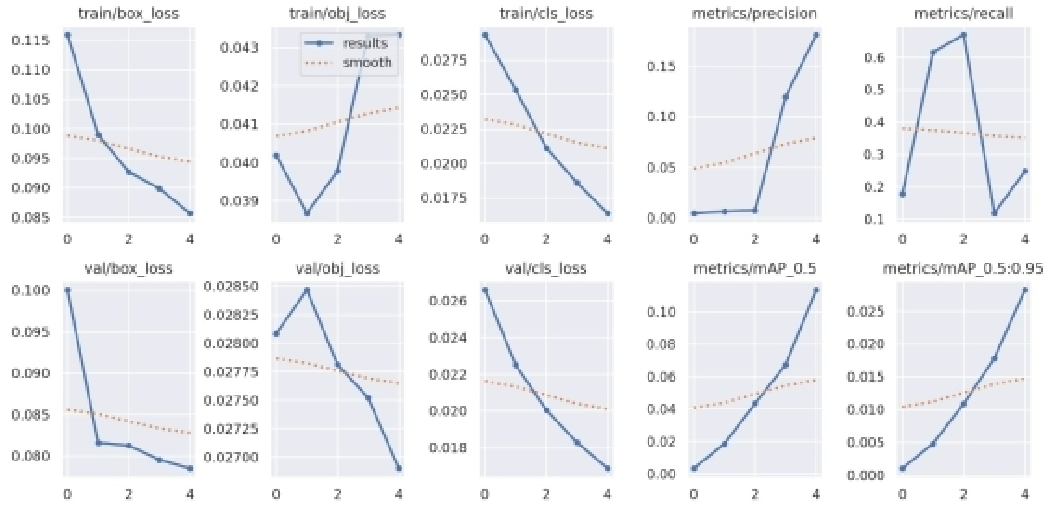
- **Mini-Batch Size:** 8.
- **Optimization Algorithm:** Adam.
  - Learning Rate: 0.01.
  - Weight Decay: $1 \times 10{-}4$.
- **Epochs:** Evaluated at 5, 10, 15, and 20.
- **Hyperparameter Search:**
  - Freezing Layers: 5, 10, 15, 18.

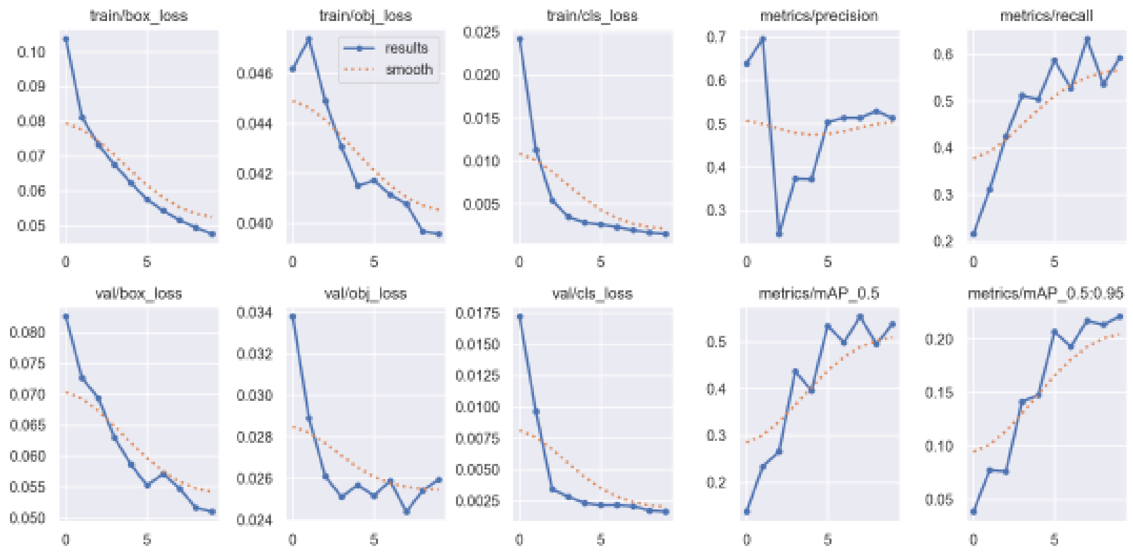
Pre and post training comparison

Metric	Pre-Tuning	Post-Training
mAP@0.5	0.539	0.671
mAP@0.5:0.95	0.221	0.323
Precision	0.513	0.658
Recall	0.591	0.696

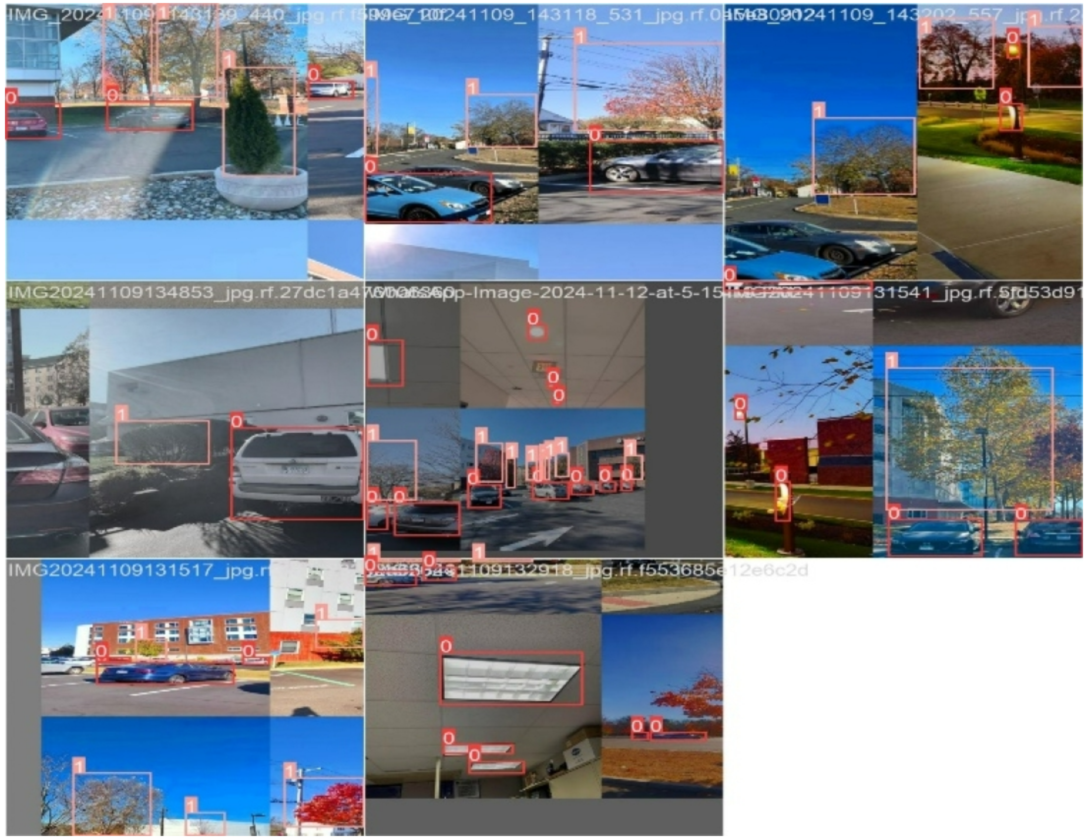
- **Metrics (Freeze=5):**
    - **Precision:** Increased from 51.3% to 65.8%.
    - **Recall:** Increased from 59.1% to 69.6%.
    - **mAP@0.5:** Increased from 53.9% to 67.1%.
    - **mAP@0.5:0.95:** Increased from 22.1% to 32.3%.

Loss Function Curves





Pre and post training comparison



Pre-Training Predictions

Post Training Predictions

Pre and post training comparison

Comparison:

- Pre-Training: The Model has Predicted inaccurately and few bounding boxes are misaligned.
- Post Training: Accurate bounding boxes around Trees and Lights.

Key Findings: Best performance with **Freeze=5** and **15 epochs**.

- Trees were detected more accurately (mAP@0.5=84.7%) than Lights (mAP@0.5=50.1%).
- Minimal freezing preserved gradient flow for better feature learning.

Thankyou!