DATA255

Part 2

1. **Gradient Descent**

   Gradient Descent is a method for finding the optimal settings in a model by gradually minimizing its errors. Imagine it like walking down a slope to the lowest point, where each step is guided by the slope's direction, called the gradient. In machine learning, this "low point" represents the least error, and each adjustment, guided by a "learning rate," helps the model gradually improve its accuracy. By making these small, calculated steps, Gradient Descent fine-tunes the model's parameters to reduce errors as much as possible.

2. **List 3 regularization techniques and explain**

   Regularization techniques help models avoid overfitting, which is when a model becomes too tailored to training data and struggles with new information. Here are three main methods:

   L1 Regularization: Adds a penalty based on the absolute values of weights, pushing some to zero, which can simplify the model by removing certain features.

   L2 Regularization: Also penalizes large weights but shrinks them gradually instead of setting them to zero, creating a smoother model less prone to overreacting to minor data variations.

   Dropout: This approach temporarily deactivates random neurons during training, compelling the model to rely on multiple pathways rather than depending on any single one, making the model more adaptable and versatile.

3. **Activation functions**

Activation functions are essential in neural networks, allowing neurons to decide if they should activate based on incoming data, enabling complex data patterns. Common activation functions include:

ReLU (Rectified Linear Unit): Allows positive inputs to pass through while setting negative ones to zero, which helps speed up learning and avoids common issues like vanishing gradients.

Sigmoid: Maps values between 0 and 1, which is helpful for binary outcomes, though it can reduce gradient strength in deeper networks.

Softmax: Often used in the final layer for multi-class tasks, it turns outputs into probabilities, making it easier to classify data into distinct categories.

4. **Loss function and Back Propagation**

Loss Function:

This is one of the important part in Neural network, providing an estimate of the difference between what the model has predicted and what has actually come to pass. More precisely, the loss function calculates an error or "loss" when a model makes a certain prediction.

During training, the objective is to minimize this loss, so the model can make more accurate predictions. Different loss functions are used based on the type of task. For example, Mean Squared Error (MSE) is commonly applied in regression, where predictions are continuous, while Cross-Entropy Loss is frequently used in classification problems where outcomes fall into distinct categories.

The loss function essentially provides feedback, guiding the model on how to adjust itself to improve accuracy. This feedback loop, where the model repeatedly checks and reduces its errors, is core to learning. An optimizer like Gradient Descent usually drives this adjustment process, tuning the model's parameters with each training pass. As the model continues to learn, these small adjustments gradually reduce the loss, helping the model improve its predictions and adapt better to new data.

Back Propagation:

The Back propagation refers to the process by which neural networks modify their inner parameters in order to minimize the error. After the model has calculated the loss, backpropagation works backward through each layer of the network, computing the contribution of each of those weights to the total error. From a mathematical standpoint, while an algorithm computes the gradient or rate of change of the loss with regard to each weight, it comes up with the quantity that says how much each weight should be changed with respect to improving the model's accuracy.

This backward pass is essential because it guides the model on how to improve over time. With the help of an optimizer, the model uses these gradients to update the weights, gradually decreasing the error over multiple training rounds, or epochs. By iterating through this process, the model gets closer to an optimal set of weights that minimize errors. Combined with the loss function and optimizers like Gradient Descent, backpropagation is what allows neural networks to learn intricate patterns and generalize well to new data.

5. **Epochs, Iterations, and Batch size**

Epochs:

An epoch signifies one full cycle through the entire training dataset during the training phase of a machine learning model. In essence, it means the model has had the opportunity to examine all the training data once. Training typically involves multiple epochs because a single pass is usually insufficient for the model to grasp the underlying patterns in the data. Throughout each epoch, the model refines its weights based on the loss calculated from its predictions on the training set, progressively enhancing its performance.

Iterations

An iteration is a single update of the model's parameters based on some subset of the training data. That means, it defines updating the model parameters after going through one batch of data samples sequentially from the dataset split into smaller groups or batches. The number of iterations is given as the total number of training samples divided by batch size. For instance, if there are 1,000 training samples and a batch size of 100, then it would take 10 iterations to go through one complete epoch. In each of these iterations, the model gets to update its weights and move down the loss function based on what it has learned from the data included in that specific batch.

Batch size

Batch size refers to the quantity of training samples processed in a single iteration when updating the model's weights. Instead of using the entire dataset at once, which can be computationally intensive, the data is divided into smaller batches. Smaller batch sizes can lead to more frequent updates to the model's parameters, which may help achieve

faster convergence but could introduce more variability in the updates. On the other hand, larger batch sizes tend to provide more consistent and stable gradient estimates but may require additional memory and longer processing times. Selecting the appropriate batch size is essential as it influences the model's training efficiency, speed, and overall performance.

Part 3

1. **Explain the differences between object detection, image classification, and image segmentation. (3 points)**

The easiest task of the three is image classification. In this task, a single label is assigned to the whole image, based on the contents of the image. While training, the model focuses on predicting the overall category for the image, without bothering with the location or the quantity of objects in it. Because it's a task of selecting primarily major objects or images from the given picture, it is therefore of high level and does not bear any spatial knowledge about objects.

A usual example of common image classification would be training a model that recognizes animals. If you feed it an image containing a dog, everything gets labeled as "dog." Similarly, if it is fed an image of a car, the model will mark that as "car." The critical relevance of Image Classification here is that the model looks at the whole image and associates only one label that captures what its dominant content is. For instance, a picture of a cat lying on a couch would have the label as only "cat," though there are many objects in this scene. The task becomes helpful in applications such as photo tagging since the objects identified fall into the broad classes.

detection goes a step further in that also locates them. It achieves this by drawing bounding boxes around each object that it detects, along with assigning a class label to each one. This allows the model to detect multiple objects in a single image and determine their exact positions within it. Object detection is an important area of interest in applications that require real-time recognition of multiple entities, such as autonomous vehicles and video surveillance.

It detects not only the mere presence of objects but locates them, too. Imagine analyzing a street scene with cars, pedestrians, and traffic lights. The object detection model would draw bounding boxes around each car, each person, and all the lights, labeling each appropriately. If an image contains five cars and two people, it identifies and locates each of those separately. That is, object detection is essential when the identity and location of multiple objects are to be known, for example, self-driving cars that may have the ability to detect pedestrians, other vehicles, and road signs in real-time to make proper driving decisions. The finest and most detailed task is that of image segmentation. Each pixel in the image is labeled as the class to which it belongs, thus giving a mask that describes the outline of each object or each region. That means in semantic segmentation, each pixel is assigned a class, but different instances of the same class are not distinguished from one another. For instance, all dogs in an image would be identified as "dog," regardless of how many dogs there are. On the other hand, the instance segmentation segments different instances of single objects of a given type uniquely by labeling each of the instances differently. This makes segmentation, because of its nature at the level of understanding per pixel, perfect for applications that require clear object boundaries, such as medical imaging or autonomous navigation.

Examples of image segmentation: Medical scan. It can classify every pixel in a scan to find tumors if that is the goal. Semantic segmentation will label pixels in this example that belong to

tumors, healthy brain tissue, and the background. Every pixel that belongs to a tumor will be marked, but in cases of multiple tumors, it will not tell the difference. This, in turn, means instance segmentation would find tumors but would also further extend this and distinguish them by giving each a number. This therefore would be important in applications requiring fine objects' outlines, for example, in medical diagnosis or self-governing drones needing to recognize and navigate around obstacles at a fine-grained level.

These latter tasks increase in complexity from image classification to image segmentation, in that order, in terms of the level of detail associated with visual understanding. Each serves different use cases, whether we need broad categorization, object localization, or pixel-level precision.

2. **Explain the architectures and differences between R-CNN, Fast R-CNN, and Faster R-CNN. (4 points)**

R-CNN (Region-based Convolutional Neural Network) was one of the first significant object detection breakthroughs. This architecture works by generating a set of potential object regions in an image, and it mainly proposes regions using a technique called Selective Search. After generating the region proposals, the model crops those regions and passes each through a CNN to extract features. It subsequently takes the extracted features to classify the object class into a classifier, such as an SVM, and a regressor that refines the bounding box coordinates. Even though R-CNN achieved high accuracy, it was computationally expensive because it had to process each proposal independently; thus, returning very slow inference speeds. For example, for a model with 2000 region proposals in an image, it has to run CNN 2000 times; hence, it is very inefficient for real-time applications.

Fast R-CNN was actually introduced because of the slow speed of R-CNN. It makes the whole image run through the CNN once and then processes each region proposal. It thus uses a network for feature extraction, which could be VGG or ResNet, to compute a feature map for the whole image. Projects region proposals onto this feature map From this, an RoI pooling layer is used for extracting a fixed-size feature vector for each region These feature vectors were used for classifying objects and refining the bounding boxes This architecture is much faster compared to R-CNN mainly due to avoiding repeated passing of the same region proposals through the CNN. For instance, in case an image has 2000 regions, it runs CNN once for the entire image; thus, Fast R-CNN is way faster.

Faster R-CNN further improves the approach of Fast R-CNN by avoiding an external region proposal algorithm such as Selective Search. Faster R-CNN integrates a Region Proposal Network, or simply RPN, into the neural network. The RPN shares convolutional layers with the feature extraction network and predicts potential object regions directly from the feature maps. The entire process, from proposal of regions to object detection, can now be dealt with under a unified framework. Subsequent steps in this approach include RoI pooling, classification, and bounding-box regression, similar to those in Fast R-CNN, given that the proposals have been generated from the RPN. Faster R-CNN is more efficient and hence appropriate for real-time applications by streamlining the processes of proposals generation and object detection. This allows it, for example, to detect pedestrians, vehicles, and traffic signs faster in autonomous vehicles from real-time video feeds by eliminating bottlenecks with slow proposal methods in Faster R-CNN.

Comparison:

Feature	R-CNN	Fast R-CNN	Faster R-CNN
Region Proposal Method	Selective Search (external, slow)	Selective Search (external, slow)	Region Proposal Network (RPN) (integrated)
Feature Extraction	CNN applied individually on each region proposal	CNN applied once on the whole image	CNN applied once on the whole image
Region of Interest (RoI) Processing	Cropped image regions passed through CNN	RoI pooling extracts feature maps from the CNN	RoI pooling extracts feature maps from the CNN
Speed	Slow (requires running CNN for each region proposal)	Faster than R-CNN (single CNN pass for entire image)	Fastest (integrated RPN for faster region proposal generation)
Object Classification	SVM (Support Vector Machine)	Fully connected layers (softmax for classification)	Fully connected layers (softmax for classification)
Bounding Box Regression	Separate regression for bounding box	Integrated with classification task	Integrated with classification task
End-to-End Training	No (requires multiple training steps)	Yes	Yes
Inference Time	Slowest (due to multiple CNN evaluations)	Faster (one CNN evaluation)	Fastest (one CNN evaluation + RPN)
Use Case Example	Non-real-time applications like static image analysis	Faster object detection, but not real-time	Real-time object detection (e.g., autonomous vehicles, surveillance)

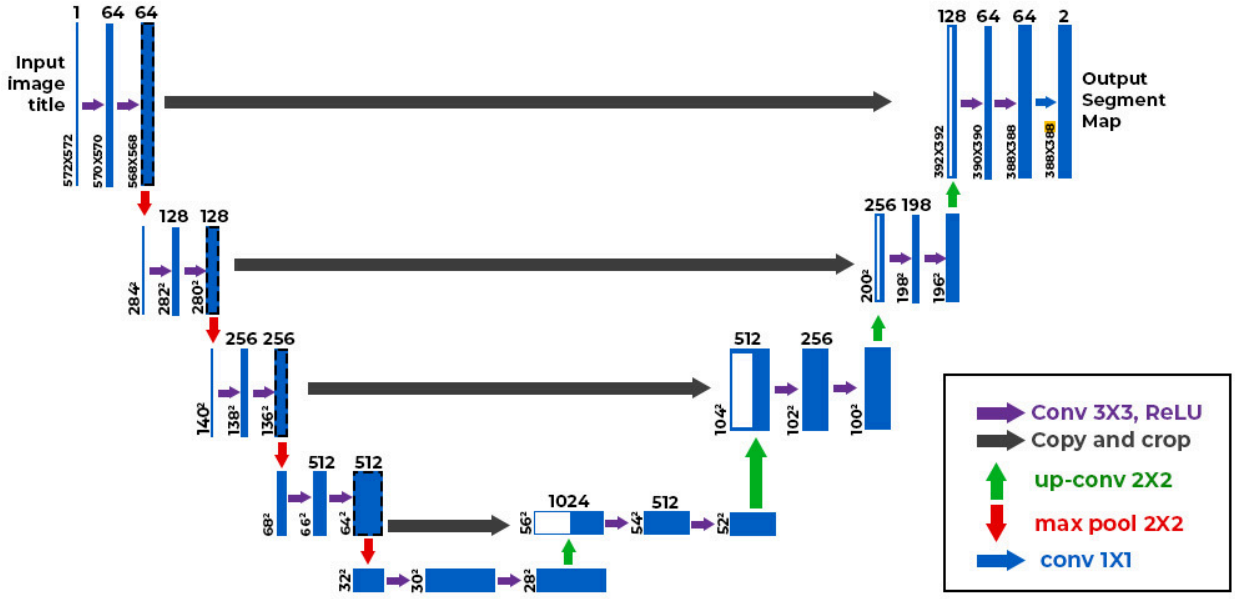
These developments have given a hint towards the trend for developing faster and more efficient object detection systems, and Faster R-CNN finds broad applications in real-time object detection scenarios.

3. **Explain what's U-net. (5 points)**

The U-Net is a CNN architecture that has been principally developed for image segmentation. It was originally developed for biomedical image segmentation but then found wide usage in the

most diverse fields. The main idea of U-Net is a pixel-level classification, meaning that each pixel in an input image is classified for the purpose of object or region identification. Furthermore, it is ideal for tasks where the exact boundaries of an object need to be determined, such as in medical imaging, satellite image analysis, or autonomous driving.

Architecture



Architecture U-Net is U-shaped, hence the reason behind its naming. U-Net has a contracting path and an expanding path. In the contraction path, it is much similar to any other typical CNN, with convolutional layers and max-pooling layers. Each one of them downsamples the input image in space while capturing higher-level features. The network learns from the abstract representations at higher levels of granularity of the image at this stage and can hence recognize even complex structures present in the input.

What makes U-Net distinctive is the expanding path, symmetric to the contraction path, which restores the spatial resolution of the image. First, the network uses transposed convolutions-or

upsampling layers-in the expanding path to recover the spatial dimensions progressively and get image details lost during downsampling. One key characteristic feature of U-Net is the skip connections between corresponding layers in the contracting path and expanding path. These connections thus allow the network to combine high-level information from the contracting path along with the fine details from the earlier layers in improving the model's ability for better segmentation.

Skip connections are going to be crucial to the performance of U-Net. The spatial information, which will get lost in the deeper layers of the network, will be preserved by them. Without these skip connections, the upsampling process may not recover the subtler details effectively. It yields a model that generates detailed segmentation maps even to pixel-level discrimination of objects. Examples include medical imaging, where U-Net will segment individual cells or tissues with a high degree of precision. Hence, it finds wide applications in tumor detection or organ segmentation.

Eventually, the power of the U-Net includes the efficiency of training with rather limited datasets, which is extremely important in biomedical imaging, for example, considering serious limitations that exist in obtaining labeled data. This is achieved through the practice of data augmentation, creating artificial variations of training images, hence improving the generalization capability of the model. Overall, versatility and accuracy have made U-Net popular for performing segmentation tasks in many industries.

Use Cases:

U-Net has been used in medical imaging extensively, from tumor segmentation and organ delineation to cell tracking. Its ability to do a pixel-level segmentation makes it perfect for the detection of cancerous tissues from MRI or CT scans, thus helping clinicians in diagnosis and treatment planning. Beyond health care, this architecture also plays a very crucial role in autonomous driving, segmenting driving scenes to help self-driving cars identify roads, pedestrians, and obstacles accurately. It enables a vehicle to possess much-enhanced safety and efficiency.

Further, U-Net finds its application in the analysis of satellite images for land use classification, building detection, and monitoring environmental changes. The high accuracy with which it segments aerial images assists greatly in urban planning, deforestation tracking, and disaster response. In manufacturing, U-Net automates defect detection, increasing quality control, while in agriculture, U-Net helps farmers detect diseased crops or estimate yields from drone and satellite images to optimize resource utilization and improve their productivity.

4. **List at least 3 widely used metrics in the object detection industry and explain them in detail. (3 points)**

1. Intersection over Union (IoU):

IoU is one of the most usable metrics amongst all the metrics involved in object detection that give us an idea of how well our predicted bounding box matches the ground truth. IoU between the Predicted bounding box and Actual object's bounding box gives the ratio between intersection area and union area. It can be mathematically written as:

IOU = Area of Overlap/ Area of Union

It ranges from 0 to 1, where 1 means that the predicted box perfectly matched the ground truth. In real scenarios, IoU ≥ 0.5 or some other threshold is considered detection for a true positive. IoU is crucial in this case because it does not only account for the fact of whether the object was detected but also how good the localization was. That is, the higher IoU means that the predicted bounding box is very close to the true object location.

2. Mean Average Precision (mAP):

The most commonly used metric is Mean Average Precision, shortly mAP for Object Detection Models performance evaluation; indeed, it synthesizes precision and recall at various IoU thresholds into one balanced number expressing the capability of the model to detect objects. Precision measures how many of the objects predicted are true, whereas Recall measures how many of the actual objects are found. AP is calculated to quantify the area under the Precision-Recall curve for each class, then the average of AP values over all classes gives the mAP. mAP for a certain IoU threshold-for example, mAP@0.5-spells out the model's precision when its predictions have at least an IoU of 0.5 with ground truth. The higher the value of mAP, the better the overall performance for multiple classes and instances; it is hence an important metric in competitions and benchmarks, such as the COCO dataset, where models are commonly evaluated on many IoU thresholds.

3. F1 Score (Harmonic Mean of Precision and Recall):

The F1 Score is another common evaluation metric in object detection, especially when balancing precision and recall is important. It is the harmonic mean of precision and recall,

which gives a single score that reflects both aspects of model performance. The formula for the F1 Score is:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

For instance, with high precision in object detection, most of the predicted objects are correct, while with high recall, it would mean that most of the actual objects have been detected. The F1 Score balances both to ensure that the model doesn't skew towards just precision-where there are fewer false positives-or recall, where more detections were returned. It works best for situations when classes of objects are imbalanced, or the cost of FPs and FNs is close.

5. **Explain what's Non-Maximum Suppression and how it works.**

Non-Maximum Suppression, or NMS for short, plays an important role in the post-processing technique of object detection. It removes duplicate bounding boxes and retains only the ones most correct. In object detection, the models usually adopt predicting multiple bounding boxes on the same object commonly with YOLO or Faster R-CNN, and with different overlap levels. Although that may provide great confidence in the existence of the object, this leads to many overlapping boxes for just one object and creates clutter in the final output. NMS resolves this by reducing the number of boxes and making sure only the most confident prediction remains per object, hence enhancing clarity and precision in detection.

First, NMS ranks all the predicted bounding boxes according to their scores. Each score represents how confident the model is that that box truly contains an object of interest. It selects

the one with the highest confidence score first and marks this as the most accurate. It then calculates IoU for all other boxes that overlap with this selected box. IoU gives the extent of overlap between two boxes; if it is above the pre-set threshold-which is taken as 0.5-the overlapping boxes are considered redundant, and suppression occurs. This process keeps going, with the selection of the next highest-scoring box while IoU is calculated once again until all of the overlapped remaining boxes are kept or suppressed.

The process further involves NMS reducing the clutter by retaining only one bounding box per object with the assurance of clear and nonoverlapping predictions that are included in the final output. For example, in a scene containing several cars, without the NMS, the model might just return many overlapping boxes for every car. Therefore, the cleaning that NMS does leaves only the most confident prediction per car, hence making detection much easier to read. This technique is very important for real-world applications with autonomous driving and video surveillance since clean and non-redundant detections are essential for speed and accuracy in decision-making.

13. Submit a short report on the model you chose for Part 1 and why. Include the IOU results in the report. Discuss your observations and the hyperparameters you used for the model.

Convolutional Layers:

- Conv1: Input images are three color channels on which 32 convolutional filters of size 3x3 are applied.
- Conv2: Depth increases to 64 filters, which allows the model to learn higher-level features.

- Conv3: Further increased to more than twice the number, at 128 filters, which captures high abstract features.

2. **Pooling Layer**:
    - A dropout layer is added with a rate of 0.2 to prevent overfitting by setting a fraction of the input units to zero at random during training.
3. **Dropout**:
    - A dropout layer with a rate of 0.2 is incorporated to reduce the risk of overfitting by randomly setting a portion of the input units to zero during training.
4. **Fully Connected Layers**:
    - A fully connected layer (fc1) reshapes the flattened feature maps into a vector of size 256.
    - The model has two output layers:

      Classification Output (fc_class): Produces class probabilities for the 25 road sign categories.

      Bounding Box Output (fc_bbox): Provides the predicted coordinates for the bounding boxes surrounding each detected sign.

Justification for Design Choices

The Convolutional Architecture: Because the convolutional layers learn the spatial hierarchies of the features from image data, convolutional layers have an edge in handling image classification problems over traditional fully connected neural networks.

Pooling: Max pooling helps reduce computational complexity while keeping the model invariant to small translations of the input image. This property is highly desirable for object detection tasks where the exact position of an object can vary.

Dropout Regularization: Overfitting, especially due to the relatively small size of many datasets, is being reduced by dropout to enable generalization capabilities of the model by reducing its reliance on specific neurons.

Dual Outputs for Class and Bounding Box: This architecture enables the model to learn class labels and bounding box predictions parallel, which becomes necessary for effective object detection.

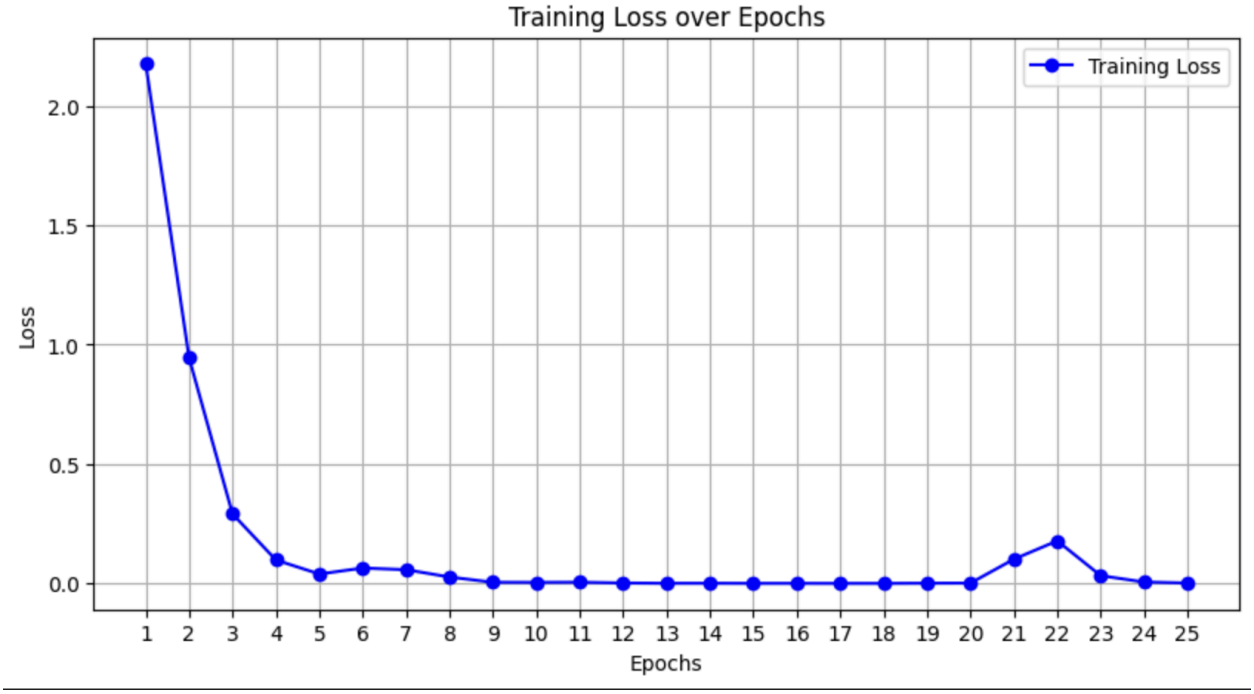
Training Approach:

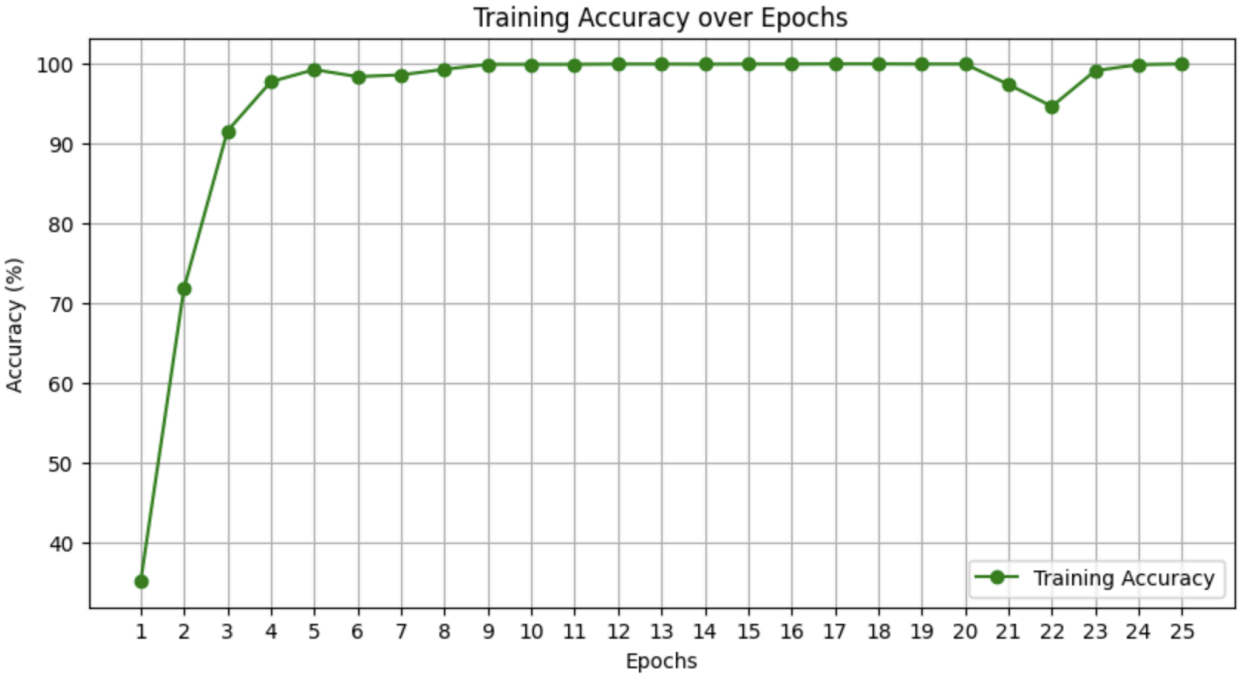
Loss Function: CrossEntropyLoss is employed for the classification output, which is quite fitting for multiclass classification problems. Efficiency and adaptive learning rate capabilities make Adam a very good choice for an optimizer.

Performance Metrics: Loss and accuracy with respect to model performance have been monitored during 25 epochs of training. This iterative feedback is key to how much a model learns.

The chosen CNN architecture is specifically designed for the road sign detection task because it efficiently extracts features, minimizes dimensionality, mitigates overfitting, and accommodates multi-class classification. By integrating convolutional layers, pooling, dropout, and fully connected layers, the model is capable of accurately identifying and locating different road signs.

This architecture enhances performance and guarantees reliability in real-world situations, where variations in image conditions can greatly affect detection accuracy.





IOU Results:

The results from the training loop show that your model achieves high IoU scores throughout training, consistently above **0.999**, which is a very good indicator that your model's bounding box predictions align very well with ground truths. Also, the primary loss decreases across epochs, meaning that it learns effectively. However, there are a few noticeable spikes in the loss, such as those in epochs 19, 20, and 21, which might indicate some batch-specific challenges-maybe hard examples or variance in the data distribution.

An IoU of more than 0.999 indicates that the model makes quite accurate predictions of bounding boxes with minimal spatial errors relative to ground truths. The gradual decrease in the primary loss across epochs-except for occasional spikes-indicates effective learning of both parts, the classifier and regression. Spike of Loss: These spikes could be due to some difficult samples of loss that hint at what a model may fail in certain data cases. Indeed, the IoU loss function provided a clear and interpretable measure about the alignment between the predicted and true bounding boxes in order to guide the model for precise localization. Overall training seems to be pretty stable despite some fluctuations and high IoU - an indicator of robust bounding box prediction toward the end of training. These results would suggest that your model will be well-suited for tasks that require precise object localization.

```
Epoch [1/25], Loss: 0.0008, IoU Loss: 0.9997
Epoch [2/25], Loss: 0.0020, IoU Loss: 0.9997
Epoch [3/25], Loss: 0.0007, IoU Loss: 0.9997
Epoch [4/25], Loss: 0.0009, IoU Loss: 0.9997
Epoch [5/25], Loss: 0.0006, IoU Loss: 0.9997
Epoch [6/25], Loss: 0.0017, IoU Loss: 0.9997
Epoch [7/25], Loss: 0.0010, IoU Loss: 0.9997
Epoch [8/25], Loss: 0.0016, IoU Loss: 0.9997
Epoch [9/25], Loss: 0.0019, IoU Loss: 0.9996
Epoch [10/25], Loss: 0.0019, IoU Loss: 0.9996
Epoch [11/25], Loss: 0.0020, IoU Loss: 0.9996
Epoch [12/25], Loss: 0.0012, IoU Loss: 0.9996
Epoch [13/25], Loss: 0.0006, IoU Loss: 0.9997
Epoch [14/25], Loss: 0.0014, IoU Loss: 0.9996
Epoch [15/25], Loss: 0.0010, IoU Loss: 0.9996
Epoch [16/25], Loss: 0.0012, IoU Loss: 0.9996
Epoch [17/25], Loss: 0.0017, IoU Loss: 0.9997
Epoch [18/25], Loss: 0.0015, IoU Loss: 0.9995
Epoch [19/25], Loss: 0.1728, IoU Loss: 0.9997
Epoch [20/25], Loss: 0.0489, IoU Loss: 0.9995
Epoch [21/25], Loss: 0.0116, IoU Loss: 0.9998
Epoch [22/25], Loss: 0.0026, IoU Loss: 0.9998
Epoch [23/25], Loss: 0.0007, IoU Loss: 0.9997
Epoch [24/25], Loss: 0.0006, IoU Loss: 0.9997
Epoch [25/25], Loss: 0.0004, IoU Loss: 0.9997
```

11. YOLOv8 (You Only Look Once Version 8) is the newest version released in the series of YOLO with the biggest emphasis on detecting objects with both fast pace and notable precision. The series is offered in multiple models – Nano (YOLOv8n), Small (YOLOv8s), Medium (YOLOv8m) to be some – to serve different applications and hardware capabilities. The smallest of all the models, YOLOv8n, is centered on efficiency and is the best option for real time work

in devices that have limited resources. On the other hand, YOLOv8s is fairly light in weight but achieves higher accuracy than its Nano counterpart. The most advanced of the three, YOLOv8m, is directed at applications that need high levels of accuracy and can accept the weight of additional computational requirements.

In this study, all three models of YOLOv8, i.e., YOLOv8n, YOLOv8s, YOLOv8m were trained on road sign detection data set with an aim of estimating precision, recall, mean Average precision (mAP) along with some loss metrics. As far as YOLOv8n is concerned, the model was found to achieve good results in terms of speed and consistency as it relates to its ability to perform convergence which is ideal in applications where limited computing power is available. On the other hand, Similarly, high levels of precision and recall with only a small amount of overfitting were achieved in the case of YOLOv8s which balanced accuracy and efficiency well within the target range.

Code Approach:

The code presents a detailed pipeline for training and evaluating the YOLOv8 models (both the nano and small versions on a single class road sign detection dataset and goes from the data setup stage to the fine tuning stage). The first of such steps is downloading, unzipping, and mounting the desired dataset from Google Drive, afterward, we can load a configuration file (data.yaml) which contains information regarding the classes and their respective amounts. The code also contains an EDA section, whose objective is to count the number of images and labels in the train, validation and test sets in order to check if the dataset is correct. A function ‘display_random_samples_with_class_index’ displays random images of associated class indices

to provide context for the structure of the dataset and how the visuals should look like for the model being built.

The code then proceeds to the second stage which is Training of YOLOv8 with the default configurations applied to both models, YOLOv8n and YOLOv8s. Different hyperparameters are used for training the models such as the number of epochs, batch size, learning rate, weight decay and the optimizer. The training process is different for various configurations of model size and parameters, therefore, Nano models were trained with varying configurations in order to determine the effect of changing parameters on mean Average Precision (mAP) and Intersection over Union (IoU). The fine-tuning phase investigates data augmentations, including Mosaic and MixUp, and more sophisticated setups, such as cosine learning rate scheduling with layer freezing to increase the detection accuracy, while preventing overfitting. After the training, test predictions are performed on test data and subsequently, the IoU is evaluated by using the areas of predicted and true bounding boxes.

In the end, the last procedures of the post-training include evaluation and visualization of the obtained results. Predictions are made in the YOLO format, embedded in a CSV submission file where the highest confidence prediction of each image is linked to the appropriate class. For a evalutaion, some important evaluation parameters F1 score curves, confusion matrices, and sample predictions are presented.

12. The code files are in two parts with the first file trying 3 variations of YOLOv8n models and the second model containing two variations of YOLOv8s and one default version of YOLOv8m. Following are the different configurations that were tried along with data augmentation and

transformations. For some models, patience was used for early stopping so that if the model was found to not improve, the training would stop and the next settings could be tested.

Parameters and Tuning Details:

For YOLOv8n:

Default (8n Default Parameters):

- Model: yolov8n.pt (nano version).
- Parameters:
    - Epochs: 10
    - Batch size: 16
    - Image size: 640

Fine-Tuning :

- Model: yolov8n.pt
- Modified parameters:
    - Epochs: 100
    - Batch size: 32
    - Learning rate (lr0): 0.001
    - Momentum: 0.937
    - Weight decay: 0.0005
    - Image size: 640
    - Optimizer: AdamW

Enhanced Tuning:

- Model: yolov8n.pt
- Modified parameters:

    - Epochs: 150
    - Batch size: 64
    - Learning rate (lr0): 0.0005
    - Image size: 704
    - Optimizer: AdamW
    - Augmentations: Enabled (augment=True), Cosine learning rate scheduling (cos_lr=True), Patience for early stopping: 10

For YOLOv8s:

Default (8s Default Parameters):

- Model: yolov8s.pt (small version).
- Parameters:
    - Epochs: 10
    - Batch size: 16
    - Image size: 640

Fine-Tuning (8s Experiment):

- Model: yolov8s.pt
- Modified parameters:
    - Epochs: 100
    - Batch size: 16
    - Learning rate (lr0): 0.001
    - Weight decay: 0.0001
    - Optimizer: Adam
    - Augmentations: Enabled (augment=True), Mosaic (1.0), MixUp (0.1)

    - Warmup parameters:
        - warmup_epochs: 3
        - warmup_momentum: 0.9
        - warmup_bias_lr: 0.05

For YOLOv8m:

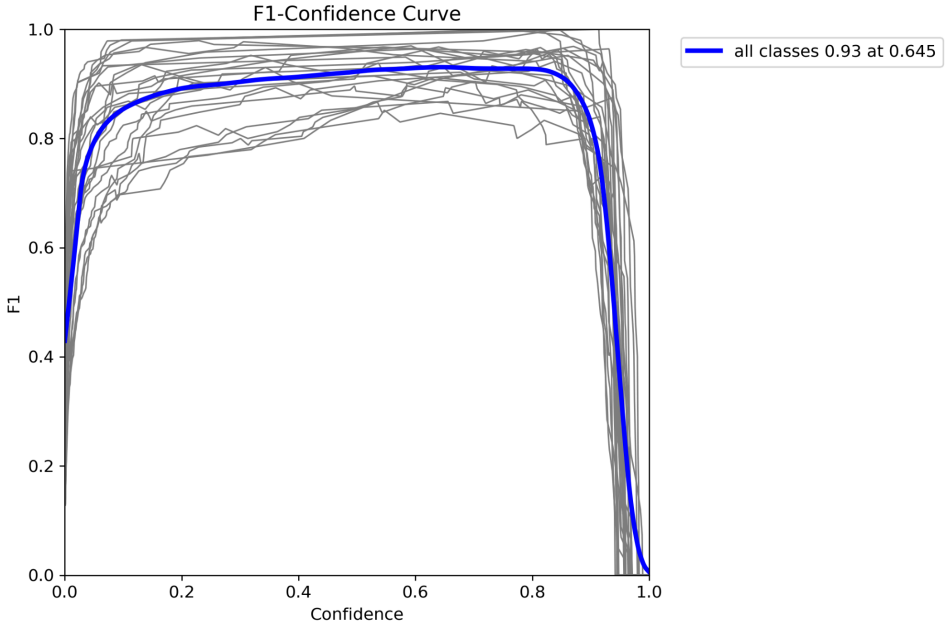
Default (8m Default Parameters):

- Model: yolov8m.pt (medium version).
- Parameters:
    - Epochs: 10
    - Batch size: 16
    - Image size: 640

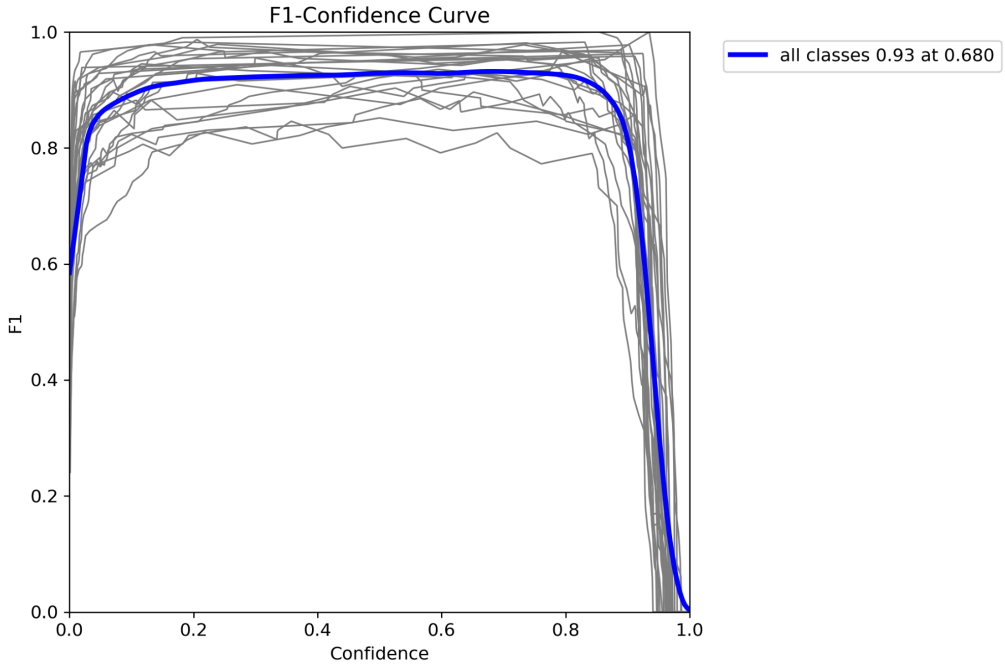
Results:

The best performing model in terms of mAP was YOLOv8m and enhanced tuning for YOLOv8n. After training the models, the IOU was calculated on the test dataset and the average was taken with 8m having 0.8562 and 8n having 0.8384 and 8s having average IoU of 0.8710. This was one factor of comparing the different models, the other factors are shown below and discussed as well.

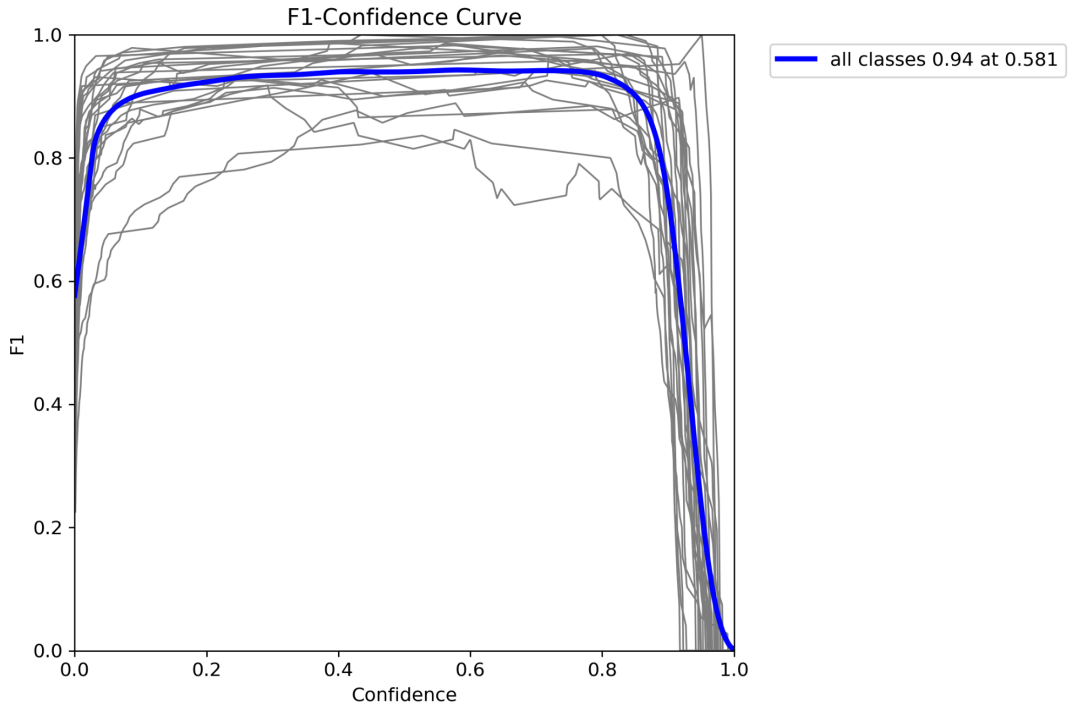
F1 Curve Graphs from the Best Models Amongst YOLOv8n, YOLOv8m and YOLOv8s



YOLOv8n - F1 Confidence Curve



YOLOv8s - F1 Confidence Curve



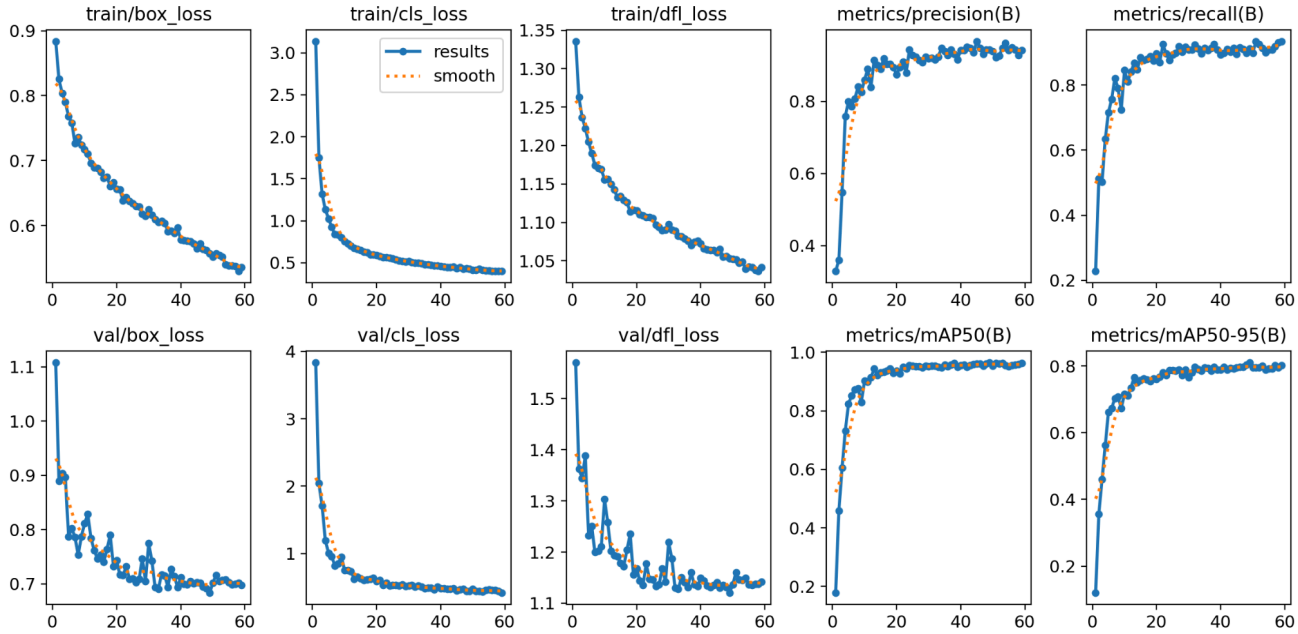
YOLOv8n - F1 Confidence Curve

As far as F1 scores are concerned with regard to the three models, YOLOv8n, YOLOv8s, and YOLOv8m although all have good performance, yet some slight variation exists. For the case of YOLOv8n, the F1 score is approximately 0.93 at a confidence margin of 0.645 meaning the accuracy is well balanced but confidence is relatively low in comparison to other models. YOLOv8s also reaches F1 scores of about 0.93 but at a confidence margin of 0.68 which means it is possible to have similar F1 scores with greater reliance towards predictions. YOLOv8m on the other hand has an F1 score of 0.94 at a confidence margin of 0.581 which depicts greater ability for the model to maintain a balance between precision and recall at low confidence margins. These findings imply that YOLOv8m might have a slightly better grasp of the key features of the data set and therefore, generate lesser misclassifications.

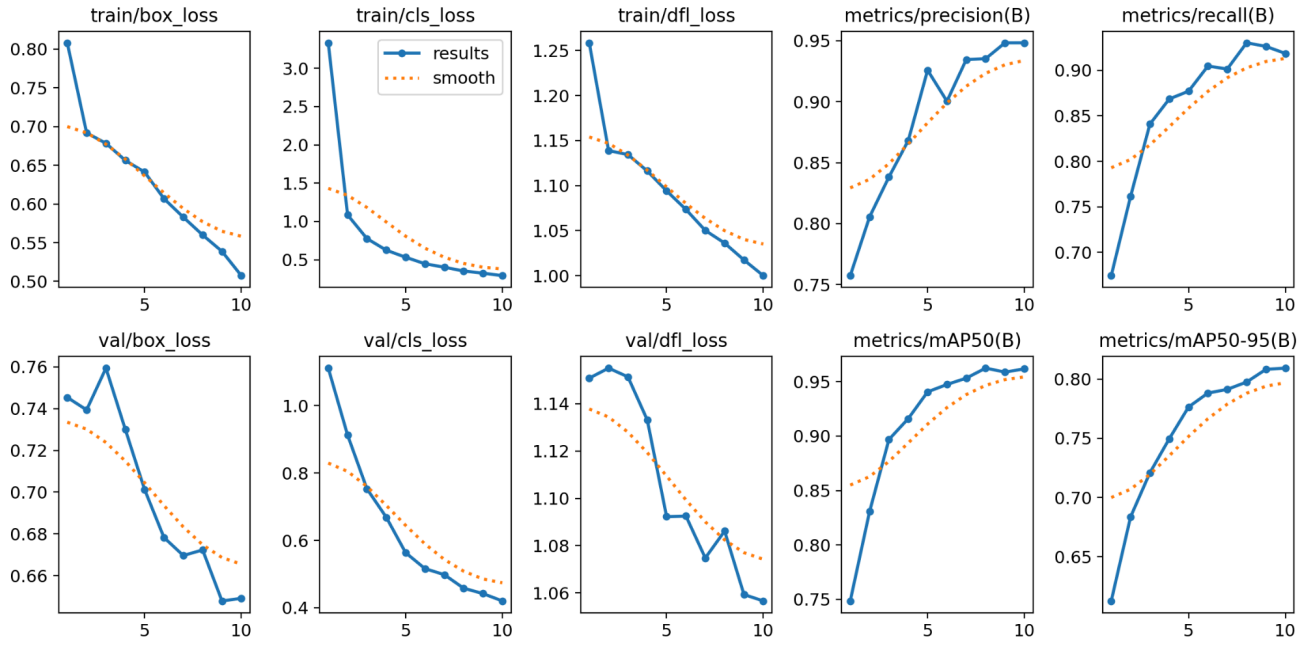
Considering these F1 scores, it is the YOLOv8m model that stands out overall in balancing accuracy and confidence, scoring the best F1 score at the optimal threshold. Following closely,

but slightly behind is YOLOv8s and YOLOv8n who also have similar F1 scores but a higher confidence threshold to achieve them. Thus, for this dataset, the hierarchy in descending order would be as follows; YOLOv8m, YOLOv8s and then YOLOv8n. Although YOLOv8m might be more beneficial in cases where accuracy is the most required, there are still potential sites for the use of YOLOv8s and YOLOv8n, especially where a faster inference time or lower computational resources are needed. In particular, YOLOv8n still remains a very resource efficient model with great performance and is thus suited for low resource environments.

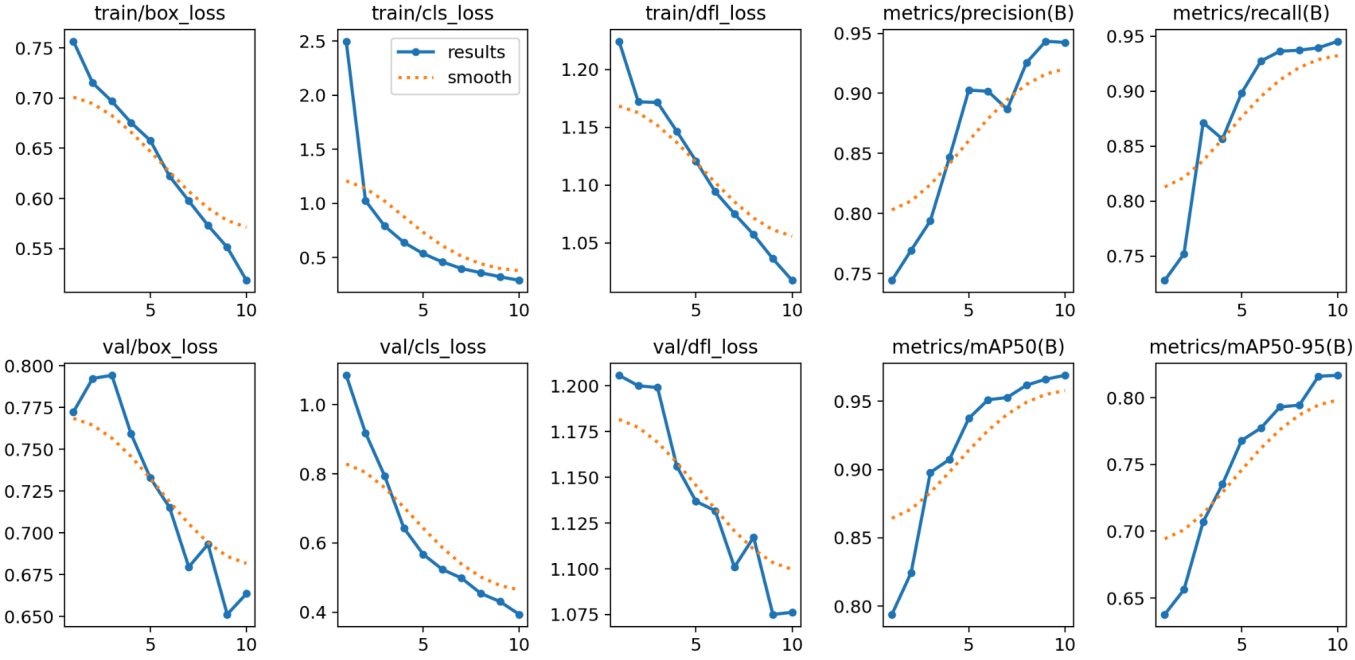
Training and Loss Metric Graphs from the Best Models Amongst YOLOv8n, YOLOv8m and YOLOv8s



YOLOv8n



YOLOv8s



YOLOv8m

In turn, we trained and validated the three models YOLOv8n, YOLOv8s, and YOLOv8m. For all the models, the loss metrics (Box loss, classification loss, and distributional focal loss) show a decreasing curve; similarly, in the precision, recall, and mAP, an uptrend is observed. But the rate of improvement and the consistency of improvements is not always the same for all of them. Being the lightest model, YOLOv8n's convergence is quite rapid as it achieves high precision and recall around the 60th epoch, however its losses are still somewhat higher than that of YOLOv8s and YOLOv8m. YOLOv8s maintains the most progressive curve; for this model, the stage of losses seems to be constantly decreasing, and there is a close calibration between the validation and training metrics, suggesting there is not much overfitting. The main point of interest concerning the training of the models is that as YOLOv8m is the largest of the three, it was able to finish with lower loss and retain the highest recall and mAP scores near the end of the training, showcasing a good fit for learning intricate features of the dataset. This refinement in YOLOv8m occurs in only 10 epochs, indicating that it does in fact converge rapidly with fewer epochs and sustains lower losses as well as attaining a high precision.

Based on these metrics, YOLOv8m stands out as the best model, with the lowest loss metrics, highest precision and recall, and robust mAP scores, indicating excellent accuracy and generalization capability. YOLOv8s ranks closely behind, with competitive precision and recall values and minimal overfitting, making it a solid choice for balanced performance in accuracy and speed. YOLOv8n, while slightly behind in terms of loss and mAP, offers good performance and quick convergence, making it ideal for scenarios where computational efficiency is prioritized over the highest possible accuracy. Thus, the ranking from best to least effective

based on training and validation metrics and most efficiency for resource computation would be YOLOv8m, YOLOv8n, and YOLOv8s.