# Lab 2: YOLO final-report

CHI YEONG HEO[1],
[1]School of Computer Science and Engineering, Pusan National University, Busan 46241 Republic of Korea

## 1. Introduction

This report presents an analysis of the implementation of You only look once(YOLO)[2] network, training on Pascal Visual Object Classes (VOC) dataset[1].

## 2. Methodology

### 2.1. Dataset

The PASCAL Visual Object Classes (VOC) dataset[1] is an extensively utilized benchmark in computer vision research, particularly in the domains of object detection, image classification, and segmentation. This dataset was initially introduced as part of the PASCAL VOC challenges, which were conducted annually from 2005 to 2012 to drive advancements in visual object recognition. It contains images annotated with bounding boxes and class labels across multiple object categories, providing a rich and diverse training resource for model development.

For this implementation, the training dataset comprises 4,300 images sourced from both the VOC 2007 and VOC 2012 datasets. The dataset encompasses 20 object classes, categorized as follows:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

### 2.2. Data Augmentation

Data augmentation is a technique designed to increase the effective size and diversity of a training dataset through random transformations, enhancing the model's generalization ability by exposing it to varied representations of the original images.

In this context, images are resized to 448×448 pixels, and pixel values are scaled to the range of [-1, 1] following data augmentation.

#### 2.2.1. Random Blur

Random Blur applies a probabilistic blurring effect, using a 5x5 kernel with a 50% probability, to simulate realistic image blurring due to motion or defocus. This approach helps the model become resilient to common real-world image distortions.

#### 2.2.2. Random Brightness

Random Brightness adjusts the brightness of the image by a random factor between 0.5 (darker) and 1.5 (brighter) to simulate varied lighting conditions, which enables the model to adapt to diverse illumination scenarios encountered in practical applications.

#### 2.2.3. Random Hue

Random Hue shifts the hue of the image to simulate different lighting environments, such as color tints or white balance variations, using a random adjustment factor between 0.5 and 1.5.

#### 2.2.4. Random Saturation

Random Saturation modifies the color intensity in an image, either enhancing or diminishing it to mimic varying levels of color richness. This implementation applies a random adjustment factor between 0.5 and 1.5.

#### 2.2.5. Random Horizontal Flipping

Random Horizontal Flipping mirrors images horizontally with a 50% probability. This technique introduces orientation variability, helping the model learn orientation-invariant features, which enhances its performance on unseen test data.

#### 2.2.6. Random Scaling

Random Scaling adjusts the perceived size of objects by scaling the image with a factor between 0.8 and 1.2 while maintaining the original height. This scaling variability enhances the model's robustness to changes in object size.

#### 2.2.7. Random Translation

Random Translation shifts the image by a random proportion (up to 20%) in both horizontal and vertical directions. This shift mitigates model sensitivity to object position within the image. Bounding boxes are adjusted accordingly, retaining only those fully visible within the new boundaries.

#### 2.2.8. Random Cropping

Random Cropping selects a random portion of the image (between 60% and 100% of the original width and height) and crops it to introduce spatial variability. This forces the model to generalize by learning broader object features rather than relying solely on specific object positions within the image. Bounding boxes are retained for objects visible within the cropped area.

### 2.3. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an iterative optimization algorithm extensively utilized for training machine learning models, particularly neural networks. The fundamental objective of SGD is to minimize a given loss function by iteratively updating the model's parameters in the direction of the negative gradient, thereby approaching the local or global minimum. Unlike traditional Gradient Descent, which evaluates the gradient using the entire dataset, SGD performs updates based on a randomly selected subset, or mini-batch, of the data. This stochastic process introduces noise into the optimization trajectory, which can facilitate escaping local minima and accelerate convergence.

A variant of SGD employed incorporates momentum, a technique that enhances the convergence properties of the standard SGD algorithm. Momentum is designed to accelerate SGD by adding a fraction of the previous update vector to the current update, thereby damping oscillations and stabilizing the optimization path. The PyTorch library implements Nesterov momentum based on the formula from [3].

### 2.4. Leaky Rectified Linear Unit (Leaky ReLU)

The Leaky Rectified Linear Unit (Leaky ReLU) is an activation function extensively used in neural networks due to its ability to address the "dying ReLU" issue seen with standard ReLU functions. Formally, Leaky ReLU is

**Algorithm 1:** Stochastic Gradient Descent with Momentum and Nesterov

**Input:** $\gamma$ (lr), $\theta_0$ (params), $f(\theta)$ (objective), $\lambda$ (weight decay), $\mu$ (momentum), $\tau$ (dampening), *nesterov*, *maximize*

**for** $t = 1$ *to* ... **do**
- $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
- **if** $\lambda \neq 0$ **then**
  - $g_t \leftarrow g_t + \lambda \theta_{t-1}$
- **if** $\mu \neq 0$ **then**
  - **if** $t > 1$ **then**
    - $b_t \leftarrow \mu b_{t-1} + (1-\tau) g_t$
  - **else**
    - $b_t \leftarrow g_t$
  - **if** *nesterov* **then**
    - $g_t \leftarrow g_t + \mu b_t$
  - **else**
    - $g_t \leftarrow b_t$
- **if** *maximize* **then**
  - $\theta_t \leftarrow \theta_{t-1} + \gamma g_t$
- **else**
  - $\theta_t \leftarrow \theta_{t-1} - \gamma g_t$

**return** $\theta_t$

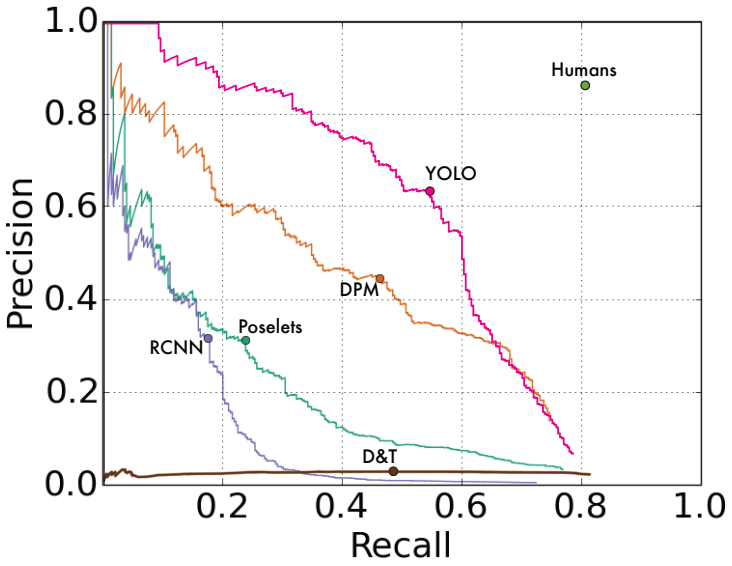


Figure 2: Precision-Recall Curve that is presented in original YOLO paper[2]

defined as:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases}$$

where $x$ denotes the input to the neuron, and $\alpha$ is a small positive constant (often 0.01). In contrast to standard ReLU, which outputs zero for negative inputs, Leaky ReLU provides a small, non-zero gradient for negative values of $x$. This non-zero gradient allows neurons receiving negative inputs to continue learning, reducing the likelihood of neuron inactivity. Leaky ReLU is particularly effective in deep networks where gradient flow across layers is essential for effective learning. For this implementation, $\alpha = 0.1$ is used across all Leaky ReLU layers.

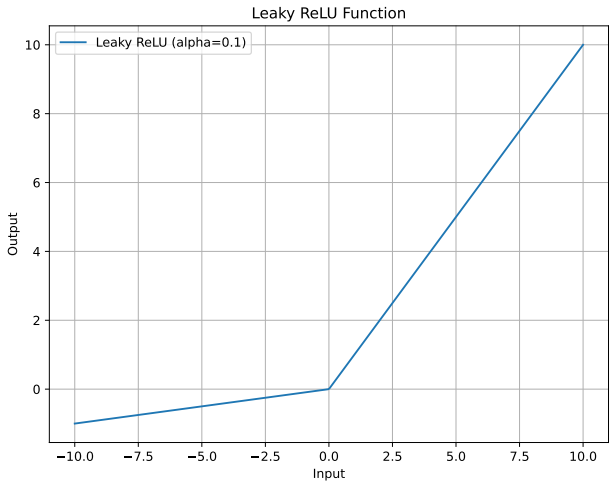


Figure 1: Leaky Rectified Linear Unit

### 2.5. mean Average Precision (mAP)

Mean Average Precision (mAP) is a widely used evaluation metric in object detection tasks, quantifying the accuracy of a model's predictions by considering both the localization and classification of objects. The mAP metric is used to determine whether the model is trained appropriately.

The calculation of mAP involves several steps:

**Intersection over Union (IoU)**: For each predicted bounding box, the Intersection over Union (IoU) is computed, representing the overlap ratio between the predicted and ground-truth bounding boxes. IoU is calculated as the area of overlap divided by the area of union of the two boxes. A higher IoU value indicates a better match between the prediction and the ground truth. Typically, a prediction is considered correct if its IoU exceeds a threshold, commonly set at 0.5 (IoU = 0.5).

**Precision-Recall Curve**: Based on the IoU threshold, each predicted bounding box is classified as either a true positive (correct detection) or a false positive (incorrect detection or duplicate). Using these classifications, a precision-recall curve is constructed, plotting precision (the ratio of true positives to the sum of true positives and false positives) against recall (the ratio of true positives to the sum of true positives and false negatives) Fig 2. This curve visualizes the model's performance in terms of its ability to detect all instances of objects (recall) while minimizing incorrect detections (precision).

**Average Precision (AP)**: The Average Precision (AP) score for each class is obtained by calculating the area under the precision-recall curve. This score provides an aggregate measure of precision across various recall levels, reflecting how well the model balances precision and recall for a given object class.

**Mean Average Precision (mAP)**: Finally, the mAP score is computed by averaging the AP scores across all object classes. This metric represents the overall detection accuracy of the model across all categories and is a robust measure of the model's performance in handling diverse object classes within an image.

In this study, mAP was implemented as the primary evaluation metric for the YOLO model, providing a comprehensive measure of its object detection performance. The metric's sensitivity to both localization accuracy and classification effectiveness makes it ideal for evaluating models with complex, multi-object detection capabilities.

## 3. Implementation Details

### 3.1. YOLO Model Architecture

The YOLO (You Only Look Once) model architecture, consists of a series of convolutional layers with leaky ReLU activation functions, interspersed with max pooling layers and followed by fully connected layers. Key structural characteristics include 22 convolutional layers, 4 max pooling layers,

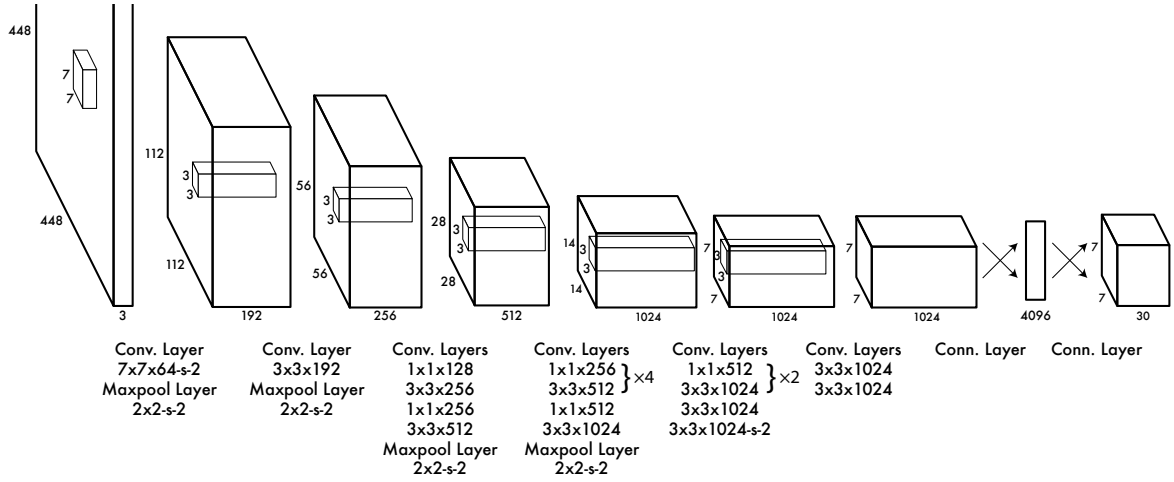


Figure 3: **Original YOLO Architecture** YOLO detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating $1 \times 1$ convolutional layers reduce the features space from preceding layers. Authors pretrain the convolutional layers on the ImageNet classification task at half the resolution ($224 \times 224$ input image) and then double the resolution for detection. Note that the model implemented in this report is different from network of original paper.

and 2 fully connected layers. Leaky ReLU activation functions are applied to all layers except the output layer. The implemented model exhibits modifications from the network structure outlined in the original paper, as summarized in Table 1.

The YOLO model processes input images by dividing them into an $S \times S$ grid of cells. Each grid cell independently predicts $B$ bounding boxes (with $B = 2$) and $C$ class probabilities (with $C = 20$). Each bounding box contains five components: $x$, $y$, $w$, $h$, and a confidence score. The components $x$ and $y$ denote the bounding box center relative to the grid cell, while $w$ and $h$ are the width and height relative to the entire image dimensions. The confidence score evaluates the probability that the bounding box contains an object, computed as the intersection over union (IoU) between the predicted and actual bounding box—this factor is crucial for constructing the loss function. Consequently, the output vector structure is $S \times S \times (B \times 5 + C) = 7 \times 7 \times 30$.

### 3.2. Loss Function

The YOLO training process optimizes a loss function:

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&\quad + \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&\quad + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2 \\
&\quad + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2 \\
&\quad + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2 \qquad (1)
\end{aligned}
$$

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell $i$ and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the $j$th bounding box predictor in cell $i$ is "responsible" for that prediction.

The YOLO training process optimizes a multi-component loss function defined in terms of position, confidence, and class probabilities. This loss function is broken down into five main components, each with specific weights to address different aspects of the YOLO model's prediction accuracy.

**Localization Loss (Coordinate Loss)**: The first two terms, weighted by $\lambda_{\textbf{coord}}$, penalize deviations in bounding box coordinates $(x, y)$ and dimensions $(w, h)$. To stabilize training and account for scale sensitivity, the width and height terms use the square root of dimensions. This is computed only for cells containing an object, enforcing model precision in the bounding box coordinates.

**Confidence Loss (Objectness Score)**: Confidence is penalized differently based on whether a cell contains an object. If a cell contains an object, the responsible bounding box predictor's confidence score $C_i$ is compared to the ground truth IoU, and if no object is present, the predictor is penalized more lightly by $\lambda_{\textbf{noobj}}$ to prevent over-penalizing empty cells.

**Classification Loss**: This component penalizes the difference between predicted class probabilities $p_i(c)$ and ground truth $\hat{p}_i(c)$ for each class in the cells containing objects. It ensures the model accurately distinguishes object classes within cells where an object appears.

In the 'yoloLoss' class, each component of this loss function is implemented as follows:

- **compute_prob_error** handles the classification error, calculating the discrepancy between predicted and target class probabilities in cells containing an object.
- **not_contain_obj_error** calculates the confidence loss in cells without objects, summing the square error for the two confidence scores from each bounding box predictor.
- **contain_obj_error** addresses localization error (coordinate and dimension discrepancies) and confidence error in cells with objects. The function dynamically assigns bounding box predictors based on their IoU with ground truth, and it uses these "responsible" predictors to compute losses on bounding box center coordinates $(x, y)$, width and height $(w, h)$, and confidence scores. It also penalizes confidence error of bounding boxes that are not "responsible".

Overall, the loss combines localization, confidence, and classification errors with adjustable weights, $\lambda_{\textbf{coord}} = 5$ and $\lambda_{\textbf{noobj}} = 0.5$, to balance accuracy in object localization and detection reliability.

## 4. Results

### 4.1. Inference Visualization

To qualitatively assess the model's performance, inference visualization was performed on a subset of 9 representative test images. As shown in Figure 4, the model generates bounding boxes around detected objects within each image, accurately predicting their classes along with associated confidence

Table 1: YOLO Architecture

| pre_train_net | | | |
|---|---|---|---|
| Name | Filters | Output Dimension | Padding |
| Conv 1 | $7 \times 7 \times 64$, stride=2 | $224 \times 224 \times 64$ | 3 |
| LeakyReLU | negative_slope=0.1 | $224 \times 224 \times 64$ | |
| Max Pool 1 | $2 \times 2$, stride=2 | $112 \times 112 \times 64$ | |
| Conv 2 | $3 \times 3 \times 192$ | $112 \times 112 \times 192$ | 1 |
| LeakyReLU | negative_slope=0.1 | $112 \times 112 \times 192$ | |
| Max Pool 2 | $2 \times 2$, stride=2 | $56 \times 56 \times 192$ | |
| Conv 3 | $1 \times 1 \times 128$ | $56 \times 56 \times 128$ | 0 |
| LeakyReLU | negative_slope=0.1 | $56 \times 56 \times 128$ | |
| Conv 4 | $3 \times 3 \times 256$ | $56 \times 56 \times 256$ | 1 |
| LeakyReLU | negative_slope=0.1 | $56 \times 56 \times 256$ | |
| Conv 5 | $1 \times 1 \times 256$ | $56 \times 56 \times 256$ | 0 |
| LeakyReLU | negative_slope=0.1 | $56 \times 56 \times 256$ | |
| Conv 6 | $3 \times 3 \times 512$ | $56 \times 56 \times 512$ | 1 |
| LeakyReLU | negative_slope=0.1 | $56 \times 56 \times 512$ | |
| Max Pool 3 | $2 \times 2$, stride=2 | $28 \times 28 \times 512$ | |
| Conv 7 | $1 \times 1 \times 256$ | $28 \times 28 \times 256$ | 0 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 256$ | |
| Conv 8 | $3 \times 3 \times 512$ | $28 \times 28 \times 512$ | 1 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 512$ | |
| Conv 9 | $1 \times 1 \times 256$ | $28 \times 28 \times 256$ | 0 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 256$ | |
| Conv 10 | $3 \times 3 \times 512$ | $28 \times 28 \times 512$ | 1 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 512$ | |
| Conv 11 | $1 \times 1 \times 256$ | $28 \times 28 \times 256$ | 0 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 256$ | |
| Conv 12 | $3 \times 3 \times 512$ | $28 \times 28 \times 512$ | 1 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 512$ | |
| Conv 13 | $1 \times 1 \times 256$ | $28 \times 28 \times 256$ | 0 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 256$ | |
| Conv 14 | $3 \times 3 \times 512$ | $28 \times 28 \times 512$ | 1 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 512$ | |
| Conv 15 | $1 \times 1 \times 512$ | $28 \times 28 \times 512$ | 0 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 512$ | |
| Conv 16 | $3 \times 3 \times 1024$ | $28 \times 28 \times 1024$ | 1 |
| LeakyReLU | negative_slope=0.1 | $28 \times 28 \times 1024$ | |
| Max Pool 4 | $2 \times 2$, stride=2 | $14 \times 14 \times 1024$ | |
| Conv 17 | $1 \times 1 \times 512$ | $14 \times 14 \times 512$ | 0 |
| LeakyReLU | negative_slope=0.1 | $14 \times 14 \times 512$ | |
| Conv 18 | $3 \times 3 \times 1024$ | $14 \times 14 \times 1024$ | 1 |
| LeakyReLU | negative_slope=0.1 | $14 \times 14 \times 1024$ | |
| post_net | | | |
| Name | Filters | Output Dimension | Padding |
| Conv 19 | $3 \times 3 \times 1024$ | $14 \times 14 \times 1024$ | 1 |
| LeakyReLU | negative_slope=0.1 | $14 \times 14 \times 1024$ | |
| Conv 20 | $3 \times 3 \times 1024$, stride=2 | $7 \times 7 \times 1024$ | 1 |
| LeakyReLU | negative_slope=0.1 | $7 \times 7 \times 1024$ | |
| Conv 21 | $3 \times 3 \times 1024$ | $7 \times 7 \times 1024$ | 1 |
| LeakyReLU | negative_slope=0.1 | $7 \times 7 \times 1024$ | |
| Conv 22 | $3 \times 3 \times 1024$ | $7 \times 7 \times 1024$ | 1 |
| LeakyReLU | negative_slope=0.1 | $7 \times 7 \times 1024$ | |
| Fully Connected layers | | | |
| FC 1 | $50176 \times 4096$ | 4096 | |
| LeakyReLU | negative_slope=0.1 | 4096 | |
| Dropout | $p = 0.5$ | 4096 | |
| FC 2 | $4096 \times 1470$ | 1470 | |

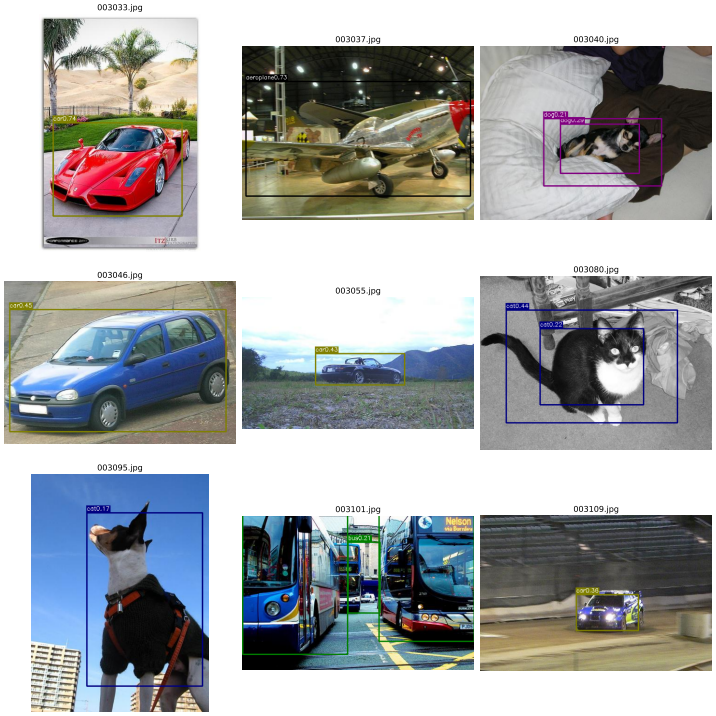


Figure 4: Inference results of the trained model on a subset of 9 test images.

scores. Notably, in image 003095.jpg, the model incorrectly identifies a dog as a cat, highlighting a case of misclassification which may warrant further refinement of model parameters or training data.

### 4.2. Test Set Performance

The model's quantitative performance was evaluated using the mean Average Precision (mAP) metric, calculated specifically for a subset of object classes from the Pascal VOC dataset: 'aeroplane', 'bicycle', 'bus', 'car', 'cat', and 'dog'. Figure 5 illustrates the precision-recall curves for each class, providing a comprehensive view of the model's detection capabilities.

The Average Precision (AP) for each individual class was computed as follows:

- Aeroplane AP: 61.77%
- Bicycle AP: 65.94%
- Bus AP: 59.35%
- Car AP: 67.94%
- Cat AP: 74.89%
- Dog AP: 78.01%

The mean Average Precision (mAP) across these selected classes is therefore 67.98%, which reflects the model's overall detection accuracy and class prediction performance within this specified subset of object categories.

## 5. Conclusion

This report presented the training, evaluation, and analysis of a YOLO-based object detection model, focusing on a subset of the Pascal VOC dataset. A thorough data preprocessing techniques are utilized, facilitate stable training and improve model convergence. The model's performance was assessed using mean Average Precision (mAP), which served as a comprehensive metric to capture both the precision and recall for each class.

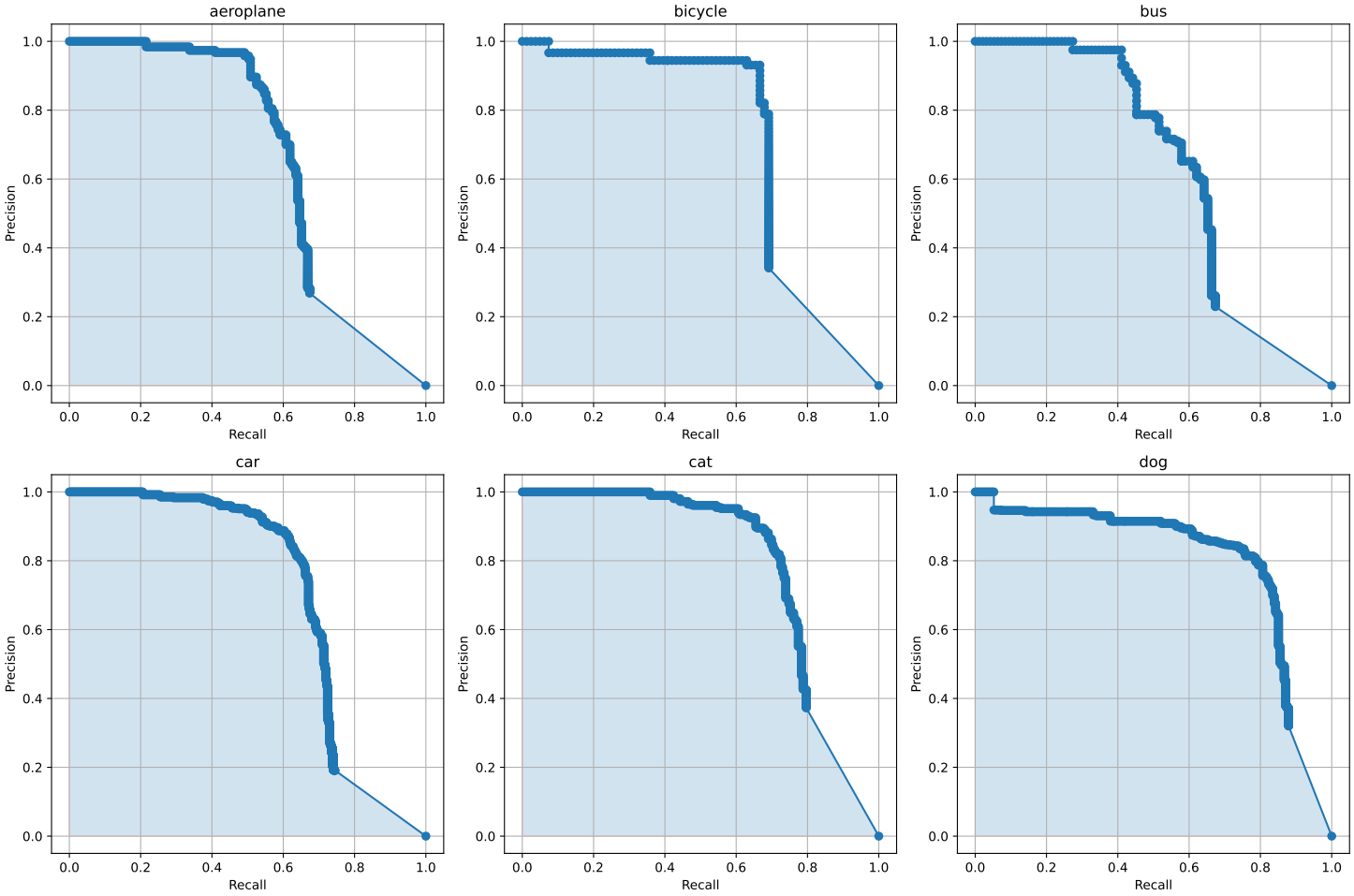


Figure 5: Precision-Recall curves for a subset of Pascal VOC classes: aeroplane, bicycle, bus, car, cat, and dog.

Experimental results show that the model achieved an overall mAP of 66.26% across the selected classes, with individual Average Precision (AP) values varying based on class complexity and object features. The visualization of inference results and precision-recall curves provided further insights into the model's ability to accurately detect and classify objects. Notably, higher AP values were observed for classes with distinct visual features, such as 'cat' and 'dog,' indicating the model's efficacy in distinguishing clear object boundaries.

[1] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[2] J Redmon. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[3] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.