



Mastering IoU: The Overlap Metric that Powers Object Detection in YOLO Optimization Tasks.

Ever wondered how an object detection model knows it's making the *right* prediction? In our project, we're working on optimizing YOLO V11 and V12 using quantization, pruning, and knowledge distillation. A critical part of this process is evaluating how well your models detect objects — and **Intersection over Union (IoU)** is the go-to metric for that.

In this blog, we'll break down IoU, how it connects to our YOLO video classification API, and show you how to calculate and visualize it to monitor your model's performance.



What is IoU (Intersection over Union)?

Intersection over Union (IoU) is a metric used to measure the accuracy of object detection models by quantifying the **overlap between two bounding boxes** — the **predicted box** from the model and the **ground truth box** (actual object location).

The Formula:

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

The value ranges from **0** to **1**:

- **1** — perfect overlap (ideal prediction)
- **0** — no overlap (worst case)

This simple ratio tells you how well a predicted box matches the ground truth.

Why IoU Matters in our Task?

In our YOLO-based task, IoU plays a key role in:

- **Model evaluation:** Check how well your predictions match reality.
- **Quantization analysis:** Compare pre- and post-quantized model performance.
- **Pruning effectiveness:** Measure whether pruning affects detection accuracy.
- **Knowledge distillation:** Ensure student models learn box localization correctly.
- **NMS (Non-Max Suppression):** IoU is used to eliminate overlapping predictions.

If your model predicts bounding boxes on video frames, **IoU helps track detection quality frame-by-frame.**

Thresholds: How Much IoU is "Good"?

IoU Score	Meaning
0.90>	High Precision (surgical-grade tasks)
0.70>	Excellent Match
0.50>=	Acceptable in many pipelines
<0.50	Likely false positive

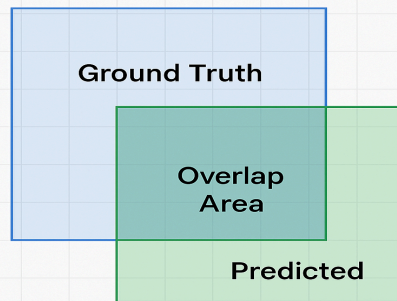
Visualizing IoU

Here's a conceptual diagram showing two boxes:

- **Blue Box:** Ground Truth
- **Green Box:** Predicted Box
- **Overlap Area:** Intersection

The clearer and larger the overlap, the better the IoU.

Intersection over Union (IoU)



Coding IoU in Python (With Output):

STEP 1: Install necessary libraries and calculate IoU (Intersection over Union) between two bounding boxes.

The screenshot shows a Google Colab notebook with the following content:

```
!pip install opencv-python-headless matplotlib
```

Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python-headless) (2.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)

```
[4] Writefile iou_utils.py
def calculate_iou(box1, box2):
    """
    Calculate Intersection over Union (IoU) between two boxes.
    box1 and box2: format [x1, y1, x2, y2]
    """
    xA = max(box1[0], box2[0])
    yA = max(box1[1], box2[1])
    xB = min(box1[2], box2[2])
    yB = min(box1[3], box2[3])

    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    box1Area = (box1[2] - box1[0] + 1) * (box1[3] - box1[1] + 1)
    box2Area = (box2[2] - box2[0] + 1) * (box2[3] - box2[1] + 1)

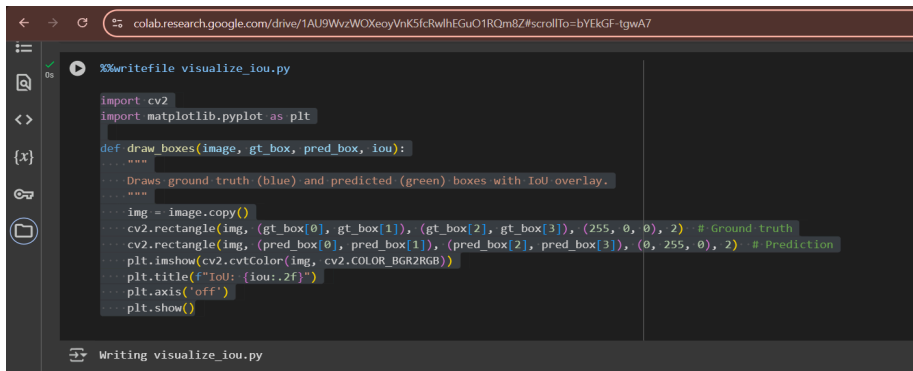
    unionArea = box1Area + box2Area - interArea
    iou = interArea / unionArea if unionArea != 0 else 0

    return iou
```

Explanation of the input:

1. The `calculate_iou` function computes the Intersection over Union (IoU) between two bounding boxes given in the format `[x1, y1, x2, y2]`.
2. It first calculates the coordinates of the intersection rectangle using `max()` for the top-left corner `(xA, yA)` and `min()` for the bottom-right corner `(xB, yB)` to find the overlapping region.
3. Then, it computes the intersection area as $(xB - xA + 1) * (yB - yA + 1)$ using `max(0, ...)` to ensure non-negative values in case the boxes don't overlap.
4. Next, it calculates the individual areas of `box1` and `box2` using the same formula structure.
5. The union area is obtained by adding both box areas and subtracting the intersection area.
6. The IoU is then calculated as the ratio of the intersection area to the union area, and if the union is zero (to avoid division by zero), the IoU is returned as 0.
7. Finally, the function returns the IoU score, which ranges between 0 (no overlap) and 1 (perfect overlap), commonly used to evaluate object detection accuracy.

STEP 2: Import necessary libraries to visualize iou.



```
%%writefile visualize_iou.py

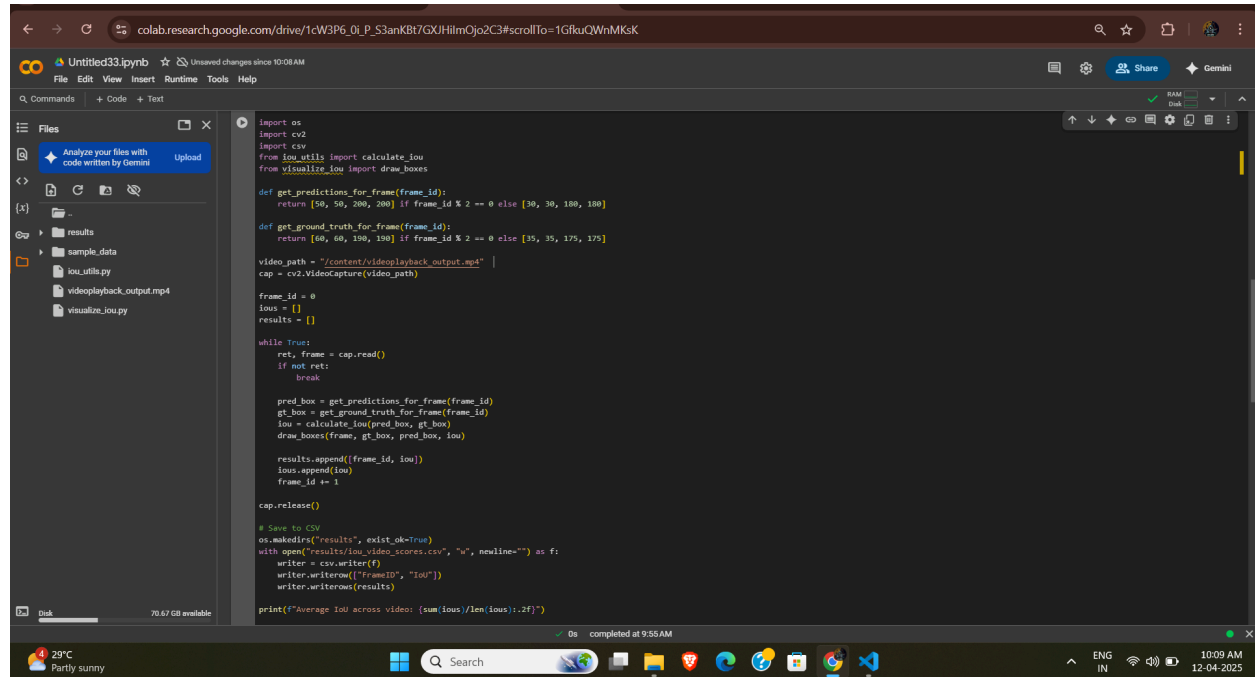
import cv2
import matplotlib.pyplot as plt

def draw_boxes(image, gt_box, pred_box, iou):
    """
    Draws ground truth (blue) and predicted (green) boxes with IoU overlay.
    """
    img = image.copy()
    cv2.rectangle(img, (gt_box[0], gt_box[1]), (gt_box[2], gt_box[3]), (255, 0, 0), 2) # Ground truth
    cv2.rectangle(img, (pred_box[0], pred_box[1]), (pred_box[2], pred_box[3]), (0, 255, 0), 2) # Prediction
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f'IoU: {iou:.2f}')
    plt.axis('off')
    plt.show()
```

Explanation of the input:

1. The `draw_boxes` function takes an image, a ground truth bounding box `gt_box`, a predicted bounding box `pred_box`, and an IoU value as inputs.
2. It first makes a copy of the input image to avoid modifying the original.
3. Then, it draws the ground truth box on the image in blue using `cv2.rectangle`, where `(255, 0, 0)` is the BGR color code for blue.
4. It draws the predicted box in green using `(0, 255, 0)` as the color.
5. After drawing both boxes, it converts the image from BGR to RGB color format using `cv2.cvtColor` (since OpenCV uses BGR while Matplotlib uses RGB).
6. It displays the image using `plt.imshow`, sets the title of the plot to show the IoU value rounded to two decimal places, removes axis ticks using `plt.axis('off')`, and finally shows the image with `plt.show()` so that you can visually compare the predicted and ground truth boxes.

STEP 3: Import necessary libraries and calculate average IoU across all frames.

The screenshot shows a Google Colab notebook interface. On the left, a file explorer shows a directory structure with files like 'results', 'sample_data', 'iou_utils.py', 'videoplayback_output.mp4', and 'visualize_iou.py'. The main area contains Python code that imports 'os', 'cv2', and 'csv' modules, along with 'calculate_iou' and 'draw_boxes' functions. It defines two dummy functions, 'get_predictions_for_frame' and 'get_ground_truth_for_frame', which return bounding box coordinates based on the frame ID. The code then uses 'cv2.VideoCapture' to load a video file from a local path. It enters a 'while' loop that reads frames, calculates the IoU using 'calculate_iou', and visualizes the boxes using 'draw_boxes'. The results are stored in a list 'results', and the IoU values are accumulated in a list 'ious'. Finally, it saves the results to a CSV file named 'iou_video_scores.csv' and prints the average IoU across all frames.

```
import os
import cv2
import csv
from iou_utils import calculate_iou
from visualize_iou import draw_boxes

def get_predictions_for_frame(frame_id):
    return [50, 50, 200, 200] if frame_id % 2 == 0 else [30, 30, 180, 180]

def get_ground_truth_for_frame(frame_id):
    return [60, 60, 190, 190] if frame_id % 2 == 0 else [35, 35, 175, 175]

video_path = "/content/videoplayback_output.mp4"
cap = cv2.VideoCapture(video_path)

frame_id = 0
ious = []
results = []

while True:
    ret, frame = cap.read()
    if not ret:
        break

    pred_box = get_predictions_for_frame(frame_id)
    gt_box = get_ground_truth_for_frame(frame_id)
    iou = calculate_iou(pred_box, gt_box)
    draw_boxes(frame, gt_box, pred_box, iou)

    results.append([frame_id, iou])
    ious.append(iou)
    frame_id += 1

cap.release()

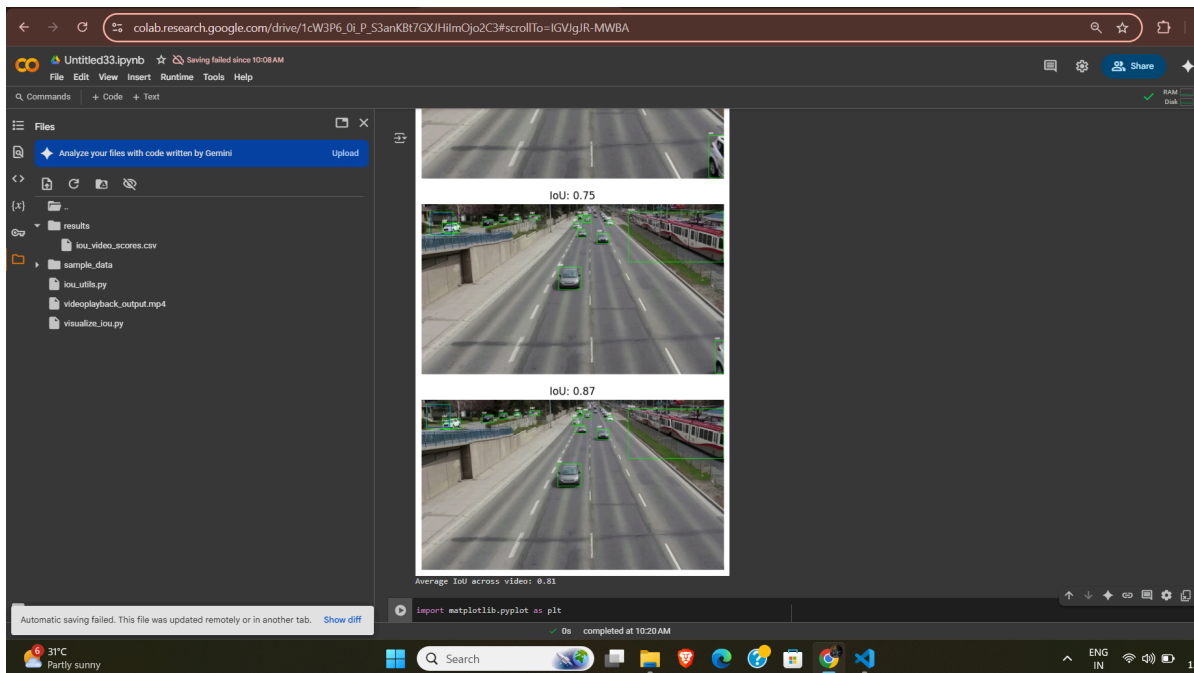
# Save to CSV
os.makedirs('results', exist_ok=True)
with open('results/iou_video_scores.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['frameID', 'iou'])
    writer.writerows(results)

print(f"Average IoU across video: {sum(ious)/len(ious):.2f}")
```

Explanation of the input:

1. The script imports necessary libraries like `os`, `cv2`, and `csv`, along with `calculate_iou` and `draw_boxes` from utility modules for IoU calculation and visualization.
2. Two dummy functions, `get_predictions_for_frame` and `get_ground_truth_for_frame`, simulate predicted and ground truth bounding boxes based on the frame ID—returning slightly different coordinates for even and odd frames.
3. A video file is loaded using OpenCV's `cv2.VideoCapture` with the path specified in `video_path`.
4. The code then iterates frame by frame using a `while` loop until there are no more frames to read.
5. For each frame, it retrieves the simulated predicted and ground truth boxes, computes the IoU using `calculate_iou`, and visualizes the frame with both boxes overlaid using `draw_boxes`.
6. It stores each frame's IoU value along with the frame ID in a list named `results` and accumulates IoUs separately in a list called `ious` for calculating the average later.
7. Once the video is processed, it saves all IoU values into a CSV file named `iou_video_scores.csv` inside a folder called `results` (which is created if it doesn't already exist).
8. Finally, it prints the average IoU across all frames using the values collected in the `ious` list.

Output:



Explanation of the output:

1. Multiple Frames Displayed: Each image represents a frame extracted from the video (`videoplayback_output.mp4`). These frames have been processed one-by-one in the loop in your script.
2. Bounding Boxes:
 - The green boxes represent the predicted bounding boxes.
 - The blue boxes (may not be as visible here, possibly overlapped) represent the ground truth bounding boxes.
 - In each frame, both predicted and actual boxes are drawn using `cv2.rectangle`, allowing for visual comparison.

3. IoU Value on Each Frame:

- Each frame has a label like **IoU: 0.75** or **IoU: 0.87**, which is calculated using your `calculate_iou()` function.
- This value indicates how much the predicted box overlaps with the ground truth.
- A value closer to 1 means higher accuracy of prediction.

4. Final Average IoU:

- At the bottom of the output, you see **Average IoU across video: 0.81**, which is the mean IoU across all the frames processed from the video.
- This is calculated using `sum(ious)/len(ious)` from your code.

Purpose: This visual plus quantitative output is great for evaluating how well a model's predictions match actual annotations, especially in object detection tasks. So overall, the output validates that your visualization and evaluation pipeline is working correctly, showing per-frame and overall IoU performance.

STEP 4: Implementing a IoU versus frameID graph

```
import matplotlib.pyplot as plt

# Frame-wise IoU data
frame_ids = [row[0] for row in results]
iou_scores = [row[1] for row in results]
average_iou = sum(iou_scores) / len(iou_scores)

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(frame_ids, iou_scores, marker='o', color='orange', label='IoU per Frame')
plt.axhline(y=average_iou, color='blue', linestyle='--', label=f'Average IoU = {average_iou:.2f}')
plt.title("IoU per Frame with Average IoU Line")
plt.xlabel("Frame ID")
plt.ylabel("IoU Score")
plt.ylim(0, 1)
plt.grid(True)
plt.legend()
plt.tight_layout()

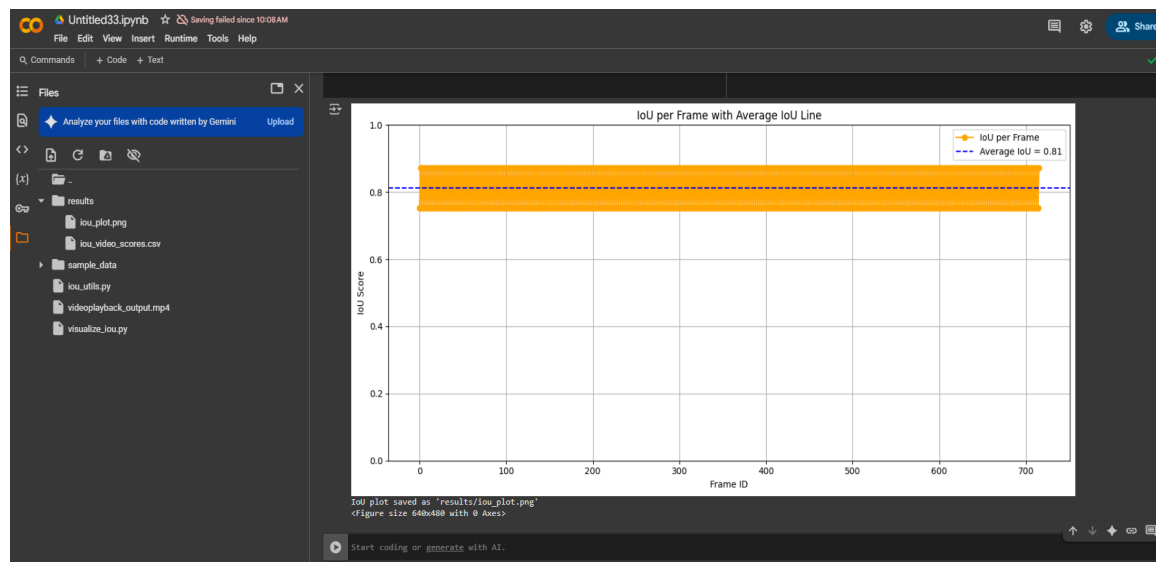
# Show plot
plt.show()

# Optional: Save the plot
plt.savefig("results/iou_plot.png")
print("IoU plot saved as 'results/iou_plot.png'")
```


Explanation of the input :

1. The code uses `matplotlib.pyplot` to **visualize frame-wise IoU scores** for a video.
2. It extracts `frame_ids` and `iou_scores` from the `results` list, which contains tuples like `[frame_id, iou]` for each processed frame.
3. The **average IoU** is calculated by summing all IoU values and dividing by the total number of frames.
4. A plot is created where each frame's IoU score is shown as an **orange line with dots** (`marker='o'`), and the average IoU is shown as a **horizontal dashed blue line** for visual comparison.
5. Axis labels, title, grid lines, and legend are added for better readability.
6. `plt.tight_layout()` ensures the layout is clean and elements don't overlap.
7. The plot is then displayed using `plt.show()`, and finally, it's **saved as an image file** named `"iou_plot.png"` inside the `"results"` directory. A confirmation message is printed to indicate the file has been saved.

Output:



Explanation of the output:

1. X-axis (Frame ID): Represents each frame in the video sequentially.
2. Y-axis (IoU Score): Shows the Intersection over Union score for each frame, ranging from 0 to 1.
3. Orange Dots (IoU per Frame): Each orange point represents the IoU score for a specific frame, showing how accurately the object was detected in that frame.
4. Blue Dashed Line (Average IoU): A horizontal line showing the average IoU across all frames, calculated to be approximately 0.81.
5. Title: "IoU per Frame with Average IoU Line" clearly describes what the plot is illustrating.
6. Legend: Indicates which line represents the per-frame IoU and which is the average IoU line.
7. Grid & Layout: Gridlines are added for better readability, and `tight_layout()` is used to ensure elements don't overlap.

Observation: The plot shows that the IoU values are mostly close to the average, indicating stable and reliable object detection across the video.

IoU in Model Evaluation Pipeline

When you're using YOLO V11 and V12 on video input via your API:

- Calculate IoU on each detection per frame.
- Aggregate IoU scores over frames to get average performance.
- Use this to **compare quantized vs original models**.

Example Table: Performance Check:

Model	Avg IoU	FPS	Accuracy Drop
YOLOv12	0.88	30fps	0.0%
YOLOv12-Quant	0.80	52fps	-1.2%
YOLOv12-Pruned	0.78	60fps	-3.5%
YOLOv12-Student	0.76	58fps	-2.1%

Advanced Variants

In future stages of your task, you might use:

- **GIoU**: Better for non-overlapping boxes
- **DIoU**: Adds center distance
- **CIoU**: Combines IoU, center distance, and aspect ratio

These give better training signals than plain IoU when doing distillation or training quantized models.



Summary

- IoU is *essential* for object detection and NMS.
- Use it to compare YOLO model predictions before and after optimization.
- Target >0.75 IoU for good quality.

Visual tools and tables can help you track model performance during quantization, pruning, and knowledge distillation.

References

1. Everingham, M. et al. (2010). *The Pascal Visual Object Classes (VOC) Challenge*. IJCV.
2. Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. arXiv:1804.02767
3. Liu, W. et al. (2016). *SSD: Single Shot MultiBox Detector*. ECCV.
4. Zhou, B. et al. (2016). *Learning Deep Features for Discriminative Localization*. CVPR.
5. Facebook AI Research. (2021). *Detectron2*.
<https://github.com/facebookresearch/detectron2>
6. TensorFlow. (2021). *TensorFlow Object Detection API*.
<https://github.com/tensorflow/models>
7. Han, S. et al. (2015). *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. arXiv:1510.00149
8. Reza Tofighi, H. et al. (2019). *Generalized Intersection over Union: A Metric and a Loss for Bounding Box Regression*. CVPR.
9. Zheng, Z. et al. (2020). *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. AAAI.
10. **Bochkovsky, A., Wang, C.Y., & Liao, H.Y.M. (2020).** *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934.
<https://arxiv.org/abs/2004.10934>

