Evaluating object detection with custom YOLOv8 architecture for enhanced speed and accuracy

Hossein Hashemi [a]

[a] *hossein.hashemi.ir@ieee.org*

October 2024

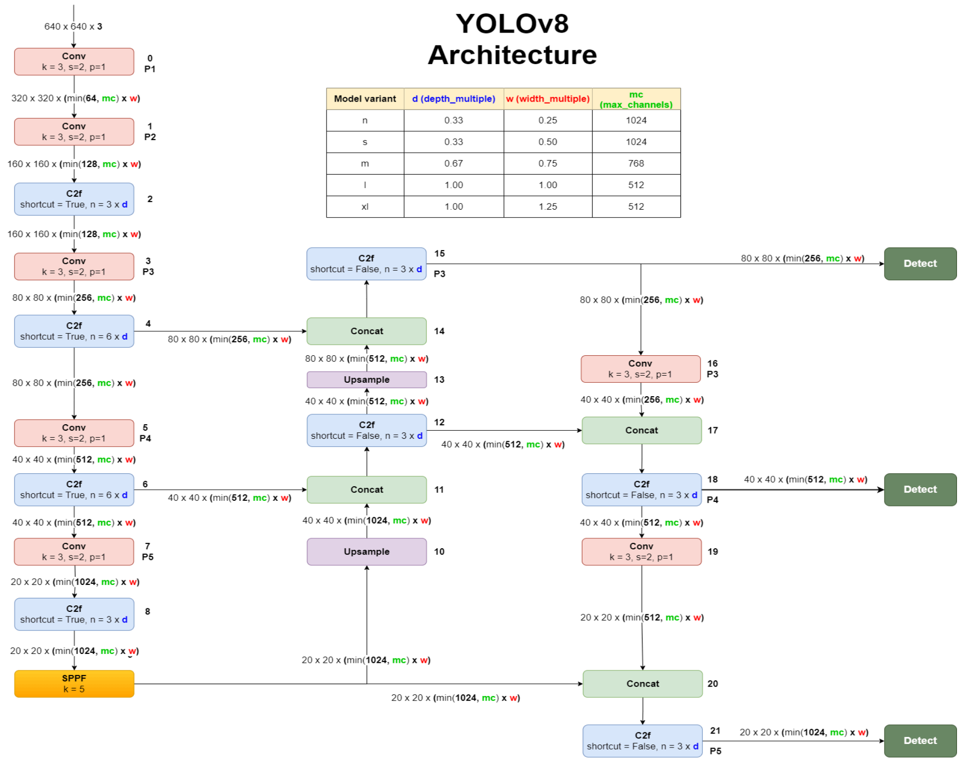
Abstract

This report focused on the modifications made to the YOLOv8 architecture to optimize it for the “CLEVR-Hans” dataset. The goal is to achieve a model to enhance speed and efficiency while preserving high object detection accuracy for our dataset. The key adjustment involves reducing the number of layers and modifying channels, optimizing components such as C2f and SPPF. in the end the evaluation compares the original and modified models which is showing that the modified version offers significantly faster performance with only a minimal decrease in accuracy, making it well-suited for real-time applications.

Keywords: (Object detection, YOLOv8, CLEVR-Hans dataset)

1. Introduction of YOLOv8 Architecture

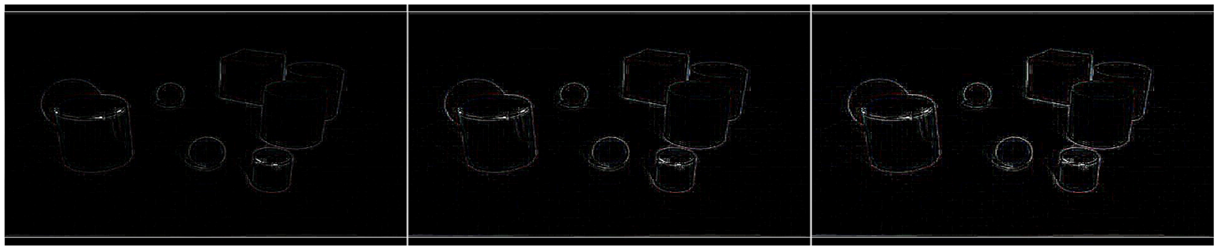
YOLOv8 is a cutting-edge object detection model which is designed to deliver high accuracy in identifying and detecting objects with various scales and environments. At the heart of YOLOv8s high performance capabilities is its well engineered architecture which by breaking down each layer, we can have a better understanding of them. below is a very short overview of these important layers:


YOLOv8
Architecture
640 x 640 x 3
Conv k = 3, s=2, p=1 0 P1
320 x 320 x (min(64, mc) x w)
Conv k = 3, s=2, p=1 1 P2
160 x 160 x (min(128, mc) x w)
C2f shortcut = True, n = 3 x d 2
160 x 160 x (min(128, mc) x w)
Conv k = 3, s=2, p=1 3 P3
80 x 80 x (min(256, mc) x w)
C2f shortcut = True, n = 6 x d 4
80 x 80 x (min(256, mc) x w)
Conv k = 3, s=2, p=1 5 P4
40 x 40 x (min(512, mc) x w)
C2f shortcut = True, n = 6 x d 6
40 x 40 x (min(512, mc) x w)
Conv k = 3, s=2, p=1 7 P5
20 x 20 x (min(1024, mc) x w)
C2f shortcut = True, n = 3 x d 8
20 x 20 x (min(1024, mc) x w)
SPPF k = 5 9
Upsample 10
40 x 40 x (min(1024, mc) x w)
Concat 11
C2f shortcut = False, n = 3 x d 12
40 x 40 x (min(512, mc) x w)
Upsample 13
80 x 80 x (min(512, mc) x w)
Concat 14
C2f shortcut = False, n = 3 x d 15 P3
80 x 80 x (min(256, mc) x w)
Detect
Conv k = 3, s=2, p=1 16 P3
40 x 40 x (min(256, mc) x w)
Concat 17
C2f shortcut = False, n = 3 x d 18 P4
40 x 40 x (min(512, mc) x w)
Detect
Conv k = 3, s=2, p=1 19
20 x 20 x (min(512, mc) x w)
Concat 20
C2f shortcut = False, n = 3 x d 21 P5
20 x 20 x (min(1024, mc) x w)
Detect

Model variant	d (depth_multiple)	w (width_multiple)	mc (max_channels)
n	0.33	0.25	1024
s	0.33	0.50	1024
m	0.67	0.75	768
l	1.00	1.00	512
xl	1.00	1.25	512

- **Convolutional Layers (Conv):**

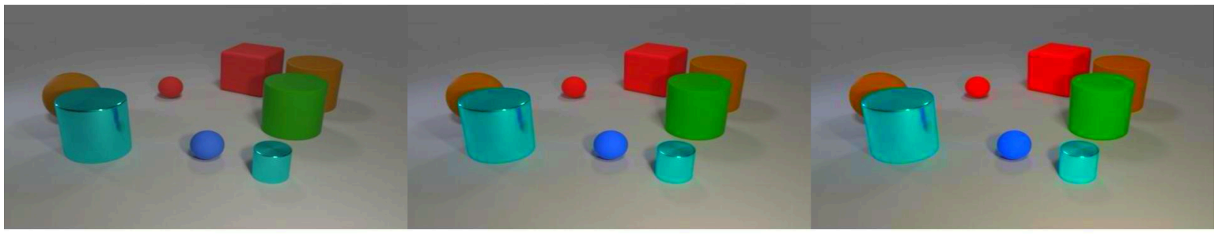
These layers perform initial feature extraction from the input image, reducing its dimensions while increasing the depth to capture more complex features.



These layers can detect basic shapes and edges of objects like cubes, spheres, and cylinders.

- **C2f (Cross-Stage Partial Connections)**:

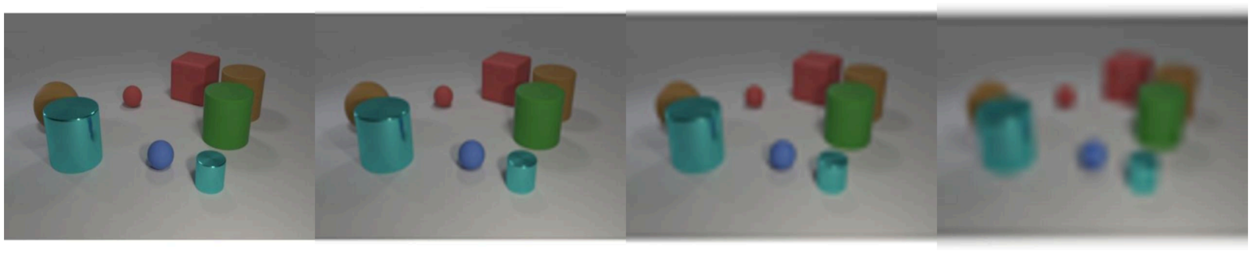
These blocks, marked as C2f, use shortcut connections to improve feature propagation and reduce the vanishing gradient problem, enhancing the network's ability to learn intricate patterns.



C2f blocks help in recognizing the subtle differences in object attributes, such as color, size, and material, distinguishing a red metal cylinder from a blue sphere.

- **Upsample and Concatenate:**

The architecture employs upsampling layers to increase the spatial resolution of the feature maps. These upsampled maps are then concatenated with feature maps from previous two layers to combine detailed spatial information with high-level features.

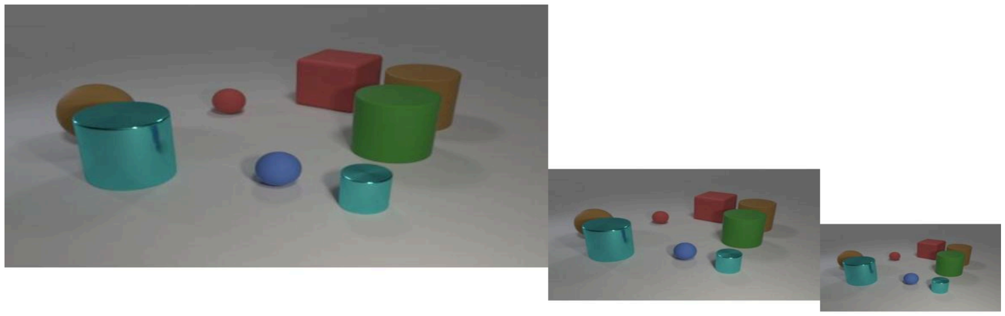


Upsampling and concatenation help models detect small, closely placed objects in complex scenes while preserving fine details.

.

- **SPPF (Spatial Pyramid Pooling - Fast):**

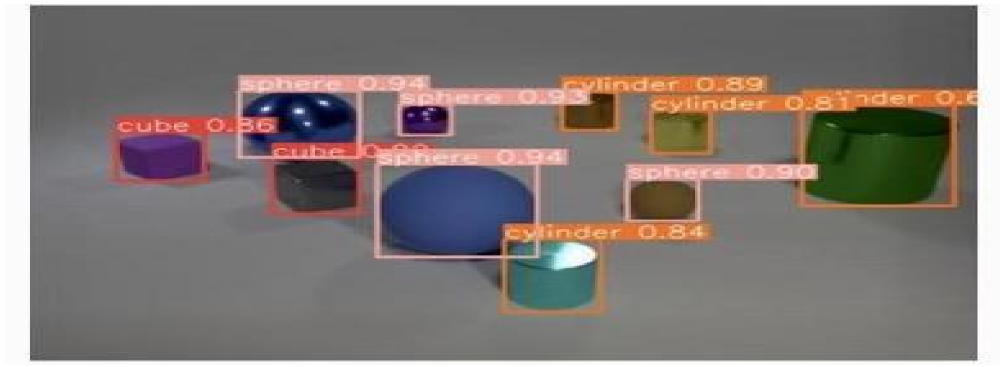
The SPPF module aggregates features from different scales, which helps in capturing contextual information and improving object detection performance at multiple scales.



The SPPF module helps detect objects of different sizes in the same scene, ensuring that small objects like tiny cubes and large objects like big spheres are detected accurately regardless of their size differences.

- **Detection:**

The final layers, labeled as Detect, are responsible for generating the output predictions, including bounding boxes and objectness scores, at multiple scales.



Detection heads would output the precise locations and classes of various objects (cubes, spheres, cylinders) in the scene.

---

2. Dataset Description

For this project, I used the CLEVR-Hans[1] dataset, which, although not originally designed for object detection, presents a significant challenge due to its complex variety of objects (cubes, spheres, and cylinders) that vary in color, size, and material. This complexity makes it ideal for testing the limits of object detection models. Since the original dataset is quite large, I generated a smaller set of 174 images using Roboflow[2] to focus more effectively on training the models.

- **Data Splitting for Balance:**

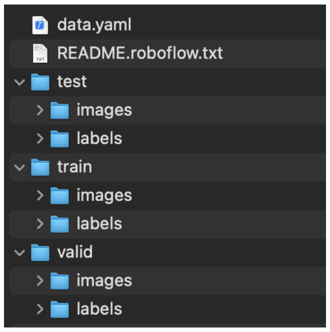
To evaluate the model effectively, I split the dataset into three parts: 70% for training (123 images), 20% for validation (35 images), and 10% for testing (18 images). This split was carefully chosen to balance the

[1] https://github.com/ml-research/CLEVR-Hans/tree/main
[2] https://universe.roboflow.com

need for sufficient training data while keeping enough samples aside to validate and test the model's performance.

- **Data Organization:**

The dataset is organized into three main folders (train, valid, and test) each containing subfolders for images and labels. This structure ensures the data is easily accessible and ready for training and evaluation.



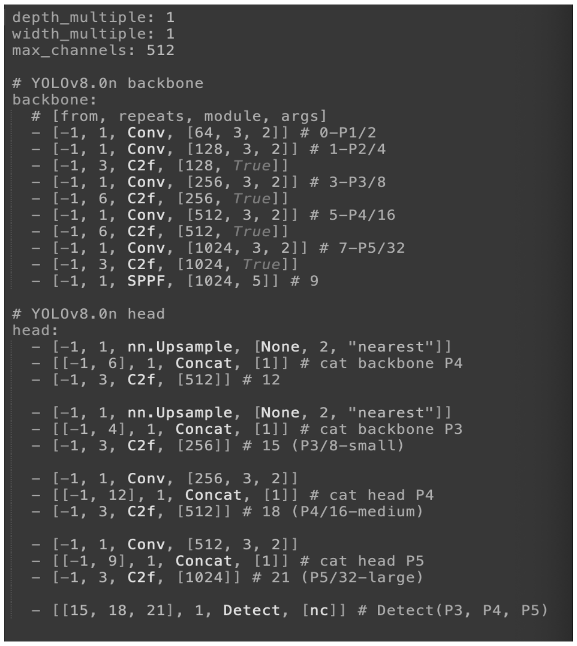
```
data.yaml
README.roboflow.txt
test
  images
  labels
train
  images
  labels
valid
  images
  labels
```

Each image has a matching label file that contains the object class (cube, cylinder, sphere) and the bounding box coordinates. This simple setup allows the YOLOv8 model to efficiently process the data for training, validation, and testing. This version is brief, clear, and directly aligned with the project's goals of improving YOLOv8 for better speed and accuracy in object detection.

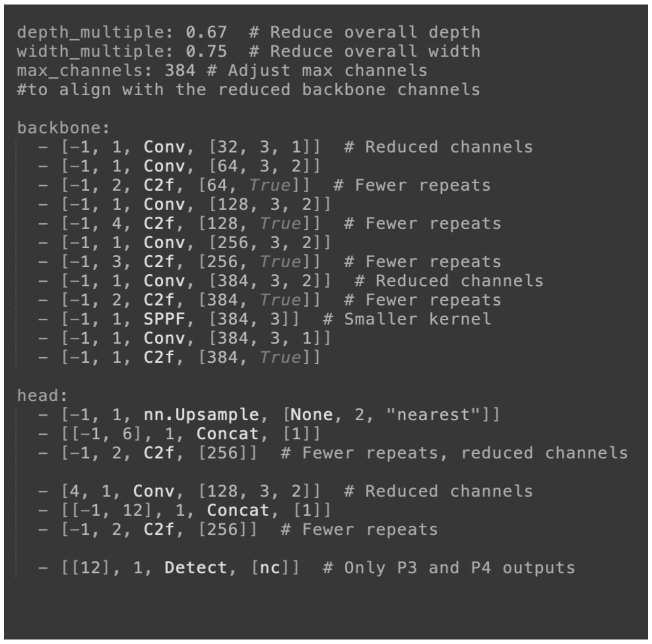
---

3. Modifying YOLOv8 Architecture for CLEVR-Hans Dataset

I made several key modifications to enhance YOLOv8 performance on the CLEVR-Hans dataset, focusing on speed and efficiency.



```
depth_multiple: 1
width_multiple: 1
max_channels: 512

# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 3, C2f, [512]] # 12

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 3, C2f, [256]] # 15 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]] # cat head P4
  - [-1, 3, C2f, [512]] # 18 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 9], 1, Concat, [1]] # cat head P5
  - [-1, 3, C2f, [1024]] # 21 (P5/32-large)

  - [[15, 18, 21], 1, Detect, [nc]] # Detect(P3, P4, P5)
```



```
depth_multiple: 0.67  # Reduce overall depth
width_multiple: 0.75  # Reduce overall width
max_channels: 384 # Adjust max channels
#to align with the reduced backbone channels

backbone:
  - [-1, 1, Conv, [32, 3, 1]]  # Reduced channels
  - [-1, 1, Conv, [64, 3, 2]]
  - [-1, 2, C2f, [64, True]]  # Fewer repeats
  - [-1, 1, Conv, [128, 3, 2]]
  - [-1, 4, C2f, [128, True]]  # Fewer repeats
  - [-1, 1, Conv, [256, 3, 2]]
  - [-1, 3, C2f, [256, True]]  # Fewer repeats
  - [-1, 1, Conv, [384, 3, 2]]  # Reduced channels
  - [-1, 2, C2f, [384, True]]  # Fewer repeats
  - [-1, 1, SPPF, [384, 3]]  # Smaller kernel
  - [-1, 1, Conv, [384, 3, 1]]
  - [-1, 1, C2f, [384, True]]

head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]]
  - [-1, 2, C2f, [256]]  # Fewer repeats, reduced channels

  - [4, 1, Conv, [128, 3, 2]]  # Reduced channels
  - [[-1, 12], 1, Concat, [1]]
  - [-1, 2, C2f, [256]]  # Fewer repeats

  - [[12], 1, Detect, [nc]]  # Only P3 and P4 outputs
```

Original model vs Modified model

First, I reduced the number of layers from 286 to 141 by removing redundant layers. This cut down computational load without a significant loss in accuracy, allowing the model to focus on essential features for object detection. I adjusted the number of channels to streamline feature extraction, optimizing for critical attributes like color, shape, and material, which are important for this dataset.

Also fine-tuned the C2f blocks to help the model share information between layers more effectively and remove unnecessary data. This made the model simpler while still capturing important details about the objects. Simplifying the SPPF module improved speed and allowed the model to handle various object sizes more effectively. Finally, I optimized the detection layers for cubes, spheres and cylinders and this reduced complexity in predictions while keeping accuracy high and making the model faster and ideal for real-time detections.

---

4. Implementation

- **Google Colab Pro+** : I used Google Colab Pro+ for faster GPU access (A100 GPU) and longer runtimes which helped speed up the training process.
- **Google Drive Setup**: Google Drive was mounted in Colab, and a folder named "YOLOv8_Optimization" was created to store datasets, models, and outputs in an organized manner.
- **YOLOv8 Architecture**: The standard YOLOv8 architecture file (yolov8.yaml) was downloaded, and a custom version (Modify_YOLOv8.yaml) was directly uploaded from a Google Drive link.
- **Model Training**: Two models were trained:
    1. **Original YOLOv8**: Using the standard architecture.
    2. **Modified YOLOv8**: Trained with our customized architecture for better and faster detection.
- **Inference**: After training, inference was performed on test images, and the results were saved, including statistics and visual plots, to compare the performance of both the original and modified models.

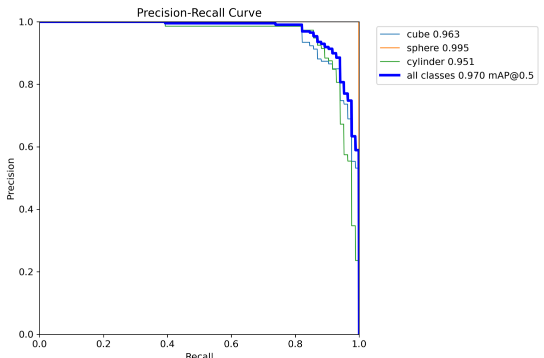
---

5. Experimentation and Results

Mean Average Precision (mAP)

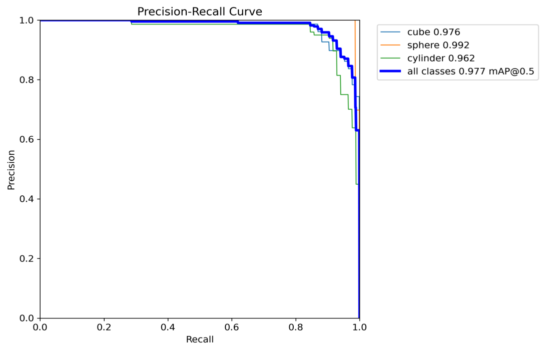
The key metric used for evaluating object detection performance is Mean Average Precision (mAP). It provides a comprehensive measure of the models ability to detect objects by computing the precision recall curve for each classes and averaging the results across them.

- **Original YOLOv8 Model**:
    - mAP50: **0.977**



- mAP50-95: **0.859**

- **Modified YOLOv8 Model**:
  - mAP50: **0.970**
  - mAP50-95: **0.848**



While there is a slight decrease in the mAP for the modified model, the trade-off in accuracy is compensated by significant improvements in speed and model size.

---

6. Inference Speed and Model Efficiency

The modifications made to the YOLOv8 architecture led to significant improvement on performance, particularly in inference speed and model size with minimal impact on accuracy. Below are the real performance metrics:

Metric	**Original YOLOv8**	**Modified YOLOv8**	**Improvement (%)**
Epoch Completion Time (hours)	0.107	0.057	46.73% faster
Inference Speed (ms)	3.6	1.9	47.22% faster
Model Size (MB)	79.3	9.3	88.27% smaller
mAP50	0.977	0.970	-0.72% (slightly lower)
mAP50-95	0.859	0.848	-1.28% (slightly lower)

These improvements show that the modified model is significantly faster and more efficient, making it ideal for real-time object detection applications.

---

7. Conclusion

As part of my computer vision course at Sapienza University, this project led me to explore YOLO versions in much more depth, helping me better understand how different model architectures work.

I focused on optimizing YOLOv8 to improve speed and efficiency by using trial and error method with training many models to better understand yolov8 and find out the best fit for our dataset, by reducing layers, optimizing channels, and simplifying components like C2f blocks and the SPPF module, I significantly lowered the model's computational load and size. Despite these changes, the modified model maintained high accuracy, with only a slight drop in performance (mAP), which was balanced by a substantial boost in speed and inference time. This makes the model ideal for real-time object detection applications where efficiency is key.

The results confirm that the modified YOLOv8 is highly effective in resource-limited environments. The faster inference speed, smaller model size, and shorter epoch completion time show the success of these changes, making it a solid choice for real-time use.