Yolo-v8 model
Auto Insurance Claim Damage & Severity Detection Project

Teresa Babío Núñez - 23201415
Daniel Quesada - 23201470

August 13, 2024

1 Introduction

Object detection is the task of identifying instances of objects belonging to specific classes within an image or video. It involves locating objects in an image using bounding boxes and assigning class labels to the detected objects. Essentially, object detection combines image classification and object localization. In addition to the classification output, it includes the bounding box coordinates that define the position and size of each object. This problem can be approached from either a neural network-based or non-neural perspective.

Object detection algorithms are broadly classified into two categories based on how many times the same input image is passed through a network:

- One-stage: a single pass of the input image to make predictions about the presence and location of objects in the image. It processes an entire image in a single pass, making them computationally efficient. YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process an image.
- Two-stage: the first pass is used to generate a set of potential object locations, and the second pass is used to refine these proposals and make final predictions. This approach is more accurate than single-shot object detection but is also more computationally expensive.

You Only Look Once (YOLO) is a state-of-the-art, real-time object detection algorithm introduced in 2015 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in their famous research paper "You Only Look Once: Unified, Real-Time Object Detection". The authors frame the object detection problem as a regression problem instead of a classification task by spatially separating bounding boxes and associating probabilities to each of the detected images using a single convolutional neural network (CNN). One important role to understand this model is the metrics used when defining the final bounding boxes.

The YOLOv8 model introduces significant advancements compared to its predecessors, with the most notable being the adoption of anchor-free detection. This shift enhances the efficiency of Non-Maximum Suppression (NMS), a crucial post-processing step that reduces the number of overlapping bounding boxes by retaining only the box with the highest confidence score. As a result, YOLOv8 can identify and locate objects in images and videos with impressive speed and accuracy. In addition to object detection and localization, YOLOv8 also supports image classification and instance segmentation, a feature that was not available in earlier versions—making it a versatile tool for various computer vision tasks.

Instance segmentation is a deep learning-driven computer vision task that predicts the exact pixel-wise boundaries of each individual object instance in an image.

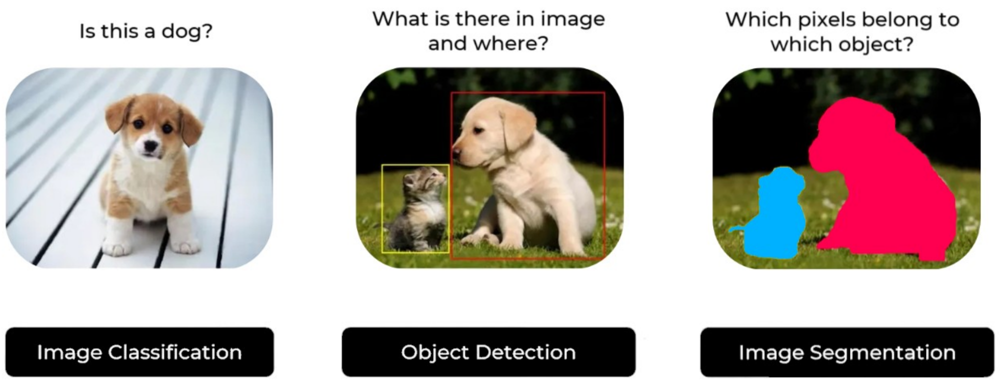

Is this a dog?
What is there in image and where?
Which pixels belong to which object?
Image Classification
Object Detection
Image Segmentation

Figure 1: Problems that can be solved by using YOLOv8

2 Metrics

Intersection over Union (IoU) is a common metric, that ranges between 0 and 1, used to evaluate the performance of an object detection algorithm. It measures the overlap between the predicted bounding box (P) and the ground truth bounding box (G).Higher IoU indicates the predicted bounding box coordinates closely resembles the ground truth box coordinates.

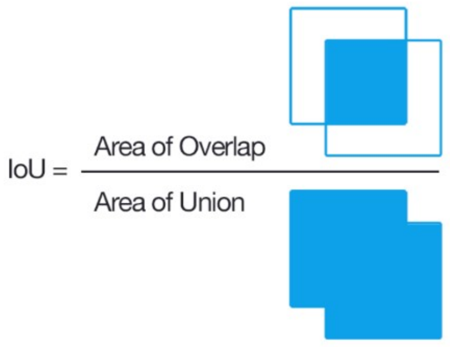


Figure 2: Intersection over Union (IoU) Illustration

The false positives are the bounding boxes whose IoU is smaller than the set IoU threshold, and the true positives are the predictions with an IoU higher than the threshold. With these definitions, recall and precision can be calculated.

These basic metrics lead to the definition of a metric that summarizes the precision-recall curve: the Average Precision (AP) of a class, which is calculated as the weighted mean of precisions at each threshold; the weight is the increase in recall from the previous threshold. The average of the AP metrics across all classes is referred to as Mean Average Precision (mAP). This metric incorporates the trade-off between precision and recall and considers both false positives (FP) and false negatives (FN). This property makes mAP a suitable metric for most detection applications.

Now that mAP is defined theoretically in our model, it will be used as follows:

- **mAP@50**: The model's precision-recall performance at a single IoU threshold of 0.50.
- **mAP@50-95**: (sometimes also referred to as mAP) averages the AP for all classes, where each AP is calculated over multiple IoU thresholds, from 0.50 to 0.95 in increments of 0.05.

3 Yolo model

3.1 YOLOv8 variants

The YOLOv8 is formed by different models (variants) that offers different trade-offs between accuracy, speed, and model size. The variants are divided based on the difference in the value of the parameters *depth_multiple* (d), *width_multiple* (w), and *max_channel* (mc), as it will be seen in Table 1. There are five variants:

1. n: smallest model, fastest inference but lowest accuracy.
2. s: small model, good balance of speed and accuracy.
3. m: medium model, higher accuracy than small models with moderate inference speed.
4. l: large model, highest accuracy but slowest inference.
5. xl: extra-large model, best accuracy for resource-intensive applications.

In the project, the m model is being used due to the difficulty of recognizing certain types of car damages. The values that characterize each variant are listed below:

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

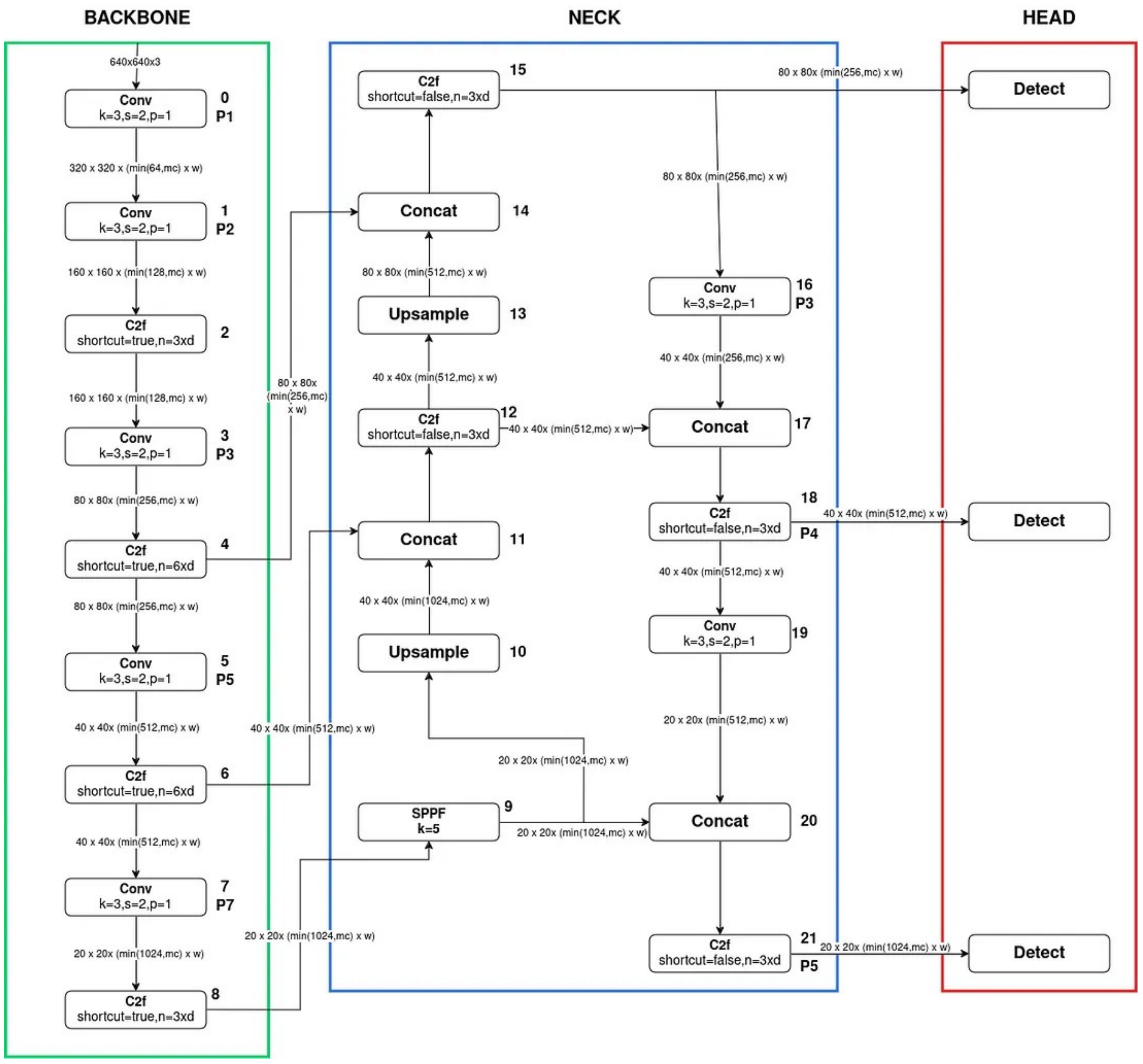
Table 1: Model Variant Parameters

- **depth_multiple** (d): This parameter controls the number of Bottleneck Blocks in the C2f block, determining the overall depth of the network. When this parameter is set to a value less than 1, it reduces the number of layers, resulting in a smaller and faster model, though this may come at the cost of accuracy. Conversely, setting the parameter to a value greater than 1 increases the number of layers, which enhances the model's capacity and potentially improves accuracy, but it also makes the model larger and slower to run.
- **width_multiple** (w): This parameter adjusts the number of channels in the convolutional layers, effectively scaling the network's width. A value less than 1 reduces the number of channels, making the network thinner, which leads to a smaller and faster model, though it may compromise accuracy. Conversely, a value greater than 1 increases the number of channels, widening the network. This results in a larger model with potentially higher accuracy but requires more computational resources.
- **max_channels** (mc): This parameter sets an upper limit on the number of channels allowed in the network. It is a safety measure to prevent the model from becoming too wide (i.e., having too many channels), especially when the *width_multiple* is set high. This can help control the model size and prevent overfitting.

3.2 Architecture

The neural networks used for localization problems have the same basic structure: backbone, neck, and head.

- The **backbone** is a Convolutional Neural Network (CNN) that acts as a feature extractor for the input image.
- The **neck** combines the features acquired from the various layers of the Backbone module.
- The **head** predicts the classes and the bounding boxes of the objects, which constitute the final output produced by the object detection model.



BACKBONE
NECK
HEAD
640x640x3
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=3xd 2
Conv k=3,s=2,p=1 3 P3
80 x 80x (min(256,mc) x w)
C2f shortcut=true,n=6xd 4
Conv k=3,s=2,p=1 5 P5
40 x 40x (min(512,mc) x w)
C2f shortcut=true,n=6xd 6
Conv k=3,s=2,p=1 7 P7
20 x 20x (min(1024,mc) x w)
C2f shortcut=true,n=3xd 8
SPPF k=5 9
Upsample 10
Concat 11
C2f shortcut=false,n=3xd 12
Upsample 13
Concat 14
C2f shortcut=false,n=3xd 15
Conv k=3,s=2,p=1 16 P3
Concat 17
C2f shortcut=false,n=3xd 18 P4
Conv k=3,s=2,p=1 19
Concat 20
C2f shortcut=false,n=3xd 21 P5
Detect
Detect
Detect

Figure 3: Yolov8 architecture.

Backbone

First, it is explained basic blocks used in the architecture.

3.2.1 Conv block

A convolutional block (Conv Block) is the most basic block in the architecture, which consists of the following:

- **Conv2d** layer (convolution applied to height and width). This layer is characterized by:
  - k: Number of filters or kernels.
  - s: Stride.
  - p: Padding.
- **BatchNorm2d** layer: A technique used to improve training stability and convergence speed by ensuring that the numbers passing through the network are not too large or too small.
- **SiLU** (Sigmoid Linear Unit) or **Swish** activation function, defined as $\text{SiLU}(x) = x \cdot \sigma(x)$, where $\sigma(x) = \frac{1}{1+e^{-x}}$.

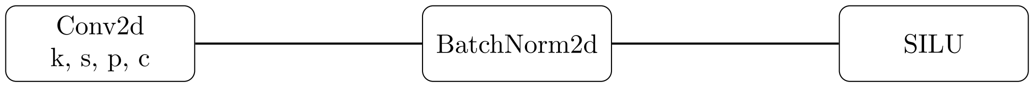


Figure 4: Basic Conv Block Structure

3.2.2 Bottleneck Block

The bottleneck block is a specialized structure within the neural network that incorporates a Conv Block along with a shortcut connection. The behavior of this block depends on the value of the *shortcut* parameter:

- If *shortcut* = true, the shortcut connection is implemented within the bottleneck block. This means that the input bypasses the Conv Blocks and is added directly to the output of the block.
- If *shortcut* = false, the input passes through two Conv Blocks in series without any shortcut connection.

Shortcut Connection: Also known as a skip connection or residual connection, a shortcut connection is a direct pathway that bypasses one or more layers in the network. This connection enables the gradient to flow more effectively through the network during training, helping to mitigate the vanishing gradient problem. By allowing the gradient to bypass certain layers, the model can learn more efficiently, particularly in deep networks.

In the context of a bottleneck block, the shortcut connection provides a mechanism for the model to bypass the convolutional layers when necessary. This allows the model to retain the original input (through identity mapping) if the convolutional layers do not contribute additional useful information. As a result, the model can more easily learn the identity function when it is beneficial, which can lead to improved performance, particularly in deeper networks.

In YOLOv8, the use of a shortcut connection within the bottleneck block is controlled by the `shortcut` parameter. This parameter can be set to `true` to enable the shortcut, allowing the input to bypass the convolutional layers, or set to `false` to force the input through the layers without bypassing. The setting of this parameter is typically specified in the model's configuration file or directly within the code that defines the network architecture.

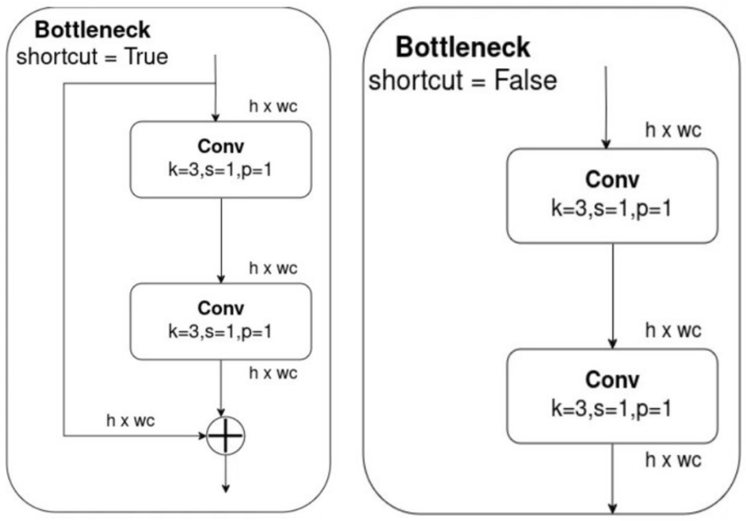


Figure 5: Bottleneck block

3.2.3 C2f block

The C2f block in deep learning architectures, such as YOLO, begins with a convolutional block that extracts features from the input data. This resulting feature map is then split into two paths: one is processed through a series of Bottleneck blocks, which reduce the feature map's dimensionality, and the other bypasses the Bottleneck blocks, going directly to a Concat block. The number of Bottleneck blocks used is controlled by the *depth_multiple* parameter, allowing for dynamic adjustment of the model's depth. After the Bottleneck block processes one part of the feature map, it is concatenated with the unprocessed part, combining detailed and abstracted features. The combined feature map is then passed through a final convolutional block, preparing it for the next stage of the model. This structure efficiently balances computational resources with the richness of feature representations, enhancing the model's ability to make accurate predictions.

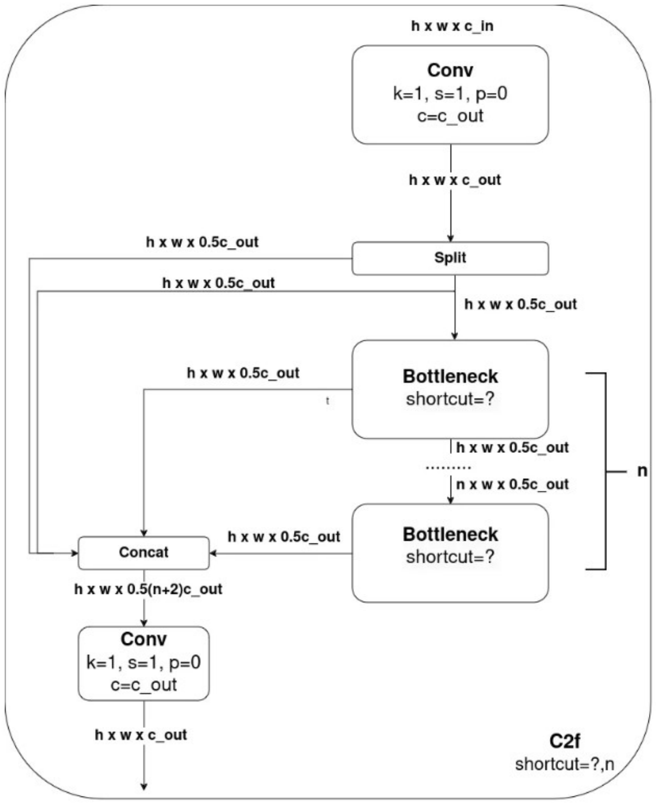


Figure 6: C2f block

Spatial Pyramid Pooling Fast (SPPF) Block

The SPPF (Spatial Pyramid Pooling Fast) block is a key component in certain deep learning architectures, designed to enhance the model's ability to process images of varying sizes. The block begins with a convolutional layer, which extracts features from the input image. These features are then passed through three sequential `MaxPool2d` layers. Each `MaxPool2d` layer downsamples the spatial dimensions of the feature map, capturing dominant features while reducing the computational complexity of the subsequent operations. The output from each `MaxPool2d` layer is then concatenated, combining multiple levels of spatial information into a single, richer feature map. This concatenated feature map is subsequently processed by a final convolutional layer, which prepares it for the next stage of the model.

The concept behind Spatial Pyramid Pooling (SPP) is to divide the input image into a grid and pool features independently from each grid cell. This approach allows the network to effectively handle images of different sizes, making it particularly useful in object recognition tasks where objects may appear at varying scales. However, traditional SPP can be computationally expensive due to the use of multiple pooling levels with different kernel sizes. SPPF, or SPP-Fast, simplifies this process by employing a more efficient pooling scheme—often using a single fixed-size kernel—thereby reducing the number of computations required while still capturing essential spatial information. The concatenation step in SPPF ensures that the model leverages a diverse set of spatial features, enhancing its ability to recognize objects at various scales.

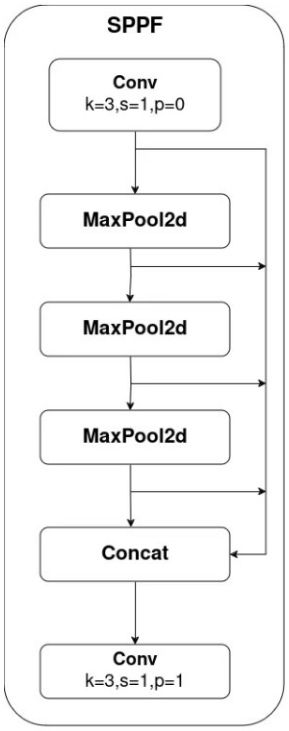


Figure 7: SPPF block

3.2.4 Detect block

The Detect Block in YOLOv8 is responsible for identifying and locating objects within an image. Unlike earlier versions of YOLO, YOLOv8 uses an anchor-free approach, which means it directly predicts the center of an object rather than calculating the offset from predefined anchor boxes. This anchor-free method simplifies the detection process by reducing the number of bounding box predictions, which speeds up the post-processing phase where candidate detections are filtered.

This block operates through two distinct paths: one for predicting bounding boxes and the other for predicting classes. Each path consists of two convolutional blocks that extract and refine the necessary features. These are followed by a single `Conv2d` layer in each path that outputs the final Bounding Box loss and Class Loss, allowing the model to accurately predict both the location and the category of each object in the image.

Anchor Box: An anchor box is a predefined rectangular shape used in object detection models to help the network predict the location and size of objects in an image. These boxes serve as reference points, with the model learning to adjust its predictions relative to these predefined shapes. By using multiple anchor boxes of different sizes and aspect ratios, the model can better detect objects of various scales within an image. Anchor boxes are particularly useful in scenarios where objects vary significantly in size and position, helping the model generalize better across different types of objects.

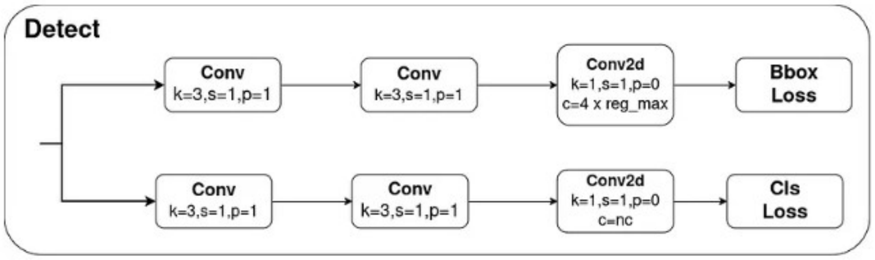


Figure 8: Detect block

3.2.5 Backbone

In Block 0, the processing starts with the input image of size $640 \times 640 \times 3$, which is fed into a convolutional block with a kernel size of 3, stride of 2, and padding of 1. The spatial resolution is reduced when $stride = 2$ is used. As a result, the convolutional block produces a feature map of 320×320, because the kernel moves in 2-pixel increments.

To determine the output channels of the convolutional block, the following formula is used:

$$\min(64, mc) \times w.$$

Here, 64 is the base output channel, mc is the *max_channel*, and w is the *width_multiple*. For example, if we are using the YOLOv8 model variant n, the final output channel becomes $\min(64, 1024) \times 0.25 = 64 \times 0.25 = 16$. This calculation is performed for every convolutional block in the architecture.

Block 2 is a C2f block that contains two parameters: *shortcut* and (n). The *shortcut* parameter is a boolean that indicates whether the Bottleneck block uses a shortcut connection. The parameter (n) determines how many bottleneck blocks are used inside the C2f block. In the case of Block 2, (n) is calculated as:

$$(n) = 3 \times d,$$

where d is the *depth_multiple*. For example, in the YOLOv8 model variant n, the *depth_multiple* is 0.33 (as shown in Table 1), so the number of bottleneck blocks used inside the C2f block becomes $(n) = 3 \times 0.33 = 0.99$, which rounds to 1 bottleneck block. In the C2f block, the resolution of the feature map and the output channels remain unchanged.

In the C2f blocks of the backbone, such as those in Blocks 4 and 6, part of the split outputs goes to the concat block in the neck.

In Block 9, the SPPF Block is used after the last convolutional layer of the C2f block in the Backbone. The main function of the SPPF block is to generate a fixed feature representation of objects at various sizes within an image, without resizing the image or introducing spatial information loss. This allows the model to effectively detect objects of different sizes while preserving the original structure of the image.

3.2.6 Neck and Head Section

The neck section is responsible for upsampling the feature map and combining the features acquired from the various layers of the Backbone section. Each time a C2f block is applied along the backbone structure, the output is smaller in size than the input. Therefore, to join the data coming from different stages of the C2f blocks with different sizes, it is necessary to upsample the output data from the deeper C2f blocks.

The upsample layer in the Neck section doubles the spatial dimensions (height and width) of the feature map without altering the number of output channels. In YOLO's Neck, this layer typically uses techniques like nearest-neighbor interpolation, bilinear interpolation, or deconvolution to achieve the upsampling.

The Concat Block sums the output channels of the blocks that are being concatenated without changing the resolution. For example, in Concat 4, the output of the C2f block in Block 4 is joined with the upsampled output in Block 13.

The head section is responsible for predicting the classes and bounding boxes of the objects, which constitutes the final output produced by the object detection model. The first Detect block in the Head section specializes in detecting small objects, using input from the C2f block in Block 15. The second Detect block specializes in detecting medium-sized objects, using input from the C2f block in Block 18. The third Detect block specializes in detecting large objects, using input from the C2f block in Block 21. Large objects are the most challenging to detect, which is why the model places more emphasis on their detection within its architecture.

3.3 How does it work?

The summarize steps of yolo models in general are the following ones:

1. **Input Processing:** The input image is divided into a grid (typically with a size of 13×13 or 26×26). Each grid cell is responsible for predicting objects within its spatial domain. In our example, the grid is of size 4×4. With the existence of a grid, it is possible to detect one object per grid cell instead of one object per image. For each grid cell, a vector can be encoded to describe the cell.

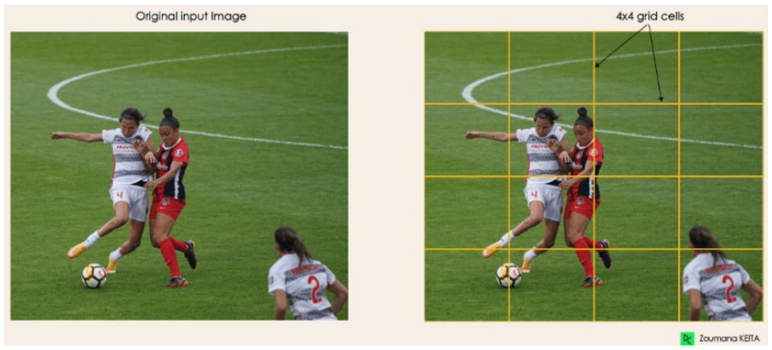


Figure 9: Image divided by the grid.

2. **Bounding Box Regression:** The next step is to determine the bounding boxes, which correspond to rectangles highlighting all the objects in the image. YOLOv8 assigns a confidence score to each object class, indicating the likelihood of that class being present in the bounding box. The classification score (in this case, there are two labels: player and ball) ranges from 0.0 to 1.0. These confidence levels capture the model's certainty that there exists an object in that cell and that the bounding box is accurate. The output for a two-label classification will be:

$$Y = [pc, bx, by, bh, bw, c1, c2]$$

   - pc corresponds to the probability score of the grid cell containing an object. For instance, all the grids in red will have a probability score higher than zero. The image on the right is a simplified version, where the probability of each yellow cell is zero (insignificant).
   - bx, by are the x and y coordinates of the center of the bounding box relative to the enveloping grid cell.

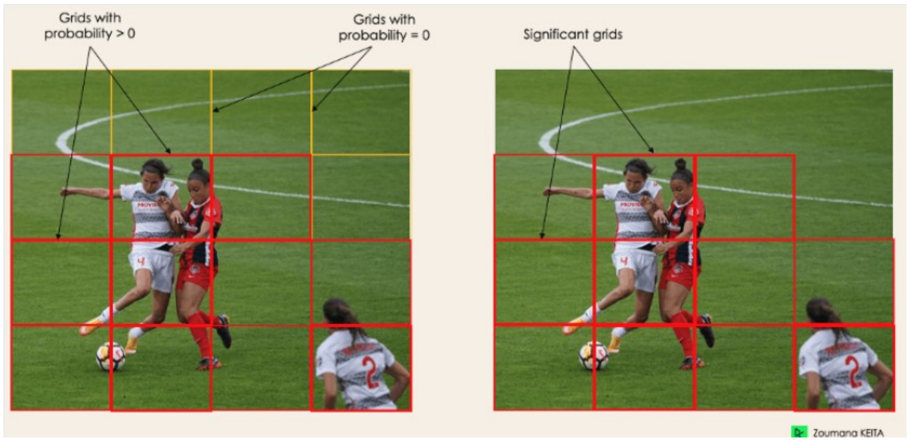
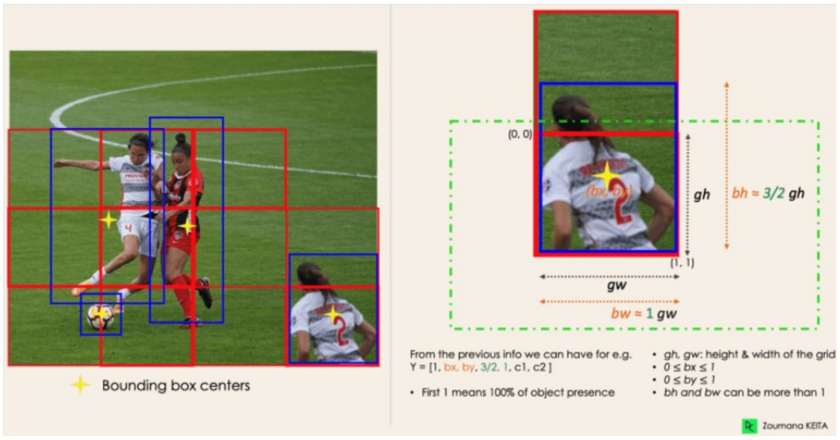


Figure 10: Cells with a probability $pc > 0$.

- bh, bw correspond to the height and width of the bounding box relative to the enveloping grid cell. For example, if the height of the grid is h and the height of the bounding box output is $\frac{3}{2}$, then the actual height of the bounding box is $\frac{3}{2} \times h$.
- $c1$ and $c2$ correspond to the two classes. The number of classes can be adjusted according to the specific use case.



From the previous info we can have for e.g.
Y = [1, bx, by, 3/2, 1, c1, c2]

- First 1 means 100% of object presence

- gh, gw: height & width of the grid
- $0 \leq bx \leq 1$
- $0 \leq by \leq 1$
- bh and bw can be more than 1

Figure 11: On the left, the grid cells are shown in red, while the bounding boxes obtained from the regressions are highlighted in blue. On the right, the outputs of the predictions are explained.

3. **Intersection Over Union (IOU):** A single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The IOU is used to discard irrelevant grid boxes, retaining only those that are relevant. Users define a threshold, and YOLO computes the IOU for each grid cell. Finally, predictions from grid cells with an IOU $\leq$ threshold are ignored, while those with an IOU $>$ threshold (where the intersection of the bounding boxes and their corresponding cell divided by the union of these two exceeds the threshold) are considered.

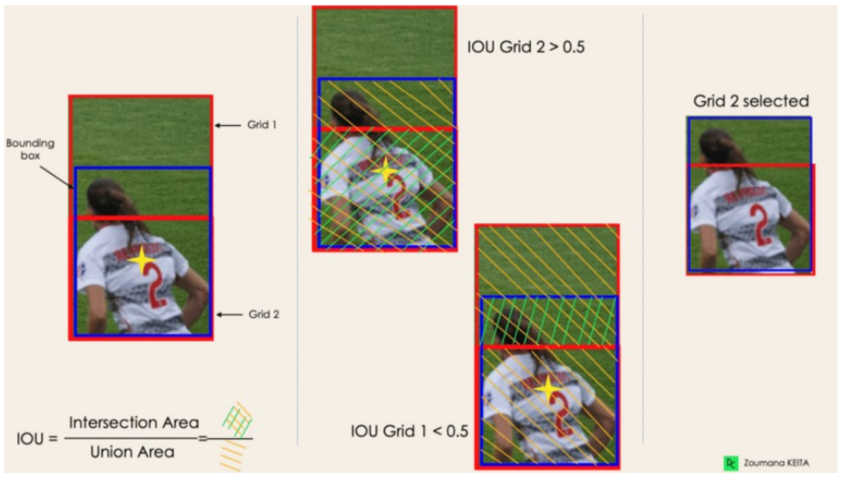


Figure 12: An example applying the grid selection process to the bottom-left object. The object originally had two grid candidates, but only "Grid 2" was selected in the end.

4. **Non-Max Suppression (NMS):** Even after applying IOU, an object might still have multiple bounding boxes with IOU values above the threshold, which could introduce noise if all boxes are retained. Non-Max Suppression (NMS) addresses this by keeping only the bounding box with the highest probability score for each object. The process involves:

    - Sorting all bounding boxes by their confidence scores in descending order.
    - Selecting the box with the highest confidence score and removing all other boxes that have a high overlap (IOU) with this box.
    - Repeating the process for the remaining boxes, ensuring that only the most relevant bounding boxes are retained.

    This method ensures that only the most accurate bounding box is selected for each object, reducing the possibility of redundant or overlapping detections.

It is mentioned that one object can be predicted per grid cell. However, it doesn't necessarily have to be just one object. For instance, it can be configured to detect multiple objects per cell. If two bounding boxes are detected per cell, the vector that defines the cell will have twice as many components, representing the probability of each object and the dimensions of the boxes:

$$C_{1,1} = (P_c^1, B_x^1, B_y^1, B_w^1, B_h^1, P_c^2, B_x^2, B_y^2, B_w^2, B_h^2, C_1, C_2),$$

Similarly, instead of predicting only two classes, this method can be extended to handle multi-class classification problems. In general, each cell can be described using a vector with dimensions $B \times 5 + C$, where B is the number of bounding boxes and C is the number of classes. If the image is divided into an $S \times S$ grid, it can be represented as a tensor with dimensions $S \times S \times (B \times 5 + C)$.

3.4 Loss functions

To understand whether a prediction is accurate or not, we use a Loss function or Cost Function. This is a mathematical function that evaluates how far off the prediction is from the original value. The estimated loss is then used to update the model parameters through an optimizer, such as Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (ADAM). The loss function used in YOLOv8-seg is defined as a weighted sum of three individual losses. The weights control the relative importance or contribution of each loss to the final combined loss, which is optimized during training. This approach enables the model to perform both object detection and semantic segmentation simultaneously:

- **Bounding Box Loss:** This loss calculates the error between the predicted and ground truth boxes' geometry. It measures how well the model predicts the size and location of the bounding boxes.
- **Objectness Loss:** This loss determines how confident the model is about the presence of an object within the bounding box. It contrasts the model's predicted probability of an object being present with the ground truth value.
- **Segmentation Loss:** This loss quantifies how close the predicted segmentation map is to the ground truth map. It measures how effectively the model performs the semantic segmentation task.