Building YOLO Algorithm for Object Detection from Scratch

Sohaila Kandil[a], Mennatullah Ashraf[a]

[a] *Egypt Japan university of science and technology,*

Abstract

In our project, we implemented the YOLO version 1 algorithm for object detection on the Pascal dataset. We followed the algorithm's design as outlined in the paper ”You Only Look Once: Unified, Real-Time Object Detection” (1). The implementation was carried out using Python and the PyTorch library for training the YOLO model on the Pascal dataset (2).

Keywords: YOLO, CNN, Forward propagation, Backward propagation, feature map

1. Introduction

In image processing, our primary tasks involve image classification and image localization. Image classification aims to identify the classes of objects within an image, while image localization focuses on pinpointing the location of these objects in the image. The YOLO algorithm is specifically designed for object detection and localization, accomplishing this by detecting object classes and localizing them using bounding boxes.

The YOLO algorithm operates by dividing the image into a grid of cells ($S \times S$), where S is an integer, and in our case, $S = 7$, as illustrated in Figure 1. A Convolutional Neural Network (CNN) model is then employed to produce a matrix containing information for each cell in the image. This information encompasses the class associated with the cell, the probability that this cell might contain a center of an object, and the coordinates (x, y) of the bounding box's center relative to the cell, along with the width (w) and height (h) of the bounding box.

To enhance the YOLO algorithm, we assign each cell the responsibility of detecting two bounding boxes, allowing us to train both. The final output selects the box with a higher probability of containing an object in the cell.

This means that the output of the YOLO network for one image will be a matrix of dimension $(S \times S, C + B \times 5)$, where c is the number of classes existing in the dataset, and B is the number of bounding boxes per cell, which is two in our case.

Following the assignment of one bounding box detection responsibility to each cell, we proceed to filter the resulting bounding boxes based on the probabilities of an object's presence in each cell. This reduction process employs a technique known as non-maximum suppression, which will be elaborated upon in Section 1.3.

1.1. YOLO network design

The YOLO network consists of a Darknet (CNN) network. The output of this network is then forwarded to a fully connected layer to produce the model's predictions.

Following the architecture depicted in Figure 2, RGB images with three channels and dimensions of 448×448 are used as input. We apply 64 kernels of size 7×7 to the image with a stride of 2 and padding of size 2×2. The output of the first layer is then of size $224 \times 224 \times 64$. Progressing through the network, the output shape eventually becomes $7 \times 7 \times 1024$. This input signifies the probability for 1024 features to exist in each cell of the image, with the total number of cells being $7 \times 7 = 49$.

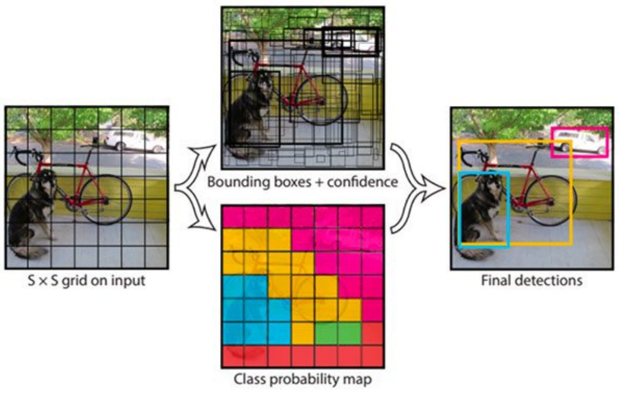
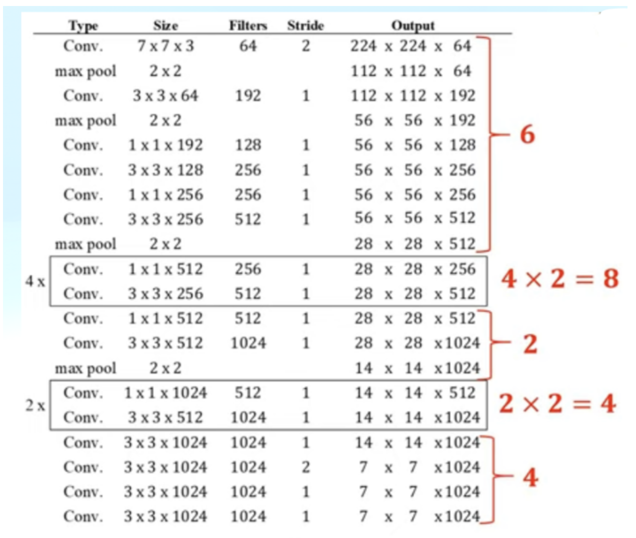


Figure 1: Extracting the feature map of an image to get bounding boxes predictions (1)

Afterward, the matrix is flattened into a 1D vector of size 50176, as illustrated in Figure 3. This vector is then input into a fully connected layer with an output size of 4096. It is important to note that due to limitations in time and computational power, the output of the fully connected layer has been constrained to a size of 496 in our implementation. Subsequently, the output is reshaped to a size of $7 \times 7 \times 30$. This final output constitutes the feature map of the image.

The feature map represents the bounding boxes of each cell in the image. The first two dimensions of the feature matrix denote the cells in the image, while the third dimension represents the bounding boxes for each cell. The shape of the bounding boxes in the third dimension is structured as follows: $[p_0, p_1, \ldots, p_{19}, c_0, x_1, y_1, w_1, h_1, c_2, x_2, y_2, w_2, h_2]$. Here, p_i represents the probability that class i is the class of the ob-



	Type	Size	Filters	Stride	Output	
	Conv.	7 x 7 x 3	64	2	224 x 224 x 64	6
	max pool	2 x 2			112 x 112 x 64	
	Conv.	3 x 3 x 64	192	1	112 x 112 x 192	
	max pool	2 x 2			56 x 56 x 192	
	Conv.	1 x 1 x 192	128	1	56 x 56 x 128	
	Conv.	3 x 3 x 128	256	1	56 x 56 x 256	
	Conv.	1 x 1 x 256	256	1	56 x 56 x 256	
	Conv.	3 x 3 x 256	512	1	56 x 56 x 512	
	max pool	2 x 2			28 x 28 x 512	
4 x	Conv.	1 x 1 x 512	256	1	28 x 28 x 256	$4 \times 2 = 8$
	Conv.	3 x 3 x 256	512	1	28 x 28 x 512	
	Conv.	1 x 1 x 512	512	1	28 x 28 x 512	2
	Conv.	3 x 3 x 512	1024	1	28 x 28 x1024	
	max pool	2 x 2			14 x 14 x1024	
2 x	Conv.	1 x 1 x 1024	512	1	14 x 14 x 512	$2 \times 2 = 4$
	Conv.	3 x 3 x 512	1024	1	14 x 14 x1024	
	Conv.	3 x 3 x 1024	1024	1	14 x 14 x1024	4
	Conv.	3 x 3 x 1024	1024	2	7 x 7 x1024	
	Conv.	3 x 3 x 1024	1024	1	7 x 7 x1024	
	Conv.	3 x 3 x 1024	1024	1	7 x 7 x1024	

Figure 2: convolutional neural networks in YOLO architecture (3)

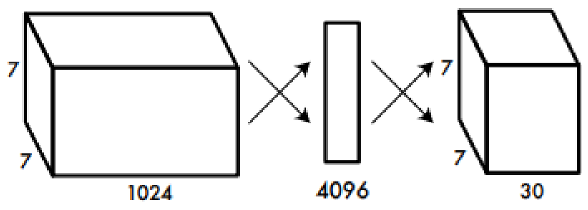


Figure 3: The fully connected layer in the YOLO network (1)

ject predicted by the cell for i in the range [0, 19], as the Pascal dataset we are working on comprises 20 classes. x_i, y_i signify the center points of bounding box i for the cell, where the left upper corner edge of the cell is represented by $(x, y) = (0, 0)$, and the right upper corner of the cell is represented by $(x, y) = (1, 1)$. w_i, h_i represent the width and height of bounding box i as perceived by the cell.

1.2. YOLO loss function

In order to train the yolo algorithm It is important to calculate the error using an error function and then optimize the model parameters using backward propagation.

The following equation calculates the location error for the bounding boxes:

$$L_{\text{loc}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \tag{1}$$

$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right], \tag{2}$$

Where λ_{coord} is a weight assigned to the location error, $\mathbb{1}_{ij}^{\text{obj}}$ is a vector of size $S \times S$ such that $\mathbb{1}_{ij}^{\text{obj}}[i] = 1$ if bounding box j at cell i is responsible for predicting an output, and $\mathbb{1}_{ij}^{\text{obj}}[i] = 0$ otherwise. (x_i) represents the predicted x-coordinate for the center point, $\hat{x}_i$ is the actual x-coordinate of the bounding box, (y_i) is the predicted y-coordinate for the center point, $\hat{y}_i$ is the actual y-coordinate of the bounding box. The same convention applies to the weights $\hat{w}_i$ and heights $\hat{h}_i$ of bounding box i.

In this equation, we calculate the mean squared error for the location of bounding boxes at each cell i (for i from 1 to S^2), but only consider the error of bounding boxes from cells responsible for predicting them. This is achieved using the indicator function $\mathbb{1}_{ij}^{\text{obj}}$. The points (x_i) and (y_i) represent the center point coordinates of the best bounding box (the one with the highest probability c_i) for each cell.

The Probability error of the bounding boxes is represented in the following formula:

$$L_{\text{conf}} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \right], \tag{3}$$

Where λ_{noobj} is a constant number. This error is calculated by computing the mean square error of the probability errors in the bounding boxes. The calculation includes negative errors for those bounding boxes detected using $\mathbb{1}_{ij}^{\text{obj}}$ and positive errors for bounding boxes detected using $\mathbb{1}_{ij}^{\text{noobj}}$. Once again, the points (x_i) and (y_i) represent the center point coordinates of the best bounding box (the one with the highest probability c_i) for each cell.

The class error is calculated using the following function

$$L_{\text{cls}} = \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2, \tag{4}$$

The overall loss for the predicted bounding box outputs is computed by summing up the earlier individual losses in the subsequent equation:

$$L_{\text{total}} = L_{\text{loc}} + L_{\text{conf}} + L_{\text{cls}}, \tag{5}$$

1.3. Getting Bounding Boxes from the Feature Map

Now after getting the feature map for the image the output will seem something like the upper middle image in figure 1 where each cell is outputing a bounding box. what we now want to do is to execlude all the unncecssary bounding boxes. we are going to use non max suppression algorithm for that purpose.

2. Methodology

The YOLO algorithm was applied to the PASCAL dataset, which consists of 16,550 images for training and 4,952 images for testing. For computational efficiency and time considerations, only 100 images will be used for training. Each image may feature one or more of the following 20 objects for classification: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, and TV monitor.

The bounding boxes for each image are represented by four values: the class number predicted by the bounding box, along with the relative coordinates (x, y, w, h) of the bounding box. These coordinates are relative to the entire image, with the upper-left corner represented by the point (x, y) = (0, 0) and the lower-right corner represented by the point (1, 1).

The code predicting the bounding boxes fails in the following four steps:

1. Designing yolo model
2. Defining the loss function
3. Loading and reprocessing the dataset
4. Training the model
5. Testing

2.1. *Designing yolo model*

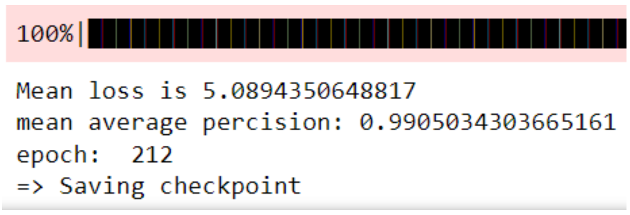
For constructing the yolo model we followed the following algorithm:

1. Construct darknet network: construct an array that contains specifications of each convolutional layer in the model based on the specifications early discussed in section 1.1. each index of its indexes contain a tuple of size four of the form (kernel size , number of filters filters , stride , padding)
2. The output of each convolutional layer undergoes batch normalization to normalize the data. Following this, the leaky ReLU function is applied to each convolutional layer to avoid linearity.
3. Create the convolutional layers based on the specifications in step one and two.
4. Create the Fully connected layers based on the specifications in section 1.1 except that the first hidden layer in this network is of size 496 instead of 4096 in order to time and computation limitations.
5. Create the Fully connected layers based on the specifications in section 1.1 except that the first hidden layer in this network is of size 496 instead of 4096 in order to time and computation limitations.Then apply the leaky ReLU function on this 496 size layer.
6. define the forward propagation function as forward. In this function the inputed data will be first fed to the darknet network, then the output will be flatened and fed to the fully connected layer.

2.2. *Loss Function Definition*

To calculate the loss of the bounding boxes tensor outputed from our model we apply the following algorithm:

1. Define the model's hyperparameters: $\lambda_{\text{coord}} = 5$, $\lambda_{\text{noobj}} = 0.5$, chosen from the paper. These values emphasize training the midpoint of bounding boxes while minimizing the impact of positive error bounding boxes in our predictions.
2. Create a $\mathbb{1}_{ij}^{\text{noobj}}$ tensor, where indexes are set to 1 (indicating the responsibility of the bounding box at this index for detecting the object) if the corresponding bounding box P value is one in the target bounding boxes vector.



```
100%|████████████████████████████████████████

Mean loss is 5.0894350648817
mean average percision: 0.9905034303665161
epoch:  212
=> Saving checkpoint
```

Figure 4: Convolutional neural networks in YOLO architecture

3. Construct a tensor containing predicted bounding box coordinates (x, y, square root of w, square root of h) after non-max suppression, considering only one bounding box with the highest probability for each cell.
4. Multiply the predicted bounding boxes tensor from step 3 by the $\mathbb{1}_{ij}^{\text{noobj}}$ tensor to exclude all positive error bounding boxes.
5. Repeat steps 3 and 4 for the target bounding boxes tensor, without the non-max suppression step, as the target vector already contains one bounding box for each cell, representing the true bounding box for that cell.
6. Calculate the mean square difference between the predicted and target coordinate tensors from steps 4 and 5, respectively, to obtain the coordinate error of the predicted bounding boxes.
7. Calculate the class and probability errors as specified in section 1.2.
8. add the error of the coordinates class and probability errors to obtain the final loss value of the prediction.

2.3. *Training the Model*

1. Load the dataset and set the image size to 448 * 448 * 3 channels (RGB), matching the input size specified for the YOLO network in Section 1.1.
2. Divide the data into batches of size 12 using a data loader. Batching enhances model accuracy and accelerates training.
3. Define a training function that processes the data from the data loader, trains it using the YOLO model defined in Section 2.1, and employs backward propagation.
4. Repeat step 3 until reaching the maximum number of iterations (epochs), set to 600, or until the mean average precision of bounding boxes reaches 0.99 and the number of epochs passed is greater than or equal to 200.
5. Begin tracking the maximum mean average precision once it reaches 0.9. Save the model at each instance when a new maximum value for mean average precision is achieved during training iterations.

3. Results

The final mean average precision we obtained from training the data set is 0.9905 and the mean loss is 5.089 as shown in figure 4.

after plotting some images from the dataset we obtained the images shown in figures some of which are correctly classified and the bounding box on them is accurate and some of them are not.

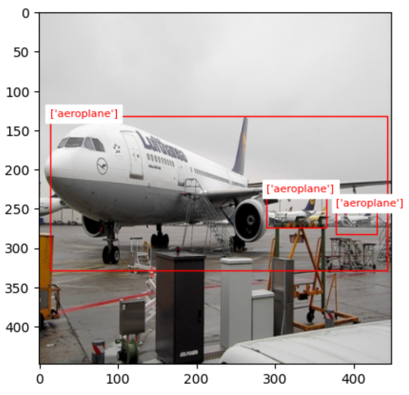


Figure 5: A correct output

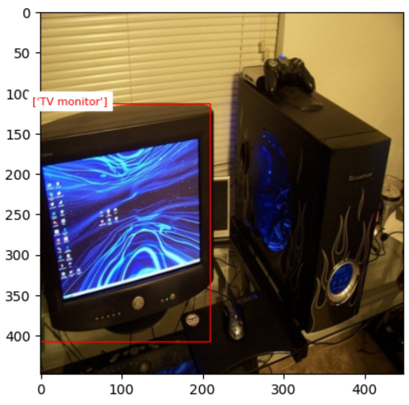


Figure 6: A correct output

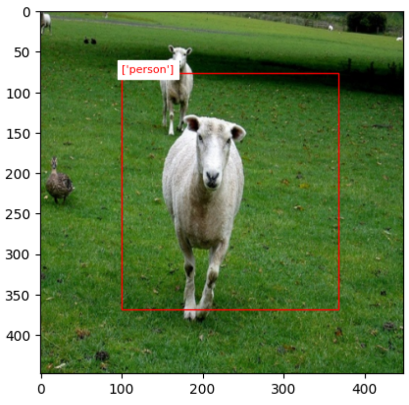


Figure 7: bounding box is correct the the prediction is not correct

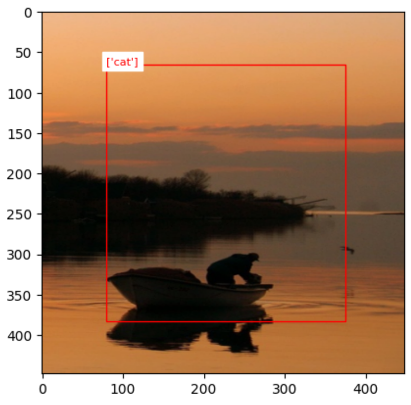


Figure 8: bounding box is not accurate and the the prediction is not correct

References

[1] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[2] "Pascal visual object classes (voc) challenge," http://host.robots.ox.ac.uk/pascal/VOC/, 2005, accessed: January 3, 2024.

[3] MLForNerds. Yolo object detection tutorials. Accessed: January 3, 2024. [Online]. Available: https://github.com/MLForNerds/YOLO-OBJECT-DETECTION-TUTORIALS/blob/main/YoloV1.pdf