Comparison of the results of Yolo algorithm implemented with different CNN

Nikola Andrić Mitrović
nikola.andricmitrovic@studenti.unipd.it

Daniele Trentin
daniele.trentin.2@studenti.unipd.it

Alessandro Cusinato
alessandro.cusinato@studenti.unipd.it

Abstract

YOLOv1 is a single-stage object detection algorithm that has been shown to be effective for a variety of tasks. The training for this particular architecture usually involves on two stages, a first training on image classification, which involves only partially the whole model, and a second one on the object detection task. In this paper, we implement YOLOv1 from scratch using different backbones in order to skip the first step of training and speed up the process. We tried with VGGNet16 and ResNet18 comparing the results of the the different implementations using different parameters on the algorithm and different hyperparameters for training. The entire process was based on the PASCAL VOC 2012 dataset. Our results shown that these chosen backbones don't reach the minimal standard to be a good backbone for this particular task. They are not complex enough to learn all the features needed to get state-of-art object detection, but their results reveal how the different parameters and hyperparametrs influence the learning process in this kind of implementation.

1. Introduction

Object detection is one of the main problems in computer vision that involves identifying and locating objects within an image or video. It is a core component of many computer vision applications, such as self-driving cars, surveillance systems, and image search engines. In our project, we chose to incorporate a YOLO algorithm to develop an advanced object detection model. To optimize the training process and assess various outcomes, we used transfer learning from renowned neural networks such as ResNet and VGGNet, which were pre-trained for object recognition. This approach not only accelerated training but also enabled us to analyze the diverse results obtained.

YOLO (You Only Look Once) is a real-time object detection algorithm that use a single deep convolutional neural network (CNN) that become popular due to its speed and accuracy. It was introduced for the first time in 2016 by by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi in their paper "You Only Look Once: Unified, Real-Time Object Detection". One of the key features of YOLO is its use of anchor boxes, these are pre-defined bounding boxes that are used to detect objects. The procedure splits the image into a grid and each cell is responsible for predicting the bounding box coordinates for the objects it contains. YOLO also predicts the probability that each cell contains an object, as well as the class of the object.

ResNet and VGGNet are both convolutional neural network architectures used in computer vision tasks such as image classification. ResNet (Residual Network) is a deep neural network architecture introduced by Microsoft Research that utilizes residual connections, which allow information to bypass a few layers and directly propagate to subsequent layers. This approach enables the network to learn more effectively and improves gradient flow during training. VGGNet (Visual Geometry Group Network) on the other hand was developed by the Visual Geometry Group at the University of Oxford. It is characterized by its uniform architecture, where several layers with smaller-sized filters are stacked together. We imported for both network a version pre-trained over ImageNet dataset.

Our experiment revealed that VGG16 and ResNet18 models are not complex enough for the object detection task. VGG16 performed slightly better than ResNet18. Certain parameters and new features like AdamW and cosine annealing learning rate had a significant impact on the results. Striking a balance between effectiveness and computational demands is crucial.

2. Related work

Our aim was to recreate the original algorithm of the first version of YOLO following the paper published in 2016 by Joseph Redmon. Compared to the original work, our approach focuses on using transfer learning both to im-

prove the performance of YOLO in object recognition and to lighten the training process.

In the field of computer vision, there are numerous alternatives to YOLO. Here are the alternatives we considered most valid:

- **SSD (Single Shot MultiBox Detector)**[1]: SSD is another object detection algorithm known for its speed and accuracy. It uses a series of feature maps of different sizes to detect objects of various scales within the image. SSD also utilizes a set of anchor boxes to improve the precision of object predictions.
- **Faster R-CNN (Region-based Convolutional Neural Network)**[2]: Faster R-CNN is an object detection algorithm that combines a region-based approach with a convolutional neural network (CNN). It uses a CNN to extract features from the image and then proposes promising regions to search for objects. It is known for its high accuracy but may be slower compared to other algorithms.
- **RetinaNet**[3]: RetinaNet is an object detection algorithm designed to address the trade-off between accuracy and speed. It uses a structure called a "feature pyramid network" (FPN) to detect objects of different scales within the image. Additionally, it employs a technique called "focal loss" to effectively handle the problem of a high ratio of background regions to object-containing regions.

Although there are many alternatives, we decided to base our project on YOLO because we wanted to create a highly efficient model without sacrificing too much on accuracy, and YOLO allows for such a compromise.

3. Dataset

The **PASCAL VOC**(Visual Object Classes) **Dataset** is a comprehensive collection of over 17000 images. The images have different size and contains one or more visible objects, which make that database particularly suitable for our purpose. The objects are organized into 20 different categories: aeroplanes, bicycles, boats, buses, cars, motorbikes, trains, bottles, chairs, dining tables, potted plants, sofas, TV/monitors, birds, cats, cows, dogs, horses, sheep, and people. Each image is paired with information about the image itself and the objects it contains, specifically, for our task, we utilized the class of each object and the four parameters (x_1, x_2, y_1, y_2) that define the coordinates over x and y directions of the bounding box(which is a rectangle).

3.1. Preprocessing

In order to make the element of this database suitable for our task we preprocessed both, labels and images, of all the elements of the dataset.

Following the original paper we first converted the coordinates of the boxes. In fact the original algorithm works storing the boxes using four parameters: coordinates of the center, height and width. That is functional because the algorithm first split each image into a grid and it needs to localize easily, thanks to the coordinates of the center, in which sector the object is located.

For what concerns images, we resized all of them into a resolution of 224x224 pixels, which is the input dimension for our neural network. We preferred that size instead of 448x448 because a small resolution allows a faster training computation and our task needs a lot of computing power(and time).

Following the resizing process, we modified all the labels to fit the modified images and than normalized all the values into a range $\{0, 1\}$ to make them more suitable to the deep convolutional architecture. For class labeling, we assigned a number between 0 and 19 to each class and kept a record of these assignments.

3.2. Data augmentation

The dataset turns out to be particularly extensive and complete, in fact it contains objects photographed from different perspectives, fully or partially and also(though less) under different lighting conditions. Neverthless as a result of the poor results of the first attempts and in an effort to provide more robust learning, especially with respect to brightness conditions and image quality, we decided to extend the training set. In order to do that we develop some simple transformation to modify the images and adapt the labels consequentially. The transformations we implemented are able to flip the images, to blur it using a filter, to change randomly light condition and colours and to zoom over a provided factor.

4. Methods

To implement the YOLO algorithm we followed the original paper[4] for what concern the overall implementation. The main differences are on the model, where we implemented transfer learning testing different backbones.

4.1. Model

The model described in the paper consists of 24 convolutional layers and 2 fully connected layers. In their implementation, the first 20 layers were pre-trained individually, while the remaining layers were added later and trained specifically for object detection. In our implementation, we opted to utilize ResNet and VGGNet as backbones for our network. These backbones were pre-trained for object classification. Following the backbone, we appended two fully connected layers, similar to the architecture described in the paper, and trained the entire model for object detection.

Visual Geometry Group Network(VGGNet)[5] is a deep convolutional neural network architecture that achieved state-of-the-art performance on the ImageNet(74.4% of accuaracy[6]). Considering that it is the same dataset used in the paper for pre-training, it seems a good candidate. The main characteristic of VGGNet is its uniform architecture and its depth, which allows it to capture complex hierarchical features from images. VGGNet architecture comes in different variants, named according to the total number of layers. We tested VGG16(16 weight layers) and VGG19(19 weight layers).

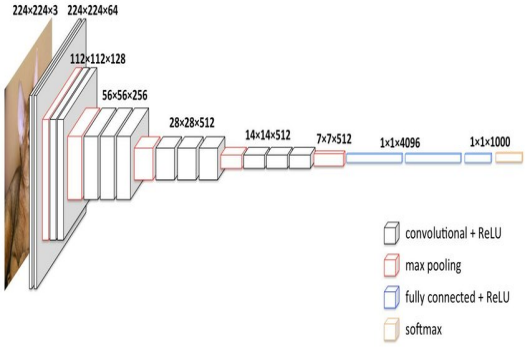


Figure 1. Architecture of VGG-16

Residual Neural Network(ResNet)[**?**] is a deep convolutional neural network architecture that was designed to address the problem of vanishing gradients and degradation in very deep neural networks. The key innovation of ResNet is the introduction of residual connections which allow the network to bypass one or more layers. This is helpful to prevent overfitting(the main issue we had to deal with) and facilitates the training of deeper architectures. Also ResNet architectures come in different variants, we focused on ResNet-18 and ResNet-50(over 76% accuracy on ImageNet[6]), trying to understand which class of complexity could be more suitable for our purpose.

4.2. Training

The YOLO algorithm is structured to split the image into grid cells and only the cell in which the center of an object resides, is the cell responsible for its detection. Each cell will predict a variable number of bounding boxes(usually 2) and a confidence score for each box. The confidence score represents the Intersection over union(IOU) between the predicted bounding box and the actual bounding box. The IOU, in practice, is the area of the intersection of the predicted and truth boxes divided by the area of the union of the same predicted and truth boxes. The prediction process usually give more than the expected boxes, even for

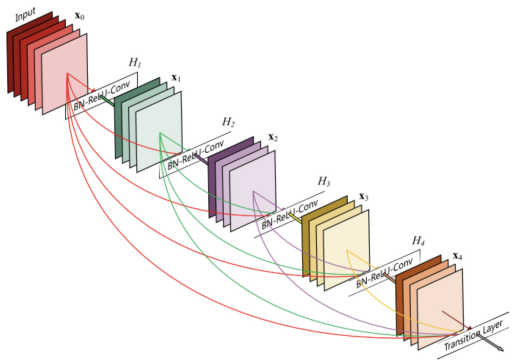


Figure 2. Overview on how the residual connection work

the same object, in these cases the best choice is made using non-maximum-suppression(NMS) over the confidence score. In addition to outputting bounding boxes and confidence scores, each cell predicts the class of the object. This class prediction is represented by a binary vector with many elements as the number of classes in the dataset. However, it is important to note that while each cell may predict any number of bounding boxes and confidence scores for those boxes, it only predicts one class. This is a limitation of the YOLO algorithm. Because there are SxS grid cells in each image, each one able to predict 5 parameters(x,y,w,h,confidence) for each of the B objects, each one classified into one of the C classes, the overall prediction of the model is a tensor of shape [SxSx(C+B*5)].

In order to implement this procedure we use the darknet framework, importing the pretrained backbones and adding the last fully-connected layers using pytorch. As utils we implemented from scratch two functions for IOU and for NMS and a third one to convert data of each images into data with respect to each cell grid.

4.3. Loss

The training process of YOLO involves several factors that contribute to the loss function and guide the optimization of the network. These factors are designed to ensure accurate localization, confidence estimation, and classification of objects.

Localization Loss(first and second factors) measures the correctness of the predicted bounding box coordinates. It is calculated using the Mean Squared Error(MSE) loss over the four parameters that define the bounding box (x, y, width, and height). The center coordinates (x, y) are given a higher weight with $\lambda_{coord} = 5$ to prioritize their accuracy. Confidence Loss indicates the presence of an object within a bounding box. The confidence loss is computed using the MSE loss over the confidence parameter. To reduce the impact of confidence predictions for boxes that don't contain objects, a weight of $\lambda_{noobj} = 0.5$ is introduced. Classifi-

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

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

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Figure 3. Loss function proposed on the paper

cation Loss evaluates the accuracy of the predicted object class. It uses the MSE loss over the predicted class probabilities. However, the classification error is penalized only if an object is present in that specific grid cell.

4.4. Evaluation

In order to get a final evaluation over the different backbones used, we computed Loss function over training, validation and test sets. We implemented also a function to compute the Mean Average Precision(maP) because is one of the most widely used metrics in the context of object detection. It works basically computing IOU between the predicted bounding boxes and the real ones.

5. Experiments

5.1. Results

In this section, we present the results obtained from our experimentation with various hyperparameters in the YoloV1 object detection algorithm using the VGG16 backbone. We utilized transfer learning by initializing the backbone with pre-trained weights on a large-scale image classification task. Our objective was to enhance the training process and elevate the mean average precision (mAP) for both the training and validation sets. We thoroughly examined the influence of various hyperparameters and evaluated their effects on the performance of the model. Our implementation is available on Github at this link: https://github.com/andricmitrovic/YOLO-object-detection

5.2. Learning Rate

We started by evaluating different learning rates to find an optimal value that would facilitate convergence and yield improved results. When using a learning rate of 1e-3, the loss quickly diverged, indicating that the model failed to learn meaningful representations. Lowering the learning rate to 3e-4 and 1e-4 reduced the divergence, but the model still struggled to converge efficiently, resulting in unstable training. We achieved better results with a smaller learning rate, 2e-5, which makes the loss converging over time. However, after 150 epochs, the mAP on the training set was 0.33, and on the validation set, it was 0.14. However, by observing the process, we noticed that the model's validation precision and validation loss quickly converged to a fixed point.

5.3. Optimizer

To further enhance the learning process, we combined a learning rate of 2e-5 with the AdamW optimizer, an improved weighted version of the classic Adam optimizer. Then, in order to make more dynamic our learning rate we decided to apply a cosine annealing learning rate scheduler. This is a type of learning rate schedule that has the effect of starting with a large learning rate that is relatively rapidly decreased to a minimum value before being increased rapidly again. The resetting of the learning rate acts like a simulated restart of the learning process. Our hope was to encourage new progresses also after the strong initial one. This approach resulted in a slight improvement, with the mAP on the training set increasing to 0.35 and on the validation set reaching 0.15. Although this indicated some progress, the performance remained suboptimal and model's validation precision and validation loss were still quick to converge to a fixed point.

The convergence of validation precision and loss to a fixed point within the early stages of training suggests the presence of early signs of overfitting. This indicates that the model's performance on the validation set starts to plateau while the training set performance continues to improve. Attempting lower learning rates of 1e-5 and 3e-6 significantly slowed down the learning process, but did not yield substantial improvements in model performance. These lower learning rates failed to overcome the limitations of the previous settings. To mitigate overfitting, our model should implement different features as regularization techniques or adjusting to the model architecture.

5.4. Batch Size

We investigated the impact of batch size on the training process as a potential means of introducing additional regularization. However, reducing the batch size did not result in noticeable improvements in the model's performance. The achieved results were similar to those obtained with larger batch sizes, indicating that batch size reduction did not provide significant benefits for the current configuration.

5.5. Last Fully Connected Layer

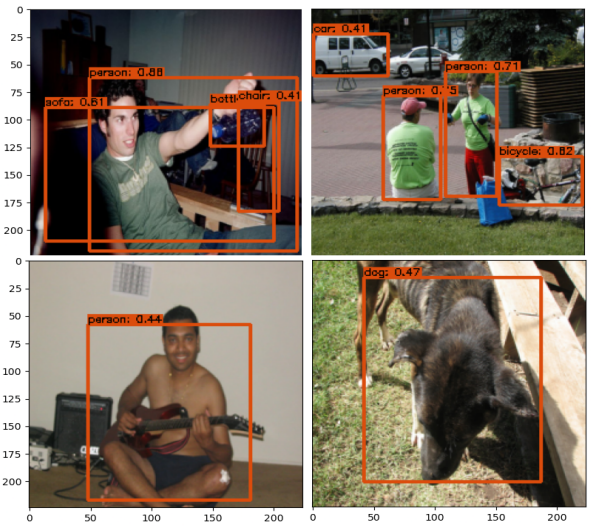
Continuing our investigation, we delved into the impact of reducing the unit count in the last two fully connected layers from 4096 to 512. Despite this modification, we ob-

served that the learning process became slower; however, it did not effectively address the issue of overfitting. The results indicate that different adjustments are required to enhance modle generalization capabilities.

5.6. Confidentiality threshould

During training we noticed that the mAP depends very much on the threshold set for the confidence score of the predictions. To ensure consistent evaluation, we established a threshold of 0.5 for calculating the mAP on the training set. This threshold allows us to focus on more confident predictions. However, for the validation set, we recognized the need to assess the model's performance across a broader range of confidence levels. Therefore, we used a lower threshold of 0.1 to evaluate the model's ability to detect objects even at lower confidence levels. By incorporating this lower threshold, we aimed to provide a more comprehensive assessment of the model's performance on the validation set.

Looking directly to the results we noticed that some predictions are quite good, even with many different objects and either with some of them overlapped. However, the main part of correct prediction involves single object that occupy completely the image space, probably due to the nature as simple classifier of the pre-trained backbone.



For what concern the bad results we noticed that the main problem was not with classifying(the pre-trained backbone works well) and neither so much with confidence score, which doesn't reach high results but still gave a decent feedback. The main issue was with missed detection. From observation on the printed predictions we notice that when many object are very close one to the each other, especially when their belong the same class, almost always some of them are missed. It happened also with object small with respect to the image size. In these cases the issues probably deals with the resolution of the images or the split size of Yolo but are both parameters that will increase the computational complexity(and time) of the training process. In addiction, lowering the threshold, we noticed that many boxes got a very low confidence score when the element present a particular colour(prediction on black cats were particularly bad) and that some object were learned in pairs. It means that it seems like if the neural network learned that if there is a horse there will be also a person and if there is a sitting person there will be a chair too. It cause some random prediction boxes, that have a very low score, but that indicate this particular behaviour. All these are observations based on the empirical analysis of the prediction made by our network on the validation set.

5.7. Alternative Backbone: ResNet18

We also explored the use of the ResNet18 model as an alternative backbone in the YoloV1 object detection

algorithm. However, the results obtained with ResNet18 were not satisfactory. The learning process with ResNet18 was considerably slower compared to VGG16. After an extended training period, the model achieved a mAP of only 0.1 on both the training and validation sets. These results, along with the limited computational resources at our disposal, prevented us from fully exploring the YoloV1 implementation with ResNet18 as the backbone. The fact that the model shown no overfitting neither learning progresses indicates that probably it is too simple to generalize the features needed for the task.

Hyperparameters	mAP on Training Set	mAP on Validation Set
Learning Rates: 1e-3; 3e-4; 1e-4; 2e-5;	Diverged	Diverged
Learning Rate: 2e-5	0.33	0.14
Learning Rate: 2e-5 + AdamW + Cosine Annealing	0.35	0.15
Learning Rates: 1e-5; 3e-6;	/	/
Batch Size reduction	0.35	0.15
FC Layers Units: 512	0.35	0.15
Alternative Backbone: ResNet18	0.1	0.1

6. Conclusion

Our experimentation shown that the detection object task, also with relatively small image size, is a task that requires a model complexity that is not reachable with models as VGG16 and ResNet18. ResNet18 got very bad results probably for its particularly architecture. Thanks to residual connection this network is able to make itself "shorter" in order to gain a better generalization and prevent overfitting. But, in our case, it is evident how a less deep architecture is not enough to learn the task properly. It can be seen as a lower bound to the complexity of a network that want to deals with that problem. On the other hand VGG16 got slight better result despite the simplier architecture, showing that probably a deeper version of ResNet, as ResNet50 or ResNet101, will be surely a good starting point as backbone for a future Yolo implementation. For what concerns the implementation of the Yolov1 algorithm we noticed that some parameters, more than others, have a very strong influence on the result, like for example the different weights on the loss function and the threshold on acceptable confidential scores. Finally, regrading the hyperparameters, we observe the efficacy of new features as AdamW and cosine annealing learning rate, that represent the best improvements on our results, and the importance on find a good balance with learning rates between the effectiveness and the computational needs, which are very high for our particular task.

References

[1] Dumitru Erhan Christian Szegedy Scott Reed Cheng-Yang Fu Alexander C. Berg Wei Liu, Dragomir Anguelov. Ssd: Single shot multibox detector, 2015. https://arxiv.org/abs/1512.02325.

[2] Ross Girshick Jian Sun Shaoqing Ren, Kaiming He. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. https://arxiv.org/abs/1506.01497.

[3] Ross Girshick Kaiming He Piotr Dollár Tsung-Yi Lin, Priya Goyal. Focal loss for dense object detection.

[4] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection, 2016. http://pjreddie.com/yolo/.

[5] Andrew Zisserman Karen Simonyan. Very deep convolutional networks for large-scale image recognition, 2015. https://arxiv.org/pdf/1409.1556.pdf.

[6] paperswithcode.com. Image classification on imagenet, 2023. https://paperswithcode.com/sota/image-classification-on-imagenet?tag$_filter = 3$.