# Implement of YoloV3

*Yachen Wang, Chenjiayi He, Jianxiao Yang, Yuxi Ge*
wyachen@bu.edu hcjy@bu.edu yangjx@bu.edu yuxig5@bu.edu

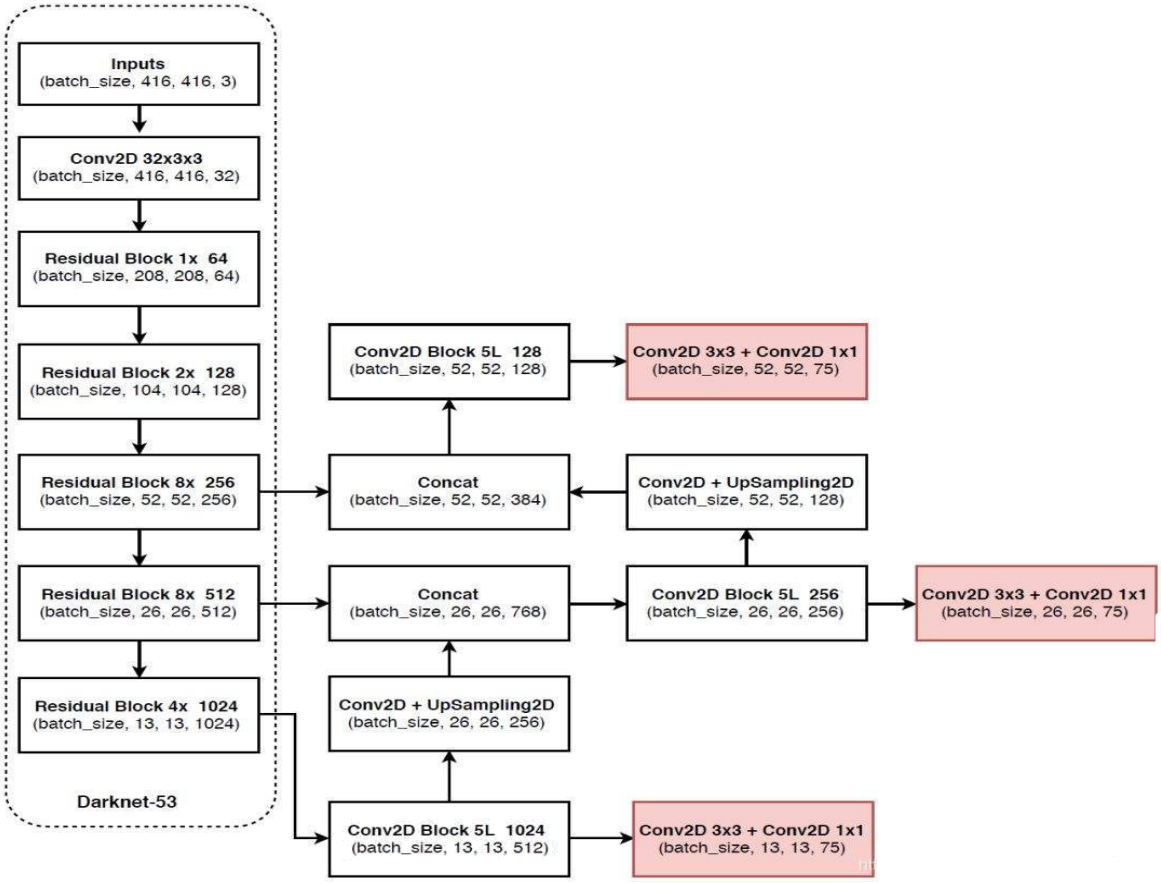



Figure 1. The Architecture of YoloV3

## 1. Task

The objective of this project is to implement the YOLOv3 (You Only Look Once version 3) object detection model. YOLOv3 is a state-of-the-art deep learning model for real-time object detection, capable of detecting and classifying a large number of objects in an image with high accuracy and efficiency. Unlike traditional object detection models that require multiple stages of processing, YOLOv3 performs object detection in a single end-to-end neural network, resulting in faster and more accurate detection. It is widely used in a variety of applications such as autonomous driving, surveillance systems, and robotics. In this project, we will train the YOLOv3 model on a custom dataset and evaluate its performance on a test set. The implementation will be done using PyTorch, a popular deep learning framework.

## 2. Related Work

YOLO (You Only Look Once): The original YOLO model proposed in 2015 by Redmon et al. that introduced the idea of object detection using a single neural network.

YOLOv2: An improved version of YOLO proposed in 2016, which introduced the use of anchor boxes and batch normalization to improve detection accuracy.

RetinaNet: A state-of-the-art object detection model proposed by Lin et al. in 2017, which uses a focal loss function to address the class imbalance issue in object detection.

SSD (Single Shot MultiBox Detector): A popular object detection model proposed by Liu et al. in 2016, which uses a multi-scale feature extraction network and multi-scale anchor boxes to improve detection accuracy.

## 3. Approach

3.1 Darknet-53

In our project, we use the Darknet-53 as our backbone feature extraction network, which is shown on the left of our main architecture(Figure 1). What is special about this backbone network is that it uses the Residual Net, which is characterized by its ease of optimization and its ability to improve the accuracy. Figure 2 shows the general structure of a residual net that we used.

We first performed a convolution with a kernel size of 3X3, and a step size of 2. This convolution will compress the width and height of the input feature layers, and get a feature layer in the meanwhile. After that, we perform a 1X1 convolution and a 3X3 convolution on that feature layer, and add the layer itself to that result, then we can get the final residual structure.

By continuously doing the downsampling, a process of series of convolutions, we can deepen the network significantly.

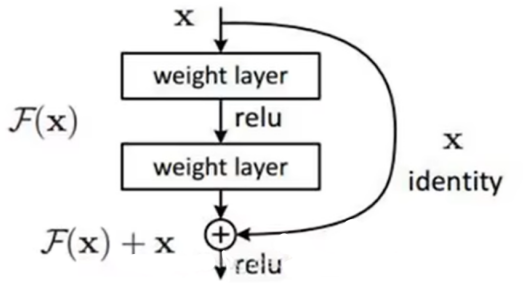



Figure 2. The Residual Net

3.2 Predict from the extracted feature

The process of predicting from the extracted feature can be divided into two parts.

Firstly, to build an FPN (Feature Pyramid Network) to enhance feature extraction. A total of three feature layers are extracted for this process, those three feature layers are located in different positions of the backbone: Darknet53, the middle layer, the lower middle layer, and the bottom layer, and the shapes of those three feature layers are (52,52,256), (26,26,512), and (13,13,1024) respectively. Then, we use those three layers for the construction of the FPN layers. The feature pyramid can fuse the feature layers of different shapes to help extract better features.

Secondly, to use a YOLO head to predict. The Yolo head is one 1X1 convolution which serves for adjusting the numbers of channels plus one 3X3 convolution which serves for feature extraction.

3.3 Decoding

The decoding process of YoloV3 is divided into two steps:

First add each grid point to its corresponding x_offset and y_offset, and the result after adding is the center of the prediction frame.

Then use the combination of the prior frame and h, w to calculate the width and height of the prediction frame. In this way, the position of the entire prediction frame can be obtained.

Below shows the process of how we transfer the prior box to the predicted box.

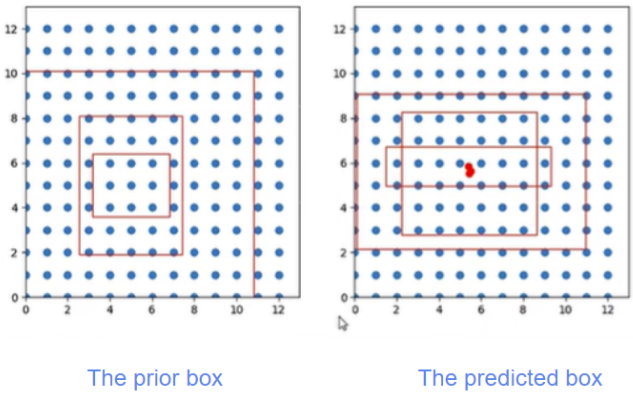



Figure 3. The prior box and the predicted box

After the final prediction results are obtained, score sorting and non-maximum suppression screening are performed.

This part is basically the common part of all object detection. It discriminates for each class:

1. Take out the boxes and scores of each category whose score is greater than self.obj_threshold.
2. Use the position and score of the box for non-maximum suppression.

## 4. Datasets


Figure 4. The VOC2007 DataSet

The DataSet we used in our project is VOC2007.The VOC2007 dataset contains images with 20 different object categories, including aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog,

horse, motorbike, person, pottedplant, sheep, sofa, train, and tvmonitor. The dataset includes both training and testing subsets, and each image is labeled with one or more object categories and a bounding box for each object. In general, the image tensor dimensions depend on the specific implementation and framework being used. However, for YOLOv3, the input images are typically resized to a fixed size, such as 416 x 416 pixels, and converted to a tensor with dimensions (batch_size, 3, 416, 416). The output tensor dimensions depend on the specific architecture of the YOLOv3 model and the number of bounding boxes and object categories being detected.

**5. Evaluation Metrics**

In our project, we use the mAP as the evaluation metrics, and here provides how the mAP value is gotten.

TP: the number of predicted boxes whose IoU>0.5

FP: the number of predicted boxes whose IoU<=0.5

FN: the number of unpredicted boxes

Precision: TP/(TP+FP)

Recall: TP/(TP+FN)

AP: The area under the Precision-Recall curve

mAP: the mean value of AP for each category

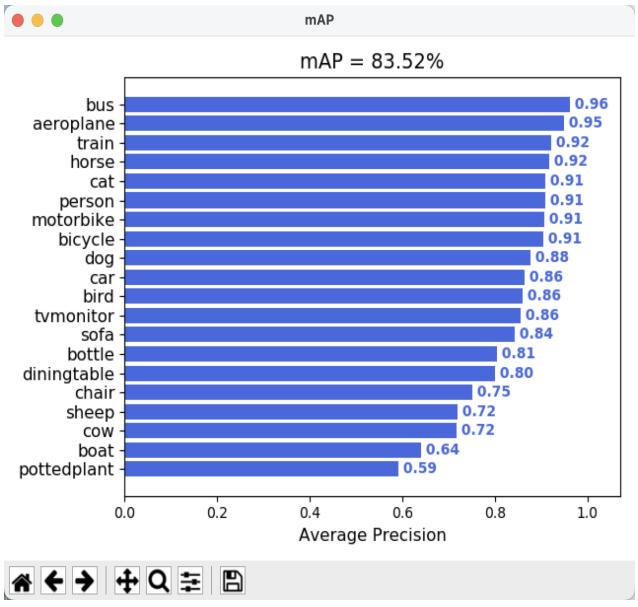
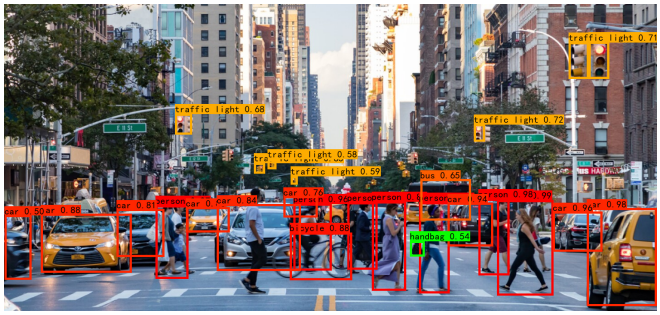



Figure 5. The mAP for Yolo V3

The mAP value of 83.5 for our YOLOv3 model indicates that the model has a high detection accuracy across all classes. This high mAP value is a result of the model's ability to accurately localize and classify objects present in the input images, while also maintaining a low rate of false positives. The YOLOv3 model's real-time detection speed, combined with its high mAP score, makes it an excellent choice for a wide range of object detection applications, such as autonomous driving, video surveillance, and robotics.

**6. Results**

As picture shown, we successfully implement the model and generate target detection. The number in picture means model thinks this may be a percentage of an item(the higher the better). Most of the number is above 70% and they are detected.So we think we are successful. Just as metrics shows, our model can detect most of the items in a precision of more than 70%.



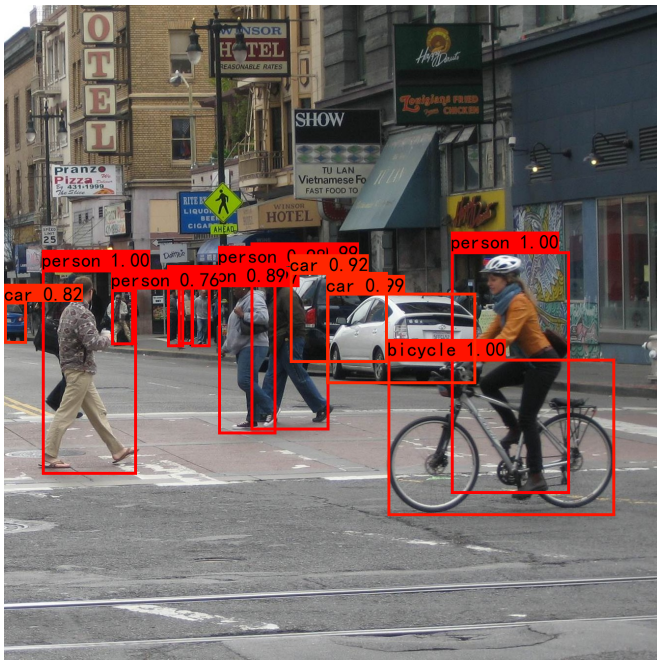



Figure 6. The result of YoloV3

**7. Conclusion**

Understand the core principles of YOLO3: By reimplementing YOLO3, I better understand its network structure (such as Darknet-53), loss function and training strategy. This helped me understand how YOLO3 transforms the object detection task into a regression problem, and how to use a single forward pass to detect objects.

Multi-Scale Detection: Implementing YOLO3 will help me learn how to detect objects at different scales, which is a general technique that can be applied to other computer vision tasks.

Practice deep learning and computer vision techniques: Reimplementing YOLO3 will make me more familiar with deep learning frameworks (such as TensorFlow, PyTorch, etc.) and computer vision libraries (such as OpenCV). These skills are very valuable for other deep learning and computer vision projects.

Model tuning and optimization: During implementation, I need to tune YOLO3 according to my own hardware and dataset. This will teach me how to tune hyperparameters like network parameters, learning rate, and how to use data augmentation techniques to improve the generalization of the model.

Performance Evaluation: By reimplementing YOLO3, I learned how to use various evaluation metrics (such as mAP, IoU, etc.) to measure the performance of object detection models. This is equally important for evaluating other object detection algorithms.

Comparison:

There are several other object detection models that have been widely used in the computer vision community. Each model has its own set of strengths and weaknesses.

When comparing R-CNN, SSD, and YOLOv3 in terms of model evaluation, we can analyze their strengths and weaknesses in terms of accuracy, speed, and generalization.

1.Accuracy:

•R-CNN: R-CNN is known for its high accuracy in object detection tasks, as it combines region proposals with CNN features. However, its accuracy is generally lower than that of its more advanced successors like Faster R-CNN and YOLOv3.

•SSD: SSD is a single-stage detector that eliminates the need for a separate region proposal step, making it faster than two-stage detectors like R-CNN. However, its accuracy can be lower than that of two-stage detectors, particularly for smaller objects.

• YOLOv3: YOLOv3 is an advanced version of the YOLO family that introduces several improvements, such as multi-scale predictions and better handling of small objects, resulting in higher accuracy compared to SSD and even some two-stage detectors.

2.Speed:

•R-CNN: R-CNN's main downside is its slow speed, making it unsuitable for real-time applications. It is the slowest among the three models discussed here.

•SSD: SSD is faster than R-CNN and even some two-stage detectors like Fast R-CNN and Faster R-CNN, as it eliminates the need for a separate region proposal step.

•YOLOv3: YOLOv3 is designed to be fast and efficient while maintaining high accuracy. It is generally faster than two-stage detectors and can perform real-time object detection on powerful hardware.

3.Generalization:

•R-CNN: Since R-CNN relies on selective search for generating region proposals, it might struggle to generalize to novel object categories and detection scenarios.

•SSD: SSD can generalize better than R-CNN due to its single-stage nature, but it might struggle with detecting smaller objects.

•YOLOv3: YOLOv3 has been shown to generalize well to different object categories and detection scenarios. Its architecture and loss functions have been designed to handle a wide range of object sizes and aspect ratios.

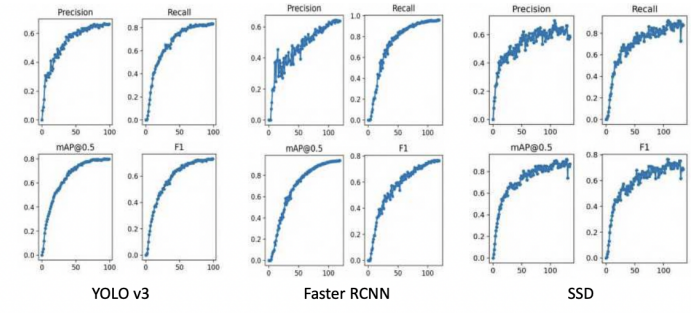



Figure 7. The evaluation of YoloV3, F-CNN,SSD

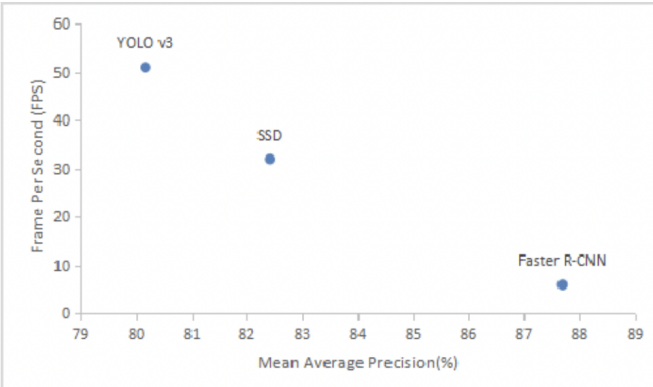



Figure 8. performance of YoloV3, F-CNN,SSD

In summary, R-CNN has high accuracy but is relatively slow and may struggle to generalize to novel object categories. SSD is faster than R-CNN but may have lower accuracy, especially for small objects. YOLOv3 strikes a balance between accuracy and speed while providing better generalization capabilities.

Solve practical problems: Implementing YOLO3 will give me more confidence to apply it to practical scenarios, such as intelligent monitoring, autonomous driving, robot navigation, etc. This hands-on experience will be invaluable for future projects and research.

In conclusion, reimplementing YOLO3 will give me a deeper understanding of the latest technologies and methods in the field of object detection, improve my deep learning and computer vision skills, and the ability to solve practical problems

**Appendix A. Detailed Roles**

Shown in Table 1 below.

**Appendix B. Code repository**

Our github link:
https://github.com/Yangjianxiao0203/CS523_YoloV3_self

**References**

1) J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In CVPR, 2016.
2) J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. In CVPR, 2017.
3) T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In CVPR, 2017.
4) W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In ECCV, 2016.

Table 1. Team member contributions

| Name | Task | File names | No. Lines of Code |
|---|---|---|---|
| Jianxiao Yang | Implement the net of YoloV3, calculate the loss in Yolo;<br>Implement predict and train for one epoch of Yolo;<br>help training the model and predicting | darknet.py<br>yolo.py<br>yolo_training.py<br>fit.py | 1000 |
| Yuxi Ge | Writing comments for each block of code<br>Implemented training process | Train.py<br>test.py | 300 |
| Yachen Wang | Implement the anchor boxes;<br>Implement the decoding part and draw the bounded box on the picture. | bbox.py<br>dataloader.py<br>utils.py<br>callback.py | 600 |
| Chenjiayi He | Responsible for the prediction process, to perform the object detection of an image.<br>use the mAP as the evaluation metrics | get_map.py<br>predict.py<br>yolo.py | 260 |