Computer Vision Midterm Checkpoint

Boyuan Yao*
School of Data Science
19307110202@fudan.edu.cn

Zixuan Chen*
School of Data Science
19307110339@fudan.edu.cn

Yanjun Shao*
School of Data Science
19307110036@fudan.edu.cn

Abstract

We finished image classification and object detection tasks for this midterm project. In image classification, we use AlexNet[11] and ResNet[8] as baseline, then boost their performance with CutMix data augmentation techniques proposed by Yun et al.[17] and MixUp, CutOut methods mentioned in [18] [4]. In object detection, we trained Faster R-CNN[16] on dataset VOC2007[5] and VOC2012[6] with backbone ResNet-50[8] pretrained on ImageNet[3]. For Faster R-CNN trained on VOC07 and VOC07+12, we scored mAP 65.7% and 79.1% on VOC07 test set respectively. We also trained YOLOv3[14] from scratch on VOC07+12, and scored mAP 48.3% on VOC07 test set. All the source code are available from the aggregate github repo: https://github.com/super-dainiu/cv-midterm. Also, we applied our Faster R-CNN model to predict a video clip, which can be found here.

1 Introduction

Image classification is the most basic task in the field of computer vision. AlexNet[11] started the deep learning fever with a new high score on ImageNet[3] dataset. And ResNet[8] solved the problem that deeper neural network is much more difficult to train due to overfitting by using residual connection to guide deeper layers to learn the residual part of features, and achieved a better score. After that, deeper network became a new trend in CNN. Moreover, as GPU capability grows, it is easier to train deeper networks now. However, for small datasets such as CIFAR-100[10], there is only a small number of images for each category, and data augmentation is an effective solution to this limitation. In fact, in [8], they already applied some data augmentation methods such as random horizontal flip, random crops and basic weight decay to boost the performance of the network. Later in [18] [4] [17], the MixUp, CutOut and CutMix methods are formally introduced as techniques to diversify the training data in order to achieve a better performance on test data.

For image classification part, we take AlexNet and ResNet as baselines to show how data augmentation methods can improve the network performance. We follow [17] such that we validate the result on test set directly instead of splitting the training set into training set and validation sets.

Object Detection using deep learning has seen a boom in the recent couple of years. The main problem in object detection focused on localizing and recognizing distinct objects in images and videos with features extracted by deep neural networks. Object detection with deep learning can be grouped into two genres: “two-stage detection” and “one-stage detection”, where the former frames the detection as a “coarse-to-fine” process while the later frames it as to “complete in one step”.

For object detection part, we re-implemented "one-stage detection" method YOLOv3 and "two-stage method" Faster R-CNN with backbone ResNet-50 and compared their performance. For Faster R-CNN, we trained two models on VOC07 and VOC07+12, and latter result is much better than the

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

first one. For YOLOv3, due to the limitation of resources, we failed to obtain a pretrained version of Darknet-53. However, the object detection results were relatively satisfying. A detector class is available at my github repository.

2 Image Classification: Network Structure

2.1 AlexNet

AlexNet is a simple CNN starts with convolution layer to transform the input $3 \times 227 \times 227$ images into $96 \times 55 \times 55$ features and a ReLU layer, before the max pooling layer, they use local response norm to process the features and after the max pooling layer, the features have the size of $96 \times 27 \times 27$. With another combination of convolution layer, ReLU unit, local response norm and max pooling, the shape of features becomes $256 \times 13 \times 13$. After three Conv-ReLU combination and a max pooling layer, the features have the shape of $256 \times 6 \times 6$ and with two linear-ReLU combination and a fully-connected layer as classifier, we could get the score for each class. The followings summary the types of each layer and figure1 shows the details of the above structure.

(1) Convolution layer + ReLU layer
(2) Local response norm
(3) Max pooling layer
(4) Convolution layer + ReLU layer
(5) Local response norm
(6) Max pooling layer
(7) Convolution layer + ReLU layer $\times 3$
(8) Max pooling layer
(9) Fully connected layer + ReLU layer
(10) Fully connected layer + cross entropy loss

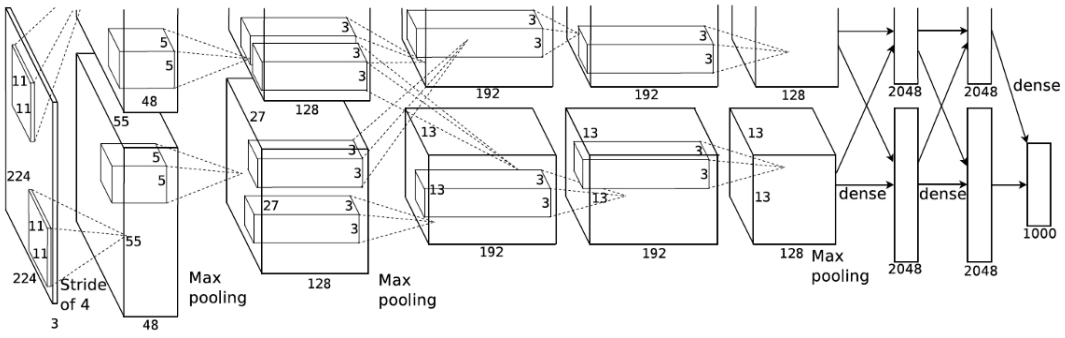


Figure 1: Overall Structure of AlexNet

The local response norm of the AlexNet is computed by following equation

$$b^i_{x,y} = a^i_{x,y} \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}, \tag{1}$$

where the N is the total number of kernels in the layer and n defines the number of "adjacent" kernel maps at the same spatial position. It could be seen as a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. At the time Alex Krizhevsky wrote the paper, it was a novel way to improve

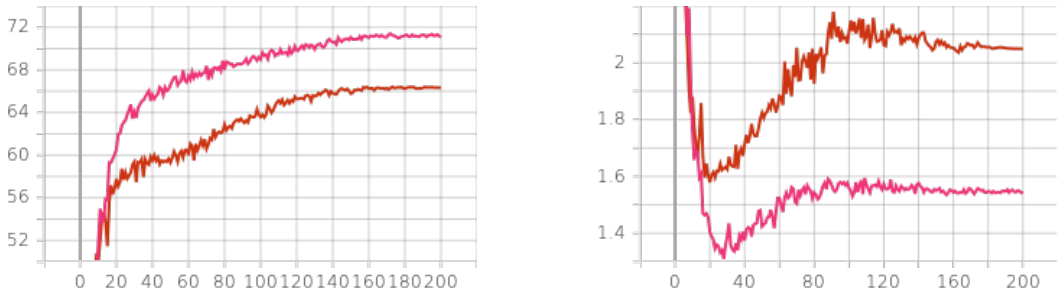
the generalization of network. In our baseline, we replace it with batch normalization introduced in [9], which applies normalization on each dimension of vector $x = (x^{(1)}, x^{(2)}, \ldots, x^{(d)})$ with the following equation

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right]}} \tag{2}$$

And to ensure that the introduced transformation could represent identity transform, after normalization, the vector would be scaled and shifted with the following equation

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)} \tag{3}$$

With batch normalization, we could achieve the following boost to the performance on CIFAR-100 dataset with baseline data augmentation methods, i.e., random crops with padding and random horizontal flip.



(a) Accuracy on test set

(b) Loss on test set

Figure 2: The performance comparison between original AlexNet structure and the one optimized with batch normalization. The pink lines indicate the accuracy and loss history of optimized model, the red lines indicate the original model. This experiment used cosine annealing technique introduced in [12] for learning rate decay.

Figure2 shows the edge of batch normalization and we used the optimized model in the later session of data augmentation. Notice that the original AlexNet is designed to finish the image classification task on ImageNet, the images in ImageNet have the shape of $3 \times 224 \times 224$, so we up sampling the images in the CIFAR-100 to fit the AlexNet input.

2.2 ResNet

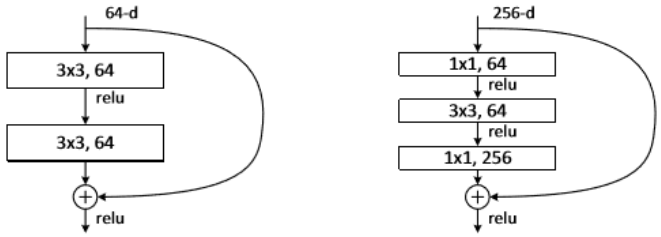
ResNet[8] introduced residual connection to guide the network to learn the "residual" of the reality and former learned feature, so that the difficulties found in former study that deep networks are harder to train and perform worse than shallow networks could be overcome. The figure3 follows show some details of ResNet structure.

The original ResNet is designed to deal with image classification on ImageNet too, but luckily the [8] also mentioned the redesigned structure for image in the shape of CIFAR-10 (also CIFAR-100). It consists of residual blocks with smaller scale and remove the max pooling layer. The first layer is 3×3 convolutions, then going through a stack of $6n$ layers with 3×3 convolutions on the feature maps of size $\{32, 16, 8\}$ respectively, the number of filters are $\{16, 32, 64\}$ respectively (with subsampling performed by conlutions with a stride of 2). The network ends with a global average pooling, a number of classes-way fully-connected layer to compute the score and using cross-entropy loss. Moreover, if we use the bottleneck structure in right part of figure3a there will be $9n + 2$ layers instead of $6n + 2$ layers. The following table summarizes the architecture:

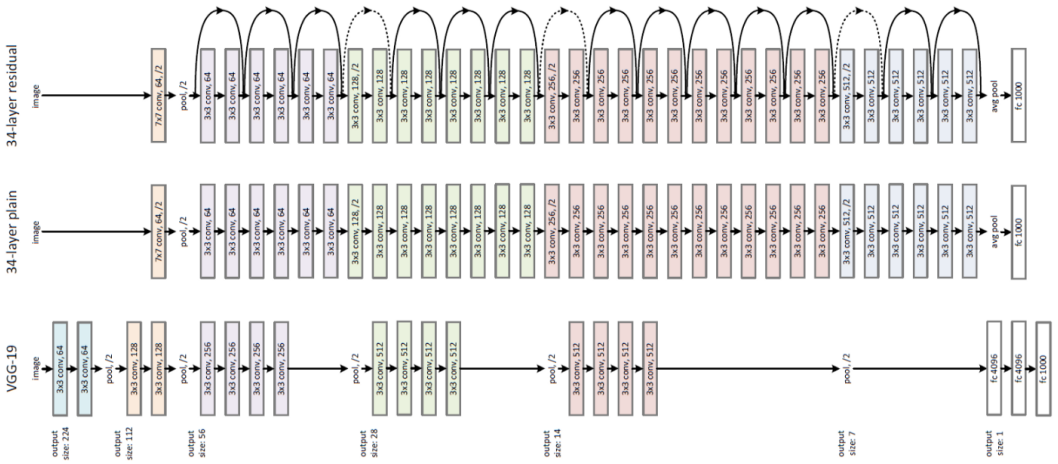
output map size	32×32	16×16	8×8
# layers	$1 + 2n \quad (1 + 3n)$	$2n \quad (3n)$	$2n \quad (3n)$
# filters	16	32	64

(4)

We also implement the original ResNet structure, replace the 7×7 convolution layer with the 3×3 one and remove the max pooling layer to make it more suitable for CIFAR input images. The smaller structure of ResNet mentioned above has smaller number of parameters, for example, the one with 110 layers ($9n + 2$ bottleneck structure $n = 12$) has only around 1.1M parameters, while the refined ResNet18 has around 11M parameters. We will show that the outcome of two networks looks almost the same in the latter section and show that why refined ResNet18 might be a better idea in some degrees.



(a) Blocks in ResNet



(b) Structure of ResNet-34

Figure 3: Details of ResNet Structure

3 Image Classification: Data Augmentation

In our baseline, we use random horizontal flip and random crops (with 4 zero padding and crop 32×32 from the padded image) as a basic augmentation approach (and some normalization preprocessing). Based on it, we implemented the CutMix, CutOut, MixUp approaches mentioned in [17] [4] [18] and compare the performance with the baseline. Figure4 show some examples of baseline images

(a) (b) (c)

Figure 4: Baseline images

The CutMix[17] generate a new training sample $(\tilde{x}, \tilde{y})$ by combining two training sample $(x_A, y_A), (x_B, y_B)$ with the following operation

$$\begin{aligned} \tilde{x} &= \mathbf{M} \odot x_A + (\mathbf{1} - \mathbf{M}) \odot x_B \\ \tilde{y} &= \lambda y_A + (1 - \lambda) y_B \end{aligned} \tag{5}$$

Where $\mathbf{M} \in \{0, 1\}^{W \times H}$ is a mask and $\odot$ is element-wise multiplication. The λ indicates the combination ratio. The figure7 shows some examples of CutMix approach

(a) (b) (c)

Figure 5: CutMix images

Instead of using two training examples to generate a new training instance, CutOut[4] simply mask the chosen box of the given training instance to create the instance, the followings are some of the examples

(a) (b) (c)

Figure 6: CutOut images

The last approach is MixUp introduced in [18], it using an affine combination to combine two training instances

$$\begin{aligned} \tilde{x} &= \lambda x_A + (1-\lambda) x_B \\ \tilde{y} &= \lambda y_A + (1-\lambda) y_B \end{aligned} \tag{6}$$

Where the λ here has the similar meaning as in CutMix approach, the followings are some examples

(a) (b) (c)

Figure 7: MixUp images

In our experiment, we choose λ from Beta distribution $\mathrm{Beta}(\alpha, \alpha)$, where the α is a hyperparameter could be tuned by user.

4 Image Classification: Training Strategy

In this task, we implement the training strategy mentioned in [12], combining restart and cosine annealing to control our learning rate. In each phase of training, we use following learning rate decay strategy to train $T_{\max}$ epochs

$$\begin{aligned} \eta_t &= \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right), \quad T_{\text{cur}} \neq (2k+1)T_{\max} \\ \eta_{t+1} &= \eta_t + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 - \cos\left(\frac{1}{T_{\max}}\pi\right)\right), \qquad T_{\text{cur}} = (2k+1)T_{\max} \end{aligned} . \tag{7}$$

We could multiply a factor T_{mult} on $T_{\max}$ after each phase of training as suggested in [12]. With this restart strategy and learning rate decay approach, the training history will be smoother and could achieve almost the same or even better performance without a lot of tuning effort and even smaller number of epochs. Compare to the old training method, where researchers tuned the learning rate once they spot that the training process falls into a plateau, this approach could be a better way to train. For example, compare to the original training process introduced in [17], they decayed the learning rate when they train 150 epochs and 225 epochs, but in fact, even with the carefully designed timing for learning rate decay, you could see test error has fallen into a plateau for a long time before the decay. Besides, the training process with such manual control could be time consuming. So we finally decided to use the above method to train our networks.

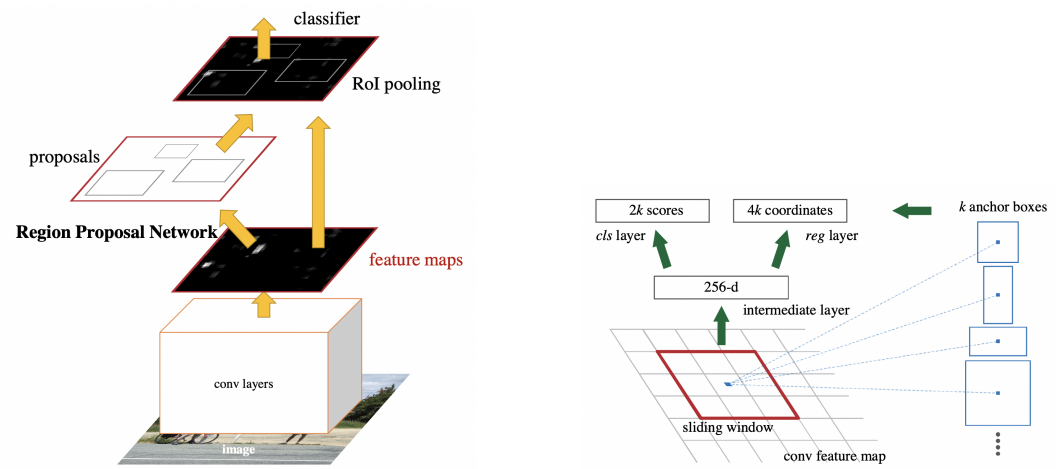
5 Object Detection: Faster R-CNN

In this section, we will introduce Faster R-CNN, the "two-stage detection" network used in our object detection task. Unfortunately, the official code of Faster R-CNN is in Matlab, so we found an unofficial code implemented in early versions of Pytorch and did some modification on it so that it can adapt to modern version of Pytorch.

To provide an overall description of Faster R-CNN, it is composed of two modules. The first module is a deep fully convolutional network that proposes regions, called Region Proposal Network (RPN),

and the second module is the Fast R-CNN[7] detector that uses the proposed regions. Generally speaking, the overall pipeline of Faster R-CNN is that it first derive a feature map from backbone CNN, which is then fed to RPN so that some Regions of Interest are proposed. Then these proposals are fed to RoI pooling layer together with feature maps to generate feature vectors. The feature vectors are passed into the classifier to complete the object detection task.

The overall architecture is shown in figure 8a. Later in this section we will introduce all the key structures of Faster R-CNN respectively.



(a) Overall Architecture of Faster R-CNN

(b) Region Proposal Network (RPN)

Figure 8: The architecture of Faster R-CNN and Region Proposal Network.

5.1 Feature Extracting

The first step of Faster R-CNN is extracting features from raw pixel images, where basic Convolutional Neural Network (CNN) is used. As CNN can be seen as two parts: the first part is several convolution layers and pooling layers, which can produce a feature map; the second part often consists of fully connected layers, which is referred to as classifier. As in Faster R-CNN we need a feature map of the original image, so we utilize the first part of existed CNN architecture to extract the feature map. In practice, this part is often derived from CNN model pretrained on ImageNet.

5.2 Region Proposal Networks

Region Proposal Network (RPN) is the second key part of Faster R-CNN. The RPN works on the output feature map returned from the last convolutional layer and output several proposed regions. Generally speaking, it mainly has following advantages:

1) It could be trained end-to-end as a neural network and be customized according to the detection task.
2) It produces better region proposals compared to generic methods like **Selective Search** and **EdgeBoxes**.
3) It shares a common set of convolutional layers together with Fast R-CNN, thus can be merged with Fast R-CNN into a single network.

As shown in figure 8b, in RPN, based on a rectangular window of size $n \times n$, a sliding window passes through the feature map. For each window, several candidate region proposals are generated, and will be filtered based on their "objectness score".

Anchor According to figure 8b, for each window, K region proposals are generated. Each proposal is parameterized by: **scale** and **aspect ratio**, according to a reference box called **anchor box**. The number of K is generally decided by combinations of scales and aspect ratios, and some anchor

variations are shown in the figure. With anchor boxes used, the network is able to offer scale-invariant object detectors as the anchors exist at different scales. The multi-scale anchors are key to share features across the RPN and the Fast R-CNN detection network.

For each $n \times n$ region proposal, a feature vector is extracted, and it is then fed to 2 sibling fully-connected layers. The first is named *cls*, representing a binary classifier that generates the objectness score for each region proposal. This layer has 2 outputs, the first is for classifying the region as a background and the second is for classifying the region as an object. Specifically, if the first element is 1 and the second is 0, then the region is classified as background, and classified as an object otherwise. The second is named *reg*, which returns a 4-dimension vector defining the bounding box of the region by bounding box regression.

Objectness Score For training the RPN, each anchor is given a positive or negative **objectness score** based on the Intersection-over-Union (IoU), which is the ratio between the area of intersection between the anchor box and the ground-truth box to the area of union of the 2 boxes, ranging from 0.0 to 1.0. Specifically, the objectness score is determined based on the following four conditions:

1) An anchor with an IoU higher than threshold, always set as 0.7, with any ground-truth box is given a positive objectness label.
2) If there is no anchor with an IoU higher than threshold, then assign a positive label to the anchor with the highest IoU with a ground-truth box.
3) An anchor with all IoUs less than threshold is given a negative objectness score, meaning that the anchor is classified as background.

Notice that anchors that are neither positive or negative do not contribute to the training objective.

Bounding Box Regression As the anchor proposed first may not be accurate, we adopt the following bounding box regression:

$$
\begin{aligned}
t_x &= (x - x_a)/w_a & \qquad t_y &= (y - y_a)/h_a \\
t_w &= \log(w/w_a) & \qquad t_h &= \log(h/h_a) \\
t_x^* &= (x^* - x_a)/w_a & \qquad t_y^* &= (y^* - y_a)/h_a \\
t_w^* &= \log(w^*/w_a) & \qquad t_h^* &= \log(h^*/h_a)
\end{aligned}
$$

where x, y, w and h denote the box's center coordinates and its widht and height. Variables x, x_a, and x^* are for the predicted box, anchor box, and ground-truth box respectively (likewise for y, w, h). This can be thought of as bounding-box regression from an anchor box to a nearby ground-truth box.

5.3 Region of Interest (RoI) Pooling

Region of Interest (RoI) pooling produced the fixed-size feature maps from non-uniform inputs by doing max-pooling on the inputs. Specifically, it has two inputs:

- Feature map obtained from the backbone convolutional neural network.
- N proposed regions, known as Region of Interests from RPN. Each proposal has five values, the first one indicating the index and the rest are proposal coordinates.

RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. All values from all sub-grids represent the feature vector.

5.4 Feature Sharing between RPN and Fast R-CNN

The core idea of this is that RPN and Fast R-CNN share the same convolutional layers. These layers exist only once but are used in the 2 networks. It is often called as **layer sharing** or **feature sharing**. This is a great contribution of Faster R-CNN that effectively reduced the number of parameters.

5.5 Loss Function

The loss function of Faster R-CNN is defined as:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*). \tag{8}$$

Here, i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive and 0 otherwise. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box and t_i^* is that of the ground-truth box associated with a positive anchor. The classification loss L_{cls} is log loss over two classes. For regression loss, we use $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$, where R is smooth L_1 loss:

$$Smooth_{L_1} = \begin{cases} 0.5 * x^2, & |x| < 1 \\ |x| - 0.5, & otherwise \end{cases}$$

The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors $p_i^* = 1$ and is disabled otherwise. The outputs of *cls* and *reg* layers consist of $\{p_i\}$ and $\{t_i\}$ respectively. These two terms are normalized by N_{cls} and N_{reg} and weighted by a balancing parameter λ.

6 Object Detection: YOLOv3

In this section, we will reproduce the original YOLOv3[14] from scratch. You only look once (YOLO) was once a state-of-the-art, real-time object detection system, widely used in the industry fields. It outperforms most real-time object detection algorithms with surprisingly low inference time per image, while its accuracy is hardly sacrificed. My Pytorch implementation aims to minimize the code redundancy. All the source code is online at my github repo: https://github.com/super-dainiu/yolov3. As no pre-trained backbone is available, it seems hard to reach the state-of-the-art performance.

From the original YOLO paper [14] and [15], the authors introduced a "single-stage detection" pipeline for object detection. They formulated object detection as a regression problem to spatially separated bounding boxes and associated class probabilities, and optimized the whole model end-to-end directly on detection performance. Darknet [13] was a light-weighted but powerful backbone network for feature extraction. The extracted features will then be sent to a bunch of convolution layers to produce predictions over class probabilities and bounding boxes of different scales. YOLOv3 was an incremental improvement over YOLOv1 and YOLOv2, but the main architecture was "nothing like super interesting, just a bunch of small changes". The inference accuracy has not been improved, but the inference time was shortened significantly, which made it possible for real-time (50FPS) object detection on Titan X GPU.

Figure 9 provides a sketch of the YOLO design. Later we will introduce how the whole system works in practice.

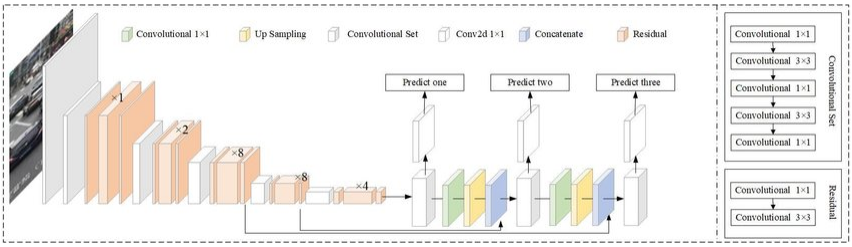


Figure 9: The YOLOv3 Model

6.1 Feature Extracting

Darknet-19 appeared early in 2013. It is an open source neural network framework written in C and CUDA. In 2018 paper of YOLOv3[14], another deeper and more efficient Darknet-53 was proposed to improve the inference accuracy. Both of the networks failed to compete with Resnet-152 on Imagenet results, but the Darknet structure better utilizes the GPU, making it more efficient to evaluate and thus faster. The following figure 10 shows the architecture of Darknet-53.

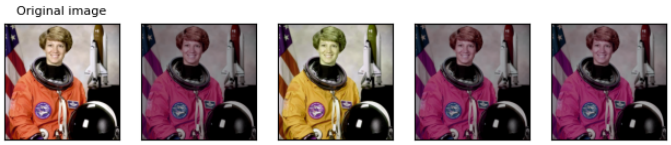
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 10: Overall Architecture of Darknet-53

6.2 Data Augmentations Using Albumentations

Albumentations[2] is a fast and flexible image augmentation library. The library is widely used in industry, deep learning research, machine learning competitions, and open source projects. We do data augmentations on our training set with the help of Albumentations library. Random transforms on images and bounding box can avoid overfitting and thus improve the performance for real-world applications.

(a) Posterization



(b) Color Jitter

(c) CLAHE

Figure 11: Data Augmentation Examples

Basic augmentations such as horizontal flips, affine rotations, channel shuffle, blurring and random crop are applied with probabilities, respectively. Other transformations such as posterization11a, color jitter11b and CLAHE11c also activate randomly. All the images are resized to 416×416 with padding and longest max size.

6.3 Bounding Box Prediction

YOLOv3 system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell is obliged to predict a number of bounding boxes and the probabilities for each class. And each bounding box consists of 5 predictions: x, y, w, h, and Pr(Object).

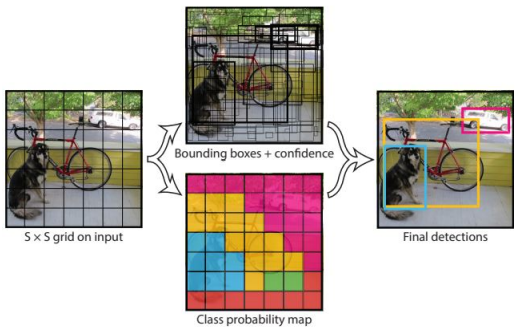


Figure 12: YOLOv3 models detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B × 5 + C) tensor.

Within each cell, a prior of bounding box width and height at different scale is given. In YOLOv3 paper [14], the authors used k-means clustering to determine the bounding box priors. They selected 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales. Here, the anchors are given by,

$$[(0.28, 0.22), (0.38, 0.48), (0.90, 0.78)]$$
$$[(0.07, 0.15), (0.15, 0.11), (0.14, 0.29)]$$
$$[(0.02, 0.03), (0.04, 0.07), (0.08, 0.06)]$$

The network predicts 4 coordinates for each bounding box, t_x, t_y, t_w, t_h. If the grid cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height (p_w, p_h), then the predictions correspond to figure 13.

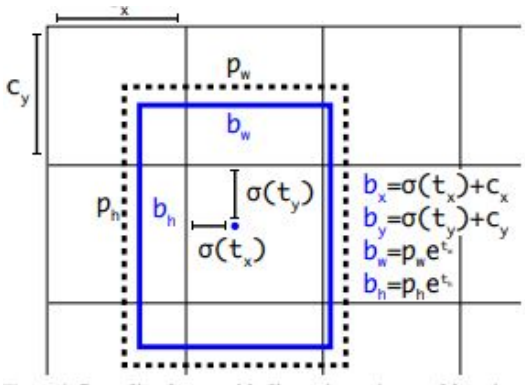


Figure 13: YOLOv3 predicts the width and height of the box as offsets from cluster centroids. It predicts the center coordinates of the box relative to the location of filter application using a sigmoid function.

6.4 YOLO Loss Function

Unlike YOLOv1[15], YOLOv3[14] use Binary Cross Entropy (BCE) loss for no objectness prediction, and use a logistic regression for objectness prediction. The ultralytics' implementation provide both YOLOv1 loss and YOLOv3 loss for training, but in my implementation, I mixed up two versions of loss into a single one. Practically, MSE loss in YOLOv1 is used for objectness score and BCE loss is used for no objectness prediction. The MSE loss for box prediction is tied to the offset with respect to the bounding box priors of each grid cell. During training we optimize the following, multi-part loss function:

$$
\begin{aligned}
L_{\text{YOLO}} = \lambda_{\text{class}} & \sum_{i=0}^{S^2} \mathbb{I}_i^{\text{obj}} \sum_{c \in \text{classes}} \text{CrossEntropyLoss}(p_i(c), \hat{p_i}(c)) \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{noobj}} \text{BinaryCrossEntropy}(C_i, \hat{C}_i) \\
& + \lambda_{\text{obj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{obj}} \text{MSELoss}(C_i, \text{iou} * \hat{C}_i) \\
& + \lambda_{\text{box}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{obj}} \text{MSELoss}((t_x, t_y, t_w, t_h), (\hat{t}_x, \hat{t}_y, \hat{t}_w, \hat{t}_h))
\end{aligned}
$$

where $\mathbb{I}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{I}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction. Since λs in the loss function can be arbitrary parameters for our tuning, we will mark the YOLO loss as $L(\lambda_{\text{class}}, \lambda_{\text{obj}}, \lambda_{\text{noobj}}, \lambda_{\text{box}})$. The details of parameter selection will be discussed later in the experiment section.

7 Datasets

In this section we will give a detailed description of datasets for image classification and object detection tasks respectively.

7.1 Image Classification: CIFAR-100

The CIFAR-100 dataset[10] (Canadian Institute for Advanced Research, 100 classes) is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. There are 600 images per class. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). There are 500 training images and 100 testing images per class.

7.2 Object Detection: VOC2007 and VOC2012

VOC2007[5] This dataset is the official dataset of *The PASCAL Visual Object Classes Challenge 2007*, aiming to recognize objects from a number of visual object classes in realistic scenes. It is fundamentally a supervised learning problem in which a training set of labeled images provided. There are twenty selected object classes listed as follows:

- *Person*: person
- *Animal*: bird, cat, cow, dog, horse, sheep
- *Vehicle*: aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor*: bottle, chair, dining table, potted plant, sofa, tv monitor

Specifically, this dataset consists of a set of images; each image has an annotation file giving a bounding box and object class label for each object in one of the twenty classes present in the image. This dataset has in total 9,963 images, containing 24,640 annotated objects. For trainval set, there are 5,011 images with 12,608 annotated objects. For test set, there are 4,952 images with 12,032 annotated objects.

VOC2012[6] This dataset is the official dataset of *Visual Object Classes Challenge 2012*. It has the same object classes as VOC2007. This dataset could be seen as an upgraded version of VOC2007, but the test set is not public, so we only have its trainval set. Specifically, it has 11,540 images with 27,450 annotated objects.

For visualization, we show several images in VOC dataset with bounding boxes and class labels annotated in figure 14.

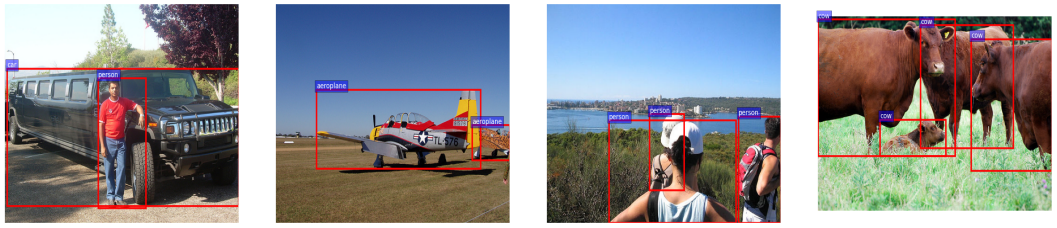


Figure 14: Visualization of four images in VOC dataset with bounding boxes and object classes annotated.

8 Experiments for Image Classification

8.1 Training Details

In our experiments concerning CIFAR-100 image classification, we use the refined AlexNet mentioned in section2.1, ResNet-110 with smaller structure and refined ResNet-18 mentioned in section2.2. The ResNet-110 using the bottleneck structure so that the 110 layers come from equation $9n + 2$ with $n = 12$, and refined ResNet-18 use the almost the same structure mentioned in [8], just remove the max pooling and change the shape of input convolution layer. Numbers of parameters are shown bellow:

Network	Refined AlexNet	ResNet-110	Refined ResNet-18
# parameters	58.7M	1.17M	11.2M

(9)

All the training use SGD with momentum as optimizer, with 0.9 momentum and 1e-4 weight-decay. We choose $\alpha = 10$ for $\mathrm{Beta}(\alpha, \alpha)$ distribution when applying CutMix augmentation, and $\alpha = 1$ for MixUp and CutOut augmentation. For AlexNet, we start the cosine annealing process with initial learning rate 0.01 and train 200 epochs without restart mentioned in [12]. For ResNet-110 and refined ResNet-18, we start with initial learning rate 0.25 and apply two restarts. Number of epochs for each phase are 40, 80, 160, with 280 total epochs.

8.2 Results

The figures in 15 show the results of image classification. The figure15a15b show accuracy and loss history of AlexNet. The gray, dark blue, azure, pink and red lines indicate CutMix augmentation, MixUp augmentation, CutOut augmentation, baseline method and the original AlexNet result (with local response norm) respectively. Figure15c15d show history of ResNet-110 with bottleneck. The azure, green, orange and dark blue lines indicate CutMix, Mixup, CutOut and baseline method. Figure15e15f show the history of refined ResNet-18, the orange, pink, red and green lines indicate CutMix, MixUp, CutOut and baseline method respectively. The bellow table summary the results.

Method / Accuracy / Network	Refined AlexNet	ResNet-110	Refined ResNet-18
baseline	71.35%	75.71%	77.14%
CutMix	74.47%	79.78%	80.46%
MixUp	72.53%	78.49%	79.59%
CutOut	72.55%	77.10%	78.49%

(10)

We could see that simply alter the structure of ResNet-18 might be a better choice above all the networks. Though ResNet-110 has the smallest number of parameters (about 10% of refined ResNet-18), but the GPU memory consumption is almost the same as ResNet-18, because the much bigger number of residual connections needs more memory when apply backpropagation during training. Moreover, the refined ResNet-18 could be trained faster (only takes about 15% time as the ResNet-110 needs). But if we just focus on the network performance, we might say ResNet-110 is better because we could achieve similar accuracy with much smaller number of parameters. And the refined AlexNet shows that batch normalization works well on generalization, it raise about 5% performance with baseline method once we applied it. But AlexNet itself is out-of-date, the intention we done these experiments just out of curiousity that how much could batch normalization do.

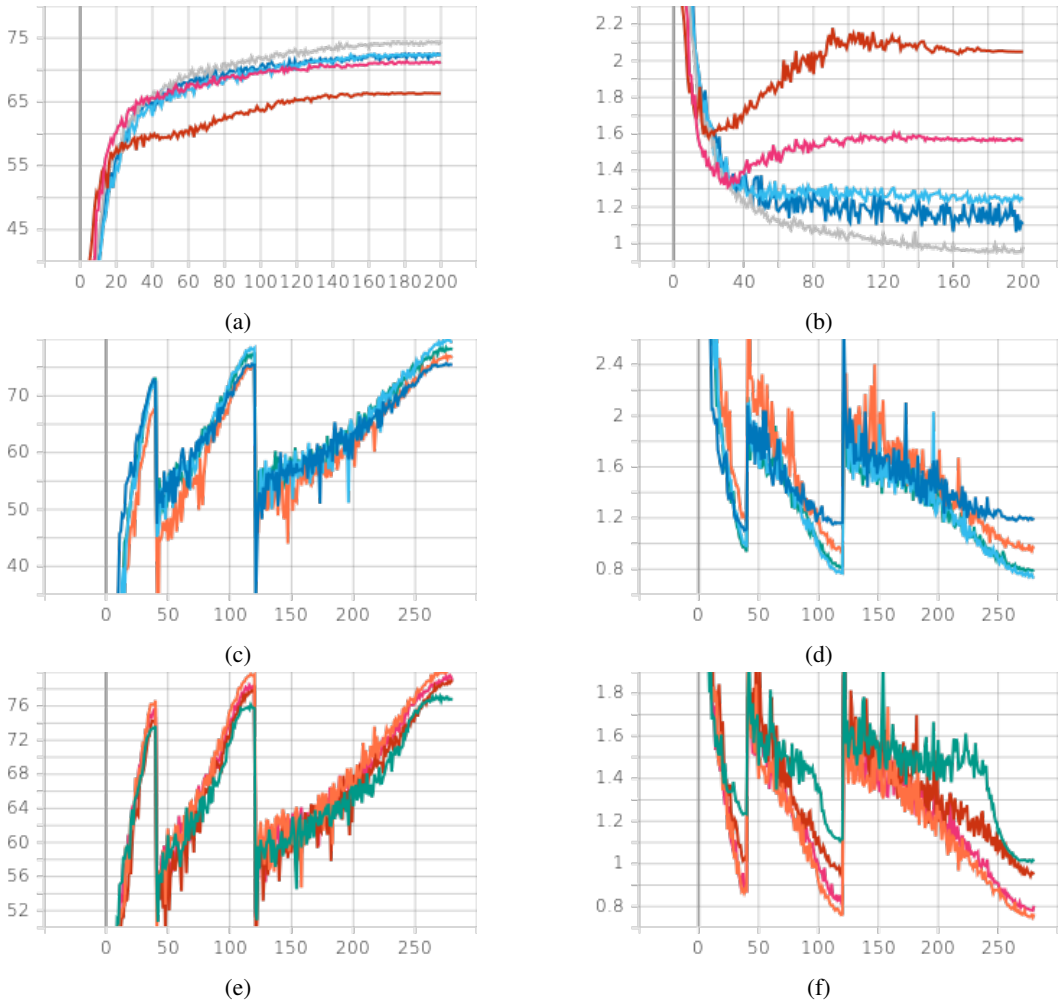


Figure 15: Results of image classification, figures in the left panel show the accuracy on the test set while the ones in the right panel show the error history on test instances

9 Experiments for Object Detection: Faster R-CNN

To train Faster R-CNN on VOC dataset, we first trained our model on VOC2007, then trained our model with previous hyperparameters on the merged dataset VOC07+12, and we then got a very good result on VOC2007 test data. The following of this section are the training details of our experiment with Faster R-CNN.

9.1 Dataset

The first dataset we used is VOC2007. The official dataset already provides us with both training set, validation set and test set. Specifically, the training set has 2,501 images with 6,301 annotated objects, validation set has 2,510 set with 6,307 annotated objects and test set has 4,952 images.

The second dataset we used is VOC07+12. This set is a combination of VOC2007 and VOC2012. It has in total 16,551 images for training and validation. We randomly split the dataset with the proportion 9:1 for training and validation.

Note that there's no new test data in VOC07+12, we test our model only on VOC2007 test set.

9.2 Loss Function

The loss function we use is the same as we defined in equation 8. Here we set $\lambda = 1$.

9.3 Training Details

Fine tuning As the whole architecture has a large number of parameters, and the training of deep convolutional network, namely ResNet-50, is very difficult due to our lack of computation ability and size of dataset, so we use fine tuning such that we can fully exploit the feature extracting ability of the pretrained deep convolutional network. Specifically, we use ResNet-50 model pretrained on ImageNet to initialize our backbone, and randomly initialize the hyperparameters of RPN with normal distribution. Then we freeze the ResNet-50 feature extracting part to learn the parameters of RPN until convergence, and finally unfreeze the backbone to train the whole network together so that there is only slight adjustments of parameters.

Learning Rate Decay We first set our initial learning rate as 1×10^{-4}, while for the first four epochs, the learning rate is smaller than initial learning rate. We choose this strategy so that we can find a good descending direction at starter. Then as the learning rate reaches initial learning rate, we apply cosine annealing strategy introduced as equation 7 for learning rate decay. Indeed in our experiment, this strategy works very well and the loss decreases steadily.

Optimizer We choose *adam optimizer* to optimize our model and set the momentum as 0.9. We choose this because *adam* leads to a good convergence result than SGD and is less sensitive to pre-set hyperparameters.

Other details We set the initial learning rate as 1×10^{-4} and minimum learning rate as 1×10^{-6}. The number of epochs is 100, with 50 for freezing training and 50 for unfreezing training. During freezing training, we set the batch size as 8, and 4 for unfreezing training. The input images are resized to 600×600. The whole experiment is carried out with RTX 3090, 24GB.

10 Experiments for Object Detection: YOLOv3

First of all, we have to point out that due to the limitations of computational resources, our backbone of Darknet-53 has not been pretrained on Imagenet. This will significantly affect the results of mAP on our test set. We trained YOLOv3 on a customized VOC07+12 dataset which consists of 16,551 images for training and 4,952 images for testing. The following sections will discuss the details and observations of our experiments.

10.1 Training Details

The whole experiment is conducted with one NVIDIA A40, 48GB. Initial learning rate is set to 1×10^{-4} and the weight decay is set to 5×10^{-4}. For comparison experiments, we train each model with 100 epochs. After choosing a set of good parameters, the model will be trained for 250 epochs in total. Cosine annealing learning rate scheduler is used to help search for global optimum, and automatic mixed precision is used to speed up the training process.

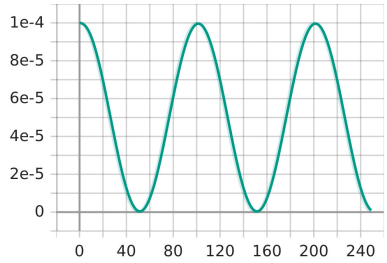


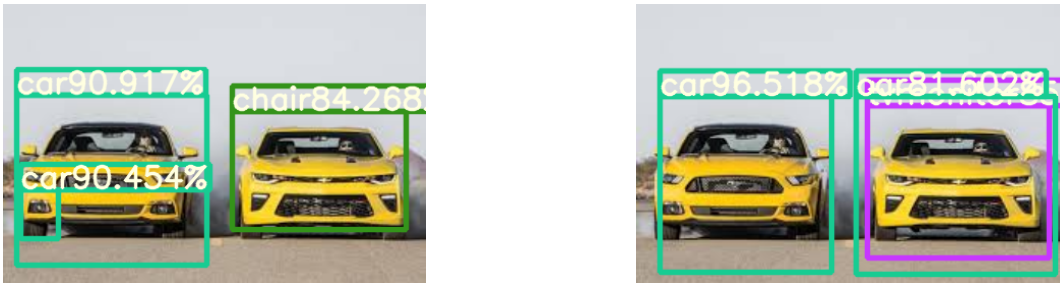
Figure 16: Learning rate fluctuation follows cosine annealing scheduler

10.2 Loss Function

Frankly speaking, the choice of loss function decides the results of our experiments. Recall that YOLO loss is summarized as $L(\lambda_{\text{class}}, \lambda_{\text{obj}}, \lambda_{\text{noobj}}, \lambda_{\text{box}})$. Our objective is to find a better set of parameters that provides the best results in application. We will begin with all equal weights. The selection of parameters might be heuristic, but our selection ($L(1, 10, 1, 10)$) did achieve a good result on test set. We will illustrate the parameter selection process with examples.

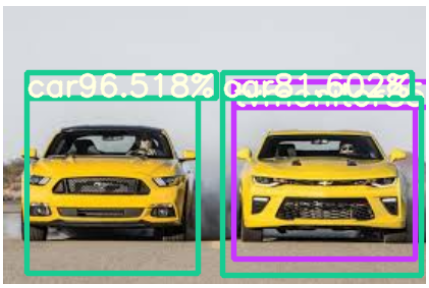
(a) $L(1, 1, 1, 1)$ (b) $L(1, 1, 10, 1)$

Figure 17: **L(1, 1, 1, 1) v.s L(1, 1, 10, 1)** When we experiment over $L(1, 1, 1, 1)$ as the loss function, the result of detecting multiple objects was disappointing. As is shown in 17a, our detector succeeded in identifying the dominating class of the objects. However, it failed to exploit the quantitative feature of the objects. Therefore, the first idea is to increase the penalty for failing to mark potential objectness, but the result 17b was dissatisfying. This indicates that we should not waste time fine tuning it.

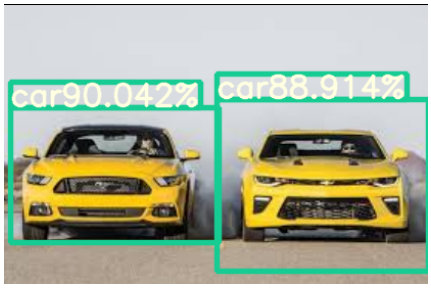


(a) $L(1, 1, 1, 1)$ (b) $L(1, 1, 1, 10)$

Figure 18: **L(1, 1, 1, 1) v.s L(1, 1, 1, 10)** The second experiment was successful. The intuition was to penalize over the bounding box errors, so that every box can be fixed correctly to the target before non-maximum suppression eliminates the separate bounding boxes for each object. As is shown in 18b, our detector succeeded in identifying multiple cars. The remaining problem was the redundancy of proposed boxes.



(a) $L(1, 1, 1, 10)$



(b) $L(1, 10, 1, 10)$

Figure 19: **L(1, 1, 1, 10) v.s L(1, 10, 1, 10)** To remove the unexpected bounding boxes, we increase the penalty for no objectness. The detector started to avoid proposing random bounding boxes. But there was no significant improvement in mAP. More discussion over this phenomena could be found in the rebuttal part of YOLOv3 paper[14]

.

11 Results for Object Detection

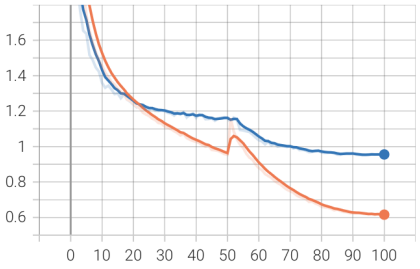
11.1 Proposal Boxes of RPN

We visualize a proper proportion of top score proposed regions produced by the RPN. They are shown in figure 20. We can see that the more close to non-background objects, the more proposed regions are.

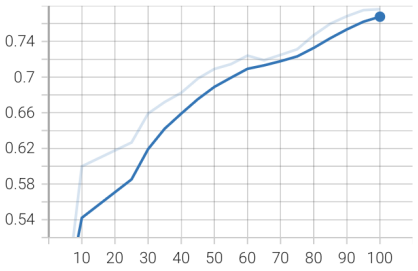
Figure 20: Visualization of four images in VOC dataset with bounding boxes and object classes annotated.

11.2 Training History

Faster R-CNN The loss history of training Faster R-CNN with backbone ResNet-50 on VOC07+12 is shown in figure 21a. For the first 50 epochs, we freeze the backbone to update the parameters of RPN only. From the validation, we can see that at around 45th epoch, the model almost converges. For the last 50 epochs, we unfreeze the backbone and this caused a spike of training loss, this is because the number of training parameters increases and the network can adjust all of them immediately, so the loss increases. Moreover, the validation loss start to decrease, because the feature extracting network can update its parameters so that to become more self-adjusted to this VOC dataset. At around 90th epoch, we can see that both validation loss and training loss hardly change, this means that the network converges. The $\text{mAP}_{0.5}$ value on validation set is shown in figure 21b, due to the time needed to compute mAP, we record it every 5 epoch. We can see that it finally reached around 77% on validation set and implies convergence.



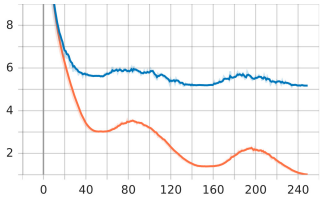
(a) Loss history of training on VOC07+12, with orange curve being training loss and blue one validation loss.



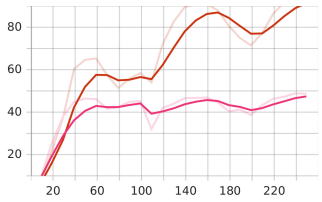
(b) $mAP_{0.5}$ of validation set, we record it every 5 epoch.

Figure 21: Loss history and mAP history of Faster R-CNN on VOC07+12.

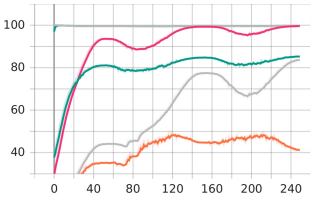
YOLOv3 The cosine annealing strategy works well in YOLOv3 case. We can see the loss history22a of training and testing decreases periodically. But unfortunately, the problem of overfitting could not be solved. We have tried out different methods including weight decay and data augmentations, but it seems too hard to minimize the gap between training and testing loss. It is plausible to blame the fact that our Darknet-53 had not been pretrained on Imagenet before being incorporated to the architecture. Other techniques such as Dropout, CutMix, and Mosaic, which were proposed in YOLOv4[1] might be helpful, but due to the limitation of time, I have no time to implement them from scratch. The test mAP was raised to 48% after 250 epochs, but we did not witness an improvement of the detection performance in practice, compared with the checkpoint after 150 epochs.



(a) Loss history of training on VOC07+12, with orange curve being training loss and blue one validation loss.



(b) $mAP_{0.5}$ of train set (orange) and test set (pink), we record it every 10 epoch.



(c) Other metrics for reference. Pink and green line for class accuracy on train set and test set respectively. Grey and orange for object accuracy.

Figure 22: Loss, mAP and other useful metrics of YOLOv3 on VOC07+12.

11.3 Model Evaluation

We trained two models with ResNet-50 as backbone on VOC2007 and VOC07+12 respectively, and test both of them on VOC2007 test set. The evaluation of both models are shown in table 1. Here we compute $mAP_{0.5}$, and set confidence score as 0.5 and NMS IoU as 0.3. We can see that Faster R-CNN trained on VOC07+12 performs far better than the one trained on VOC07, which indicates that the size of training dataset plays a vital role in training a good model. Moreover, our Faster R-CNN model did a poor job on tiny objects, with only 60.6% on bottle and 49.2% on plant. We speculate that there are two main theories that can explain this:

- **Subsampling:** Let us suppose that we have a tiny object with size 15×15, and subsampling rate is always 16, it means that this tiny object does not count even one pixel on feature maps.
- **Receptive Field:** As in convolutional networks, the receptive field is always very large, so the feature map may contain many features surrounding this tiny object, thus the feature owned by this object on feature map is very limited, and this can cause the poor detection result of tiny objects.

The mAP of YOLOv3 is slightly disappointing. This is largely affected by the fact that our backbone Darknet-53 had not been pretrained on Imagenet. There could still be some improvement, as in original YOLOv3[14] paper, the best mAP for VOC07+12 is 57.9. We have not yet exploit the full potential of our backbone.

Table 1: Results on VOC2007 test set with Faster R-CNN, backbone ResNet-50 and YOLOv3.

model	data	mAP	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
FR-CNN	07+12	79.1	82.1	84.7	79.3	71.3	60.6	87.7	87.2	89.7	61.4	83.8	77.3	87.4	88.9	84.1	82.0	49.2	80.9	79.4	87.1	77.6
FR-CNN	07	65.7	68.5	77.6	66.7	52.1	41.7	75.1	79.6	79.9	43.4	61.4	63.7	75.7	81.0	71.5	74.5	40.7	54.5	63.2	76.3	66.3
YOLOv3	07+12	48.6	58.0	60.9	34.8	32.4	17.4	63.0	69.9	57.5	29.5	44.7	48.4	48.9	59.3	63.3	52.8	19.1	49.0	51.5	56.9	55.3

And mIoU and accuracy of each model is shown in table 2. We can see the result remains consistency: the better a model is, the higher its mIoU and accuracy are.

Table 2: mIoU and accuracy of object detection models

model	data	mAP	mIoU	Accuracy
Faster R-CNN	VOC07	65.7	81.0	89.5
Faster R-CNN	VOC07+12	79.1	88.6	94.1
YOLOv3	VOC07+12	48.4	70.4	85.1

11.4 Result Visualization

The test result of Faster R-CNN model is visualized in figure 23. We can see that our Faster R-CNN model performs pretty well on regular size objects such as person, motorcycle and dog. However, it still misclassifies some objects, for example, classifying wood background for chairs, and can not recognize small figures sometimes. The speculations proposed above and the lack of certain kind label of data may be a reasonable explanation for this.

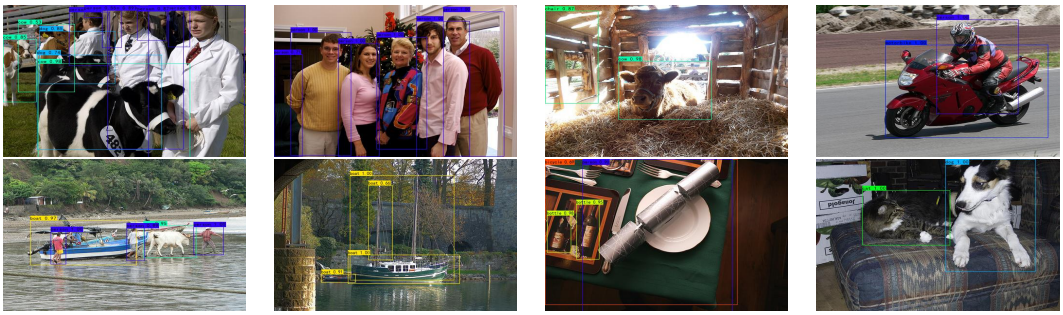


Figure 23: Faster R-CNN test result visualization

The test result of YOLOv3 model is visualized in figure 24. We can see the final results of our detector was relatively satisfying, despite that it cannot deal with the situations where multiple objects are present in the images. In our experiments of video detections, however, YOLOv3 performs quite well. It detects the video frame over 30FPS on my personal laptop, which enables real-time object detection with the camera. You may also try out the real-time detection using the detection class uploaded to the github repo.

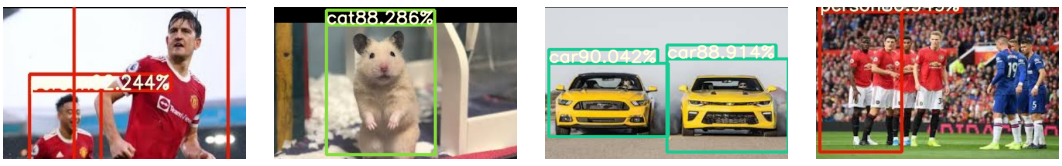


Figure 24: YOLOv3 test result visualization

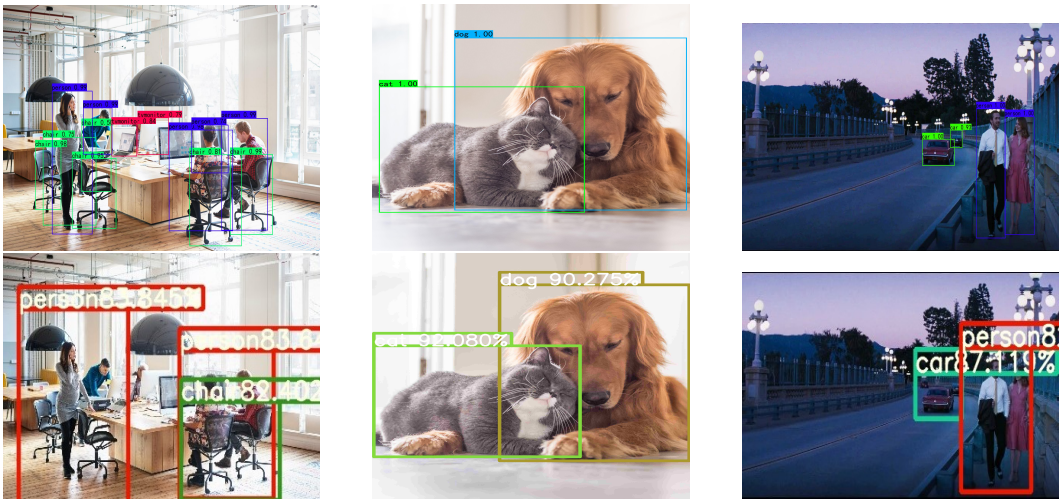


Figure 25: Comparison of Faster R-CNN (above) and YOLOv3 (below) on customized data

Figure 25 shows the result on customized data. Faster R-CNN did well in detecting multiple small objects in the environment. However, YOLOv3 failed to distinguish different items from each other.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. URL `https://arxiv.org/abs/2004.10934`.

[2] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL `https://www.mdpi.com/2078-2489/11/2/125`.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html, 2007.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[7] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. doi: 10.1109/ICCV.2015.169.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[12] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[13] Joseph Redmon. Darknet: Open source neural networks in c, 2013.

[14] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. URL `https://arxiv.org/abs/1804.02767`.

[15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015. URL `https://arxiv.org/abs/1506.02640`.

[16] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL `http://arxiv.org/abs/1506.01497`.

[17] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[18] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.