Computer Vision Midterm Project

Miaopeng Yu
School of Data Science
Fudan University
19307130203@fudan.edu.cn

Yi Shao
School of Data Science
Fudan University
19307130113@fudan.edu.cn

Xinkai Lv
School of Data Science
Fudan University
19300180115@fudan.edu.cn

Abstract

In the first part of the project, we use several CNNs to build classifiers for CIFAR-100, and try to use some augmentation methods like cutmix, cutout ,and mixup to improve the model's accuracy. And we finally get a top-1 accuracy of 83.01% and top-5 accuracy of 95.70% using the WideResNet-28. In the second part of the project, we train VOC 2007 using Faster R-CNN and YOLO v3, by the help of pretrained models, we finally gain the mAP0.5 of 77.44% and 86.83% respectively, we also test both models on several images that do not belong to the VOC dataset and get promising results.

Repo: `https://github.com/Bbbstin/CV_Midterm`

Best model: `https://github.com/Bbbstin/CV_Midterm/releases/tag/publish`

1 Introduction

1.1 Introduction to the Two Tasks

In the first part of the project, we need to train some CNN models using the dataset CIFAR-100. At the same time, compare the performance of using CNNs with cutmix, cutout, mixup and baseline (without using the three augmentation methods mentioned here). And visualize three images after using these three methods.

In the second part of the project, we need to train two object detection models - Faster R-CNN and YOLO v3 on the VOC dataset. And visualize the proposal box of Faster R-CNN on four test images. After training, test the two trained models on three new images that do not belong to the VOC dataset, and visualize the results.

1.2 General Organization of this Report

Since the two tasks are not quite related, we decided to divide the report into two parts. The outline of this report is as follows. In section 2, we describe the details of the background and implementation of task 1. In section 3, we describe the details of the background and implementation of task 2. In section 4, we give our conclusion of this project. We can see that we just provided a general organization here, a more detailed organization of each task is provided in 2.1 and 3.1.

2 Task 1: Train CNNs on CIFAR-100

2.1 Introduction to Task 1

In the first task, we first use ResNet-18 as our baseline network. We applied the three different augmentation methods - cutmix, cutout and mixup to the baseline, and get the accuracy of using them. And cutmix performs one of the best among them, and has an accuracy of 79.47%. After that, we apply the cutmix to the following CNNs we are testing. And WideResNet-28 reaches top-1 accuracy of 83.01%, and top-5 accuracy of 95.70%.

The organization of section 2 is as follows. In section 2.2, we introduce the dataset CIFAR-100, and how we divide this dataset into the training set and the test set. In section 2.3, we introduce the three augmentation methods we are using in this task - cutmix, cutout and mixup, and visualize three images after using these methods. In section 2.4, we briefly introduce the network structure we are using, like ResNet (we choose ResNet-18 as our baseline), ResNeXt, Dual Path Network (DPN), Deep Layer Aggregation (DLA), etc. Since we think the report will be tedious if we show the models detailedly, we only show the advantages and the basic idea of these models. In section 2.5, we show the implementation details like the batch size, learning rate, optimizer and so on. In section 2.6, we report our results using these models, and visualize the training loss and the testing loss, and the training and testing accuracy of the models using TensorBoard.

2.2 Dataset CIFAR-100 and its Division

The CIFAR-100 dataset (Canadian Institute for Advanced Research, 100 classes) is a labeled subset of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. And it has 100 classes containing 600 images each, 500 of them are training images and 100 of them are testing images. So for the totally 60,000 images, 50,000 of them are assigned to the training set, and 10,000 of them are assigned to the test set. Also, each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). And we are using the "fine" label. [1]

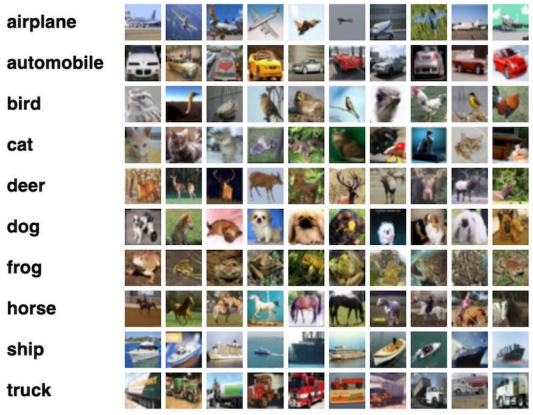


Figure 1: A subset of CIFAR-100 [1]

2.3 Augmentation Methods

2.3.1 Cutmix

CutMix is an image data augmentation strategy. Instead of simply removing pixels as in Cutout, we replace the removed regions with a patch from another image. The ground truth labels are also mixed proportionally to the number of pixels of combined images. The added patches further enhance localization ability by requiring the model to identify the object from a partial view. [2]

2.3.2 Cutout

Cutout is an image augmentation and regularization technique that randomly masks out square regions of input during training. And it can be used to improve the robustness and overall performance of convolutional neural networks. The main motivation for cutout comes from the problem of object occlusion, which is commonly encountered in many computer vision tasks, such as object recognition, tracking, or human pose estimation. By generating new images which simulate occluded examples, we not only better prepare the model for encounters with occlusions in the real world, but the model also learns to take more of the image context into consideration when making decisions. [3]

2.3.3 Mixup

Mixup is a data augmentation technique that that generates a weighted combinations of random image pairs from the training data. Given two images and their ground truth labels: (x_i, x_i) , (x_j, x_j), a synthetic training example $(\hat{x}, \hat{y})$ is generated as:

$$\hat{x} = \lambda x_i + (1 - \lambda) x_j$$
$$\hat{y} = \lambda y_i + (1 - \lambda) y_j$$

where $\lambda \sim \mathrm{Beta}(\alpha = 0.2)$ is independently sampled for each augmented example. [4]

2.3.4 Visualization of These Three Methods

We use the images on the CIFAR-100 dataset, and apply the three augmentation methods to these images. The images after transformation are shown below in Figure 2.

The image in the first column is the image to mixup or cutmix with. The images in the second column are three different original images. The images in the third column are the images after using cutmix. And the fourth column is after using cutout, and the last column is after using mixup.

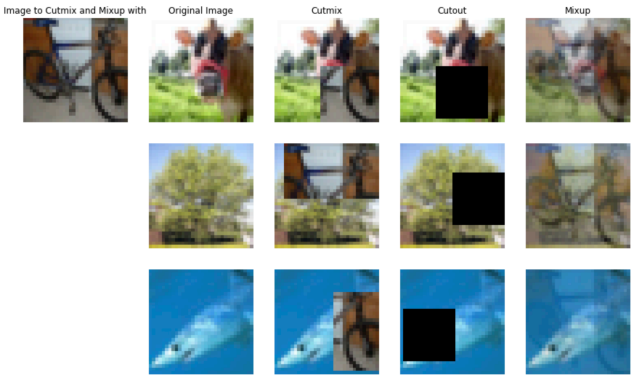


Figure 2: Visualization of the Three Methods

2.4 CNNs Model Structure

In this subsection, we will briefly introduce the network we use in task 1. Since we think the report will be tedious if we introduce each network detailedly, we only show the advantages of these models and their main ideas.

2.4.1 ResNet

ResNet introduces the residual connection to the network, which can help to decrease the intrinsic complexity of the model, and since

$$\frac{\partial(f(g(x)) + g(x))}{\partial x} = \frac{\partial g(x)}{\partial x} + \frac{\partial f(g(x))}{\partial x}$$

the shortcut connection can help to solve the gradient vanishing problem. It is after the ResNet we can train a much deeper model.

The blocks ResNet is using are shown below, the left one is called basic building block, which is used in ResNet-18/34, and the right one is called bottleneck building block, which is used in ResNet-50/101/152. [5]

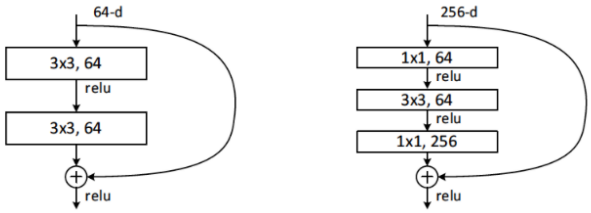
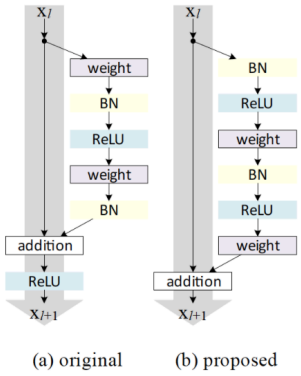


Figure 3: Building Blocks of ResNet

2.4.2 PreActResNet

In the original Residual Unit, although the Barch Normalization (BN) normalizes the signal, this is soon added to the shortcut, and thus the merged signal is not normalized. This unnormalized signal is then used as the input of the next weight layer.

On the contrary, the inputs to all weight layers have been normalized in the pre-activation version, which can help make the training more robust. And the pre-activation version just simply moves the BN and non-linear activation before the convolution layers. And this is why it is called PreActResNet. And the structure is shown below. [6]



(a) original (b) proposed

Figure 4: Residual Unit of PreActResNet

2.4.3 WideResNet

Wide Residual Network proposes that as the gradient flows through the network there is nothing to force it to go through the residual block weights and thus it can avoid learning during training, so it is possible that there is either only a few blocks that learn useful representations, namely many blocks share very little information with small contribution to the final goal. This problem is formulated as diminishing feature reuse. And as widening the residual blocks, dropout should be inserted between convolutional layers to regularize training and prevent overfitting. And the structure is shown in Figure 5 [7]

2.4.4 ResNeXt

ResNeXt is constructed by repeating a building block that aggregates a set of transformations with the same topology, it exploits the split-transform-merge strategy in an easy, extensible way. The building block of ResNeXt is shown in Figure 6 (right). [8]

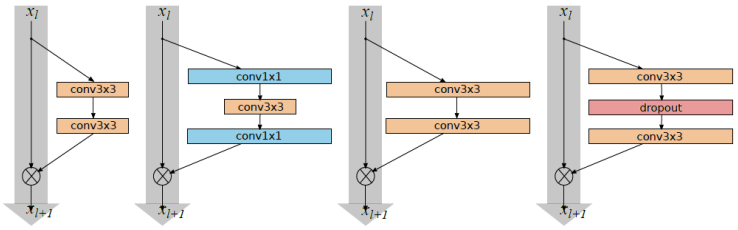


Figure 5: WideResNet Structure

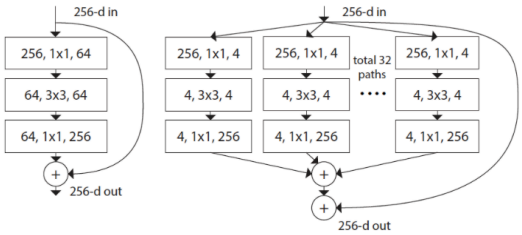


Figure 6: **left**: A block of ResNet [Resnet]. **Right**: A block of ResNeXt with cardinality=32. A layer is shown as (# in channels, filter size, # out channels)

The model (if using the building block shown in Figure 6) first splits the input's channel into 32 groups (cardinality), then perform the same thing as ResNet independently in each group, then adds them together. By using the building block with cardinality, the width of the model can be increased without increasing the number of parameters and FLOPs. [resnext] For example, the number of parameters of the building block on the left (ResNet) is approximately $256 \times 1 \times 1 \times 64 + 64 \times 3 \times 3 \times 64 + 64 \times 1 \times 1 \times 256 = 69632$, the number of parameters of the building block on the right (ResNeXt) is approximately $(256 \times 1 \times 1 \times 4 + 4 \times 3 \times 3 \times 4 + 4 \times 1 \times 1 \times 256) \times 32 = 70144$. While the left one have only 64 channels, and the right one have 128 channels.

2.4.5 DenseNet

To solve the problem that when CNNs become increasingly deep - the information can vanish and "wash out" by the time it reaches the end (or the beginning of the network), we create short paths. Instead of connecting all layers by adding them together as ResNet does, DenseNet connects all layers by concatenating them. [9]

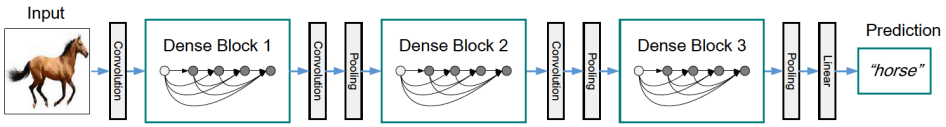


Figure 7: A Deep DenseNet with Three Dense Blocks

It is easy to train, since each layer has direct access to the gradients from the loss function and the original input signal, leading to implicit deep supervision. Also, DenseNet can reuse the features, so that some more high-level features can be produced later in the network. [9]

2.4.6 Dual Path Network (DPN)

ResNet enables feature re-usage, while DenseNet enables new feature exploration. And DPN combines both of them to enjoy the benefit of both topologies. [10] And the structure of DPN can be easily understood in Figure 8. We first pass a convolution layer for all channels. And then split the output into two parts: while the first part acts like ResNet (Residual Connection), the second part acts like DenseNet (Dense Connection). Then, we concatenate the two parts together. [10]

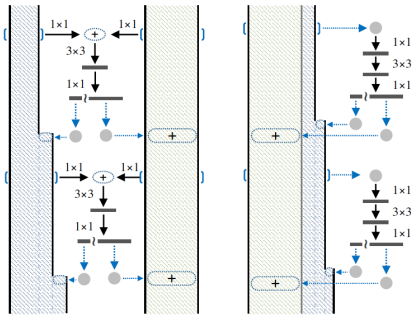
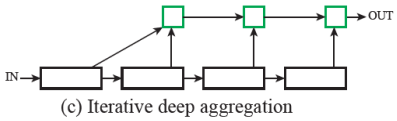


Figure 8: DPN

2.4.7 Deep Layer Aggregation (DLA)

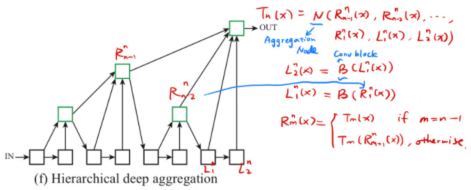
Aggregate layers can help to better fuse semantic and spatial information for recognition and localization. DLA is a tree structure CNNs built from two parts: iterative deep aggregation (IDA) and hierarchical deep aggregation (HDA). IDA focuses on fusing resolutions and scales, while HDA focuses on merging features from all modules and channels, with which shallower and deeper layers are combined to learn richer combinations. [11]



(c) Iterative deep aggregation

$$I(x_1, \cdots, x_n) = \begin{cases} x_1 & if\ n = 1 \\ I(N(x_1, x_2), \cdots, x_n) & otherwise \end{cases}$$

(a) IDA



(f) Hierarchical deep aggregation

(b) HDA

Figure 9: Structure of IDA and HDA

where in Figure 9, N is the aggregation node to combine and compress the inputs (we choose $N(x_1, \cdots, x_n) = \sigma(BatchNorm(XW) + X)$, where σ is the non-linear activation here). And for a better reading experience, we add some notations to the images.

And the structure of DLA is shown in Figure 10.

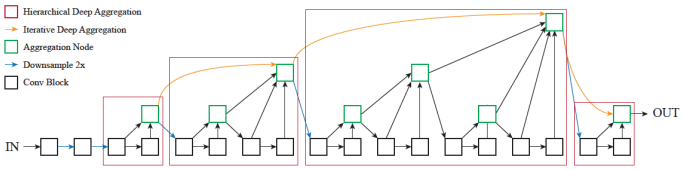


Figure 10: Structure of DLA. DLA learns to better extract the full spectrum of semantic and spatial information from a network. Iterative connections join neighboring stages to progressively deepen and spatially refine the representation. Hierarchical connections cross stages with trees that span the spectrum of layers to better propagate features and gradients. [11]

2.5 Implementation Details

We train the models on the 50,000 training set and evaluate on the 10,000 test set. The input image is normalized to $\mu = 0, \sigma = 1$ and randomly cropped to 32×32 from a zero-padded 40×40 image or its flipping. The models are trained on 1 GTX3090 with a batch size of 256, with SGD with a weight

decay of 0.0005 and a momentum of 0.9. We start with a learning rate of 0.1 and train the models for 300 epochs, reducing the learning rate using the cosine annealing with $T_{max} = 300$. And the loss function we use is cross-entropy loss, we select the best model according to the accuracy in the test set.

And since we found that for the three augmentation methods - cutmix, cutout and mixup, cutmix performs one of the best in the ResNet-18 model (we will show that later in the next subsection). Therefore, we train the later models with cutmix as well.

2.6 Results and Visualization

We first train four ResNet-18 using cutmix, cutout, mixup and none of them for 200 epochs each. The result is shown in Table 1 We can see that using these three augmentation methods, the models

Table 1: Result of Different Augmentation Methods

model	augmentation Method	top-1 accuracy	top-5 accuracy
ResNet-18	baseline	77.55%	93.53%
	cutmix	79.47%	94.47%
	cutout	77.78%	93.51%
	mixup	79.53%	94.22%

indeed become more robust, and have a better performance than the baseline. And for cutmix and mixup, the performance has improved about 2 percentage, and the improvements are almost equally good.

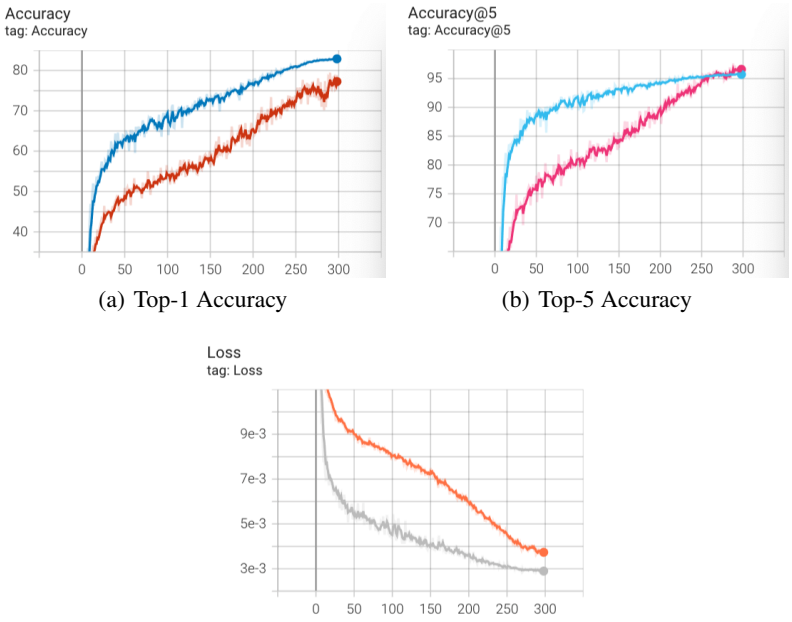
Therefore, we train the following models with cutmix for 300 epochs each, and the results are shown in Table 2.

Table 2: Result of Different Models on CIFAR-100

model	top-1 accuracy	top-5 accuracy
ResNet-50	81.30%	95.07%
PreActResNet-50	82.11%	95.36%
WideResNet-28	83.01%	95.70%
ResNeXt-29, 32×8d	80.35%	94.35%
ResNeXt-50, 32×4d	80.83%	94.79%
DenseNet-121	81.57%	94.63%
DPN-92	81.11%	94.38%
DLA-34	82.07%	95.11%

And the training and the testing loss, accuracy in the test set curve plot by TensorBoard is shown in Figure 11. These three plots are the training process of WideResNet. For the first two plots, the red curve is the training set accuracy, while the blue one is the test set accuracy. For the loss curve, the orange one is the training loss, and the gray one is the testing loss. Since we are using cutmix in the training set, the class of the image is more complex, it is reasonable that the training accuracy is lower than the testing accuracy.

We can see that WideResNet-28 performs the best among the several models, and its top-1 accuracy can reach 83.01%. However, some models do not perform as well as we expected, like ResNeXt, DPN and DLA. It is probably because the learning rate or augmentation method we choose is not suitable for these models. Due to computation resource-limited, we have not test other settings for these models. And we can try to change these settings, and we may get better performance for these models.



(a) Top-1 Accuracy

(b) Top-5 Accuracy

(c) Loss

Figure 11: Visualization of the Curves Using TensorBoard

3 Task 2: Train Faster R-CNN and YOLO v3 on VOC 2007 Dataset

3.1 Introduction to Task 2

In the second task, we train the Faster R-CNN and YOLO v3 models on the VOC 2007 dataset. Due to the computation resource-limited, we are not using VOC 2012. However, we found a well-trained model for Faster R-CNN and YOLO v3 using VOC 2007+2012 and COCO respectively, and therefore reaches a higher mAP up to 77.44% (Faster R-CNN) and 86.83% (YOLO v3). So, we will also report and compare the results of using these well-trained models and our own models. After that, we show some new images that do not belong to the VOC dataset, and apply these two models to them, and show the detection results.

The organization of section 3 is as follows. In section 3.2, we introduce the dataset VOC, and how we divide the dataset into different sets. In section 3.3, we introduce the first model Faster R-CNN. In section 3.4, we introduce the second model we are using in task 2 - YOLO v3. In section 3.5, we show the implementation details like batch size, learning rate, optimizer, etc. In section 3.6, we report our results using the two models, and visualize the training and testing loss, mAP and mIoU curve in the test set.

3.2 Dataset VOC and its Division

PASCAL VOC is a dataset for image recognition. And it is fundamentally a supervised learning problem in that training set of labeled images is provided. The twenty object classes include person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, TV/monitor. Each image in this dataset has pixel-level segmentation, bounding box annotations, and object class annotations. This dataset has been widely used as a benchmark for object detection tasks. [12]

We use VOC 2007 for the reason that it is the last year that provided the test set. Some people may use VOC 2012 in the training set as well. However, due to the computation resource-limited, we use only VOC 2007 here.

For VOC 2007, it is divided into the train/val set and the test set at about a ratio of 1:1, and the VOC 2007 dataset totally contains 99,63 images with 24,640 annotated objects. And for VOC 2012, it has only a train/val set with 11,530 images containing 27,450 annotated objects. [12]

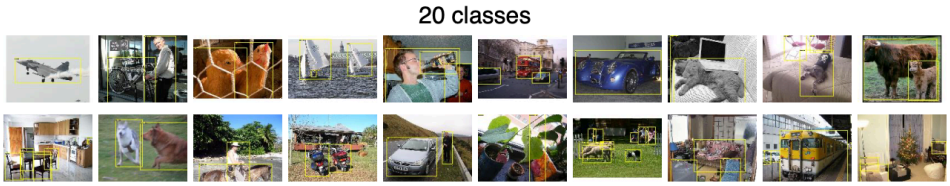


Figure 12: VOC 2007 Examples

And we divide 4952 images for the training set in this project, and a more detailed information is shown in Figure 13.

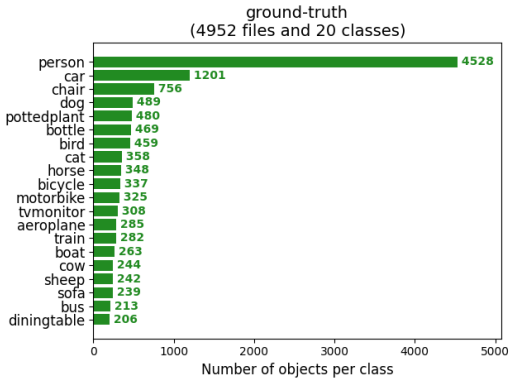


Figure 13: Ground Truth about the Training Set

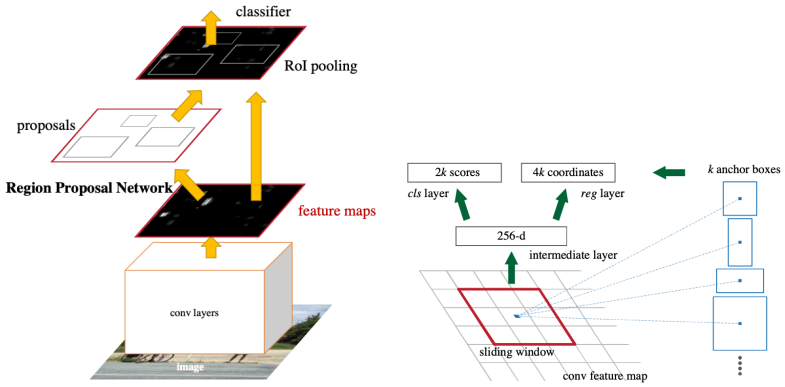
3.3 Faster R-CNN

Although region-based CNNs are computationally expensive as the originally developed in [13], their cost has been drastically reduced thanks to sharing convolutions across proposals like Fast R-CNN, which achieves near real-time rates using very deep networks, if we ignore the time spent on region proposals [14]. However, the time spent on region proposals is too long. So in Faster R-CNN, the author introduces a Region Proposal Network (RPN) that shares full-image convolution features with the detection network, and it can generate proposals with various scales and aspect ratios to tell the object detection (Fast R-CNN) where to look. And the RPN module serves as the "attention" of Faster R-CNN.

The architecture of Faster R-CNN is shown in Figure 14(a). It consists of 2 modules: RPN for generating the region proposals, and Fast R-CNN for detecting objects in the proposed regions.

After extracting features from the image using the backbone network. The next step of Faster R-CNN is generating region proposals using RPN. The RPN works on the output feature map returned from the last convolutional layer shared with the Fast R-CNN. Based on a rectangular window of size $n \times n$, a sliding window passes through the feature map. For each window, several anchor boxes are generated, and will be later filtered based on their "objectness score". Then we adjust these anchor boxes to get the proposal boxes.

After that, for all regions proposed in the image, a fixed-length feature vector is extracted from each region using the ROI Pooling layer. Then the extracted feature vectors are then classified using the Fast R-CNN and used to adjust the proposal box. Finally, the class scores of the detected objects in addition to their bounding boxes are returned. [15]



(a) Architecture of Faster R-CNN (b) Architecture of RPN

Figure 14: Architecture of Faster R-CNN and RPN

3.4 YOLO v3

YOLO is a Convolutional Neural Network for performing object detection in real-time. CNNs are classifier-based systems that can process input images as structured arrays of data and identify patterns between them (view image below). YOLO has the advantage of being much faster than other networks and still maintains accuracy.

It allows the model to look at the whole image at test time, so its predictions are informed by the global context in the image. YOLO and other convolutional neural network algorithms score regions based on their similarities to predefined classes.

High-scoring regions are noted as positive detections of whatever class they most closely identify with. For example, in a live feed of traffic, YOLO can be used to detect different kinds of vehicles depending on which regions of the video score highly in comparison to predefined classes of vehicles.

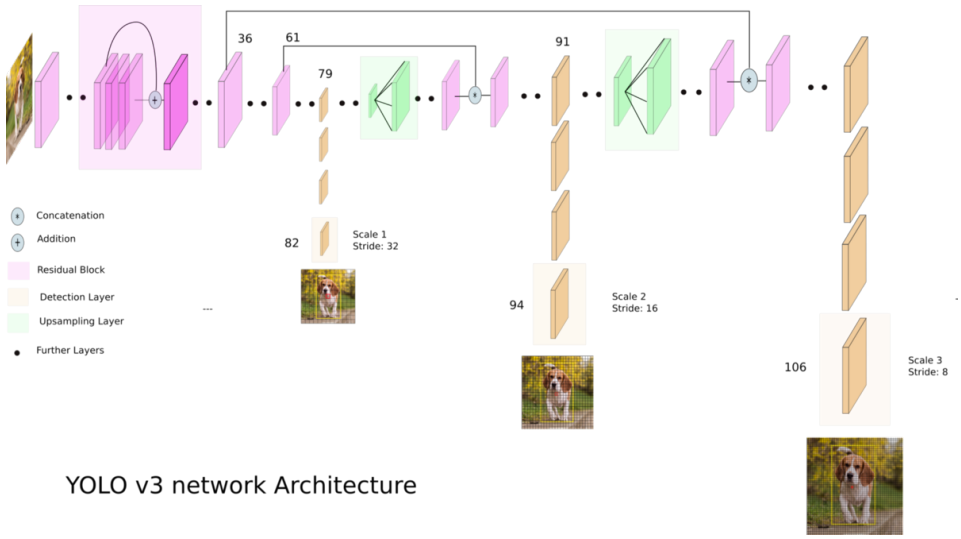


Figure 15: YOLO v3 Illustration

YOLO v3 is a real-time, single-stage object detection model that builds on YOLO v2 with several improvements. Improvements include the use of a new backbone network, Darknet-53 that utilises residual connections, or in the words of the author, "those newfangled residual network stuff", as well as some improvements to the bounding box prediction step, and use of three different scales from which to extract features.

As shown in Figure 15, 3 different scales are used to detect objects of different sizes. And several convolutional layers are added to the base feature extractor Darknet-53, which is called the backbone network. The feature map is taken from 2 layers previous and is upsampled by 2 times. A feature map is also taken from earlier in the network and merge it with our upsampled features using concatenation. This is actually the typical encoder-decoder architecture. This method allows us to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. [16]

3.5 Implementation Details

3.5.1 Implementation Details: Faster R-CNN

While training, we re-scale the images such that their shorter side is 600 pixels, since multi-scale feature extraction may improve accuracy but does not exhibit a good speed-accuracy trade-off [15]. The backbone we use in Faster R-CNN is pretrained ResNet-50. And we choose to use pretrained networks is for faster and better convergence of the network. For anchors, we use 3 scales with box areas of $128^2, 256^2, 512^2$ pixels, and 3 aspect ratios of 1:1, 1:2 and 2:1.

The optimizer we choose for this model is SGD with a weight decay of 0.00001 and a momentum of 0.9. We start with a learning rate of 0.01 and reducing the learning rate using cosine annealing. And we train the model for 150 epochs, in the first 50 epochs, we will freeze the backbone, and train the remained network only, the batch size we choose in these 50 epochs is 8. After 50 epochs, we unfreeze the backbone and train the whole Faster R-CNN together, the batch size we choose in the last 100 epochs is 4.

To reduce the redundancy, we adopt non-maximum suppression (NMS) on the propoosal regions based on their cls scores, we fix the IoU threshold for NMS at 0.3, and keep 12,000 anchor boxes according to their scores while training, and 6,000 anchor boxes while testing. After that, we keep 600 proposal boxes according to their scores while training, and 300 proposal boxes while testing. And we set the confidence to be 0.5.

And the loss function we choose is the same as the original paper. [15]

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

where p_i is the predicted probability of anchor i being an object. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box. And the elements with a $*$ is the ground-truth, and t and t^* are normalized to [-1, 1], L_{cls} is the log loss over two classes, $L_{reg}(t_i, t_i^*) = L_1(t_i, t_i^*)$. And N_{cls} is mini-batch size, N_{reg} is the anchor locations, and we set $\lambda = 10$ to balance the weight between the two part of the loss function.

3.5.2 Implementation Details: YOLO v3

Since YOLO v3 is not easy to train, we use the pretrained model from third party. In the training process, first we resize the images to a fixed size of 416×416 pixels, then we load data and feed it to the network.

The loss function is composed of four parts: the first one is x value and y value offsets after encoding the ground truth bounding boxes for positive samples, the second and third one are the differences of confidence score between the predicted class and the ground truth class for positive and negative samples respectively, and the last one is the loss of predicted class scores and ground truth class scores.

The optimizer for YOLO v3 is SGD with nesterov momentum of 0.9. We start with a learning rate of 5e-2 and reducing the learning rate using cosine annealing to 5e-4. And we train the model for 300 epochs with 50 freezing epochs at the beginning. After 50 epochs, we unfreeze the backbone and train the whole YOLO v3 together, the batch size is 16 and 8 for these two stages respectively.

3.6 Results and Visualization

3.6.1 Visualization of Proposal Boxes

We first show four images from the VOC 2007 dataset, and visualize the proposal boxes in the image. To make the image not so messy, we only plot 100 proposal boxes in each image. The images are shown in Figure 16, where the left column is the original image, and the right column is the images with proposal boxes on them.

And we can find that, for the location with some stuff, there are more proposal boxes assigned to that place, which can help to tell the network where to look.

Figure 16: Proposal Boxes on images

3.6.2 Results of Faster R-CNN and YOLO v3

We train and evaluate our models on the PASCAL VOC 2007 dataset. Since we found well-trained models, we also report mAP on the VOC 2007 of these models. The results are shown in Table 3.

Table 3: Result of Different Models on the VOC Dataset

model	backbone	data	mAP0.5
Faster R-CNN	ResNet-50	VOC2007	69.79%
		VOC2007+2012	77.44%
YOLO v3	Darknet-53	VOC2007+COCO pretrained	86.83%

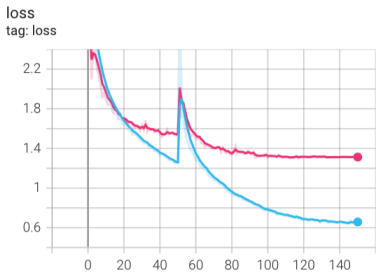
And the more detailed results are shown in Table 4

Table 4: Detailed Result of Different Models on the VOC Dataset

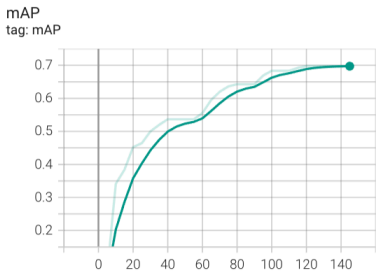
model	data	mAP0.5	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow
FRCNN	07	69.79	71.73	80.86	69.60	52.60	44.00	77.18	83.84	83.96	48.17	72.48
	07+12	77.44	80.93	85.58	75.89	68.77	57.77	83.94	86.30	89.41	58.79	80.33
YOLO v3	07	86.83	91.94	91.93	88.08	75.57	78.95	94.19	94.79	91.19	76.15	91.44

table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
65.45	80.78	86.30	75.96	79.80	42.49	64.53	66.06	79.40	70.60
75.39	87.20	87.66	81.88	82.76	48.60	77.30	76.23	86.82	77.23
83.92	91.21	92.38	91.48	91.92	61.78	83.32	88.05	92.29	86.31

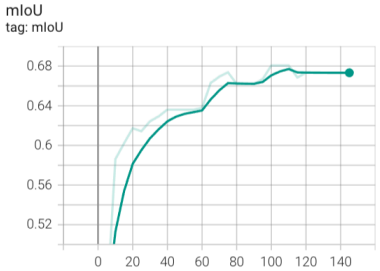
The following plots Figure 17 show the loss curve during training and testing, the mAP curve on the test set, and the mIoU curve. And in Figure 17(a), a dramatic rise in loss is due to the unfreezing process, where the loss in the backbone is also taken into account.



(a) Loss of Faster R-CNN



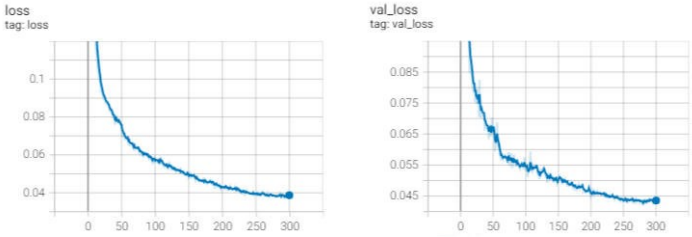
(b) mAP of Faster R-CNN



(c) mIoU of Faster R-CNN

Figure 17: Curves of Training Faster R-CNN using VOC 2007 with TensorBoard

Here comes the plots for YOLO v3. In Figure 18, we present the loss curves during training and validation, both the curves look mild because we are training on the pretrained model, which avoids drastic changes in the loss.



(a) Training Loss of YOLO v3 (b) Validation Loss of YOLO v3

Figure 18: Curves of Training YOLO v3 using VOC 2007 with TensorBoard

Noted that the number of epochs is so large that it would take much longer time if we calculate mAP every 5 epochs, so we just ignore the same work done in Faster R-CNN training process to assure efficiency.

3.6.3 Application on Real-World Scene

To test the performance of our models and make sure that they are well trained without overfitting, we will check on some realistic photos. The main elements of the following samples include target objects in VOC 2007, now let's see how our models perform on these real-world scenes.

Figure 19: Object Detection on Real-World Scene (Original Image, Faster R-CNN, YOLO v3)

Comparing the outcomes above, the bounding boxes and class accuracies are quite close on the first two instances for Faster R-CNN and YOLO v3. However, interesting things happened on the third example are: YOLO v3 returns a better bounding box for the most front navigator containing the whole body of him, while Faster R-CNN just omits his arm. Furthermore, YOLO v3 seems not to recognize the aeroplane in the background while this time Faster R-CNN does. This result is quite surprising, and after some investigation, we propose the assumption that YOLO v3 may not be able

to identify the plane with its wings folded, based on the lack of training data and the high mAP YOLO v3 gets on VOC 2007 up to 91.74%.

4 Conclusion

In this project, we first revisited and trained several CNNs models on the dataset CIFAR-100. And these models reached high accuracy and some of them like DLA are even using just a few memory while training. Finally we get the best result thanks to WideResNet-28.

Then, we revisited and trained Faster R-CNN and YOLO v3 on the dataset VOC 2007, through visualizing the training process on TensorBoard and bounding boxes on image samples, we can understand the procedure and intermediate results of training clearly. We also tested the performance of both models on real-world, it's quite enchanting to see our models apply to reality and truly work.

For further work, in order to get a higher mAP using Faster R-CNN model, we can try to pretrain the model on the COCO dataset, and then train it on the VOC dataset. Also, we can try to change the backbone to some more complex models like ResNet-101 or DLA (which is a promising model for fusing semantic and spatial information for recognition and localization) [11].

References

[1] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[2] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 6023-6032).

[3] DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.

[4] Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412.

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[6] He, K., Zhang, X., Ren, S., & Sun, J. (2016, October). Identity mappings in deep residual networks. In European conference on computer vision (pp. 630-645). Springer, Cham.

[7]Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. arXiv preprint arXiv:1605.07146.

[8] Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1492-1500).

[9] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

[10] Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., & Feng, J. (2017). Dual path networks. Advances in neural information processing systems, 30.

[11] Yu, F., Wang, D., Shelhamer, E., & Darrell, T. (2018). Deep layer aggregation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2403-2412).

[12] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2), 303-338.

[13] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[14] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[15] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28.

[16] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.