Waste Classification & Localization

Based on Convolutional Neural Networks

Zhiyang Liu, Yiwei Wang, Wandong Wu, Zilan Zhang

Abstract

A deep learning approach to classify and localize the waste object could be very efficient and accurate. The objective of this project is to take images of a single piece of waste, classify it into six classes and localize the waste object in each image. Here we dive into different optimization algorithms and apply them to improve our classification accuracy, including data augmentation, optimization method (momentum, Adam, etc.) and hyperparameter adjustment.

1 Introduction

Waste sorting has perplexed people for a long time. Wastes were sorted by hand traditionally. Although the modern category dustbin might be helpful, the confusion on which class the waste belongs to gradually makes it useless. Our motivation is to build a more intelligent, effective and stable system to process the waste. We convert this big goal into three steps, image classification, single object localization and multiple object detection. In this paper, we will mainly talk about the first two steps and put multiple object detection as our future work

Our dataset comes from https://github.com/garythung/trashnet. [1] It has 2527 images, divided to six classes: cardboard, glass, metal, paper, plastic and trash. The quantity ratio of training and validation data is four to one

Figure 1.1 – Waste Dataset (six classes)

As for neural network model, we tried AlexNet, VGG16, 11-layer customized model and ResNet18. Because the ResNet18 has much better performance in our case, our later discussion

is all based on the ResNet18 model. [2]

Generally, our model computes the score for each class, and classify the image to class with highest score. We concentrate on optimization from three aspects: Hyperparameter Adjustment, Optimization Methods and Data Augmentation. After applying these optimization methods, our best accuracy of the validation dataset increases from 67% to 92%. In single object localization part, we fine-tune the network model and attach another fully connected layer on the last layer, which does both classification and bounding box regression at the same time.

2 Related Work

2.1 Gradient Descent

Gradient descent is a general used algorithm for loss reduction. The core of this algorithm is the updating method for backpropagation.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j{}^{(i)}$$

Based on the drawback of Vanilla Gradient Descent, SGD has made some modification on it.

$$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j{}^{(i)}$$

Mini-batch stochastic gradient descent is a compromise between "vanilla" gradient descent(GD) and stochastic gradient descent(SGD):

Repeat {

for i = 1, 1+b, 1+2b, ... {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+b-1} (h_\theta(x^{(k)}) - y^{(k)})\, x_j{}^{(k)}$$

(for every j=0, 1, ...)

}

}

2.2 Optimization Method

The most popular optimization for the weight parameters update step in backward propagation can be generally separated into two parts, one is Stochastic Gradient Descent with Momentum, and the other parts called adaptive optimization method, which contain Adam, RMSprop, AdaGrad, Adadelta and etc. Simply speaking, the momentum can be described that give the point in the process of gradient descent a velocity value (computation function described as below)

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Adam is the most popular and controversial algorithm in this field today. Intuitionally, it is like to combine the momentum term with RMSprop to have a super adaptive algorithm that can

modify its gradient descent step automatically by the combination of all these things during the back-propagation step.

$$first_{moment} = beta_1 * first_{moment} + (1 - beta_1) * \nabla f(x_t)$$

$$second_{moment} = beta_2 * second_{moment} + (1 - beta_2) * \nabla f(x_t) * \nabla f(x_t)$$

$$first_{unbias} = \frac{first_{moment}}{1 - {beta_1}^t} \qquad second_{unbias} = \frac{second_{moment}}{1 - {beta_2}^t}$$

$$x -= \frac{\alpha * first_{unbias}}{\sqrt{second_{unbias}} + eps}$$

3 Image Classification

3.1 Hyperparameter Adjustment

At the beginning, we use settings like Batch size = 4, Learning rate = 0.001 with simple decay method. The running outcome for best accuracy is 67%. The plots of loss and accuracy are shown below.

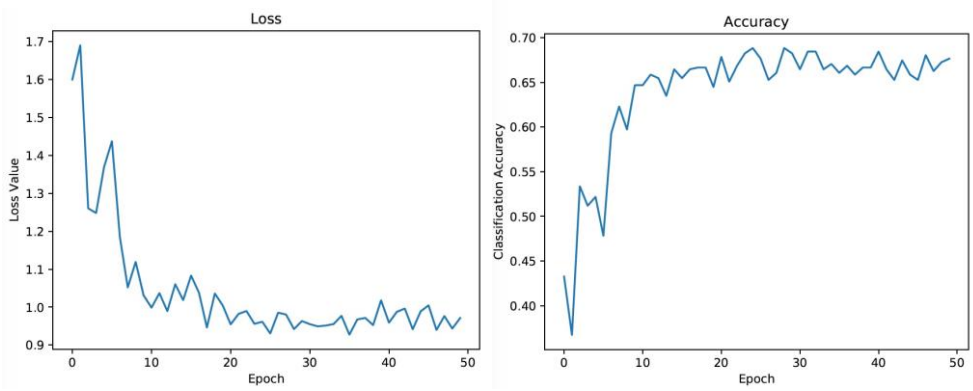

Figure 3.1 – Initial loss and accuracy

3.1.1 Mini-batch Size

To use mini-batch SGD, we have to fiddle with the mini-batch size. Here is a test result of changing the batch size to 16.

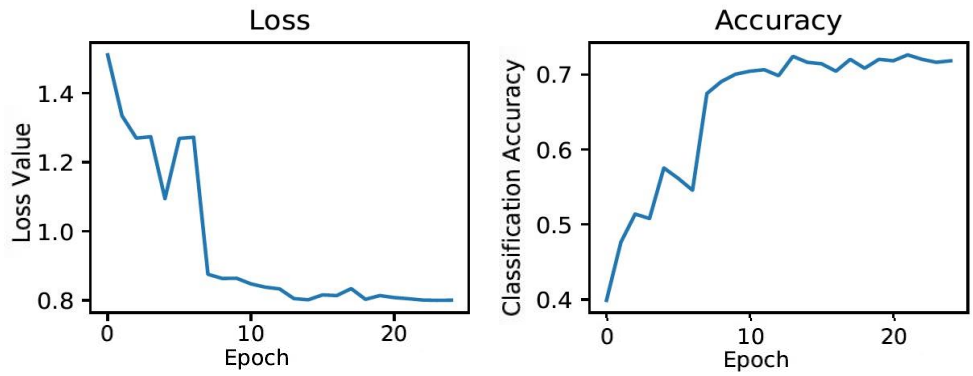

Figure 3.2 - Loss and accuracy after adjusting mini-batch size

It is shown by the plot that accuracy has improved by making this change. That means we can optimize the model by adjusting batch size. One thing that worth addressing is that a good batch size can balance the accuracy of test result and the efficiency of training the model. And with a good vectorized implementation, it could run fast than both GD and SGD.

3.1.2 Learning Rate Adjustment

The learning rate α setting is quite tricky. If the value is too small, it might take too much time to reach the global minimum. Meanwhile, the value of the loss function might vibrate between the concave and could not settle down. Generally, there are two procedures to adjust it. First, the value should be selected from 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, ... After narrowing the value to a relatively small range, we can make fine adjustment in this fixed range to maintain a better performance.

Although a bigger α leads to a faster convergence process, it may also bring problem to the algorithm. Therefore, to avoid vibration at the bottom of the concave, we add a decay method to the learning rate. In our project, we told the learning rate to decay every 7 epochs at the very beginning. Nevertheless, the performance is not satisfactory enough. By observing the loss curve, we limit the decay into a fixed range of epoch ([55, 75]). In this way, we can maintain the best performance we can get so far. More concrete information after hyperparameter adjustment are shown as follows.

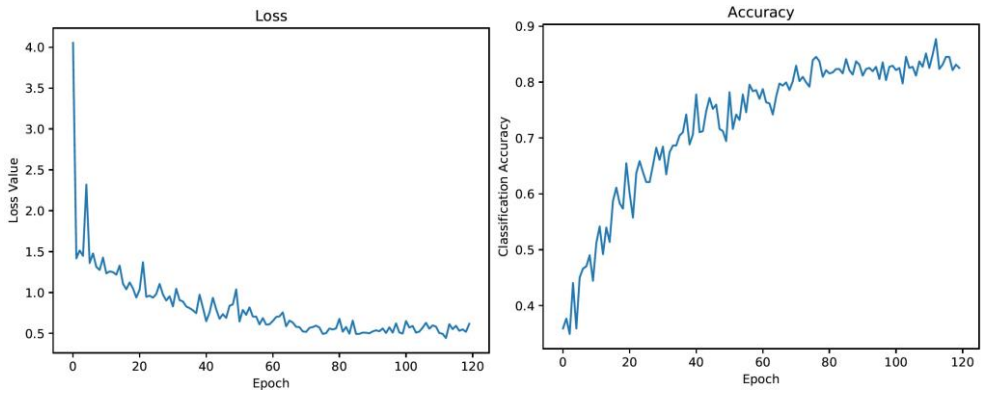

Figure 3.3 - Loss and accuracy after adjusting learning rate

The best accuracy we get after making these optimizations is 88%.

3.2 Optimization Method Selection

3.2.1 Initialization

We do a contrast test to set up same hyperparameters for each algorithm and test each algorithm's performance in this relatively fair condition (Because of the different mathematical model and each optimization has some unique hyper parameters, so the condition for the test can only be relatively fair)

We select following algorithm: Stochastic gradient descent(SGD)+Momentum, Adam, AdaGrad and RMSprop. We set the learning rate as 1e-3, learning rate decay as None, weight decay as None, Momentum as 0.9, eps (a very small value to keep the calculation result not to be NAN) as 1e-8. And Betas for Adam is (0.9, 0.99), Alpha for RMSprop is 0.99.

We use the original data set provided by Stanford University (around 2,000 images), because SGD might have a relatively slow convergence time when we set a big data set that might kill by the system (we used a Pittsburgh GPU cluster system which only allow us to use 1 hour for each time request). With the test parameter setting above we check after 80 epochs and 64images per batch the best accuracy and each epoch's performance over these 4 algorithms.

3.2.2 Result Analysis

The adaptive algorithm performed well in the initial progress, the training loss went down, the validation accuracy went high rapidly in the first several epochs, but their performance plateaus quickly and stuck in a point, like showing in the after record, in this test set Adam could reach the 82% of validation accuracy at last but reach 70% at 14th epochs, this means that Adam fluctuated between 70% and 80% for over 66 epochs and still didn't reach the peak at 88% that SGD reached at last. As to SGD, SGD with momentum made the loss value go down smoothly and reach the peak of accuracy without huge fluctuate. And some time adaptive methods and SGD with momentum get the similar low training loss, but SGD always have a higher or equal accuracy on validation set.

Besides, the most powerful adaptive optimization method, Adam, sometime could reach the same best performance as SGD, but sometimes it couldn't. As to other adaptive algorithm, RMSprop and AdaGrad, which are also popular algorithm these days, they always get a much worse performance than SGD and Adam (shown in the following record of data

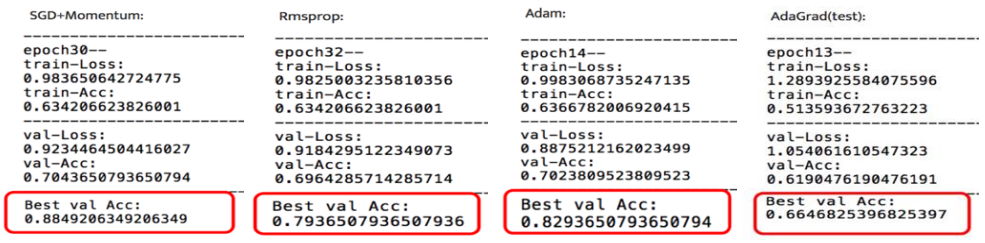

Figure 3.4 – Validation accuracy using SGD-Momentum, RMSprop, Adam and AdaGrad methods

The paper [3] test more non-adaptive and adaptive optimization algorithm on several tasks, which trains on the CIFAR-10 dataset and has the similar result with ours. It shows the adaptive algorithms seemingly powerful might have worse performance in practice.

From our perspective, the reason why adaptive might not reach the global minima and fluctuate a lot, like Zig Zag after first several epochs can be explain as, the square of gradient as well as the decay of square of gradient computation for each step and contributes to later steps computation do make Adam have the adaptive characteristic and perform well in theory, but these intermediate parameters make Adam may not converge in some case for high dimensional calculation. [4]

3.2.3 Best Solution

The adaptive algorithm has a short converge time but have no guarantee to reach the global minima. And the purely SGD algorithm is too slow for us to deal with over 10,000 images in total, 64 images per batch. So, we choose SGD + Momentum at last.

3.3 Data Augmentation

We have got the best validation accuracy 88% from our previous work. However, this project heavily depends on the quantity and quality of dataset. Considering 2500 images are not enough to train our model, we need to modify our dataset.

Firstly, we add some similar images from ImageNet, enlarging our dataset to 9000 images. and then split them to training and validation dataset with the ratio of 4:1. However, the best accuracy drops to 85% with pre-trained model (figure 3.5), and 75% with untrained model. This might because the additional images are not as clean as images in our original dataset.

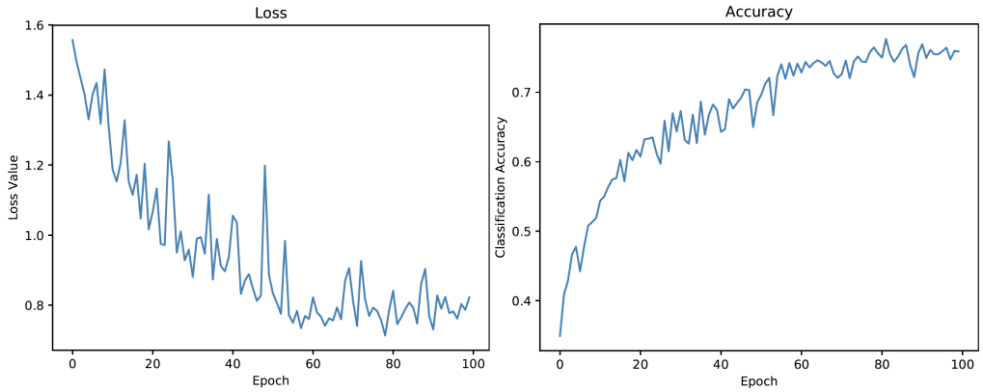

Figure 3.5 - Loss and accuracy when dataset size = 9000

Secondly, we adopt data augmentation. There are many ways to augment data, including rotation, lighting conditions adjustment, crop, ands etc. So one image can generate to multiple samples. This method can help avoid overfitting.

We use Augmentor library to augment our dataset. Specifically, we skew, rotate, flip, shear, and distort 2500 images, and every step will be run with a probability of 0.5. Eventually, we obtain a dataset with19000 images. The ratio of training to validation dataset is still 4:1. As

shown in figure3.6, the final best accuracy reaches 92%.

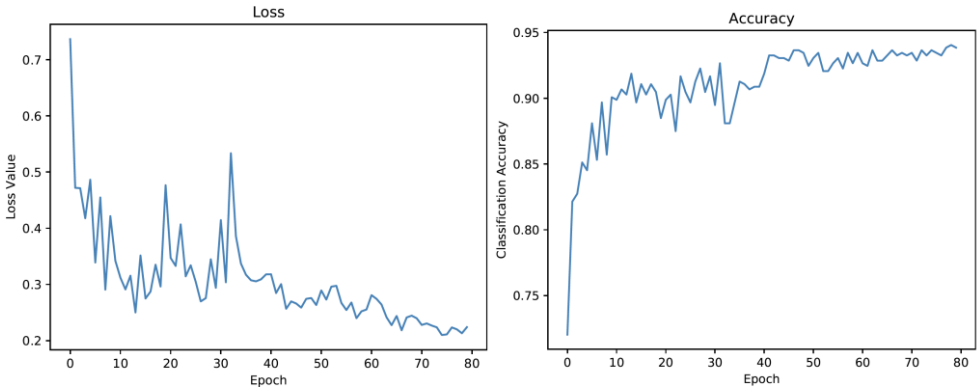

Figure 3.6 - Loss and accuracy when dataset size = 19000

3.4 Evaluation

Initially, we save the model according to the best accuracy. However, it is not always the best evaluation criteria. In our scenario, trash only accounts for about 6% in our dataset. So, we incorporate F1 measure to verify our result.

The traditional F1 measure is calculated by $F1 = \frac{2P*R}{P+R}$, where P is precision and R is recall.

The F1 score shows the harmonic average of the Precision and Recall, which stays between 0 to 1 (perfect precision and recall).

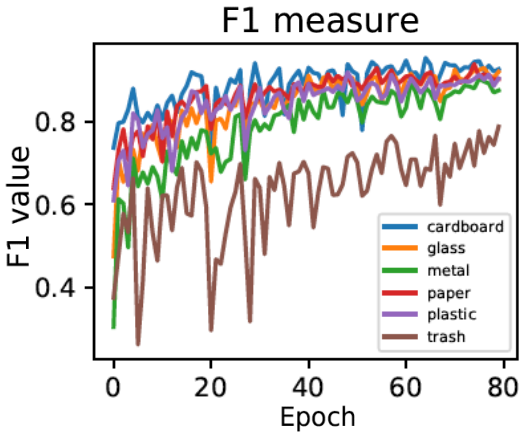

Figure 3.7 – F1 measure

Figure3.7 reveals that our dataset is not even distribution. And this might be a reason why our accuracy cannot reach over 95%.

4 Object Localization

In our case, we only achieve the object localization for single object in image. The whole process is similar with the image classification. The difference is that besides the class scores, this time we also need to output the estimated bounding box coordinates (with x_{min}, y_{min}, x_{max}, y_{max}) to localize the object in image. [5]

4.1 Neural Network Model

Here is our neural network model. The convolutional and pooling layers are same as the classification's. I will not talk about it. What we have done here is attaching another fully connected layer on the last layer. So, we have two outputs this time. One is a $\text{num\_class} * 1$ vector for classification and the other is a $4 * 1$ vector for bounding box regression. We weight and sum each loss function to get a total loss, then use the total loss to do the back propagation

to renew the weight parameters in each layer.

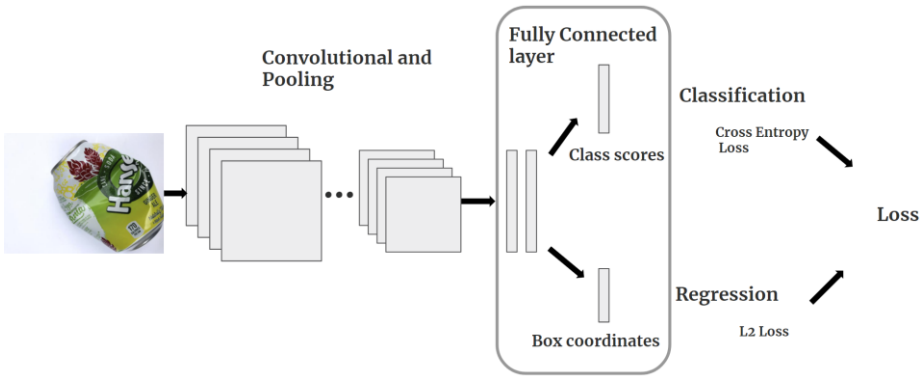

Figure 4.1 – Training process for single object localization

4.2 Loss Function

For the classification part, we still use the cross entropy loss function. Here we talk about the loss function for bounding box regression. There are two good methods to calculate the loss, intersection of union(IoU) and L2 loss. IoU is to calculate the ratio of overlapped size and union size of our output and ground truth bounding box. The equation is shown as follows.

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

The L2 loss calculates the sum of all the squared differences between output and ground truth.

$$L2\ loss = ||Output - Ground\ truth||^2$$

Here we choose the L2 loss for our loss function, because in initial iterations, the output sometimes has negative value, which makes it difficult to calculate IoU. L2 loss behaves better in this situation. Finally, we weight the cross entropy loss and L2 loss and do back propagation.

4.3 Result Analysis

The bad result is shown as follows. You can see it is totally different from the ground truth (the values in each row represent x_{min}, y_{min}, x_{max}, y_{max}). It may because when we do the data augmentation for the dataset (skew, flip. Rotate etc.), the corresponding process for the ground truth of bounding box coordinates doesn't match the process for some of the image data.

```
bbox_output_large                                    gt_bbox_large
 [[167.42932   61.420677 119.2023   103.117775]    [[ 71.   2. 192. 223.]
 [181.07994  136.2287   120.31758  270.26758 ]     [ 47.  47. 330. 498.]
 [310.76828  163.90756  160.0672   269.42636 ]     [ 83. 105. 380. 391.]
 [310.11447   87.92563  189.64609  141.42914 ]]    [106.  51. 382. 332.]]
```

Figure 4.2 – Bad results for bounding box regression

The good result looks decent. It is not exactly same as the ground truth, but is enough to help the robot arm in the future to localize the waste.

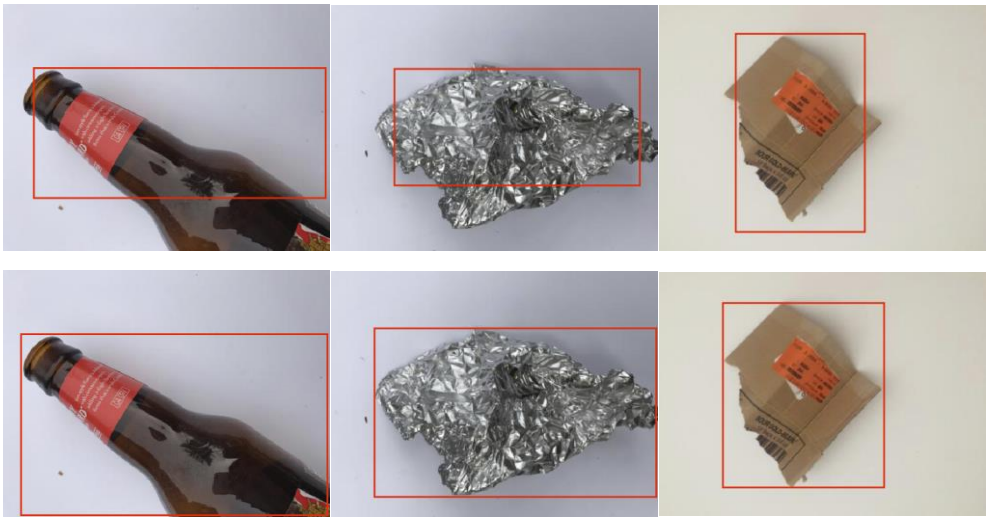

Figure 4.3 – Good results in object localization

However, the total result accuracy is very messy, and we will not talk about it here. We are still trying to fix the problem (because of data augmentation or other bugs)

5 Conclusion

Our result shows the importance of hyperparameter adjustment, optimization method and data augmentation in a convolutional neural network training process for image classification. We should apply suitable hyperparameters and methods to get a high accuracy. Also, the dataset's size is a key factor for the training result, especially in a more complicated computer vision task, like object detection.

6 Future Work

We have done image classification and single object localization so far. To maintain our aim, our next step is multiple object detection. This will equip our system to recognize multiple instances in one or more classes. More information can be referred from *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [6] and *Mask R-CNN*. [7]

Meanwhile, object detection task requires larger dataset and more ground truth for training that current dataset cannot provide. We want to set up a Data Acquisition program to receive data from everybody. We'd like to develop a simple web App and launch an environment protection topic on Twitter so that people can help protecting environment by labeling and uploading the waste photo with their phone. And our dataset comes from these Twitters.

Reference

[1] Yang, Mindy, and Gary Thung. "Classification of Trash for Recyclability Status."

[2] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[3] Ashia C. Wilson, R. Roelofs, M. Stern, N. Srebroy, and B. Recht. *The Marginal Value of Adaptive Gradient Methods in Machine Learning*. In University of California, Berkeley and Toyota Technological Institute at Chicago, May 24, 2017

[4] Sashank J. Reddi, Satyen Kale, Sanjiv Kumar. *On the Convergence of Adam and Beyond.* Google New York, Published as a conference paper at ICLR 2018

[5] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.

[6] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *Advances in neural information processing systems*. 2015.

[7] He, Kaiming, et al. "Mask r-cnn." *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017.

Contribution

All: GPU cluster environment configuration, coding debug, result analysis.

Zhiyang Liu was mainly responsible for the object localization part.

Yiwei Wang compared different CNN models and several optimization methods, and led the hyperparameter adjustment work.

Wandong Wu took charge of data collection, data preprocessing, and data augmentation.

Zilan Zhang implemented evaluation, result recording, and visualization function. Also, she assisted in hyperparameter adjustment work.