pytorch builds a garbage identification model

1. Introduction

1.1 Problem Description

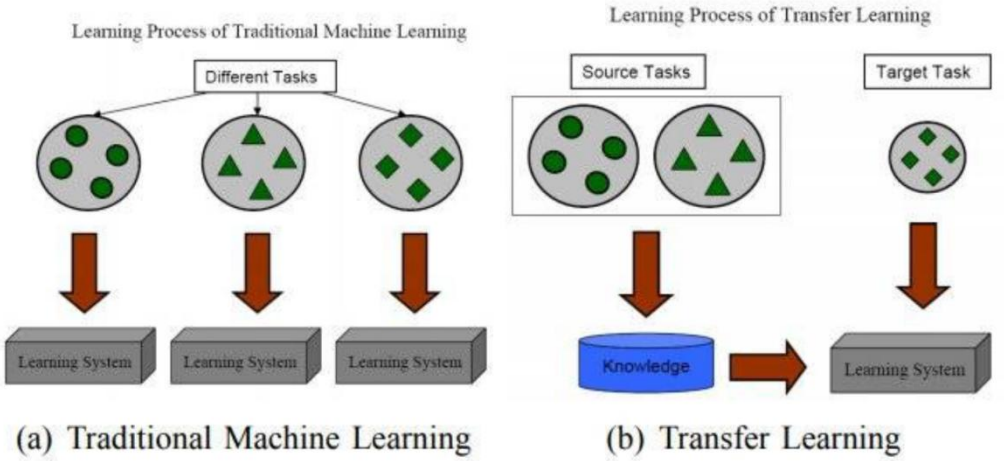
The classification of garbage is first based on image recognition, so this paper wants to use the deep learning method that has developed rapidly in recent years, a deep learning landmark algorithm Resnet , and use it to derive ResNet-18 to design a garbage classification system,which classification under 55 categories, the data includes recyclable garbage (18), kitchen waste (19), harmful garbage (9), and other garbage (9), so as to realize the intelligent identification and classification of common garbage in daily life. Improve people's awareness of garbage classification and release, and avoid environmental pollution caused by people's wrong release.

1.2 Environment

Python 3 ,pytorch ,opencv ,Google Colab ,Google drive and rent “矩池云” GPU.

1.3 Transfer Learning(迁移学习)

Transfer Learning is a subresearch field of deep learning. The goal of transfer learning is to use the similarity between data, tasks, or models to transfer the knowledge learned in the old domain to the new domain. This paper uses the transfer learning of Resnet-18 network and uses the weight parameters of its first 17 layers. The 18th layer changes the original more than 1000 outputs to 55 types of outputs in this paper.



(a) Traditional Machine Learning (b) Transfer Learning

[Source： Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345–1359.]

1.4 Resnet（残差网络）

The Development of Deep Learning From LeNet to AlexNet, to VGGNet and GoogLeNet, the depth of the network is constantly deepening. Experience has shown that the depth of the network has a crucial impact. But when the network is deep enough, just stacking more layers on top of it creates a lot of problems: the first problem is vanishing/exploding gradients, which can be solved by BNS and better network initialization; The second problem is degradation, that is, when the network is saturated with layers, adding more layers leads to optimization difficulties and higher training and prediction error, note that the higher error is not due to overfitting. This paper directly uses the pre-trained residual network and the residual network without training to make a comparison.

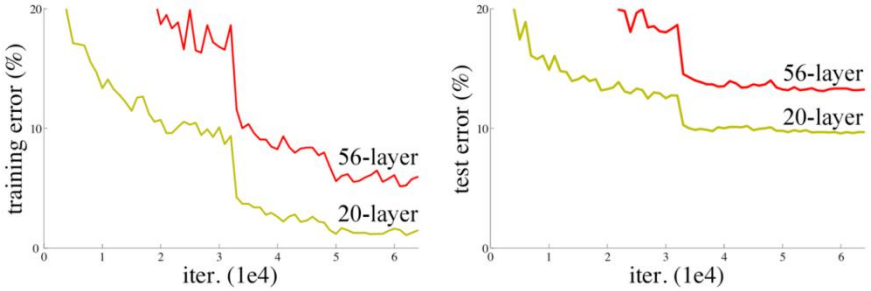


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Theory:

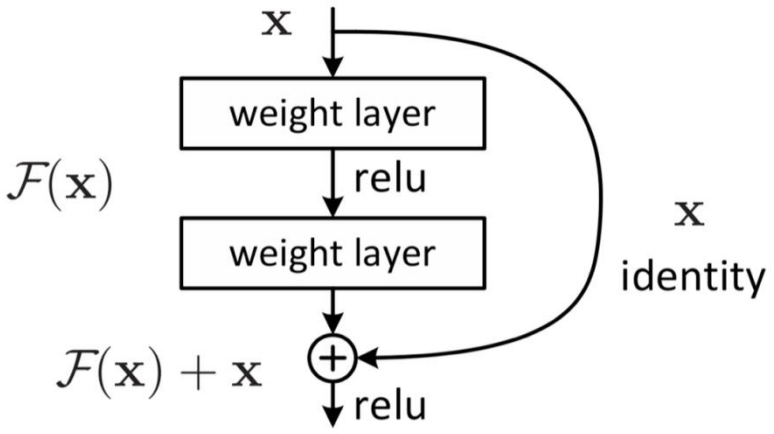


Figure 2. Residual learning: a building block.

1.5 SGD and Adam(两个优化器之间的对比)

Deep learning optimization algorithms have gone through the development process of SGD -> SGDM -> NAG ->AdaGrad -> AdaDelta -> Adam -> Nadam. SGD can reach the global optimal solution, and the best training accuracy is higher than other optimization algorithms, but it has strict requirements on the adjustment of learning rate, and it is easy to stop at the saddle point. Adam is a optimization algorithm.The memory requirement is small, and different adaptive learning rates are calculated for different parameters， it is easy to skip the saddle point, and does not need human intervention to adjust the learning rate, but it is easy to oscillate at the local minimum, there is a sudden increase in the learning rate under special data sets, resulting in non-convergence, that is to say it has all the advantages and all the shortcomings of other optimization algorithms. In this paper, stochastic gradient descent is used with models without pre-training. The pre-trained model employs Adam to achieve the highest performance.

2. Dataset Construction – My Own Dataset and Public Dataset

2.1 Data Description

First observe the data characteristics I need to complete the task, and first do simple experiments on image crawling to see the effect of data collection. Good data samples also play a vital role in the training of the model. Here, 50 keywords "French fries" and 50 keywords "mouse" are crawled. For the keyword "French fries", many irrelevant images will be crawled to increase the difficulty and accuracy of training, because the target object cannot be accurately located. For the "mouse" keyword, it looks good from the crawled images. Therefore, the data set is obtained in two ways. For kitchen waste, I choose to use images from the kaggle open source data set, and another I will crawl the rest by myself.
Crawl site at:
https://image.baidu.com/search/down?tn=download&ipn=dwnl&word=download&ie=utf8&fr=result&url=%s&thumburl=%s
Kaggle site at:
https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification

Figure 1:the bad crawl data,the keyword is “薯条”

2.2 My dataset

Goal: For the remaining 36 categories except kitchen waste, 200 to 600 images were crawled respectively.

Main Tip:

```
:param word:  抓取的关键词
:param total_page:  需要抓取数据页数  总抓取图片数量为  页数  x per_page
:param start_page:起始页码
:param per_page:  每页数量
```

Result:

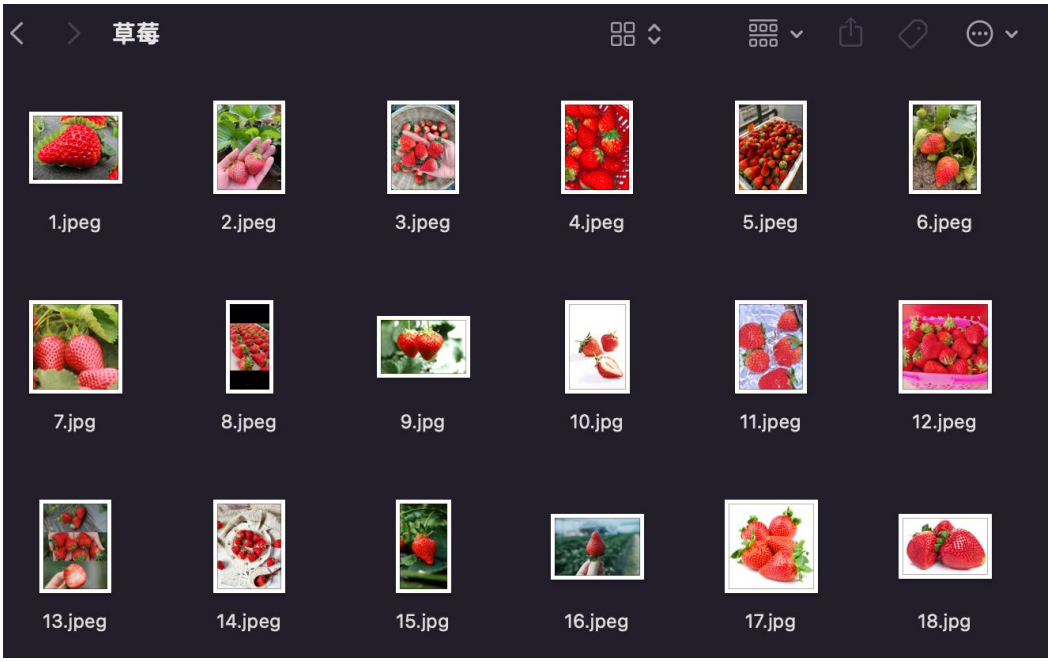


Figure 2:Show the effect,the example of keyword “草莓”

Result dataset:

3. Data processing

3.1Raw Data Preprocessing

Refer to the crawled image size ratio, use the scatter plot in plt to build pixel information into an empty list, and output the image's length and width.

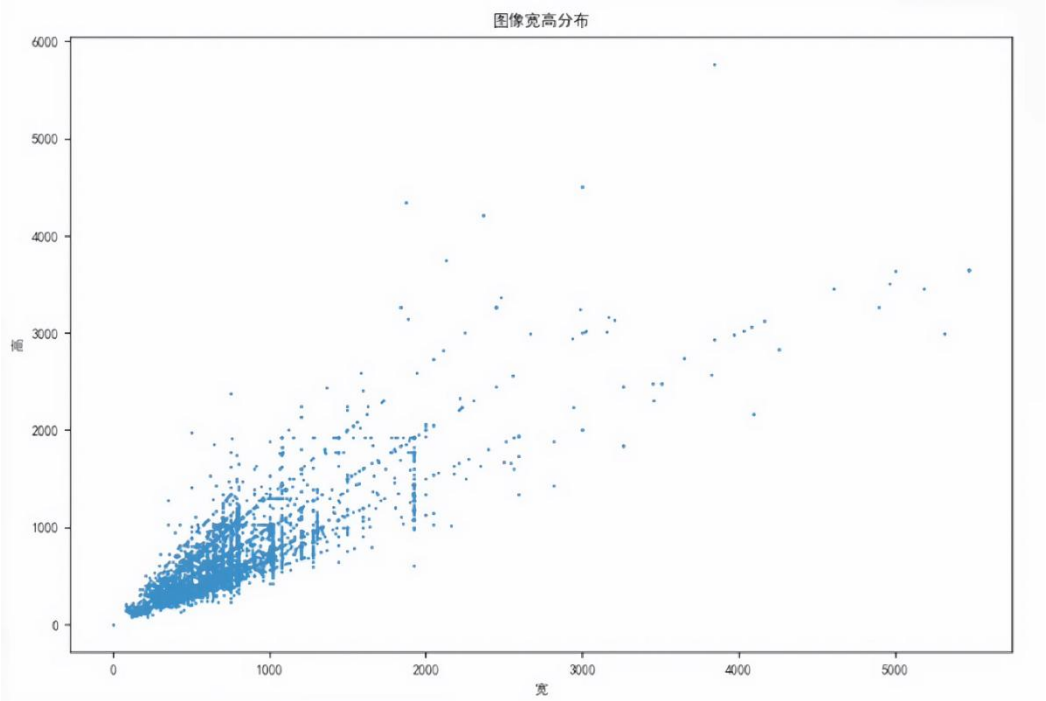


Figure 3:The scatter of image

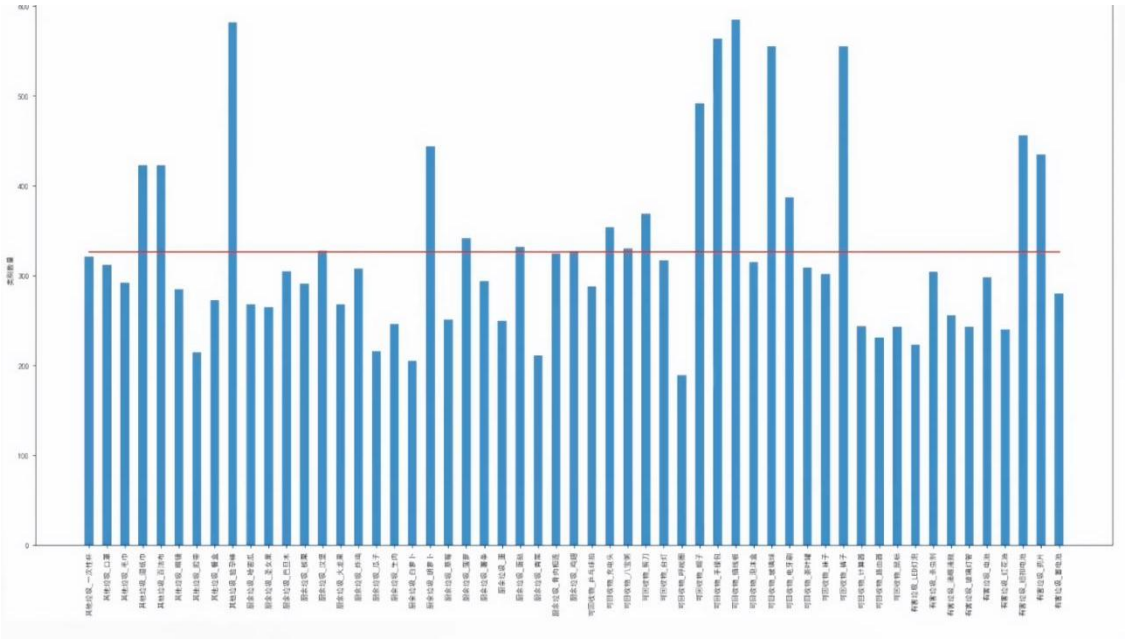


Figure 4:The distribution of image number

From figure 3, it can be seen that the image resolutions vary, so we need to perform appropriate cropping. From figure 4, it is evident that the distribution of the number of samples is uneven. This is because the parameters for image size were not set when crawling the data. Therefore, in the preprocessing step, images with one side too short or too long, and those with inappropriate width, height, and aspect ratios should be deleted.

Data Augmentation Processing:

```
# 水平翻转
```

```
def Horizontal(image):
    return cv2.flip(image,1,dst=None) #水平镜像
# 垂直翻转
def Vertical(image):
    return cv2.flip(image,0,dst=None) #垂直镜像
```

Data Balancing Processing:

```
img_root    = r"enhance_dataset"
threshold = 300

for a,b,c in os.walk(img_root):
    if len(c) > threshold:
        delete_list = []
        for file_i in c:
            file_i_full_path = os.path.join(a,file_i)
            delete_list.append(file_i_full_path)

        random.shuffle(delete_list)

        print(delete_list)
        delete_list = delete_list[threshold:]
        for file_delete_i in delete_list:
            os.remove(file_delete_i)
            print("将会删除",file_delete_i)
```

Example the enhance_dataset:

3.2Calculate the Mean and Variance of the Image

Calculating the mean and variance of the image is for normalization purposes.

```
def getStat(train_data):
    train_loader = torch.utils.data.DataLoader(
        train_data, batch_size=1, shuffle=False, num_workers=0,
pin_memory=True)
    mean = torch.zeros(3)
    std = torch.zeros(3)
```

```
    for X, _ in tqdm(train_loader):
        for d in range(3):
            mean[d] += X[:, d, :, :].mean() # N, C, H ,W
            std[d] += X[:, d, :, :].std()
    mean.div_(len(train_data))
    std.div_(len(train_data))
    return list(mean.numpy()), list(std.numpy())
```

```
100%|██████████| 15442/15442 [03:12<00:00, 80.12it/s]
([0.6391828, 0.5712495, 0.5060598], [0.21158008, 0.2197067, 0.22926107])
```

3.3 Splitting the Dataset

Train set 0.9, test set 0.1, then perform shuffle operation to randomize.

```
train_ratio = 0.9
test_ratio = 1-train_ratio

rootdata = r"enhance_dataset"

train_list, test_list = [],[]

class_flag = -1
for a,b,c in os.walk(rootdata):
    for i in range(0, int(len(c)*train_ratio)):
        train_data = os.path.join(a, c[i])+'\t'+str(class_flag)+'\n'
        train_list.append(train_data)

    for i in range(int(len(c) * train_ratio), len(c)):
        test_data = os.path.join(a, c[i]) + '\t' + str(class_flag)+'\n'
        test_list.append(test_data)

    class_flag += 1

random.shuffle(train_list)
random.shuffle(test_list)
```

3.4 Generate a Data Loader (4-dimensional)

N,C,H,W(Batch size, channel, weight, high)

```
rain_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                          batch_size=10,
                                          shuffle=True)
```

4. Model Training

4.1 ResNet-18 based CNN model

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 [3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

Tip: In this model,18 here specifies 18 layers with weights, including the convolutional layer and the fully connected layer, excluding the pooling layer and the BN layer.

Main code:

```
def train(dataloader, model, loss_fn, optimizer,device):
    size = len(dataloader.dataset)
    avg_loss = 0
    # 从数据加载器中读取 batch（一次读取多少张，即批次数），X(图片数据)，y（图片真实标签）。
    for batch, (X, y) in enumerate(dataloader):#固定格式：batch：第几批数据，不是批次大小，（X，y）：数值用括号
        # 将数据存到显卡
        X, y = X.to(device), y.to(device)
        # 得到预测的结果 pred
        pred = model(X)
        loss = loss_fn(pred, y)
        avg_loss += loss
        # 反向传播，更新模型参数
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # 每训练 10 次，输出一次当前信息
        if batch % 10 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

    # 当一个 epoch 完了后返回平均 loss
    avg_loss /= size
    avg_loss = avg_loss.detach().cpu().numpy()
```

```
    return avg_loss
```

Result:

```
Accuracy:62.8%, Avg loss: 0.023143
```

Results Analysis:

Accuracy: The model achieved an accuracy of 62.8% on the test set, indicating its ability to correctly classify about two-thirds of the samples.

Average Loss: The average loss of 0.023143 reflects the model's average uncertainty or error rate in classification. A lower loss value typically indicates better performance.

4.2 Pre-trained ResNet-18 CNN model

Suggestions for Improvement:

Based on the aforementioned model, I plan to make improvements by employing transfer learning. In the previous model, I only utilized the architecture of the Residual Network without adjusting its parameters or discussing its depth in detail. Now, I have optimized the algorithm. For instance, the first algorithm employs a ResNet-18 model trained from scratch (without pretrained weights, and it has a longer runtime).The second algorithm, however, uses a pretrained ResNet-18 model and replaces the final fully connected layer to adapt to the new classification task (55 categories). This approach is commonly referred to as transfer learning, where utilizing pretrained model weights on extensive datasets can speed up the training process and enhance performance on smaller datasets. In the second optimization and improvement plan, I focused on optimizing the learning rate and the optimizer. The first algorithm employs Stochastic Gradient Descent (SGD) as the optimizer with a fixed learning rate of 1e-3. The second algorithm uses the Adam optimizer, with an adaptive learning rate, setting different learning rates for various parameters. A higher learning rate (`learning_rate * 10`) is used for the newly added fully connected layer (fc layer), while a lower learning rate (`learning_rate`) is employed for the other layers of the pretrained model. The goal here is to quickly learn the weights of the new layer while gradually adjusting the weights of the pretrained layers. As garbage classification involves life pictures that initially don't differ much but increasingly do so later, the learning rate for the last layer is set higher.For the third algorithmic improvement, I plan to initialize the weights of the fully connected layer. The first algorithm didn't specify a method for weight initialization, thus defaulting to PyTorch's standard initialization. In the second algorithm, the newly replaced fully connected layer uses the Xavier initialization method (also known as Glorot initialization). This helps in achieving a reasonable distribution of weights at the start of training, preventing ineffective training due to vanishing gradients, and aiding the network in learning better. Finally, regarding batch size adjustments, since there are many successful cases and recommended sizes for classification tasks available online, I opted for a batch size of 128 for the first algorithm. For the second algorithm, the batch size is 32, due to increased memory usage from the pretrained model and the intention to use smaller update steps for training. These are the improvements I have made to the model.

Main code:

```
def validate(dataloader, model, loss_fn, device):
    size = len(dataloader.dataset)
```

```
    # 将模型转为验证模式
    model.eval()
    # 初始化 test_loss 和 correct,  用来统计每次的误差
    test_loss, correct = 0, 0
    # 测试时模型参数不用更新，所以 no_gard()
    # 非训练,  推理期用到
    with torch.no_grad():
        # 加载数据加载器，得到里面的 X（图片数据）和 y(真实标签）

        for X, y in dataloader:
            # 将数据转到 GPU
            X, y = X.to(device), y.to(device)
            # 将图片传入到模型当中就，得到预测的值 pred
            pred = model(X)
            # 计算预测值 pred 和真实值 y 的差距
            test_loss += loss_fn(pred, y).item()
            # 统计预测正确的个数(针对分类)
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= size
    correct /= size
    print(f"correct = {correct}, Test Error: \n Accuracy: {(100 * correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
    return correct, test_loss
```

Result:

```
Accuracy:92.1%, Avg loss: 0.000318
```

Results Analysis:

In this project, we employed a pretrained and improved Convolutional Neural Network (CNN) based on ResNet-18 to tackle a classification problem involving 55 categories. The model demonstrated excellent performance, achieving an accuracy of 92.1% and an extremely low average loss of 0.000318.

4.3 loss and accuracy comparison

Main code:

```
def DrawLoss(train_loss_list,train_loss_list_2):
    plt.style.use('dark_background')
    plt.title("Loss")
    plt.xlabel("epoch")
    plt.ylabel("loss")
    train_loss_list = train_loss_list[:10]
    epoch_list = [i for i in range(len(train_loss_list))]
    p1, = plt.plot(epoch_list, train_loss_list, linewidth=3)
    p2, = plt.plot(epoch_list, train_loss_list_2, linewidth=3)
```

```
        plt.legend([p1, p2], ["with pretrain", "no pretrain"])
        plt.show()
def DrawAcc(train_loss_list,train_loss_list_2):
        plt.style.use('dark_background')
        plt.title("Accuracy")
        plt.xlabel("epoch")
        plt.ylabel("accuracy")
        train_loss_list = train_loss_list[:10]
        epoch_list = [i for i in range(len(train_loss_list))]
        p1, = plt.plot(epoch_list, train_loss_list, linewidth=3)
        p2, = plt.plot(epoch_list, train_loss_list_2, linewidth=3)
        plt.legend([p1, p2], ["with pretrain", "no pretrain"])
        plt.show()
```

Accuracy:

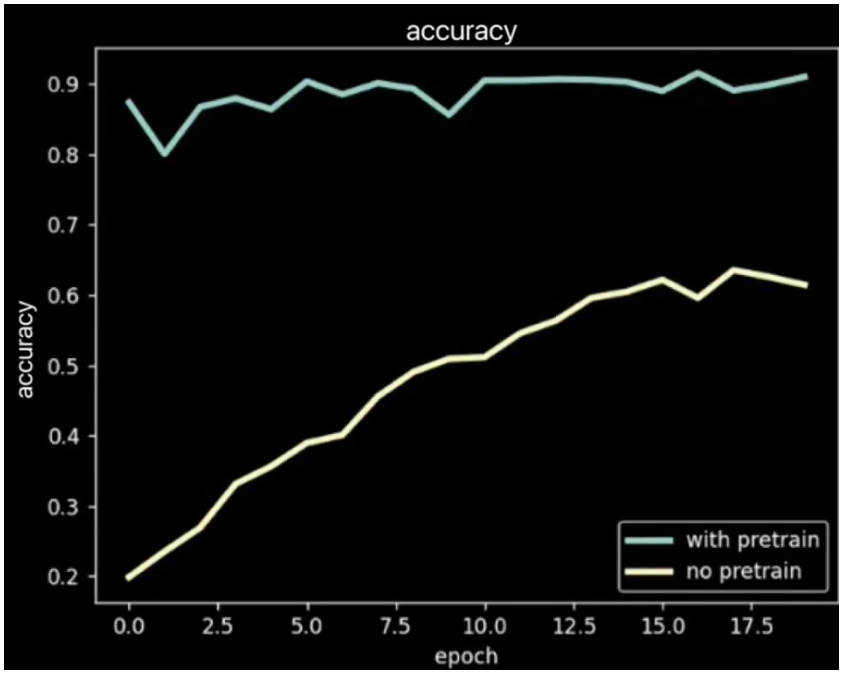


Figure 5:Compare the accuracy between with pretrain and without pretain andAs the number of epochs increases, the model's accuracy changes.

Loss:

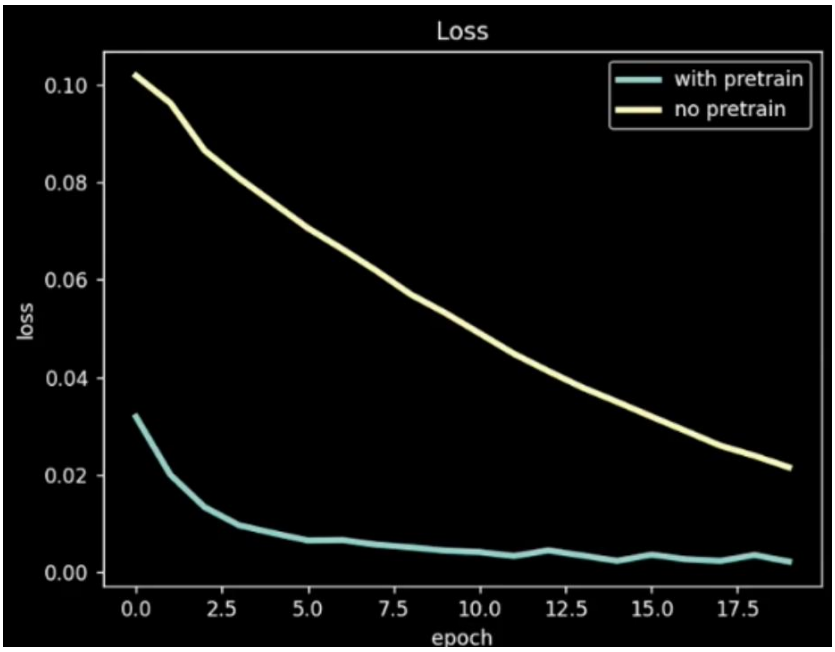


Figure 6:Compare the loss between with pretrain and without pretain and As the number of epochs increases, the model's loss changes.

Conclusion:From my perspective,In comparing a Convolutional Neural Network (CNN) based on ResNet-18 with and without pretraining, distinct differences in loss and accuracy are observed. Initially, the model without pretraining starts with a high loss, which gradually decreases and stabilizes, but its final performance still falls short of the pretrained counterpart. Regarding accuracy, even after several epochs of training, the non-pretrained model fails to reach the high accuracy level achieved by the second model.Training with a pretrained model typically begins with a lower initial loss, as the model has already learned useful features from its prior training.In transfer learning, the loss reduction curve is generally smoother and converges more quickly. In contrast, the loss reduction in the model without pretraining tends to be more erratic, especially in the early stages of training.Pretrained models are usually trained on large and diverse datasets, equipping them with feature extractors that capture more general features. These features provide useful information across different tasks, often leading to higher accuracy in the final task.As the number of epochs increases, both models show significant improvements. However, the pretrained model demonstrates better precision from the beginning of the training process.Overall, the pretrained ResNet-18 CNN model exhibits superior initial performance, smoother loss reduction, and enhanced feature extraction capabilities, resulting in higher accuracy compared to its non-pretrained counterpart.

5.Model Verification

5.1 Test the model with a single image

Input:The address of the image is enhance_dataset (my dataset after image enhancement)."

Output:The category of the image (in the same text format as the naming convention of the dataset).

Code:

```
img_path = r'enhance_dataset/可回收物_鼠标/img_鼠标_1_original.jpg'
```

```
    val_tf = transforms.Compose([    ##简单把图片压缩了变成 Tensor 模式
        transforms.Resize(512),
        transforms.ToTensor(),
        transform_BZ    # 标准化操作
    ])
```

Example:

Input:

```
img_path = r'daraset/可回收物_鼠标/img_鼠标_1_original.jpg'
```

Output:

```
预测结果为：可回收物_鼠标
```

5.2 Use a confusion matrix to evaluate the model

Main code:

```
label_names = []
data_root = r"./enhance_dataset"
for a,b,c in os.walk(data_root):
    if len(b) != 0:
        print(b)
        label_names = b

confusion = confusion_matrix(true_label, predict_label, labels=[i for i in range(len(label_names))])


plt.matshow(confusion, cmap=plt.cm.Oranges)     # Greens, Blues, Oranges, Reds

plt.rcParams["font.sans-serif"] = ["SimHei"]    # 设置中文字体
plt.rcParams["font.size"] = 8
plt.rcParams["axes.unicode_minus"] = False    # 该语句解决图像中的“-”负号的乱码问题

plt.colorbar()
for i in range(len(confusion)):
    for j in range(len(confusion)):
        plt.annotate(confusion[j,i],xy=(i,j),horizontalalignment='center',
verticalalignment='center')
```

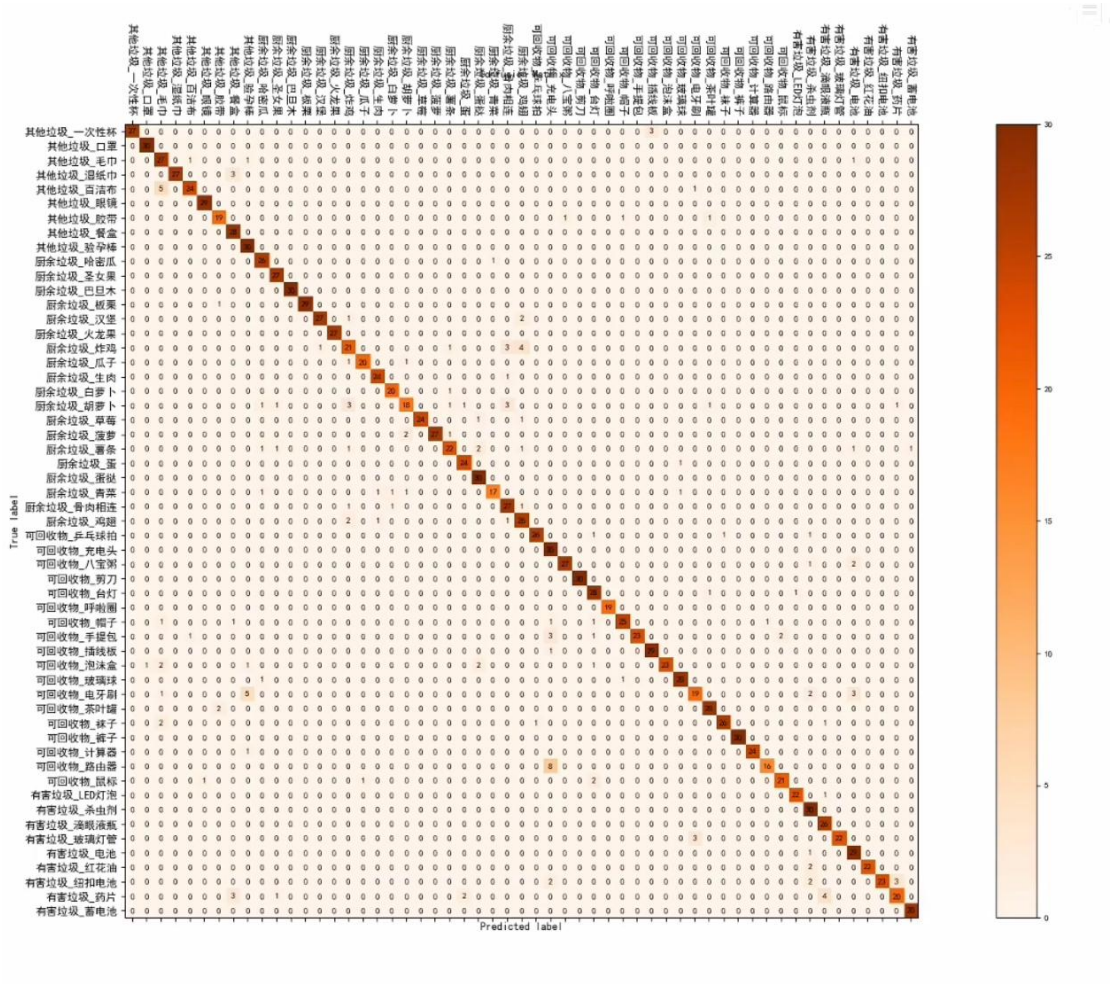


Figure 7:confusion matrix

From the chart, we can see that the model has demonstrated good performance on the garbage classification test set, with only a few misclassifications, which are confined to certain types of garbage. Moreover, the misclassified items often share common characteristics.

6.1 Case analysis

Example:

Figure:“百洁布”

Figure:“毛巾”

Analyse:From the chart, it is observed that there are several groups of misclassifications, specifically between towels and scouring pads, routers and chargers, and fried chicken and chicken wings. There are several common reasons for these misclassifications in the confusion matrix. One is the issue of data balance. To address this, I reviewed the enhance_data and found that the number of images for each category is roughly equal, around 200 each. Another potential reason is noisy data in the dataset, which indeed could contribute to misclassifications. Furthermore, upon observation, it's evident that the misclassified items share similar features, such as color and texture, which likely leads to the primary cause of the model's misclassification.

6.Conclusion:

Overall, the pretrained ResNet-18 CNN model exhibits superior initial performance, smoother loss reduction, and enhanced feature extraction capabilities, resulting in higher accuracy compared to its non-pretrained counterpart.

Reference:

[1] CS231n Convolutional Neural Networks for Visual Recognition

[2] Deep Residual Learning for Image Recognition