Report_HW1

110550128 蔡耀霆

Github: https://github.com/WhiteOuO/VRDL.git

Introduction:

In this experiment, I mainly used the ResNet50 model and the pretrained weights ResNet50_Weights.IMAGENET1K_V2.

The top-1 accuracy of IMAGENET1K_V2 is 4% higher than that of V1.

ResNet50_Weights.IMAGENET1K_V1:

These weights reproduce closely the results of the paper using a simple training recipe.

acc@1 (on ImageNet-1K)	76.13
acc@5 (on ImageNet-1K)	92.862
min_size	height=1, width=1
categories	tench, goldfish, great white shark, ... (997 omitted)
num_params	25557032
recipe	link
GFLOPS	4.09
File size	97.8 MB

ResNet50_Weights.IMAGENET1K_V2:

These weights improve upon the results of the original paper by using TorchVision's new training recipe. Also available as `ResNet50_Weights.DEFAULT`.

acc@1 (on ImageNet-1K)	80.858
acc@5 (on ImageNet-1K)	95.434
min_size	height=1, width=1
categories	tench, goldfish, great white shark, ... (997 omitted)
num_params	25557032
recipe	link
GFLOPS	4.09
File size	97.8 MB

I have chosen to use pretrained weights for image recognition, which will give me a good starting point, and then let the model gradually adapt to my dataset and learn. I plan to implement some data augmentation so that each image in my training set can be transformed through different image processing techniques to provide more comprehensive features for that class, such as horizontal flipping, slight changes in brightness and hue, and cropping the image.

In addition to image processing techniques, I also plan to use methods to exclude outliers from the training data. I believe this will help the model better learn the common features of the class from the normal data.
Furthermore, I will also apply a learning rate adjustment strategy to help my

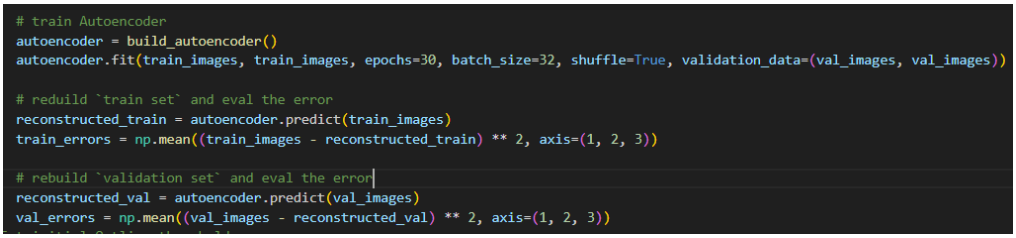
model progress more effectively.

Method:

1. **Outlier Exclusion:**

   I used a model called Autoencoder, which is an unsupervised learning method commonly used for data compression or denoising. The model consists of two parts:

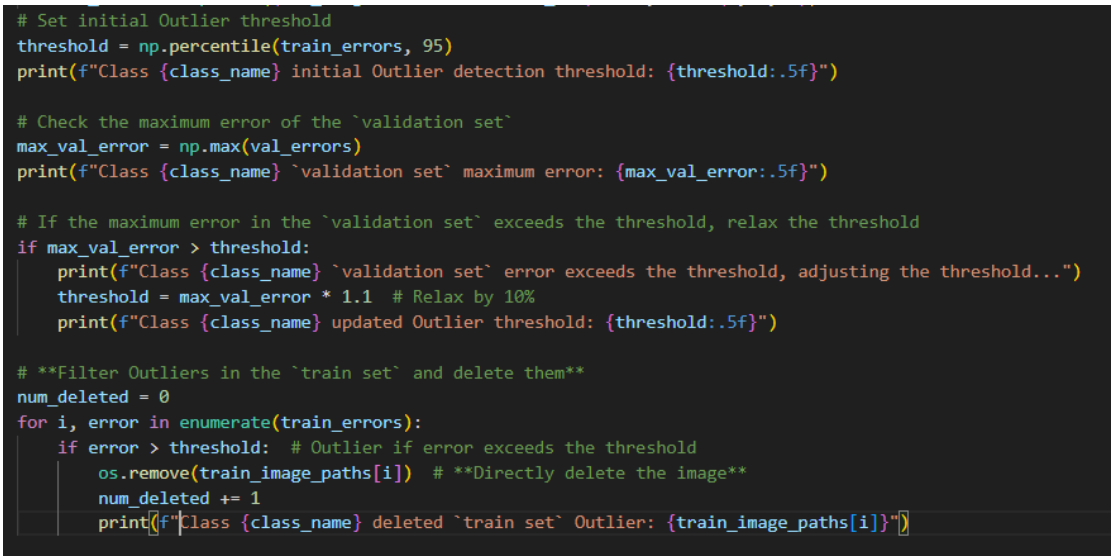
   - **Encoder**: Converts input data (images) into smaller dimensions.
   - **Decoder**: Reconstructs the input data from the smaller dimensions, aiming to make the reconstruction as close as possible to the original input.



```
# train Autoencoder
autoencoder = build_autoencoder()
autoencoder.fit(train_images, train_images, epochs=30, batch_size=32, shuffle=True, validation_data=(val_images, val_images))

# reduild `train set` and eval the error
reconstructed_train = autoencoder.predict(train_images)
train_errors = np.mean((train_images - reconstructed_train) ** 2, axis=(1, 2, 3))

# rebuild `validation set` and eval the error
reconstructed_val = autoencoder.predict(val_images)
val_errors = np.mean((val_images - reconstructed_val) ** 2, axis=(1, 2, 3))
```

   I first use the training data to train the Autoencoder. When it learns the features of the class, we ask it to reconstruct those data that have been reduced to lower dimensions.

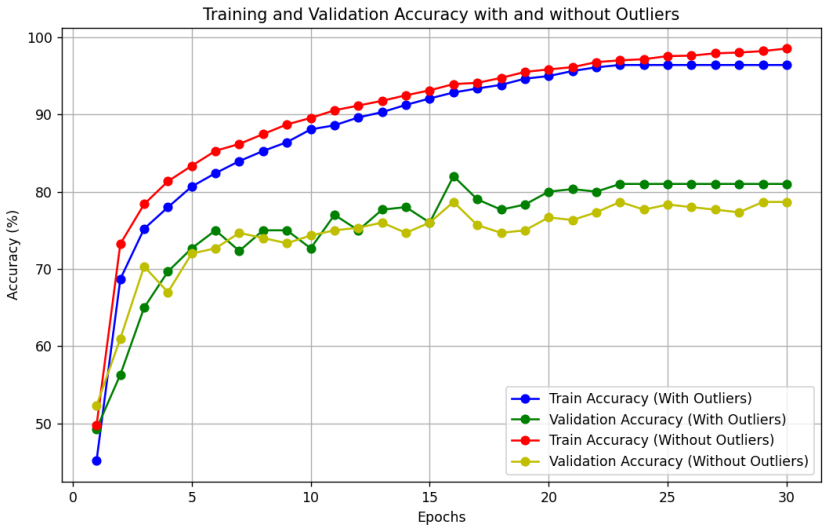
   If the reconstruction result differs significantly from the original image, we can reasonably suspect that the features of that image are not similar to the common features of the class. We can then infer that it is an outlier.(If the reconstruction error is above the threshold, I will classify the image as an outlier.)



```
# Set initial Outlier threshold
threshold = np.percentile(train_errors, 95)
print(f"Class {class_name} initial Outlier detection threshold: {threshold:.5f}")

# Check the maximum error of the `validation set`
max_val_error = np.max(val_errors)
print(f"Class {class_name} `validation set` maximum error: {max_val_error:.5f}")

# If the maximum error in the `validation set` exceeds the threshold, relax the threshold
if max_val_error > threshold:
    print(f"Class {class_name} `validation set` error exceeds the threshold, adjusting the threshold...")
    threshold = max_val_error * 1.1  # Relax by 10%
    print(f"Class {class_name} updated Outlier threshold: {threshold:.5f}")

# **Filter Outliers in the `train set` and delete them**
num_deleted = 0
for i, error in enumerate(train_errors):
    if error > threshold:  # Outlier if error exceeds the threshold
        os.remove(train_image_paths[i])  # **Directly delete the image**
        num_deleted += 1
        print(f"Class {class_name} deleted `train set` Outlier: {train_image_paths[i]}")
```

This threshold is typically set to the 95th percentile of the training errors for all images. Additionally, to avoid the validation data being misclassified as outliers, I will also compress and reconstruct the validation data and calculate the error. If the error is smaller than the threshold, we will not consider the validation data as an outlier. If the error is greater than the threshold, the threshold will be updated to 1.1 times the error from the validation data.

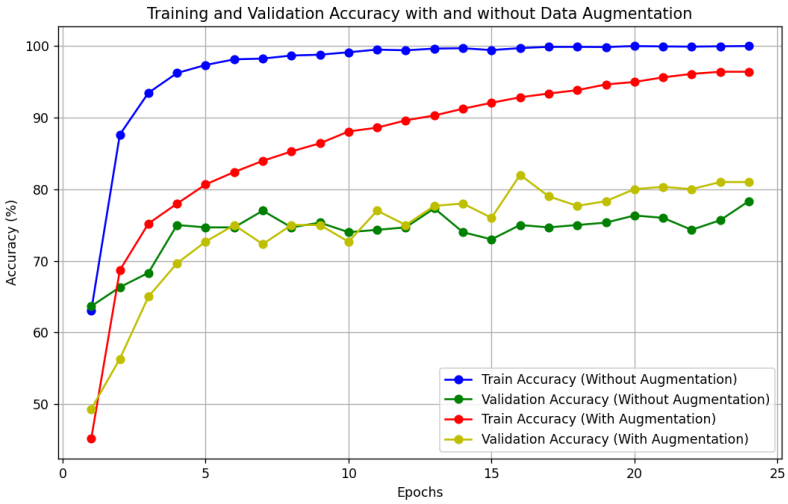
This method ensures that the validation data is not misclassified as an outlier and helps to eliminate anomalous images whose features deviate too much from the common features of other images in the same class. Here are the training results using this method and directly using the original, unmodified dataset. It can be seen that with outlier exclusion, the training stopped early at epoch 23 (early stopping), showing faster training and better results.



- With outliers, the training stopped early at epoch 23. At epoch 16, the validation accuracy reached a maximum of 82.00%.
- Without outliers, the validation accuracy reached a maximum of 79.67% at epoch 23. The training continued until epoch 30 before early stopping occurred.

2. Data augmentation
    - **Padding & Resize**: All images are padded to the same size square, ensuring that no aspect ratio distortion occurs when resizing the image.

- **RandomResizedCrop**: A random region of the image is cropped, allowing us to get a close-up, which is then scaled to 224x224, the size required by ResNet.
- **ColorJitter on Brightness, Contrast, Saturation, and Hue**: This mainly affects the RGB factors of the image. By slightly adjusting the brightness, contrast, and saturation, the model can accommodate a wider range of photographic conditions.
- **Random Rotation**: Since most of the objects in the images are plants and animals (rather than numbers or text), this technique can effectively provide variations of the features in the images from different shooting angles.
- **Random Horizontal Flip**: A 50% chance of flipping the image horizontally.



- The plot shows that training accuracy with data augmentation (red line) improves more slowly compared to the model without data augmentation (blue line). This indicates that data augmentation helps the model avoid overfitting and generalize better.
- On the other hand, the validation accuracy without data augmentation (green line) stagnates or fluctuates, showing that the model struggles to generalize well to unseen data.
- Overall, data augmentation improves the model's generalization ability, leading to better performance on the validation set.

Results:

- **Model**: seresnet152d.ra2_in1k

- https://huggingface.co/timm/seresnet152d.ra2_in1k
- **Learning Rate**:
    - Initial learning rate: 0.00015
    - Learning rate decay:

```
# Manually adjust the learning rate if no improvement
if(early_stop_counter%2==0 and early_stop_counter!=0):
    optimizer.param_groups[0]['lr'] = optimizer.param_groups[0]['lr'] * 0.8
```

    - Training strategy:
        - Early stopping: patience = 10 (Terminate the training if the validation accuracy does not improve for 10 consecutive times)
        - Reduce the learning rate by a factor of 0.8 if there is no improvement in validation accuracy for two consecutive epochs.
- **Outlier Elimination**: Applied
- **Data Augmentation**: All techniques above applied.
- **Mix up applied.**

```
Epoch 15/60 | Train Acc: 50.62% | Val Acc: 89.67% | Val Loss: 1.11989
Epoch 16/60 | LR: 0.000019440
Epoch 16/60 | Train Acc: 48.73% | Val Acc: 90.00% | Val Loss: 1.09676
Epoch 17/60 | LR: 0.000019440
Epoch 17/60 | Train Acc: 47.28% | Val Acc: 89.33% | Val Loss: 1.07621
Epoch 18/60 | LR: 0.000019440
Epoch 18/60 | Train Acc: 47.37% | Val Acc: 91.00% | Val Loss: 1.06623
 Confusion Matrix saved at: confusion_matrix18.png
Epoch 19/60 | LR: 0.000019440
Epoch 19/60 | Train Acc: 50.98% | Val Acc: 89.67% | Val Loss: 1.08875
Epoch 20/60 | LR: 0.000019440
Epoch 20/60 | Train Acc: 47.63% | Val Acc: 90.67% | Val Loss: 1.08432
Epoch 21/60 | LR: 0.000019440
Epoch 21/60 | Train Acc: 49.91% | Val Acc: 90.33% | Val Loss: 1.07803
Epoch 22/60 | LR: 0.000019440
Epoch 22/60 | Train Acc: 50.17% | Val Acc: 90.33% | Val Loss: 1.08320
Epoch 23/60 | LR: 0.000011664
Epoch 23/60 | Train Acc: 50.88% | Val Acc: 89.67% | Val Loss: 1.09774
Epoch 24/60 | LR: 0.000011664
Epoch 24/60 | Train Acc: 51.45% | Val Acc: 90.00% | Val Loss: 1.08466
Epoch 25/60 | LR: 0.000011664
Epoch 25/60 | Train Acc: 50.30% | Val Acc: 91.00% | Val Loss: 1.07863
Epoch 26/60 | LR: 0.000011664
Epoch 26/60 | Train Acc: 48.79% | Val Acc: 89.67% | Val Loss: 1.08240
Epoch 27/60 | LR: 0.000006998
Epoch 27/60 | Train Acc: 48.44% | Val Acc: 90.00% | Val Loss: 1.09235
Epoch 28/60 | LR: 0.000006998
Epoch 28/60 | Train Acc: 49.40% | Val Acc: 89.67% | Val Loss: 1.07545
Epoch 29/60 | LR: 0.000006998
Epoch 29/60 | Train Acc: 51.72% | Val Acc: 90.33% | Val Loss: 1.08927
Epoch 30/60 | LR: 0.000006998
Epoch 30/60 | Train Acc: 52.76% | Val Acc: 90.67% | Val Loss: 1.06184
Early Stopping Activated!

 Best model weights saved! Validation Acc: 91.00%
Confusion Matrix saved at: confusion_matrix.png
```

Test: public:0.933

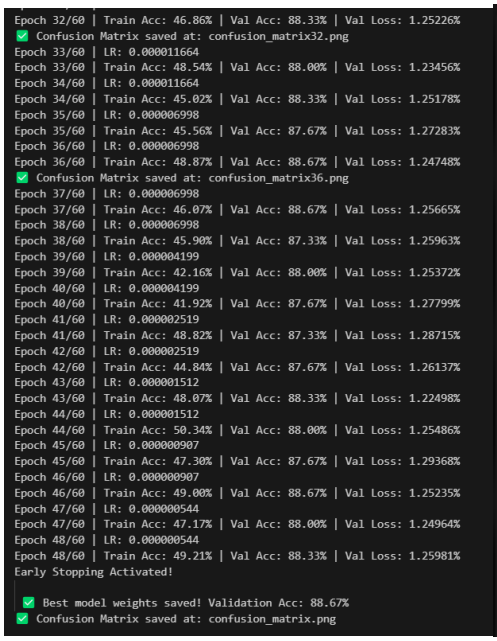
Additional Experiments:

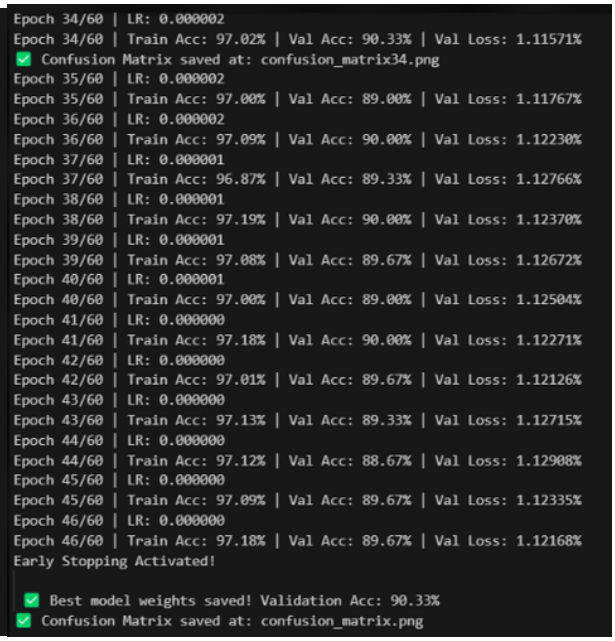
1. Mix up

Ref: https://zhuanlan.zhihu.com/p/439205252

My score has been stagnating at a certain level, and no matter how I change the model, I cannot significantly improve the score. Therefore, I believe there are a few difficult-to-handle data points in the test set that the model cannot learn well in the training process. I suspect that some of the data points have features that are mixed between two categories, which makes it difficult for the model to confidently determine which category the image belongs to. As a result, I need to make changes to my training set to help the model better handle these situations.

The method I chose is mixup, which allows you to blend two training samples at a chosen ratio to create new data. By adding these mixed samples into the training process, the model can better handle samples that may fit multiple categories at once.





Init LR:0.00015, Decay strategy: same,

Data augmentation:

```python
# ✅ Data Augmentation
train_transform = transforms.Compose([  # Data augmentation
    transforms.Resize((256, 256)),
    transforms.RandomResizedCrop(224, scale=(0.5, 1.0)),  # Increase range of scaling
    transforms.ColorJitter(brightness=0.4, contrast=0.5, saturation=0.5, hue=0.3),  # Stronger color jitter
    transforms.RandomHorizontalFlip(p=0.5),  # Increase flip probability
    transforms.RandomRotation(45),  # Increased rotation range
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1), scale=(0.9, 1.1), shear=10),  # Add affine transform for slight changes in perspective
    transforms.RandomPerspective(distortion_scale=0.3, p=0.5, interpolation=3),  # Add perspective distortion for more variation
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    transforms.RandomErasing(p=0.4, scale=(0.02, 0.25))  # Increase probability and area for random erasing
])
```

Drop out: 0.5

We can see that the speed of increase in train accuracy has a significant gap. With mixup applied, the train accuracy even drops below the val accuracy and cannot exceed 50 for an extended period. This ensures that the val accuracy does not become difficult to improve due to the train accuracy plateauing. In this experiment, the accuracy of the submitted CSV without mixup was 0.94, while with mixup it was 0.92. Although the experiment results show that the model with mixup performed worse, I believe this might be due to the learning rate being too small. Therefore, I think it would be beneficial to adjust the initial learning rate to see if this method can improve the final result.

Here is the result：

```
Epoch 44/60 | LR: 0.000005039
Epoch 44/60 | Train Acc: 47.80% | Val Acc: 87.33% | Val Loss: 1.31161%
✅ Confusion Matrix saved at: confusion_matrix44.png
Epoch 45/60 | LR: 0.000005039
Epoch 45/60 | Train Acc: 49.60% | Val Acc: 86.33% | Val Loss: 1.33063%
Epoch 46/60 | LR: 0.000005039
Epoch 46/60 | Train Acc: 49.11% | Val Acc: 86.33% | Val Loss: 1.33583%
Epoch 47/60 | LR: 0.000003023
Epoch 47/60 | Train Acc: 48.04% | Val Acc: 86.33% | Val Loss: 1.31065%
Epoch 48/60 | LR: 0.000003023
Epoch 48/60 | Train Acc: 44.59% | Val Acc: 87.00% | Val Loss: 1.29274%
Epoch 49/60 | LR: 0.000001814
Epoch 49/60 | Train Acc: 45.28% | Val Acc: 86.67% | Val Loss: 1.33054%
Epoch 50/60 | LR: 0.000001814
Epoch 50/60 | Train Acc: 45.71% | Val Acc: 87.00% | Val Loss: 1.29476%
Epoch 51/60 | LR: 0.000001088
Epoch 51/60 | Train Acc: 47.67% | Val Acc: 87.00% | Val Loss: 1.31837%
Epoch 52/60 | LR: 0.000001088
Epoch 52/60 | Train Acc: 47.14% | Val Acc: 87.33% | Val Loss: 1.27255%
Epoch 53/60 | LR: 0.000000653
Epoch 53/60 | Train Acc: 48.66% | Val Acc: 87.33% | Val Loss: 1.31278%
Epoch 54/60 | LR: 0.000000653
Epoch 54/60 | Train Acc: 49.86% | Val Acc: 87.33% | Val Loss: 1.27937%
Early Stopping Activated!

✅ Best model weights saved! Validation Acc: 87.33%
✅ Confusion Matrix saved at: confusion_matrix.png
```

Initial LR:0.0005

The result is even wrose. The submitted result is 0.91.