Homework 1: Image Classification

IOC 313551093 盧品樺

Github: https://github.com/huahua1018/NYCU_CV2025/tree/main/HW1

1 Introduction

The objective of this homework is to classify 2D images into 100 categories. To deepen our understanding of ResNet, we are required to use it as the backbone of our model and further enhance its performance by adding or modifying components.

I selected ResNeXt101_32x8d[1] as the backbone and improved its capabilities by incorporating dropout and the Convolutional Block Attention Module (CBAM)[2]. Additionally, I applied Mixup[3] for data augmentation. The details of the method are described in Section 2.

In the experiments, the model achieved an accuracy of 0.91 on the validation set. The results are presented in Section 3.

2 Method

2.1 Data Preprocessing

According to the official documentation of the pretrained ResNeXt101_32x8d[1], we perform normalization using the standard mean and standard deviation values for ImageNet. Specifically, the mean values for the three RGB channels are set to $[0.485, 0.456, 0.406]$, and the standard deviations are set to $[0.229, 0.224, 0.225]$. This ensures that the input distribution remains consistent with the pretrained model.

2.2 Data Augmentation

To enhance model robustness, I applied various data augmentation techniques to the training data. Specifically, I utilized several common transformations provided by torchvision.transforms, including:

- **RandomResizedCrop:** Randomly crops the input image to 224×224 while preserving 80%–100% of the original size to simulate scale variations.
- **RandomHorizontalFlip:** Applied with a 50% probability to improve directional invariance.
- **RandomRotation:** Rotates images within ±15° to simulate different camera angles.
- **ColorJitter:** Randomly adjusts brightness, contrast, saturation, and hue (±20%) to improve adaptability to varying lighting conditions.

Additionally, in image classification tasks, Mixup[3] is a widely used data augmentation technique. Therefore, I also applied Mixup to the training data to enhance model generalization. Mixup works by combining two images to create a new one. The new image is generated by taking a weighted sum of the two images, along with their corresponding labels.

2.3 Model Architecture

After fine-tuning various pre-trained ResNet models provided by PyTorch, including ResNet50, ResNeXt50, and ResNeXt101_32x8d, I found that ResNeXt101_32x8d achieved the best performance. Therefore, I selected this model as the backbone.

I first modify the final fully connected layer of the ResNeXt101_32x8d model to output 100 classes. And then I found that the model tend to overfit the training data. To address this issue, I added a dropout layer with a dropout rate of 0.5 after the global average pooling layer to prevent overfitting. Additionally, I froze the weights of layer 1 and layer 2 to retain their pre-trained values.

```
    self.fc = nn.Sequential(
        nn.Dropout(p=0.5),
        nn.Linear(in_features, num_classes)
    )
```

To further enhance the model's capabilities, I incorporated the Convolutional Block Attention Module (CBAM)[2]. CBAM is designed to improve the attention mechanism of the model, and I integrated it to enhance its feature selection capabilities. I inserted the CBAM module after the final layer of the ResNeXt101_32x8d model because, at this stage, the model has already extracted a rich set of features. Adding CBAM at this point helps the model better refine and select the most important features. Although I experimented with placing the CBAM module at different layers of the network, the performance was not as optimal as when it was placed after the final layer. Regarding the implementation of CBAM, I referred to [4] and integrated it into my model.

Since CBAM enhances the model's capabilities but also increases its complexity, I added a dropout layer with a drop rate of 0.2 between layer 3 and layer 4 to prevent overfitting while retaining sufficient information.

```
    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)

        x = self.layer3(x)
        x = self.dropout(x)

        x = self.layer4(x)
        x = self.cbam4(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x
```

2.4 Hyperparameters

- **Batch size:** 32
- **Epochs:** 50
- **Learning rate:** 0.0001
- **Min learning rate:** 0.000001
- **Optimizer:** AdamW
- **Weight decay:** 0.045
- **Scheduler:** ReduceLROnPlateau
- **Factor:** 0.1
- **Mixup alpha:** 0.2

3 Results

3.1 Analysis on Mixup

Fig. 1. and Fig. 2. show the training curves of the model without and with Mixup. In Fig. 1. , without Mixup, the model tends to overfit the training data, as evidenced by the validation loss being much higher than the training loss. However, in Fig. 2. , with Mixup applied, the validation loss is lower and closer to the training loss.

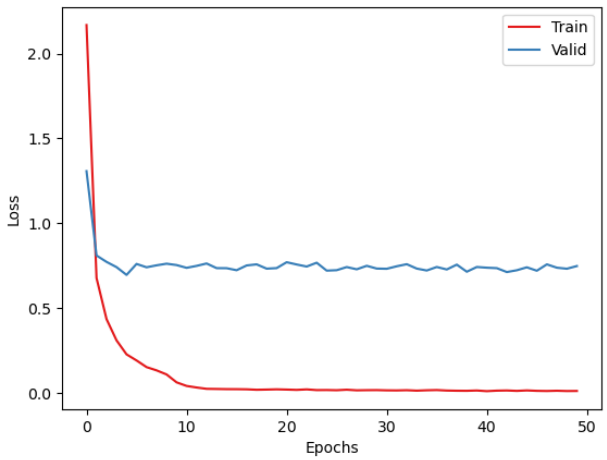

Fig. 1. Training curve without Mixup

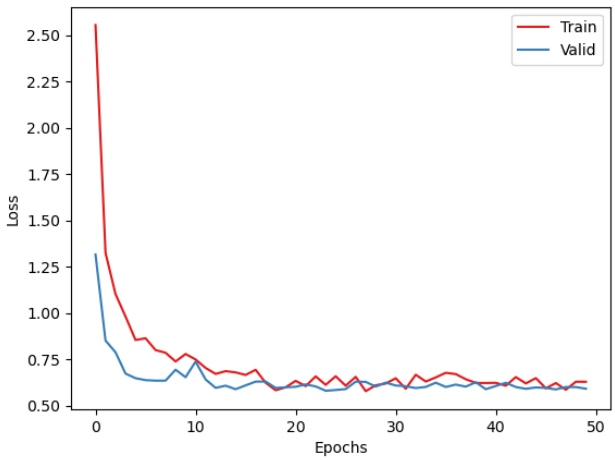

Fig. 2. Training curve with Mixup

From Fig. 3. and Fig. 4. , we can see that although both models achieve similar accuracy, the model with Mixup shows a significant reduction in validation loss. This suggests that Mixup helps improve the model's generalization ability.

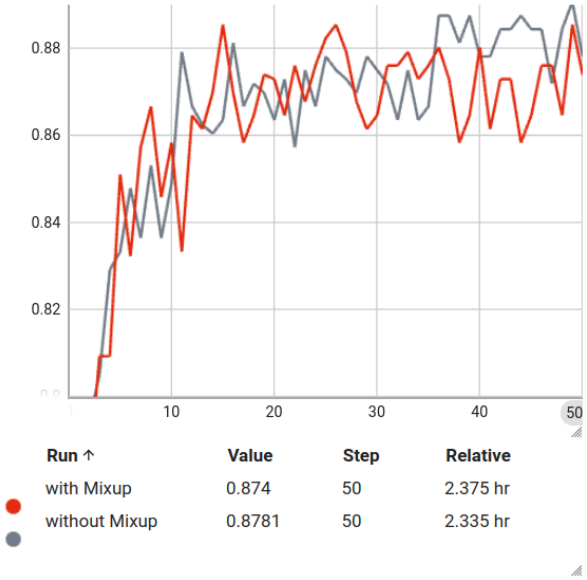

Fig. 3. Accuracy with/without Mixup

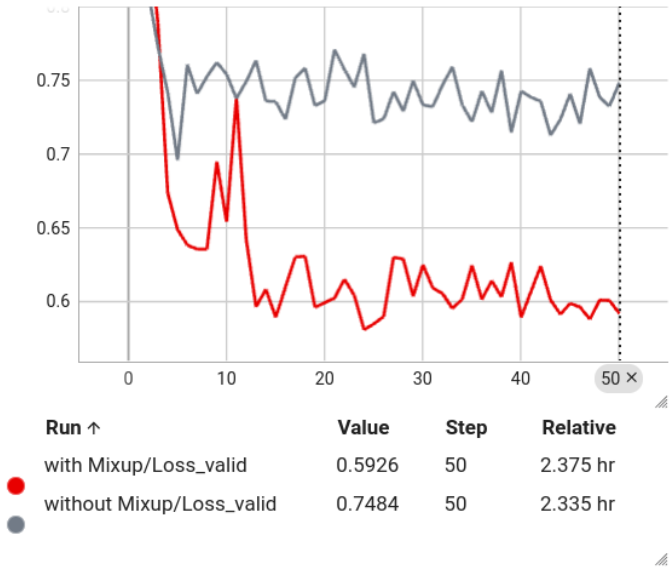

Fig. 4. Validation loss with/without Mixup

3.2 Analysis on CBAM

From Fig. 5. and Fig. 6. , it can be seen that after using CBAM, the model's validation loss increased.

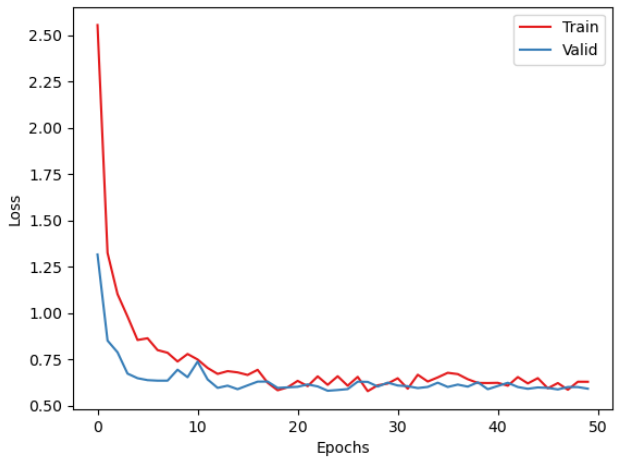

Fig. 5. Training curve without CBAM

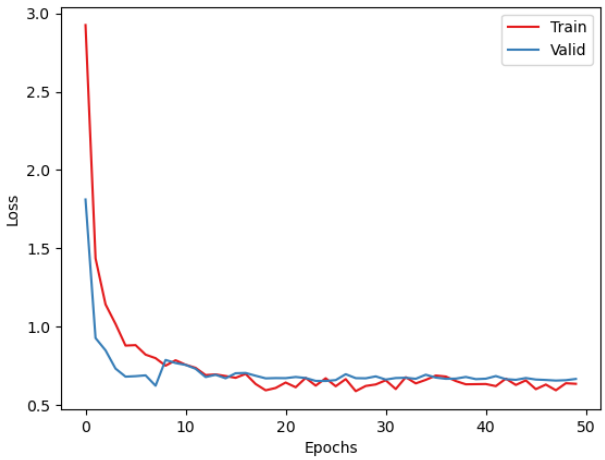

Fig. 6. Training curve with CBAM

From Fig. 5. and Fig. 6. , contrary to expectations, after adding the CBAM module, the model's accuracy decreased and the validation loss increased. I suspect this may be due to the increased model complexity. I believe overfitting might be the cause, so I next tested the effect of adding a dropout layer after layer 3.

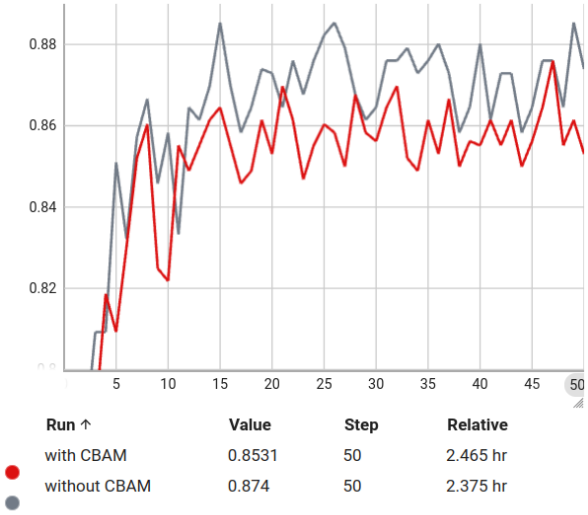

Fig. 7. Accuracy with/without CBAM

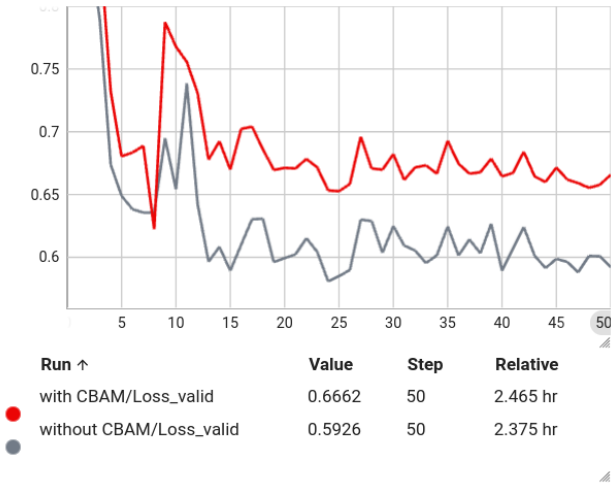

Fig. 8. Validation loss with/without CBAM

3.3 Analysis on dropout after layer 3

From Fig. 9. , we can observe that before adding dropout, the gap between validation loss and training loss was relatively large, indicating signs of overfitting. To address this, I added a dropout layer after layer 3. As shown in the Fig. 10. , the gap between validation and training loss decreased.

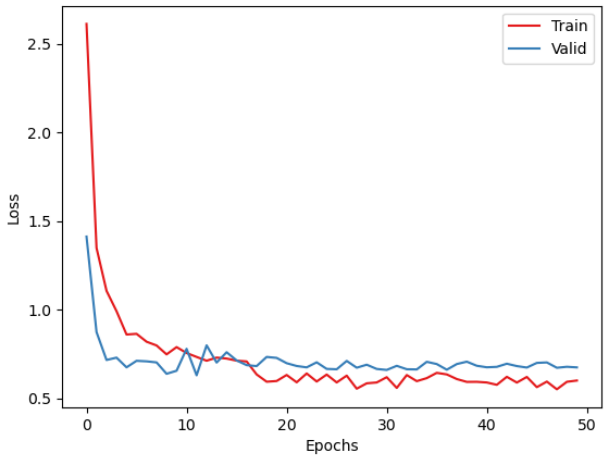

Fig. 9. Training curve without dropout after layer 3

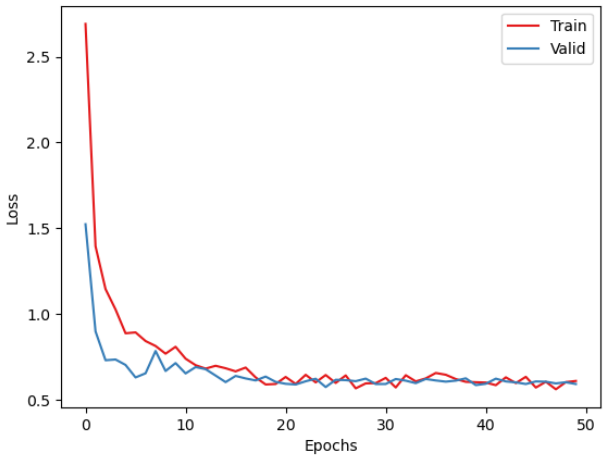

Fig. 10. Training curve with dropout after layer 3

From Fig. 11. and Fig. 12. , we can see that after adding dropout, the model's accuracy improved, and the validation loss decreased, suggesting that dropout effectively helps mitigate overfitting.

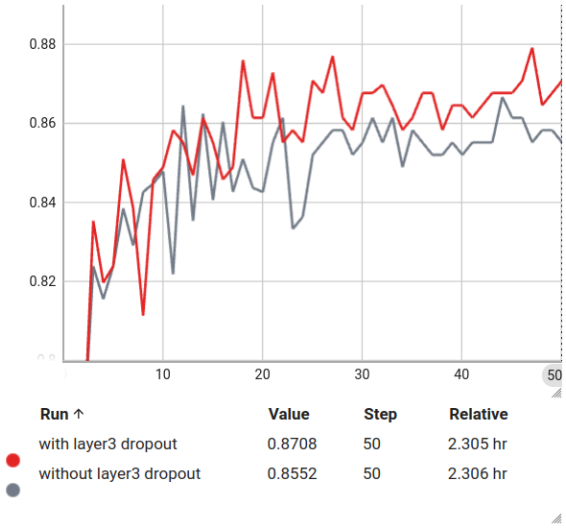

Fig. 11. Accuracy with/without dropout layer 3

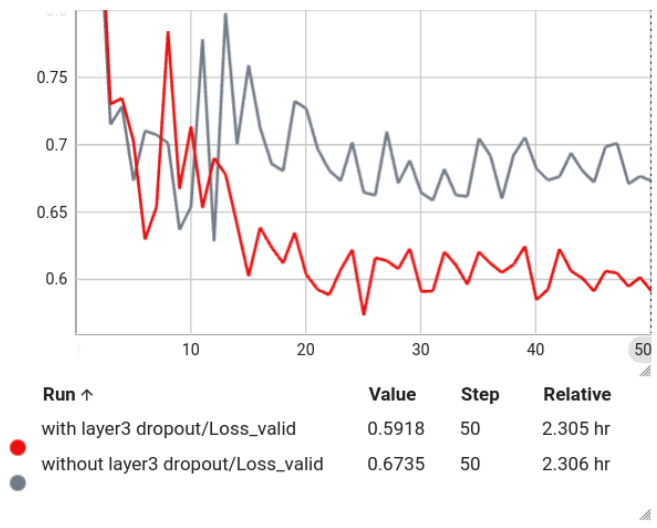

Fig. 12. Validation loss with/without dropout layer 3

3.4 Combined model performance

Combining all the aforementioned components that contribute to improving the model's learning, we obtain the results as follows.

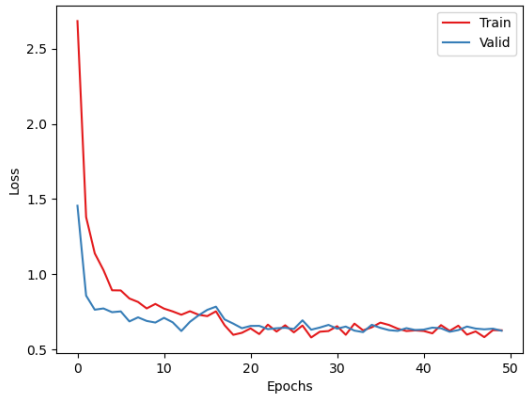

Fig. 13. Training curve of the combined model

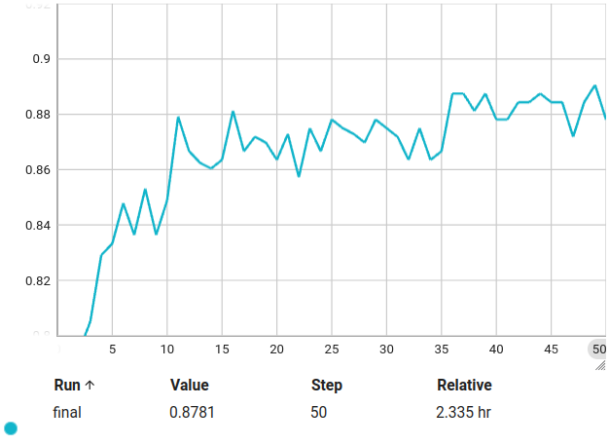

Fig. 14. Accuracy of the combined model

4 Additional experiments (optional)

I used Mixup for data augmentation, which not only increases the diversity of the data but also aims to help the model create smoother and more stable decision boundaries between classes, thereby reducing overfitting. However, I am concerned that Mixup might cause the model to primarily learn the blended images, rather than effectively learning from the original images, which could impact its performance on the validation set and test set. To address this, I randomly decide whether to apply Mixup before each iteration, with a 50% probability, in order to prevent the model from overly relying on the mixed images.

Through this approach, I aim to enhance the model's generalization ability using Mixup, while avoiding an over-reliance on the mixed images, thereby further improving the model's performance.

From Fig. 15. and Fig. 16. , we can see that the gap between the training loss and validation loss became larger with random Mixup. This is because with random Mixup, the model learns more from the original images, thus resulting in performance that lies between the scenario with no Mixup and the one with full Mixup.

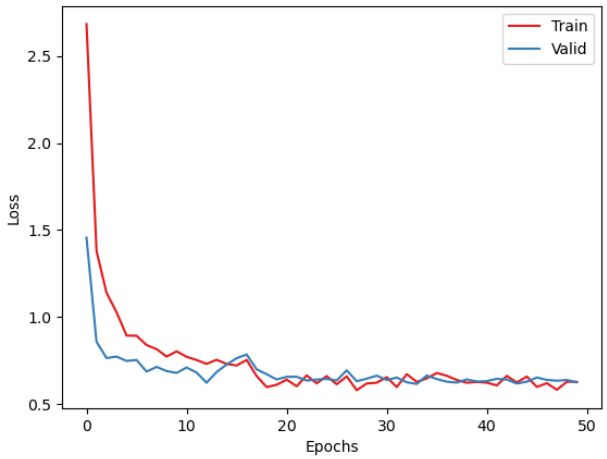

Fig. 15. Training curve without random Mixup

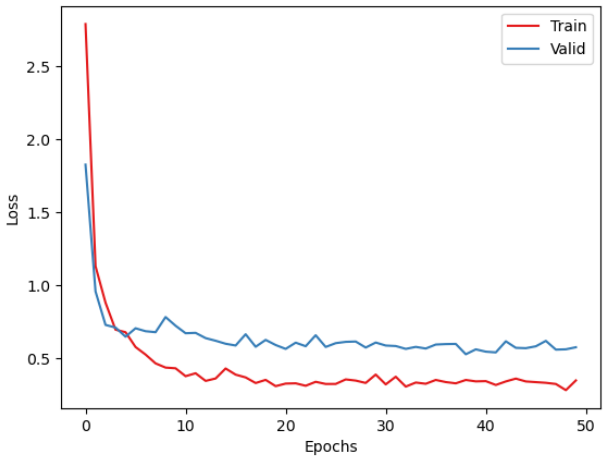

Fig. 16. Training curve with random Mixup

However, from Fig. 17. and Fig. 18. , we can observe that the model's accuracy with random Mixup is higher, and the validation loss is lower as well. This suggests that using random Mixup enhances the model's ability to generalize, leading to better performance on the validation set. By randomly applying Mixup, the model can effectively balance between learning from the original images and the augmented ones, avoiding overfitting while improving its overall robustness.

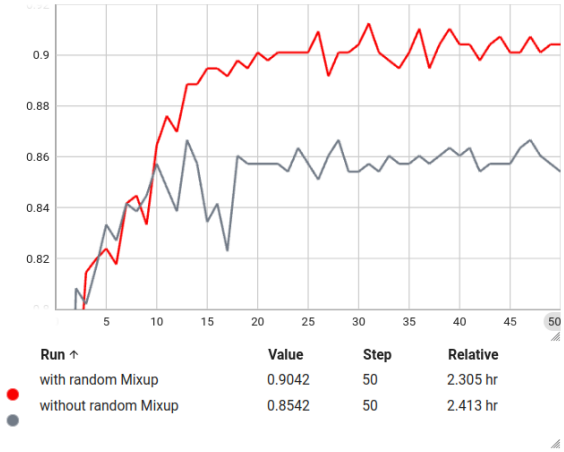

Fig. 17. Accuracy with/without random Mixup

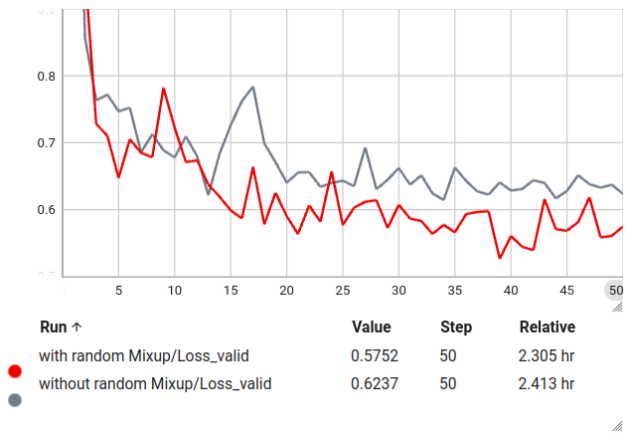

Fig. 18. Validation loss with/without random Mixup

By combining all the methods above, the final model achieves the best accuracy of **0.91** on the validation set.

5 References

[1] ResNeXt101: https://pytorch.org/vision/main/models/generated/torchvision.models.resnext32x8d.html#torchvision.models.resnext101_32x8d

[2] CBAM: https://dl.acm.org/doi/10.1007/978-3-030-01234-2_1

[3] Mixup: https://github.com/facebookresearch/mixup-cifar10/blob/main/train.py

[4] Code of CBAM: https://zhuanlan.zhihu.com/p/99261200