AI6126 Project 1 Report
Fashion Attributes Classification Challenges

YU Yue
G2202151A

Introduction

The task of the project is to identify the fashion attributes for the fashion images. Each image is annotated with 6 attributes indicating the 6 major categories it belongs to. For each category, we can regard the task as a multi-class classification. In this project, the pre-trained **ConvNeXt** model with ImageNet-1K is exploited as the backbone, and a classification layer is added to classify the images. Our implementation achieves 84.167% accuracy on the test set and ranked 4^{th} at $11{:}00am,\ 24/03/2023$. All the code can be downloaded from here. In this report, we discuss about the details of our implementation.

Results

#	User	Entries	Date of Last Entry	Overall Accuracy ▲
1	ZhanShuo	16	03/23/23	0.85100 (1)
2	wfjiang	11	03/24/23	0.84700 (2)
3	jhchiam	31	03/21/23	0.84533 (3)
4	YuYue525	227	03/23/23	0.84167 (4)

Model

In our implementation, we use pre-trained ConvNeXt [1] model with ImageNet-1K from Pytorch as our backbone. Compared with the traditional ResNet and Swin Transformer model, ConvNeXt is upgraded in the following aspects:

1. **Macro design**. ConvNeXt adjusts the number of blocks in each stage of ResNet50 to be consistent with Swin-T from $(3, 4, 6, 3)$ to $(3, 3, 9, 3)$ and replaces the ResNet-style stem cell with a “Patchify” stem implemented using a 4×4, stride 4 convolutional layer.
2. **ResNeXt-ify**. ConvNeXt uses depth-wise convolution, a special case of grouped convolution where the number of groups equals the number of channels, which is like the weighted sum operation in self-attention.

3. **Inverted Bottleneck**. ConvNeXt uses an inverted bottleneck block, and the position of the spatial depth-wise convolutions layer is moved up.

	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 [1×1, 384 1×1, 96]] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 [1×1, 768 1×1, 192]] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 [1×1, 1536 1×1, 384]] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 [1×1, 3072 1×1, 768]] × 2
FLOPs		4.1×10^9	4.5×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6	28.3×10^6

Figure 1: Detailed architecture specifications for ResNet-50, ConvNeXt-T and Swin-T. [1]

4. **Large Kernel Sizes**. ConvNeXt moves the depth-wise convolutions to the beginning of the inverted bottleneck block as preparation, and then adopts larger kernel-sized convolutions 7×7.
5. **Micro Design**. ConvNeXt adopts GELU instead of ReLU, fewer activation functions and fewer normalization layers, substitutes batch normalization with layer normalization and separate down-sampling layers.

Figure 1 shows the detailed structure of ConvNeXt compared with ResNet-50 and Swin-T. In our implementation, we use the pre-trained ConvNeXt model *convnext_large* with weights *ConvNeXt_Large_Weights.IMAGENET1K_V1* in Pytorch as the backbone. For more details, please refer to the official website.

The input of the model is the images resized to $(224, 224)$ and the output of the backbone is labels with size $(N, 1000)$. Because our task has 26 labels, we directly add a fully connected layer as the classification layer mapping from $(1000,)$ to $(26,)$. Then the **total number of parameters** of the model is $197{,}793{,}362$.

Loss Function

In our implementation, we regard the task as several multi-class classification challenges but jointly trained in one model. Specifically, for each category, we need to output one label from several attribute classes, so the task is a multi-class classification task. For the multi-class classification problem, generally we use cross-entropy loss, but in our implementation, we use focal loss [2] for better performance on the imbalanced dataset.

For the binary cross-entropy loss, the formula can be written as:

$$\mathcal{L}_{CE}(y,\hat{p}) = \begin{cases} -\log(\hat{p}), if\ y = 1 \\ -\log(1-\hat{p}), if\ y = 0 \end{cases}$$

We can use $p_t = \begin{cases} \hat{p}, if\ y = 1 \\ 1-\hat{p}, if\ y = 0 \end{cases}$ to represent the closeness between the prediction and the ground truth, the larger the p_t, the better the performance. Then the cross-entropy loss can be written as:

$$\mathcal{L}_{CE}(y,\hat{p}) = -\log(p_t)$$

By using the same annotation, the focal loss can be written as:

$$\mathcal{L}_{FL}(y,\hat{p}) = -(1-p_t)^{\gamma}\log(p_t)$$

In the formula, compared with cross entropy loss, focal loss has one more modulating factor $(1-p_t)^{\gamma}$. For those are classified accurately, $p_t \to 1$ so that the modulating factor approaches to 0, for those are inaccurately classified, $p_t \to 0$ so that the modulating factor approaches to 1. That is, compared with the cross-entropy loss, the focal loss does not change the loss for samples with inaccurate classification, and the loss for samples with accurate classification will become smaller. Therefore, focal loss is equivalent to increasing the weight of difficult-to-classify samples in the loss function, which helps to improve the accuracy of difficult-to-classify samples.

In the experiments, we also tried to add weights for each class, then the focal loss becomes $\mathcal{L}_{FL}(y,\hat{p}) = -\alpha(1-p_t)^{\gamma}\log(p_t)$ where α is the weights for the class. Specifically, $\alpha = \frac{\#\ of\ samples}{\#\ of\ samples\ in\ this\ class}$The less the number of samples in this class, the higher the weight. However, the experiment results show that without the weights, the model will perform better on the validation set. Therefore, in the implementation with the highest score, we discard this weight hyper-parameter.

Actually, before computing the focal loss for each category, we also use softmax for the output. Therefore, the whole process can be written as the following equation:

$$\mathcal{L} = \sum_{c=1}^{C}\mathcal{L}_c = \sum_{c=1}^{C}\sum_{n=1}^{N} -\left(1-\left(softmax(o_c)\right)^{(y)}\right)^{\gamma}\log\left(\left(softmax(o_c)\right)^{(y)}\right)$$

Specifically, o_c means the output of the category c, $\left(softmax(o_c)\right)^{(y)}$means after applying softmax on the output, we select the one according to the ground-truth. For example, if the ground-truth is label 3 in the category 1, we just select the third number from $softmax(o_c)$ as the prediction of the probability that the sample belongs to label 3 in the category 1. For each sample we calculate the focal loss of total 6 categories and sum them up as the final loss.

Here we draw the process to calculate the loss:

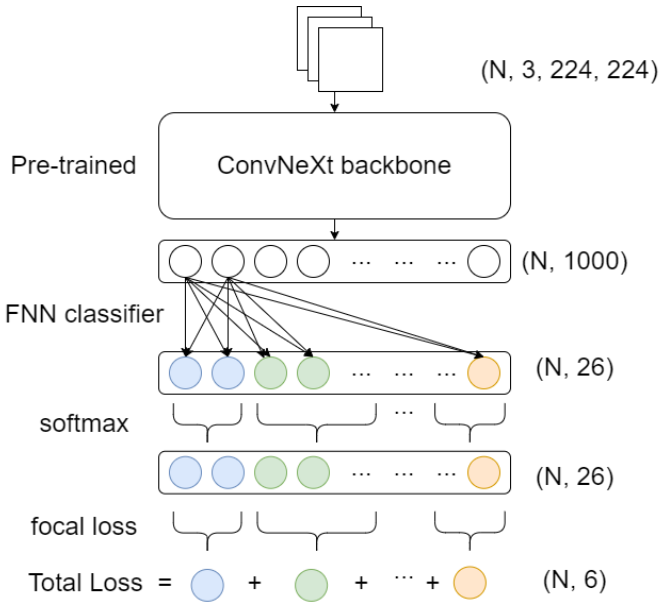

Figure 2 The structure of the model

Training Process

First of all, to find the optimal settings for training, we trained the model on the training set and validate the model on the validation set. We find with the following settings the validation set can achieve the highest accuracy: Adam optimizer with the initial learning rate $1e-3$ and no weight decay; the γ in the loss function is set to 2; a total of 50 epochs are trained, and after every 2 epochs, the learning rate drops to 0.8 of the original; the batch size is set to 64. For the data preprocessing, we just resize the original training and validation images to the size $(224, 224)$ and then use random horizontal flipping on the training images to do data augmentation. Figure 3 shows the learning curves of training and validating.

As we can see from the figure, the loss on the training is decreasing, but for validate loss, it decreases at the beginning and then increases because of overfitting. For the training and validation accuracy, both are increasing during the training.

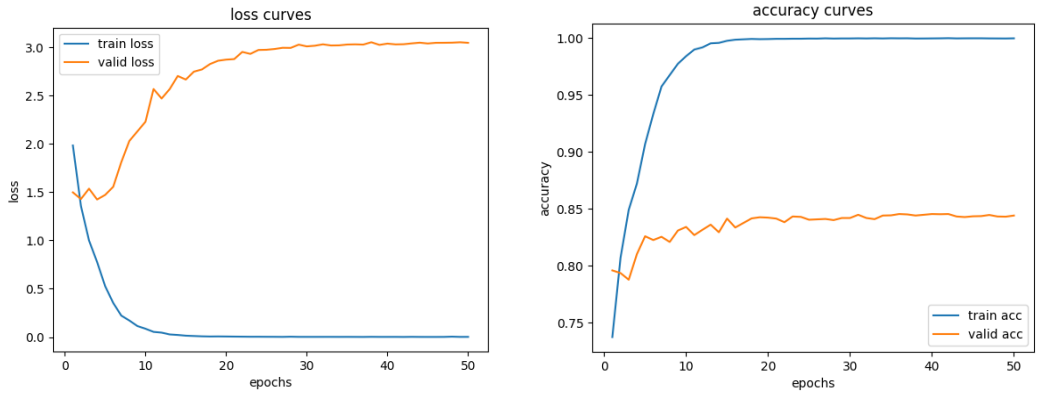

Figure 3 The loss and accuracy curves for training and validation

After finding the optimal settings, we trained the model on both training set and validation set with the same settings (I have already checked with TA Zhou Chong that we are allowed to do so). The training curves are shown in figure 4.

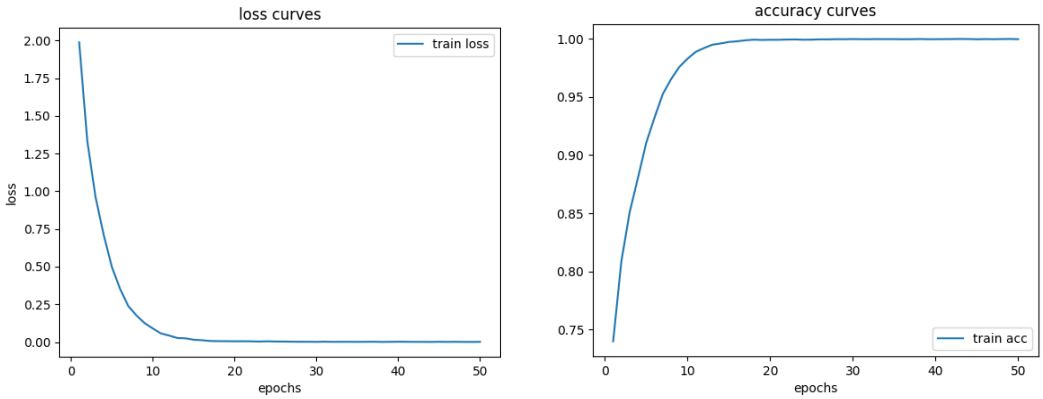

Figure 4 The loss and accuracy curves for training

Then, after training for 50 epochs, we test the model on the test set and get the highest accuracy 84.167%. The model weights can be downloaded from here.

Training Machine

We trained our model on one Nvidia A40 GPU with 30GB memory usage provided by SCSE GPU cluster, NTU.

Reference

[1] Zhuang Liu et al. *A ConvNet for the 2020s.* 2022. arXiv: 2201.03545 [cs.CV].

[2] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection.* 2018. arXiv: 1708.02002 [cs.CV].