Exploration of DeepPermNet Lab Report

521030910411 Bolun Zhang
Shanghai Jiao Tong University
zbl1677590857@foxmail.com

Abstract: Based on the paper "DeepPermNet: Visual Permutation"[1], we constructed a DeepPermNet to solve jigsaw puzzle problems. Following the description in the paper and considering the scale of problem data CIFAR-10, we adjusted the parameters of the original network and further explored optimization measures, such as adding batch normalization layers. After extensive exploration and optimization, our model achieved convergence in a relatively short time and achieved a 95.67% accuracy rate on the test set.

Key Words: ALexNet,DeepPermNet, CosineAnnealing, Cross Entropy,SGD

1 Introduction

1.1 Problem Restatement

Our task is to investigate the image jigsaw puzzle task on the CIFAR-10 dataset. Specifically, we divide an image into several sub-images, such as four pieces by horizontal and vertical splitting. Then, we shuffle these sub-images and use them as input. The goal is to design a model that can reconstruct the original order of these shuffled sub-images.

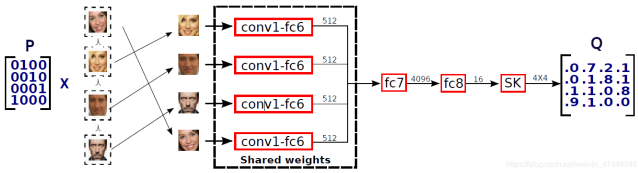

Fig. 1: Network structure in Reference Papers

1.2 Problem Standardization

To describe the problem more clearly, we standardized the shuffling process of the image by performing operations with a randomly generated permutation matrix P, as shown in the figure. Given a series of images X arranged in normal data order and a randomly generated permutation matrix P, PX is the shuffled image, and the role of the network is to predict the permutation matrix P. For the network, we need to convert the matrix into a vector as the input label. Specifically, for the P matrix in the figure, we convert it into the vector [1,2,3,0] according to the position of '1' in each row, which serves as the label category for each sub-image, representing its position in the original image. The network generates a 1*4 vector for each sub-image, which predicts the probability of belonging to each category, i.e., its position in the original image. The position with the highest probability in each vector corresponds to the predicted result of the sub-image's position in the original image. In the figure, we can see that the Q matrix is successfully predicted with the vector [1,2,3,0] after the above conversion.

1.3 Network Architecture

As shown in the figure, we first established AlexNet to process the segmented sub-images, and then concatenated the processed sub-images together. Then, we passed them through DeepPermNet to obtain the prediction results. It is worth noting that we did not directly use the SK strategy in thefigure, but instead innovatively incorporated the Softmax process into the cross-entropy calculation function under the Jittor framework. This not only helps us understand the essence of the problem but also fully utilizes the functionality of the Jittor framework to simplify the code structure.

1.4 Loss Calculation

Note that after standardization, the problem can be viewed as a sub-image classification problem, so we can use cross-entropy strategy to calculate the loss value.

2 Concrete Implementation

2.1 AlexNet

2.1.1 Background

AlexNet is a convolutional neural network architecture that was proposed by Alex Krizhevsky et al. in their 2012 paper "ImageNet Classification with Deep Convolutional Neural Networks". It consists of 5 convolutional layers followed by 3 fully connected layers. The network is designed to classify images into 1000 categories of the ImageNet dataset.

2.1.2 Implement

We have innovated upon the aforementioned architecture, which consists of convolutional layers, batch normalization layers, pooling layers, activation function layers, dropout layers, and fully connected layers.

- Convolutional layer:The convolutional layers include a first layer with an input channel of 3, output channel of 32, a 3x3 kernel size, and a padding of 1; a second layer with an input channel of 32, output channel of 64, a 3x3 kernel size, and a padding of 1; a third layer with an input channel of 64, output channel of 128, a 3x3 kernel size, and a padding of 1; a fourth layer with an input channel of 128, output channel of 256, a 3x3 kernel size, and a padding of 1; and a fifth layer with an input channel of 256, output channel of 256, a 3x3 kernel size, and a padding of 1.

- Batch normalization layer: Each convolutional layer output is followed by a batch normalization layer, and
- Max pooling layer:The first, second, and fifth convolutional layers are followed by a max pooling layer with a kernel size of 2x2 and a stride of 2.
- Activation function:ReLu
- Dropout layer:add dropout layers after the first, second, third, and fourth convolutional layers, before the fifth convolutional layer, and between the fully connected layers to prevent overfitting.
- Fully connected layer:The fully connected layers take an input of 256x2x2 and output 2048 and 1024, respectively. ReLU is used as the activation function between the fully connected layers, and the final output is the classification result.

2.2 DeepPermNet

On the basis of the network structure proposed in the paper, we innovatively designed the following network architecture.

- AlexNet Submodel: The first layer of the DeepPermNet model is an AlexNet submodel. This submodel consists of multiple convolutional layers, pooling layers, and ReLU activation functions, and is used to extract features from input images. The output of this submodel is a feature tensor with a shape of (batchsize, 256, 6, 6), where each element represents a pixel value on the feature map. These feature tensors will be the input to subsequent layers.
- Batch Normalization Layer: The second layer of the DeepPermNet model is a batch normalization layer. This layer is used to normalize the feature tensor to improve the training stability and generalization ability of the model. The output shape of this layer is the same as the input shape, which is (batchsize, 256, 6, 6).
- MLP Classifier: The third layer of the DeepPermNet model is a multi-layer perceptron (MLP) classifier. This classifier consists of multiple fully connected layers, batch normalization layers, ReLU activation functions, and dropout layers, and is used to map the feature tensor to 16 output classes. The output of this classifier is a tensor with a shape of (batchsize, 16), where each element represents a predicted score for an output class.
- L2 Regularization Layer:The final layer of the DeepPermNet model is an L2 regularization layer. This layer is used to perform L2 regularization on the output of the classifier to constrain the size of the output values. The input and output shapes of this layer are both (batchsize, 16).

Overall, the DeepPermNet model has a relatively simple structure, consisting of an AlexNet submodel, a batch normalization layer, an MLP classifier, and an L2 regularization layer. The AlexNet submodel is used to extract features from input images, the batch normalization layer is used to normalize the features, the MLP classifier is used to map the features to 16 output classes, and the L2 regularization layer is used to constrain the size of the output values. The output of the entire model is a tensor with a shape of (batchsize, 4, 4), where each element represents a predicted score for an output class.

2.3 Cross-entropy

Cross Entropy is a measure of the difference between two probability distributions. In machine learning and deep learning, cross-entropy is often used to evaluate the performance of a model by measuring the difference between the output of a model and the true label.

In classification problems, we typically represent the output of the model as a probability distribution, where each category corresponds to a probability value. Assuming that there are K categories, and pi is used to represent the probability that the model predicts to be class i, and qi is used to represent the probability that the true label is class i, then the formula for calculating cross-entropy is

$$H(p,q) = -\sum_{i=1}^{K} q_i \log p_i$$

2.4 Cosine Annealing

Cosine Annealing is an optimization algorithm commonly used in deep learning to adjust the learning rate. Its basic idea is to gradually reduce the learning rate during the training process, thereby making the model more stable and easy to converge.

The specific implementation of the Cosine Annealing strategy is to first set an initial learning rate lr0 and a total number of training epochs T. Then, during the training process, the learning rate is gradually reduced according to the form of the cosine function until the minimum value lrT is reached, and then gradually increased back to the initial value lr0. The advantage of the Cosine Annealing strategy is that it allows the model to use a larger learning rate in the early stages of training, accelerating the convergence of the model; and in the later stages of training, it can slowly reduce the learning rate to avoid the model from getting stuck in a local optimal solution.

The formula of the Cosine Annealing strategy is as follows:

$$lr_t = lr_{min} + \frac{1}{2}(lr_{max} - lr_{min}) \cdot (1 + \cos(\frac{t}{T} \cdot \pi))$$

Here, lr_t represents the learning rate at the t-th epoch, lr_{min} and lr_{max} represent the minimum and maximum learning rates, T represents the total number of training epochs, and t represents the current training epoch number. The cosine function in this formula ensures a smooth change in the learning rate, making the model more stable.

3 Experiment

3.1 Batch normalization layer

As we know, the original AlexNet architecture did not include Batch Normalization layers. As described earlier, we attempted to add these layers between the main layers and tested the running time and accuracy for 100 epochs.

Table 1: Add Batch normalization or not

Add Batch normalization	Max Accuracy(%)	Train Time (s)
T	95.67	3390.25
F	89.24	3871.21

We observed that the addition of Batch Normalization layers resulted in improvements in both accuracy and running time, with a particularly significant increase in accuracy.

3.2 Epoch

We conducted separate tests on the training time and test set accuracy corresponding to different epochs to determine the optimal number of epoches.

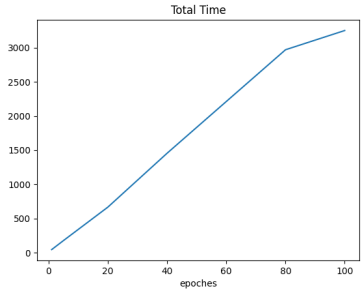

Fig. 2: epoch Time

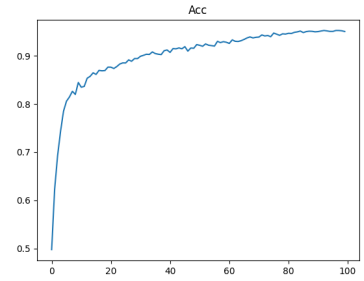

Fig. 3: epoch Acc

As shown in the figure, the model accuracy rapidly increased to over 80% before the 8th epoch, with a training time of less than 5 minutes. After that, the model accuracy increased slowly, reached over 90% at around the 40th epoch, with a total training time of about 20 minutes. We believe that, considering both efficiency and accuracy, the epoch at this point is optimal. After the 95th epoch, the training time reached 50 minutes, and the accuracy also increased to around 95%.

3.3 Learning rate adjustment

3.3.1 Problem

During the process of model testing, we found that if the learning rate is set too high, the model can converge quickly in the early stages, but the loss will stop decreasing after reaching a certain value, resulting in a final model accuracy of less than 80%. On the other hand, if the learning rate is set too low, the model may face the issue of slow or no decrease in loss.

3.3.2 Method

We realized that to address this issue, a strategy of dynamically adjusting the learning rate is needed.

- Manual adjustment: we employed a manual adjustment strategy. After observing the change of loss along with the epochs during one training process, we identified the epoch intervals where the loss decreased slowly or stopped decreasing, which were around the 10th and 30th epochs. At these two epochs, we reduced the learning rate by a certain proportion and recorded the optimal results during the adjustment process.
- Cosine Annealing: we adopted the Cosine Annealing strategy, adjusting the minimum learning rate to different values and recording the optimal results.

Table 2: Comparison of minimum learning rate

minimum learning rate	Max Accuracy(%)	Train Time (s)
1×10^{-5}	94.90	3490.34
1×10^{-8}	95.67	3390.25
1×10^{-12}	94.04	3852.25

Table 3: Comparison of Methods

Method	Max Accuracy(%)	Train Time (s)
Manual adjustment	93.25	3249.16
Cosine Annealing	95.67	3390.25

As shown in the table above, we found that the Cosine Annealing strategy demonstrated higher accuracy compared to the manual adjustment strategy, despite a similar total training time of 100 epochs.

3.4 Gradient descent optimization algorithm

During the experiment, we tried two gradient descent optimization algorithms, namely SGD and Adam. For SGD, we set the momentum to 0.9 and the weight decay to 1e-4. For Adam, we set the betas to (0.9, 0.75). Both methods were initialized with a learning rate of 0.1.We tested the accuracy and running time of the two methods, and the results are as follows: From the results, it can be seen that SGD outperforms Adam in terms of both running time for 100 epochs and accuracy.

Table 4: Comparison of Methods

Method	Max Accuracy(%)	Train Time (s)
Adam	91.35	4748.16
SGD	95.67	3390.25

3.5 The final effect

In the end, we achieved outstanding performance by innovatively adding Batch Normalization layers, adopting the Cosine Annealing learning rate adjustment strategy, and optimizing gradient descent with the SGD algorithm. Next, we present the performance in 100 epochs of our model.

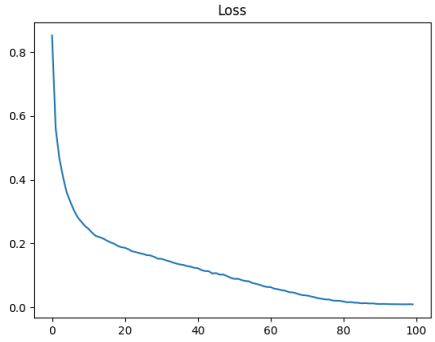

Fig. 4: Loss

As shown in the figure, the loss of the model steadily decreases and eventually converges.

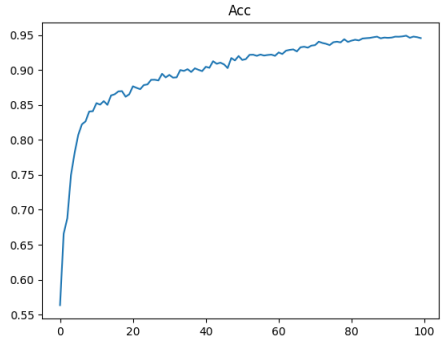

Fig. 5: ACC

In 100 epochs, our model achieved a maximum accuracy of 95.67%.

```
Train Epoch: 100 of 100 [49200/50000] Loss: 0.004606
Train Epoch: 100 of 100 [49300/50000] Loss: 0.014487
Train Epoch: 100 of 100 [49400/50000] Loss: 0.013811
Train Epoch: 100 of 100 [49500/50000] Loss: 0.016009
Train Epoch: 100 of 100 [49600/50000] Loss: 0.006903
Train Epoch: 100 of 100 [49700/50000] Loss: 0.022704
Train Epoch: 100 of 100 [49800/50000] Loss: 0.012620
Train Epoch: 100 of 100 [49900/50000] Loss: 0.018144
Acc = 0.9562
Best acc:0.9567
程序运行时间为：3390.60秒
```

Fig. 6: Result

4 Conclusion

In this experiment, we constructed a model for solving jigsaw puzzles based on the hints provided in the paper "DeepPermNet: Visual Permutation". We made significant innovative optimizations to the model, which ultimately resulted in achieving high accuracy. However, we still lack knowledge and skills to fully understand the essence of the network and optimize the model at a higher level. Perhaps the potential of the model is far beyond what we have achieved, and we still need to continuously learn and conduct research in order to develop better models.

References

[1] Rodrigo Santa Cruz et al. *DeepPermNet: Visual Permutation Learning*. 2017. arXiv: 1704.02729 [cs.CV].