Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/39/20230576

Further exploration on deep learning in flower recognition

Xiaoyu He

College of Electronics and Science Technology, Xiamen University, Xiamen, Fujian Province, 361005, China

2845120256@qq.com

Abstract. Flower recognition is an important research direction in the field of computer vision, and automatic classification of flower images through deep learning methods is of great significance for ecological environment monitoring and plant research. With this background, this study aims to further optimize the existing flower recognition system and improve its classification accuracy by adjusting key parameters. Based on the existing deep learning model and several rounds of training, this paper explores the tuning strategies for parameters such as different learning rates and weight decay to achieve higher recognition accuracy. In the experiments, this paper uses the classical flower dataset and enhances the diversity of the data through image preprocessing and data enhancement. Through a hundred rounds of training, the model in this paper achieves about 80% classification accuracy on the test set, which is significantly improved compared with the initial model. Further analysis of the results shows that by reasonably adjusting the learning rate and weight decay parameters, the model achieves certain improvements in different flower classes, demonstrating the impact of parameter tuning on the overall performance of the model.

Keywords: Deep Learning, Flower Recognition, Hyperparameter, Neural Network Model.

1. Introduction

Deep learning model optimization in flower recognition is a research direction worth exploring. For non-rigid objects such as flowers, "shallow learning" models like Support Vector Machine (SVM) and Neural Networks (NN) are less effective and cannot fulfill the recognition work well. With the continuous development of Pattern Recognition and Machine Learning (PRML) technology, a deep learning model has been proposed, which combines the shortcomings of neural networks, adds multiple hidden layers for training, and can obtain more reasonable recognition results [1]. Continuously deepening and optimizing the existing model is conducive to the further use of deep learning and artificial intelligence for the benefit of mankind.

Deep learning has made significant progress in the field of flower recognition but still faces a number of challenges and problems that may limit the improvement of model performance and the diffusion of applications. The current main optimization tools for deep learning are the continuous adjustment of parameters to train existing models and further adjustments to current architectures. However, deep learning models are usually complex and require a large amount of computational resources and time to train. Once the accuracy reaches a certain value, it is a challenging task to make the model more perfect and further improve the prediction accuracy.

In this paper, we focus on designing a new network structure or adjusting the hierarchy and parameters of the existing network in order to achieve a better capture of feature information of flower images and further improve the performance and effectiveness of the flower recognition task. The generalization ability of the model is further improved by data enhancement and regularization techniques when used. Find the optimal model parameters to further optimize the model. This paper makes a contribution to deep learning for flower recognition by improving the recognition performance through model optimization and hyperparameter tuning. This provides strong support for research and practical applications in related fields and lays a certain foundation for future exploration and development.

2. Literature review

The concept of deep learning was first introduced in the journal Science in 2006, defined as a machine learning process based on sample-trained multi-layered deep networks [2]. With the rapid progress in the field of deep learning and the rapid development of technology in the past few years, the convolutional neural network has become the most anticipated method among a host of neural network training methods by virtue of its unique advantages. With the continuous optimization and innovation of its model and system by later generations, its recognition accuracy has risen sharply. However, there are also challenges such as sample imbalance and multi-angle recognition. Deep convolutional neural networks have achieved good results in the fields of image classification, target detection, and image segmentation, and with the development of deep learning, excellent models continue to emerge. However, scholars have found that not the deeper the number of network layers the better the effect, along with the deepening of the number of network layers, problems such as gradient disappearance, gradient explosion, network degradation, and so on, have appeared [3]. Future research still needs to provide higher accuracy for flower recognition by further optimizing the model structure and improving the dataset.

When building a convolutional network model the parameters of each layer are set based on experience, for different objects in the recognition process may not be the optimal parameters, resulting in a large difference between the final output and the expected value, backpropagation is one of the most versatile methods for training multilayer feedforward neural networks [4]. Based on python, the complete process from data loading to model training, evaluation and saving is realized by defining the model, data preprocessing, training loop and model evaluation in several pieces of code. The basic data and model support are provided for the flower recognition program. Meanwhile, a convolutional neural network (CNN) consists of a series of layers stacked on top of each other, which receive the input image and perform mathematical operations (nonlinear activation functions such as ReLU,tanh) and predict the label probability at the output [5].

Based on the existing flower recognition model, this paper further adjusts and optimizes its parameters in order to achieve more accurate recognition and prediction. By adjusting the hyperparameters of the model, such as the learning rate and weight attenuation, the model is better adapted to the task of flower recognition and at the same time improves the performance of the model, and the appropriate optimization of the appropriate learning rate and the attenuation of the learning rate accelerates the convergence of the model and reduces the training time.

3. Methodology

3.1. Design introduction

The training set for this neural network is derived from web images of a collection of four separate flowers. After correctly building the required environment as well as successfully debugging the required source code. By continuously performing hyper-parameter tuning and model training, this experiment aims to successfully improve the accuracy of the flower recognition model. The impact of each hyperparameter on the model performance was gradually revealed by continuously adjusting different combinations of hyperparameters such as learning rate, optimizer, regularization, etc., as well as adjusting the model architecture and data preprocessing process during multiple training sessions. This process, although tedious, is an indispensable part of finding the optimal parameter configuration.

3.2. Design process

Training strategy means that equal number of rounds of training by constantly modifying the parameters until the maximum accuracy of the model is reached. Meanwhile, the optimizer type and regularization parameters still need to be adjusted to prevent overfitting and enhance the generalization ability of the model. In addition, subtle adjustments to each hyperparameter may have different degrees of impact. Numerous adjustments and modifications need to be made and tested to find the combination of hyperparameters that achieves the best results.

Part of the code explanation with the role of the relevant hyperparameters is as follows:

epochs = 100 (This value is the number of training rounds).

batch_size = 16 (This value is the batch size, the number of samples used for each training). The training data will be divided into small batches, each containing the specified number of samples.

img_height = 227 img_width = 227 (This value is the height and width of the network inputs).

num_classes = 20 (The value is the learning rate, which controls the size of the step in which the parameters are updated during the optimization process). A smaller learning rate may require more training rounds, while a larger learning rate may result in an unstable training process.

lr = 0.01 (The value is the learning rate decay, which is multiplied by this factor at the end of each round in order to gradually reduce the learning rate during training, thus helping the model to converge better).

lr_decay = 1e-4 (This value is the weight decay, also known as L2 regularization. It is used to control the complexity of the model and avoid overfitting. A larger weight decay limits the size of the weights.

Here is the pseudo-code for the training part

```
# 1. First create the necessary folders
if not os.path.exists(config.example_folder)::
    os.mkdir(config.example_folder)
if not os.path.exists(config.weights):.
    os.mkdir(config.weights)
if not os.path.exists(config.logs):
    os.mkdir(config.logs)
# 2. Define the neural network model
model = Model.get_net()
if torch.cuda.is_available().
    model = model.cuda()
# 3. define the optimizer and loss function
optimizer = optim.SGD(model.parameters(), lr=config.lr, momentum=0.9, weight_decay=config.weight_decay)
criterion = nn.CrossEntropyLoss().cuda()
# 4. Check whether to resume training from checkpoints
start_epoch = 0
current_accuracy = 0
resume = False # If you don't want to load the model, set it to False
if resume.
    checkpoint = torch.load(config.weights + config.model_name + '.pth')
    start_epoch = checkpoint["epoch"]
    model.load_state_dict(checkpoint["state_dict"])
    optimizer.load_state_dict(checkpoint["optimizer"])

# 5. Define data transformations for image preprocessing
transform = transforms.Compose([
    transforms.RandomResizedCrop(90),
    transforms.ColorJitter(0.05, 0.05, 0.05),
    transforms.RandomRotation(30),
    transforms.RandomGrayscale(p=0.5),
    transforms.Resize((config.img_width, config.img_height)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
# 6. Load and preprocess training data
_, train_list = get_files(config.data_folder, config.ratio)
# (code to load and preprocess training data can go here)
# 7. Start the training cycle
for epoch in range(start_epoch, config.epochs)::
# (you can add the code for the training loop here)
pass # Placeholder for training loop
# 8. (Additional code for testing, evaluating and saving checkpoints can be added here)
```

Experimental setup: utilizing PyCharm as the compiler, the model is compiled and data trained in cuda11.7, conda pytorch environment. The specific hardware information is Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz RAM16.0 GB, GPU:NVIDIA GEFORCE RTX 1660ti

The following data are the hyperparameters and accuracy of the first training:

```
epochs = 100
batch_size = 16
img_height = 227
img_width = 227
num_classes = 20
lr = 0.01
lr_decay = 1e-4
weight_decay = 2e-4
ratio = 0.2
type(accTop1) = <class 'numpy.float64'>
Test_epoch: 100 Test_accuracy: 44.12% Test_Loss: 1.587615
```

As you can see the accuracy rate is still very low at this moment, only 44.12%. Then we adjusted to confirm the optimal learning decay rate and the optimized accuracy.

```
epochs = 100
batch_size = 16
img_height = 227
img_width = 227
num_classes = 20
lr = 0.011
lr_decay = 1e-4
weight_decay = 2e-4
ratio = 0.2
type(accTop1) = <class 'numpy.float64'>
Test_epoch: 100 Test_accuracy: 74.62% Test_Loss: 0.823282
```

It can be seen that the accuracy rate has risen dramatically to 74.62 percent. Next, we further search for optimal weight decay and learning rate:

```
epochs = 100
batch_size = 16
img_height = 227
img_width = 227
```

```
num_classes = 19
lr = 0.011
lr_decay = 1e-4
weight_decay = 2e-4
ratio = 0.2
type(accTop1) = <class 'numpy.float64'>
Test_epoch: 100 Test_accuracy: 74.61% Test_Loss: 0.817638
```

It can be seen that the accuracy rate is steadily increasing, but the increase is decreasing. After multiple rounds of tweaking and testing, we were approaching the maximum value:

```
epochs = 100
batch_size = 16
img_height = 227
img_width = 227
num_classes = 19
lr = 0.011
lr_decay = 0.00099
weight_decay = 2e-4
ratio = 0.2
type(accTop1) = <class 'numpy.float64'>
Test_epoch: 100 Test_accuracy: 78.44% Test_Loss: 0.817186
```

The optimal hyperparameter combination was found to be 0.012 learning rate, 0.00011 weight decay, and 19 learning rate:

```
epochs = 100
batch_size = 16
img_height = 227
img_width = 227
num_classes = 19
lr = 0.012
lr_decay = 0.00011
weight_decay = 2e-4
ratio = 0.2
type(accTop1) = <class 'numpy.float64'>
Test_epoch: 100 Test_accuracy: 84.88% Test_Loss: 0.607745
```

It can be seen that the hyperparameters and the accuracy reach the maximum value of the normal distribution, and the accuracy increases by up to 35% compared to the initial value.

3.3. Experimental results

The analysis of the results of the experiment shows that the accuracy of the model can be significantly improved after continuous optimization and adjustment. With the constantly modified hyperparameters, the model is able to better capture the features in the data, which leads to a gradual improvement in its accuracy and performance on the test data.

Fine hyperparameter tuning and model optimization is not an easy task, but its value lies in improving the performance of the model step by step. By adjusting the learning rate, this experiment ensures the convergence speed and stability of the model during training.

In conclusion, this experiment has successfully improved the accuracy of the flower recognition task through several hyper-parameter adjustments and training. This result not only depends on persistence and patience, but also shows the importance of continuous optimization in deep learning tasks. In future research, this systematic approach to hyperparameter exploration and optimization will help improve the performance of more tasks.

4. Discussion and analysis

After suitable hyperparameter tuning, good hyperparameter choices including learning rate, optimizer, and regularization coefficients successfully accelerated the model convergence and improved the performance. After extensive tuning and testing, the model shows satisfactory performance: it achieves the task of basic flower recognition. After several rounds of training, the model achieved an accuracy of 80.88 on the test set, an improvement of almost 30% compared to the previous model. These data indicate that the optimization of the model has achieved some results. This success can be attributed to the use of multiple strategies during model design and training. First, at the beginning of the debugging process, the experiment chose a model architecture that was appropriate for the task and ensured that it was sufficiently expressive. Second, multiple convolutional networks were used to help the model better generalize to different data samples. In addition, the experiment carefully tuned the learning rate and optimizer parameters to ensure that the model converged stably during training.

Although the model was tuned to achieve strong accuracy on the test set, it still has some shortcomings in certain aspects. Firstly, it can be noted that the model performs poorly when dealing with specific categories or complex scenes, possibly due to data imbalance or difficulty in capturing category-specific features. Despite the use of data augmentation techniques to improve robustness during training, the model still struggles to achieve satisfactory accuracy when dealing with extreme variations or noisy data. Also after numerous training sessions it was further found that the model may be too sensitive to certain irrelevant features, which may lead to overfitting problems. The so-called overfitting, which is specifically manifested in the model recognition rate works well on the training set and poorly on the test set, suggesting that the model generalization ability is weak, thus affecting the recognition rate [6]. In order to further improve the model performance, it is still necessary to find more complex model architectures and finer hyper-parameter adjustments, and further optimize the data pre-processing process.

Further optimization of the data preprocessing process. This includes a number of aspects such as finer data cleaning, normalization and feature extraction. By ensuring the quality and consistency of the input data, more informative features can be provided to help the model improve its learning rate and accuracy. Secondly, more levels of representation capabilities can be introduced by exploring more complex model architectures and network layers in the expectation that the model can better capture complex patterns and relationships in the data. In addition, the regularization methods in the model can impose a focus to avoid overfitting problems. Finally, there are issues related to further tuning of the hyperparameters. More experimentation with different learning rates, optimizers and regularization parameters can be done to find the best combination. This helps the model to converge faster and achieve better performance on validation and test sets. Also, in terms of model performance in specific data distributions, category imbalances or noisy environments. The accuracy of the model in these complex situations can also be further improved by exploring methods such as migration learning, small sample learning or adversarial training. In conclusion, this experiment realizes that the model still has room for improvement in many aspects. In-depth research and experiments can be continued to further improve the performance and adaptability of the model to meet the evolving application needs.

5. Conclusion

The research in this paper aims to address the various difficulties and challenges in the field of flower identification, thus making a valuable contribution to several fields such as botany, ecology and agriculture. In the course of the research, the experiment in this paper first selected a suitable deep learning model for this task and optimized the model to some extent. By adjusting the model structure, activation function and regularization method, this experiment successfully improves the performance and generalization ability of the model. The effect of hyperparameters was also studied in depth and the performance and accuracy of the model in various scenarios were further improved by carefully tuning the hyperparameters such as learning rate, optimizer and batch size. This research work is valuable for deep learning in the field of flower recognition. By optimizing the model and accurately adjusting the hyperparameters, a more significant performance improvement was achieved, which will help to distinguish different types of flowers more accurately. This has practical applications for plant classification, ecological monitoring and agricultural production. Despite some successes, the model still has limitations such as data imbalance and generalization ability. Therefore, future work will focus on further optimizing the model structure, introducing more data enhancement strategies, and exploring more advanced deep learning techniques to improve the robustness and accuracy of flower recognition systems in complex environments.

References

[1] Shen, P. & Zhao, B.. (2017). Flower species recognition based on deep learning model. Science and Technology Bulletin (03), 115-119. doi:10.13774/j.cnki.kjtb.2017.03.024.

[2] Wu, Yuan Yuan. (2022). Research on Flower Recognition and Classification Based on Deep Learning (Master's thesis, Jiangxi University of Science and Technology). https://kns.cnki.net/KCMS/detail/detail.aspx?dbname=CMFD202301&filename=1022621022.nh

[3] Zhang, Mengyu. (2021). Flower recognition based on ResNet and attention mechanism. Computers and Modernization (04), 61-67.

[4] Zeng, Fanjing & Lei, Ming. (2019). Research on flower recognition based on convolutional neural network. Computer Knowledge and Technology (11), 185-188. doi:10.14004/j.cnki.ckt.2019.1152.

[5] Song Zilong. (2019). Flower species recognition system based on convolutional neural network. Computer Products and Distribution (12), 91+131.

[6] Liu Jiazheng . (2019). Flower species recognition based on deep migration learning model. Jiangsu Agricultural Science (20), 231-236. doi:10.15889/j.issn.1002-1302.2019.20.054.