WEP Warwick Evans Publishing

International Journal of Computer Science and Information Technology

ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 3, Number 3, Year 2024
DOI: https://doi.org/10.62051/ijcsit.v3n3.30
Journal homepage: https://wepub.org/index.php/IJCSIT/index

Design and Implementation of AlexNet Flower Classification System from the Perspective of Deep Learning

Linqi Zhou

Southwest Minzu University, College of Electronic Information, Chengdu, Sichuan province, China

ABSTRACT

In the field of computer vision, image classification is a fundamental and important task. Traditional image classification methods rely on manually designed feature extractors such as SIFT and HOG, but these methods have limited effect when processing complex image data. With the rise of deep learning technology, especially the rapid development of convolutional neural network (CNN), the accuracy and efficiency of image classification have been significantly improved. AlexNet As a classic model of deep convolutional neural network, it has become a research hotspot in the field of computer vision since it achieved excellent results in the ImageNet large-scale visual recognition competition in 2012. Flower classification is mainly based on botanical and morphological characteristics, which helps scientists and horticulturists to better understand and study different species of flowers. The classification of flowers also helps people to better understand the growth habits, ecological environment and use of flowers, which is of great significance for horticultural cultivation and biodiversity conservation. The experiment mainly uses Alexnet network through the flower image classification, flower five types, through the training set on the flower image feature extraction and model training, and validation in the test set to complete the experiment, the experiment of Alexnet network layer of the corresponding changes, in order to reduce the purpose of calculating complexity, at the same time keep the training accuracy and previous network.

KEYWORDS

Classification of flowers; the Alexnet network; and feature extraction

1. INTRODUCTION

This task aims to use a convolutional neural network model like AlexNet to classify flowers. AlexNet Is a classic deep learning model, commonly used in image recognition tasks. In the flower classification task, we can use the convolutional and fully connected layers of AlexNet to extract image features and train the classifier to identify the flower classification. AlexNet Is a classical convolutional neural network proposed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton in the 2012 ImageNet image classification competition. At that time, AlexNet had achieved excellent results in the ImageNet large-scale visual recognition competition, raising the accuracy rate of the deep learning model in the competition to an unprecedented height. AlexNet Input is a 3 x 224 224 size color image with RGB three channels. AlexNet It contains 5 convolutional layers (including 3 pooling) and 3 fully connected layers. Each convolution layer contains the convolution kernel, bias term, ReLU activation function, and local response normalization (LRN) module. The first, second, and fifth convolutional layers are followed by a maximum pooling layer, and the last three layers are fully connected layers. The final output layer is softmax, which transforms the network output into probability values to obtain a probability distribution of the samples belonging to 1000 categories, used to predict the categories of the images. Some pictures of the specific training set and test set are shown in Figure 1.

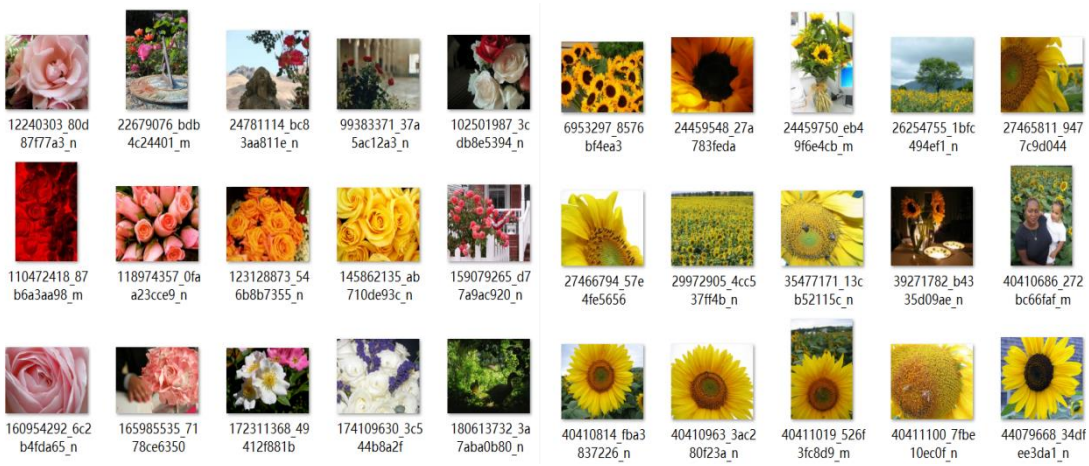

Figure 1. partitioned data set (PDS)

2. ALGORITHM DESIGN IDEAS

1) Data preparation: Collect flower image data set and divide them into training set and test set. The data set used in this experiment is the public flower data set, including pictures of species and flowers, and the test set and training set are divided by oneself;

2) Data preprocessing: preprocessing of the image, including image scaling, normalization and other operations;

3) Model construction: Use the AlexNet model for the flower classification task, and build and train the model by yourself;

4) Model training: Use the training set to train the AlexNet model, and adjust the model parameters to improve the classification accuracy;

5) Model evaluation: use the test set to evaluate the trained model, and calculate the classification accuracy and other indicators;

6) Model application: the trained model is applied to the new flower images for classification and prediction.

3. ALGORITHM IMPLEMENTATION PROCESS

3.1. Data Download and Preprocessing

After downloading the data and saving it locally, we will call the Pandas library, matplotlib library, seaborn library, os library, etc., to make the prediction preparation. The dataset is then divided into the training set and the test set according to the ratio of 7 to 3, and the procedure is shown in Figure 3.1, and the results are shown in Figure 3.2.

A data loader is then written for use to load the training dataset. Where train_dataset is a dataset object, batch_size is the size of each batch, shuffle=True indicates randomly shuffled data at the beginning of each epoch, and num_workers=0 indicates using the master process to load the data. Set the batch size to 128 here, and the process is shown in Figure 3.3.

```python
from matplotlib import pyplot as plt
import torch.utils.data as Data
from PIL import Image
import os
from torch import nn
import torch.optim as optim
from torch.nn import init
import torch.nn.functional as F
import time
import torchvision
from torchvision import transforms, datasets
from shutil import copy, rmtree
import json

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("using {} device.".format(device))

data_transform = {"train": transforms.Compose([transforms.Resize((64, 64)),
                                               transforms.RandomHorizontalFlip(),
                                               transforms.ToTensor(),
                                               transforms.Normalize((0.5, 0.5, 0.5),
                                                                    (0.5, 0.5, 0.5))]),
                  "test": transforms.Compose([transforms.Resize((64, 64)),
                                              transforms.ToTensor(),
                                              transforms.Normalize((0.5, 0.5, 0.5),
                                                                   (0.5, 0.5, 0.5))])}

data_root = os.getcwd()
image_path = os.path.join(data_root, "D:/pycode/data2")
print(image_path)

train_dataset = datasets.ImageFolder(root=os.path.join(image_path, "train"),
                                     transform=data_transform["train"])

train_num = len(train_dataset)
print(train_num)
```

Figure 3.1. Library calling and partition of the datasets

```
D:\py\Anaconda3\envs\123\python.exe D:/pycode/123/Alexnet.py
using cpu device.
D:/pycode/data2
2568
1100
using 2568 images for training, 1100 images for validation .
```

Figure 3.2. The first three data sets are shown

```python
batch_size = 128

train_loader = torch.utils.data.DataLoader(train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=0)

test_dataset = datasets.ImageFolder(root=os.path.join(image_path, "test"),
                                    transform=data_transform["test"])

test_num = len(test_dataset)
print(test_num)  # val_num = 364
test_loader = torch.utils.data.DataLoader(test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False,
                                          num_workers=0)
```

Figure 3.3. Data loader

3.2. Model Building and Training

This task selection using Alexnet network model training, AlexNet a innovation is using the Relu activation function, replace the often used S function and T function, another innovation is LRN (Local Response Normalization) local response normalization, LRN simulation neurobiological a function called lateral inhibition (lateral inhibitio), lateral inhibition refers to the activated neurons will inhibit adjacent neurons. LRN local response normalization borrows the idea of side inhibition, which makes the large response relatively larger and improves the generalization ability of the model. LRN only normalizes the adjacent regions of the data and does not change the size and dimension of the data. The LRN concept was first proposed in the AlexNet model and has been applied in GoogLenet, but the actual role of LRN is controversial. Its model information is shown in Figure 3.4.

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000, init_weights=False):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(  # 输入64×64×3
            nn.Conv2d(3, 48, kernel_size=3, stride=1, padding=1),  # 64,64,48
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 32,32,48

            nn.Conv2d(48, 128, kernel_size=3, padding=1),  # 32,32,128
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 16,16,128

            nn.Conv2d(128, 256, kernel_size=3, padding=1),  # 16. 16. 192
            nn.ReLU(inplace=True),

            nn.Conv2d(256, 256, kernel_size=3, stride=2, padding=1),  # 8,8,192
            nn.ReLU(inplace=True),

            nn.Conv2d(256, 128, kernel_size=3, padding=1),  # 8,8,128
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 4,4,128
        )
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.5),
            nn.Linear(128 * 4 * 4, 2048),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(2048, 2048),
            nn.ReLU(inplace=True),
            nn.Linear(2048, num_classes),
        )
        if init_weights:
            self._initialize_weights()
    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, start_dim=1)
        x = self.classifier(x)
        return x
```

Figure 3.4. Model building

Next model initialization, set the number of training rounds is 50 rounds, classified into five categories, learning rate of 0.001, the training process using Adam optimizer, Adam optimizer is an optimizer based on gradient descent algorithm, it combines the momentum optimization and adaptive learning rate, can be in the training neural network fast convergence and have good generalization ability. Features of the Adam optimizer include the adaptive learning rate, momentum optimization, and second-order moment estimation. It is able to adaptively adjust the learning rate of each parameter to better adapt to the gradient changes of different parameters during training. At the same time, the Adam optimizer also has the characteristics of momentum optimization, which can accelerate the convergence rate and reduce the oscillation. Moreover, the Adam optimizer also uses the second moment estimation to estimate the change of the gradient, thus better adapting to the parameter changes. Details are shown in Figure 3.5.

```
num_classes = 5
lr = 0.001
epochs = 50


net = AlexNet().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(net.parameters(), lr=lr)
```

Figure 3.5. Train

Then the model training is conducted. The training process is divided into two parts, the first part is the training set training process, the second part is the test set test part, the process is shown in Figure 3.6 and 3.7.

```
def train_epoch(net, data_loader, device):
    net.train()
    train_batch_num = len(data_loader)
    total_1oss = 0
    correct = 0
    sample_num = 0

    for batch_idx, (data, target) in enumerate(data_loader):
        t1 = time.time()

        data = data.to(device).float()

        target = target.to(device).long()

        optimizer.zero_grad()

        output = net(data)

        loss = criterion(output, target.squeeze())

        loss.backward()
        optimizer.step()

        total_1oss += loss.item()

        prediction = torch.argmax(output, 1)

        correct += (prediction == target).sum().item()

        sample_num += len(prediction)
        # if batch_idx//5 ==0:
        t2 = time.time()
        print("processing:{}/{},      {}s".
              format(batch_idx + 1, len(data_loader), t2 - t1))

    loss = total_1oss / train_batch_num
    acc = correct / sample_num
    return loss, acc
```

Figure 3.6. Training set training

```python
def test_epoch(net, data_loader, device):
    net.eval()
    test_batch_num = len(data_loader)
    total_loss = 0
    correct = 0
    sample_num = 0

    with torch.no_grad():
        for batch_idx, (data, target) in enumerate(data_loader):
            data = data.to(device).float()
            target = target.to(device).long()
            output = net(data)
            loss = criterion(output, target)
            total_loss += loss.item()
            prediction = torch.argmax(output, 1)
            correct += (prediction == target).sum().item()
            sample_num += len(prediction)
    loss = total_loss / test_batch_num
    acc = correct / sample_num
    return loss, acc
```

Figure 3.7. Test set test

Because the loss function shows overfitting and underfitting during the training, the loss judgment is optimized during the training. If the last loss function has little difference from the loss function of this round, that is, the training is stopped, and the effect can be achieved in 50 rounds of training for this task. In the process of constantly adjusting the hyperparameters, it was found that the parameters that also showed good accuracy in the training set and test set were set to the learning rate 0.001 and the step size 128, and the accuracy was repeated after 50 rounds, so the best effect was achieved in 50 rounds. The visualization maps are shown in 3.8 and 3.9.

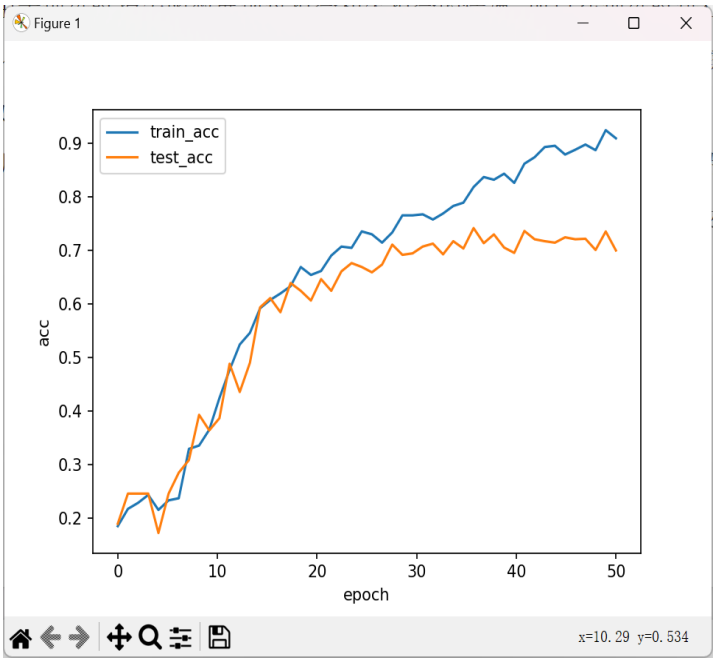

Figure 3.8. Accuracy contrast

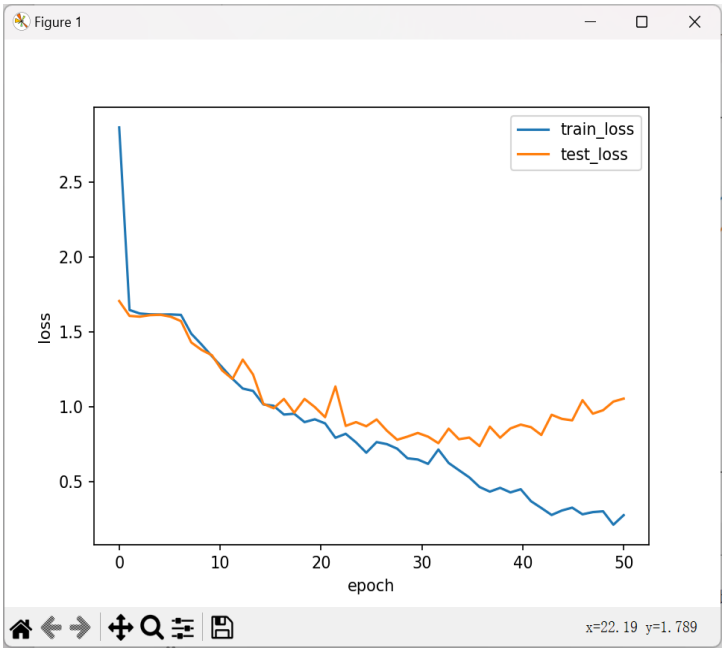

Figure 3.9. Loss rate comparison

4. RESULTS AND ANALYSIS

The task in the third part of the training results show that the model training effect did not reach the best performance, in the training set and test concentration show accuracy and loss function difference is obvious, the model generalization ability is not particularly good, need again on the basis of the improvement of the model and performance, experimental results basic classification requirements, but from the image, the model appeared overfitting phenomenon, need to solve the overfitting phenomenon to further improve the model overall performance. For the specific task of flower classification, if the data set is large and diverse, the accuracy of the model in the validation set or test set can reach a relatively high level (such as 80% -90%), but the specific value will depend on the specific characteristics of the data set and the degree of optimization of the model

Based on the Alexnet network, the flower classification task was carried out. After training and classification through the Alexnet network, the model can identify the given flower species, so as to achieve the purpose of classification.To train and improve the AlexNet model, this paper uses a dataset specifically used for flower classification. The dataset contains a variety of different kinds of flower images, and it is annotated and classified in detail. In the data preprocessing stage, the images are normalized, trimmed, and flipped to increase the diversity of the training data and improve the generalization ability of the model.

In this paper, we study the flower classification task based on AlexNet network, and improve the model performance on the flower classification task by improving the model structure and optimizing the training process. The experimental results show that deep learning technology has great applications in the field of flower classification. Future work can further explore more complex network structure, more efficient optimization algorithms, and richer data sets to improve the performance and application range of models.

REFERENCES

[1] Li Wanhu, Wu Lili. Image recognition of TCM based on the modified AlexNet [J]. Software Engineering, 2023, 26(09):38-41. DOI:10.19644/j.cnki.issn2096-1472.2023.009.007.

[2] Baoyalin, Tange. Research on embroidery image classification based on AlexNet deep learning [J]. Wool Textile Technology, 2023, 51(06):81-87. DOI:10.19333/j.mfkj. 20230204307.

[3] Liu Jinghe, Ren Yanbiao, Xue Yan, etc. Floral recognition study based on ResNet152 [J]. Computer Knowledge and Technology, 2023, 19(15):15-17. DOI:10.14004/j.cnki.ckt. 2023.0769.

[4] Liu Manman. A AlexNet vegetable identification algorithm [D] based on Gibbs sampling and residue structure. Zhejiang Sci-Tech University, 2023. DOI:10.27786/d.cnki.gzjlg. 2023.000944.

[5] D.Deepa, R.Yaswanth, Kumar K. Tomato leaf diseases classification using alexnet[C]//Eliwise Academy. Proceedings of the 4th International Conference on Computing and Data Science (Part 2). Department of Computer Science and Engineering College; 2022:8. DOI:10.26914/c.cnkihy.2022.084559.

[6] Duan Ao, Li Li, Yang Xu. Image recognition and classification algorithm based on AlexNet [J]. Journal of Tianjin Vocational and Technical Normal University, 2022, 32(01):63-66. DOI:10.19573/j.issn2095-0926.202201011.