International Journal of Computer Science and Information Technology
ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 3, Number 2, Year 2024
DOI: https://doi.org/10.62051/ijcsit.v3n2.17
Journal homepage: https://wepub.org/index.php/IJCSIT/index

Classifying Handwritten Numbers Using Convolutional Neural Networks

Sizhe Fan

Beijing University of Technology, China
Fansiz20040115@163.com

ABSTRACT

A convolutional neural network (CNN) is one of the most significant networks in the deep learning field. Since CNN has made impressive achievements in many areas, including but not limited to computer vision and natural language processing, CNNs have attracted much attention from both industry and academia in the past few years. The problem of gradient vanishing or gradient explosion tends to get worse as the depth of the model increases. In traditional neural network structures, especially in the field of image processing, since a large number of convolutional and pooling layers need to be utilized to extract features layer by layer, the model performance tends to degrade and other unfavorable situations as the number of layers accumulates. In order to solve the gradient problem that occurs during the training process of deep neural networks, the concept of residual connectivity has emerged.

KEYWORDS

MNIST; CNN; ResNet18

1. INTRODUCTION

In a traditional convolutional neural network (CNN) [5], the input data is passed on layer by layer through a series of layers of linear and nonlinear transformations. However, when the network has too many layers, the gradient may gradually decrease or even disappear during back propagation [4], causing network training to become exceptionally difficult. The depths of DL models for fault diagnosis are shallow compared with convolutional neural networks in other areas (including ImageNet), which limits their final prediction accuracies [2]. The core idea of residual connection is to allow the network to pass information directly between different layers, so that the gradient can flow back to the previous layer more smoothly, thus effectively alleviating the problem of vanishing gradient.

The main task of this experiment is to process the huge dataset, the core task of the dataset is to recognize the characters in the image. For this purpose, I need to use the technique of deep learning to complete operations such as data reading and data augmentation in order to provide sufficient data support for the subsequent character recognition task.

2. RELATED WORK

Residual Neural Network revolutionizes deep learning by introducing residual blocks. These blocks allow gradients to flow more effectively through the network, enabling deeper architectures. ResNets significantly improve performance in tasks like image recognition, outperforming prior models. This

learning process can be efficiently done by stochastic gradient descent, and the generalization error is also small if poly nomially many training examples are given [1].

In the practice of machine learning, we usually divide the original dataset into three parts: training set, validation set and test set. First, we will divide the dataset into training set and validation set. Since the model construction process needs to constantly check whether the model configuration and the degree of training are appropriate (whether it is overfitting or underfitting), we will further subdivide the training data into two parts: one part is used for actual model training, and the other is used to validate the model's performance.

The training set allows us to obtain a preliminary neural network model. Subsequently, the validation set is used to evaluate the effectiveness of the model and select the best-performing model. The validation set can be reused during the model construction process and is mainly used to assist us in optimizing the model. Finally, when the model performs well on the validation set, we use the test set to finally evaluate the performance of the model, including key metrics such as accuracy and error. The test set provides us with an objective evaluation standard for judging network performance.

2.1. Convolutional Neural Network

This feedforward neural network, which is inspired by the biological sensory mechanism, exhibits translational invariance. It utilizes a convolutional kernel to maximize the extraction and utilization of local information, thus effectively preserving planar structural information. This network structure is heavily inspired by the human visual system. Within the human visual cortex, a large number of neurons are distributed, which exhibit a high degree of limitation and hierarchy in processing visual information. Once our gaze touches an object, a series of neurons are then activated, and each layer of neurons is able to capture specific features, such as lines and edges. Higher levels are able to recognize more complex and varied features and thus identify the object we are looking at.

2.2. Input Layer

The input layer generally represents a matrix of pixels for a picture. An image can be represented by a 3D matrix. The length and width of the 3D matrix represent the size of the image, while the depth of the 3D matrix represents the color channel of the image.

2.3. Convolutional Layers

The convolutional layer, as the core component of the convolutional neural network, performs convolutional operations to effectively extract various types of features in the input image. This process generates a series of feature maps, which not only reveal basic structural information such as edges, lines and corners of the picture, but also gradually capture richer and more abstract semantic features as the network layers deepen. In the field of signal processing, the convolution operation is essentially the process of multiplying and summing two variables (one of which is subject to flip and shift operations) over a specific range. It is worth mentioning that each neuron in the convolutional layer is connected to only a portion of the input data, and this local connectivity mechanism allows the convolutional layer to focus on local features of the input data. More critically, each filter in the convolutional layer is shared, i.e., the same filter is used over the entire range of the input data, and this weight-sharing strategy greatly reduces the number of network parameters, which in turn reduces the risk of overfitting. Compared with the fully connected layer, the neurons of the convolutional layer are only partially connected to the corresponding input data, and thus feature sparse connections. This design not only reduces the amount of computation, but also improves the efficiency of feature extraction. Therefore, the convolutional layer is undoubtedly the key layer in building a convolutional neural network, and most of the computation in the network is concentrated in this layer.

2.4. Pooling Layer

The Pooling Layer in Convolutional Neural Networks (CNNs) is a key step in reducing the spatial dimensions of the data, such as reducing the height, width, and the number of channels, which not only simplifies the model structure, but also reduces the amount of computation, thus enhancing the model's generalization ability. In practice, the pooling layer first divides the feature map into multiple regions, and then performs statistical analyses on each region, which can be the maximum (i.e., maximum pooling) or the average (i.e., average pooling) within the region to obtain a new feature map. In addition to reducing the amount of computation, the pooling layer makes the model more robust to image transformations, and the features extracted by the pooling layer remain stable even if the image undergoes rotations, translations, or scale changes. At the same time, this also reduces the risk of model overfitting. Therefore, the pooling layer plays a crucial role in convolutional neural networks, which not only simplifies the model and reduces the computational complexity, but also improves the generalization ability and robustness of the model. In the operation process, max pooling takes the maximum score in the pooling window, and average pooling selects the average score of the pooling window [3]. This design makes the convolutional neural network more efficient and reliable in processing image data.

2.5. Residual Layer

The residual layer consists of two main parts: one is an ordinary convolutional or fully connected layer for extracting features, and the other is a constant mapping layer, that is, a layer where the input is equal to the output, which is used to skip some unnecessary computations. The results of these two parts are added together to get the output of the residual layer. The advantage of this design is that when the network is deep enough, performance degradation can be avoided by skipping certain layers even if they are not well learnt. At the same time, the design of the residual layer makes the training of deep networks more stable and efficient.

2.6. Fully Connected Layer

A fully connected layer, a key component of a neural network, is characterized by the fact that each neuron within the layer forms a connection to all neurons in the previous layer, and each connection is assigned a unique learning weight. This all-connected structure allows the layer to deeply mine and learn the feature representation of the input data. In neural networks, high-level abstract features of the input data can be gradually extracted by stacking multiple fully connected layers. These features not only contain the raw information of the data, but also incorporate the experience and knowledge learned by the network during the training process. This improved abstract representation capability is crucial for the network to complete complex tasks such as classification and regression. Another significant advantage of the fully connected layer is its integration capability. Since each neuron is connected to all the neurons in the previous layer, it is able to synthesize the various information learned in the previous layers to form a comprehensive and in-depth understanding of the input data. This integration mechanism allows the fully connected layer to play a key role in the decision-making process of the neural network, contributing to the accuracy and generalization of the network. In conclusion, the fully connected layer provides a powerful representation learning foundation for neural networks through its fully-connected structure and powerful feature learning and integration capabilities, enabling the network to handle a variety of complex tasks.

This experiment involves several Python libraries that play a key role in machine learning and deep learning applications. Below is a brief description of these libraries:

Os library:

Os is one of Python's standard libraries and provides functionality for interacting with the operating system.

In this experiment, the os library may be used for file path manipulation, directory traversal, environment variable management, etc., to help the experimenter better organize and manipulate data.

NumPy:

NumPy (Numerical Python) is a powerful numerical computation library for Python.

It provides high-performance multi-dimensional array objects, as well as tools for working with these arrays.

In deep learning, NumPy is commonly used for data preprocessing, transformation and storage.

PyTorch:

PyTorch is an open-source deep learning framework that allows researchers and designers to build and train neural networks using GPU-accelerated tensor computation.

3. EXPERIMENT

In this experiment, PyTorch is used for tensor generation, operations, slicing, and connectivity, providing a rich set of neural network layers and activation functions, as well as efficient automatic differentiation.

The torch.nn module contains the various layers and functions needed to build neural networks.

The ImageFolder class in the torchvision library is a handy data loader for automatically loading image data from folders and organizing it by category.

DataLoader is used to load data in bulk and provides functions such as multi-threaded loading and data disruption.

TensorBoard:

TensorBoard is a visualization tool for TensorFlow but can also be used with PyTorch to visualize various metrics during neural network training, such as changes in the loss function, changes in accuracy, and so on.

With torch.utils.tensorboard, information from the training process can be recorded and displayed in TensorBoard, helping the experimenter to better understand the model training process.

During the experiment, these libraries will work together to enable the processing of training sets, model construction, training and visualization. By organizing and calling functions and classes in these libraries wisely, experimenters can efficiently build and train deep learning models, and monitor and analyze model performance.

In the code planning, the device cuda is first defined to the CPU.

After completing the training set code, we next need to define the hyperparameters. In this experiment, we set the number of training rounds (epochs) to 30 and the number of samples per batch (batch_size) to 128. The selection of hyperparameters is a crucial part of neural network training, as they directly determine the learning rate of the model, the number of iterations, and the number of samples involved in each weight update. Although choosing hyperparameters is a complex and challenging task, this experiment uses suitable values that have been calculated to ensure that the model can have a good starting point. Specifically, we set the number of training rounds to 30, meaning that the entire dataset will be used to train the network 30 times. This value was chosen to ensure that the model has enough chances to learn the features in the data while avoiding too long training time. The number of samples per batch, on the other hand, is set to 128 to maintain the stability of the training process while making full use of the computational resources. It should be noted that although this experiment does not further optimize the hyperparameters, in practice, we usually use more advanced techniques such as cross-validation, grid search or random search to find the best combination of hyperparameters. These

techniques can help us explore the hyperparameter space more systematically and thus find parameter settings that can further improve the model performance.

In conclusion, by setting the hyperparameters reasonably, we can lay a solid foundation for the training of the neural network and provide strong support for the subsequent model optimization.

ResNet18 construction:

First of all, you need to create a class named ResNet18 , which is inherited from PyTorch's base class nn.Module and is specifically used to build neural network models.ResNet18 is a simplified version of ResNet (residual network) with a depth of 18 layers. Its core feature is the introduction of residual connectivity, an innovation that effectively solves the problem of vanishing or exploding gradients that deep neural networks may encounter during training. In the initialization method of ResNet18, the first layer of the network, conv1, is defined, containing the convolution operation, batch normalization, and the ReLU activation function, which converts the number of channels of the input image from 3 (RGB) to 64. Three blocks, block1, block2, and block3, are subsequently defined, which consist of several Basic_blocks, each of which Basic_block contains two convolutional layers, each followed by batch normalization and ReLU activation. block2 and block3 start with Basic_Block1 that may change the number of channels in the feature map. in addition, ResNet18 includes an average pooling layer avg_POOL, which is used to reduce the spatial dimensions of the feature map. and a fully connected layer fc, which converts the final feature maps into prediction scores for 10 categories. In the forward propagation method forward, the complete flow of data from input to output is defined, including convolution through the layers, activation, feature extraction, pooling, and the final classification score output. It is important to note that the specific implementation of Basic_block and Basic_Block1 is not given in the code snippet, they are the key classes for implementing the residual block structure, and the details of their implementation will directly affect the structure and behavior of the ResNet18 model. The code picture shown in Figure 1 is like this:

```
class ResNet18(nn.Module):
    def __init__():
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d, nn.BatchNorm2d, nn.ReLU
        )
        self.block1 = nn.Sequential(
            Basic_block, Basic_block, Basic_block
        )
        self.block2 = nn.Sequential(
            Basic_Block1, Basic_block, Basic_block
        )
        self.block3 = nn.Sequential(
            Basic_Block1, Basic_block, Basic_block
        )
        self.avg_POOL = nn.AvgPool2d
        self.fc = nn.Linear

    def forward(x):
        y = self.conv1(x)
        y = self.block1(y)
        y = self.block2(y)
        y = self.block3(y)
        y = self.avg_POOL(y)
        y = y.view(-1, num_features)
        y = self.fc(y)
```

Figure 1. Code of Resnet18

```
# 初始化损失列表
train_lossList, test_lossList = [], []

# 训练循环
for epoch in range(epoches):
    net.train()  # 设置训练模式
    correct, total = 0.0, 0.0

    for i, (inputs, labels) in enumerate(train_loader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (labels == predicted).sum().item()

        writer.add_scalar("loss", loss, i + 1 + epoch * len(train_loader))

    print(epoch + 1, loss.item(), 100 * (correct / total))
```

Figure 2. code of training

Training set:

Next the code that requires the training set enters the training loop, for each training cycle (epoch) it first prints the number of the current epoch and then sets the neural network model into training mode. In each epoch, the code further traverses each batch in the training data loader train_loader, including the input data and labels. In each batch, it performs a series of operations, including obtaining the batch length, moving the data to the specified device, clearing the optimizer gradient, performing forward propagation, calculating the loss, backpropagating, updating the model weights, calculating the accuracy, and recording the loss value using writer. Finally, the code prints the current epoch, number of iterations, loss value, and accuracy. The main purpose of this code is to train the neural network model by iterating the entire training dataset multiple times and updating the model weights after each batch, as well as monitoring the loss and accuracy during training. The code picture shown in Figure 2 is like this. The loss function of the final training code is shown in Figure 3, and it can be

seen that with the gradual deepening of the number of iterations, the train_loss decreases, proving the effectiveness of our proposed neural network scheme.

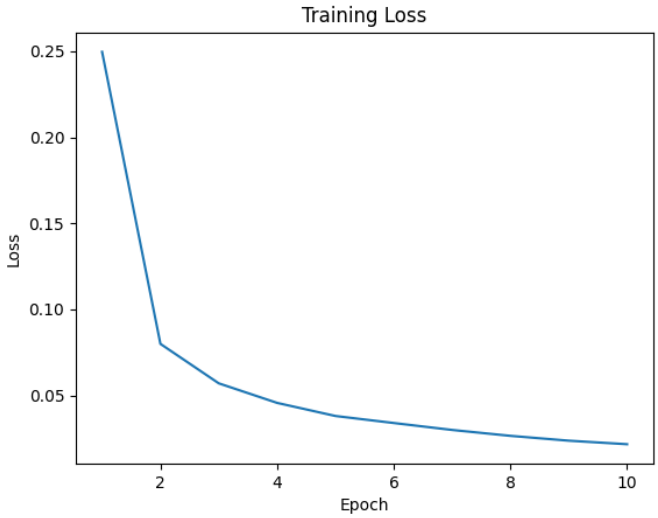

Figure 3. Training loss

Test Sets:

The code first outputs "Waiting Test!" to prompt the start of the test. Then, without calculating the gradient, iterates through each batch in the test data loader, moves the images and labels to the specified device, and gets the predicted output from the model. Then, the number of correctly predicted samples and the total number of samples are counted, and the test accuracy is calculated and output. At the same time, the test accuracy is recorded using WRITER for subsequent analysis. After that, update the learning rate according to the training progress. Finally, output "Training complete!" to indicate the end of the training process, and close the writer. through this code, you can evaluate the performance of the model on the test set and adjust the learning rate accordingly to optimize the training effect.

4. CONCLUSION

In this study, we designed and implemented a two-layer Convolutional Neural Network (CNN) using the PyTorch deep learning framework, where the first convolutional layer uses 5x5 convolutional kernels, while the second layer deploys 3x3 convolutional kernels. In order to reduce the dimensionality of the image data, we introduced a pooling layer in the network. For the MNIST handwritten digit dataset, we meticulously tuned the hyperparameters and applied a backpropagation algorithm as well as a loss function to train the model. After series of training and testing processes, the model exhibits a high accuracy rate, which is stable between 98% and 99%, proving its effectiveness and stability on the handwritten digit classification task.

REFERENCE

[1] Lyu, He, et al. "Advances in neural information processing systems." Advances in neural information processing systems 32 (2019).

[2] Maren, Alianna J., Craig T. Harston, and Robert M. Pap. Handbook of neural computing applications. Academic Press, 2014.

[3] Li, Zewen, et al. "A survey of convolutional neural networks: analysis, applications, and prospects." IEEE transactions on neural networks and learning systems 33.12 (2021): 6999-7019.

[4] Gu, Jiuxiang, et al. "Recent advances in convolutional neural networks." Pattern recognition 77 (2018): 354-377.

[5] O'shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." arxiv preprint arxiv:1511.08458 (2015).