An Analysis of Convolutional Neural Network

|  |
| --- |
| Lan Yiyuan(55614744),Sun Jingyao (55880040), Zou Shuang (55911609),  Li Yixin (55975771), Du Kejun (56075096) |
| *Department of Computing Sciences, City University of Hong Kong, Hong Kong, China* |

***Abstract*—Convolutional neural network (CNN) is an artificial neural network that widely used in object recognition and image classification. It is popular by its features of simple structure, few training parameters and great adaptability. This paper will use Colab to train and evaluate the performances of three CNN models (AlexNet, ResNet and VGG16), and do the analyze for the results.**

I. Introduction

Convolutional neural network (CNN) is an artificial neural network based on human visual cognitive model. Through its hierarchical feature extraction method, users can obtain the features of various layers from the low level to the high level. CNN is the foundation of deep learning, it plays an important role especially in the field of computer vision, such as image classification, object detection, instance segmentation and scene capture. From the LeNet proposed in the 1990s to the AlexNet proposed in paper “ImageNet Classification with Deep Convolutional Neural Networks” in 2012, CNN had gone through many developments. From VGG, GoogLeNet to ResNet and DenseNet, the layers of convolutional neural network become deeper, and the architecture becomes more complex. The solution to the gradient problem is becoming more ingenious, and the performance of CNN model is getting better. The main purpose of this project is to train and evaluate the performance of CNN model based on Colab.

We choose three classic CNN model in this project and let each of them trained two dataset to make a comparison. The model we choose is AlexNet, VGG16 and ResNet.

In the following report, the two datasets we trained will be introduced in section II, the comparison will be made in section III and the evolution of them will be done at section IV. At last, we make a conclusion at section V.

II. The Introduction of Dataset

## *A. MNIST Dataset*

MNIST Dataset is a classic data set in the field of machine learning. This dataset was first proposed by Yan Lecun in the paper Gradient-Based Learning Applied to Document Recognition (the classic CNN network model LeNet-5 was also proposed in this paper). The MNIST dataset consists of 60,000 training samples and 10,000 test samples, each of which is a 28×28 pixel grayscale handwritten digital images. These handwritten digitals have been dimensioned and be located in the center of the image. It's worth mentioning that the MNIST dataset is stored in bytes, so the entire dataset needs to be loaded into the numpy array for training and testing.

## *B. CIFAR-10 Dataset*

CIFAR-10 is a widely used data set which are labeled subsets of 80 million tiny images dataset. It consists 50000 training images and 10000 test images. There are ten classes in the dataset, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck; each class contains 6000 images. There is no overlap in the data set.

To compare with the MNIST data set, CIFAR-10 data set has the following differences: first, the CIFAR-10 data set consisted by three-channel color RGB images, while MNIST data set consisted grayscale images. In addition, the size of images in CIFAR-10 data set is 32 × 32, which is slightly larger than images in the MNIST data set. Finally, unlike the handwritten digital, CIFAR-10 samples contains objects in the real world. This means that the noisy of data set would be larger, and the objects have different proportions and features, which makes it difficult to identify them.

III. The Training Model and The Result

In this section, we use AlexNet, VGG16 and ResNet to train the MNIST and CIFAR-10 respectively, and the training times were both over 100.

## *A. AlexNet*

*1). Introduction of AlexNet*

AlexNet deepens the network structure and learns richer and higher-dimensional image features on the basis of LeNet. We select AlexNet for this project, because it is the most widely researched CNN and is a proper trade-off between speed and accuracy.

In contrast to other respondents using standard characteristics and classifier training techniques, AlexNet [1] used neural networks, particularly convolution neural networks. The model comprises of 3 fully connected layers and 5 convolutional layers. The first layer of AlexNet is used for input of a filtered image with a dimension of 227 × 227 × 3 respectively for width, height, and depth (red, green, blue). The last fully connected layer connects 1000 connected layers and the rest of the layers’ work as a feature extractor. For each input image, AlexNet can produce a 4096-dimensional feature vector that includes the hidden layer activations instantly before the output layer. AlexNet itself is a huge structure containing 650,000 neurons and 60 million parameters. Figure 1 illustrates the architecture of AlexNet [2].

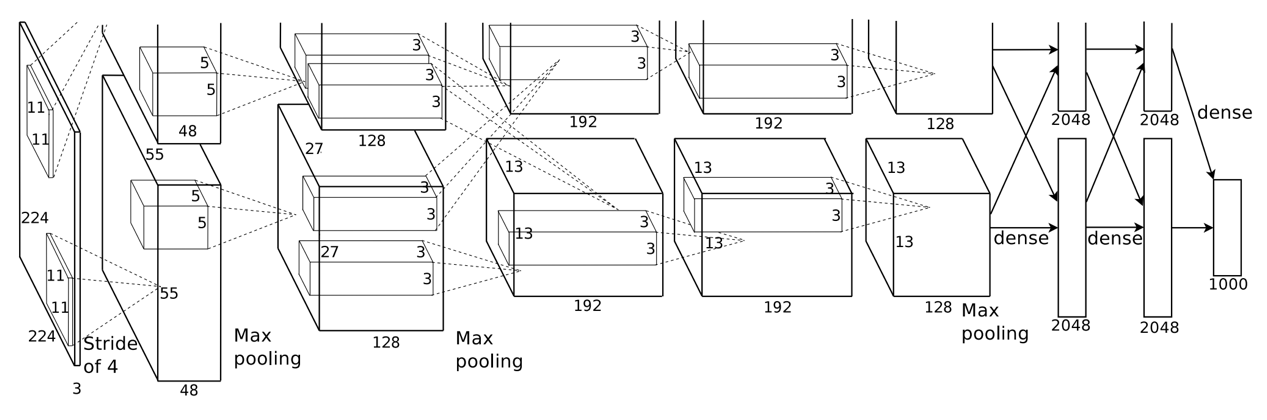


Fig. 1. The architecture of AlexNet

The model was trained on approximately 1.2 million training pictures and performed testing on 150,000 ImageNet data sets test pictures [3]. This model is very efficient for reducing the overfitting problem with the help of maintaining dropout and data augmentation.

*2). The Result of Two Dataset with AlexNet*

The results of training the MNIST and CIFAR-10 dataset and the curves of Train Loss of which trained by AlexNet is shown in figure 2. The training accuracy of MNISR is 0.98828, the result of CIFAR-10 is 79.190%. [7][8]

地图的截图

描述已自动生成

Fig. 2. The arccuracy in MNIST and CIFAR-10 with AlexNet

## *B. VGG16*

*1). Introduction of VGG16*

VGGNet divides the network into 5 segments, each segment connects multiple 3 × 3 convolutional networks in series, each segment is followed by a maximum pooling layer, and finally there are 3 fully connected layers and a softmax layer.

The model explores the relationship between the depth of the convolutional neural network and its performance. By repeatedly stacking 3 × 3 small convolution kernels and 2 × 2 maximum pooling layers, 16 ~ 19 deep convolutions are successfully constructed. Product neural network. VGGNet all uses 3 × 3 convolution kernels and 2 × 2 pooling kernels to improve performance by continuously deepening the network structure. The increase in the number of network layers will not bring about an explosion in the amount of parameters, because the amount of parameters is mainly concentrated in the last three fully connected layers.

According to the size of the convolution kernel and the number of convolution layers, VGG can be divided into 6 ConvNet Configurations: A, A-LRN, B, C, D, E Among them, D and E are more commonly used and are called VGG16 and VGG19, respectively.

The specific structure diagram of VGG 16 is shown in figure 3.

一些文字和图案

描述已自动生成

Fig. 3. Specific structure diagram of VGG16.[4]

The outstanding feature of VGG16 is simplicity, which is reflected in: 1. The convolutional layers all use the same convolution kernel parameters. 2. The pooling layer adopts the same pooling nuclear parameters. 3. The model is composed of a stack of convolutional layers and pooling layers, and it is relatively easy to form a deep network structure. Based on the above analysis, the advantages of VGG can be summarized as: Small filters, Deeper networks.

*2). The Result of Two Dataset with VGG16*

The results of training the MNIST and CIFAR-10 dataset and the curves of Train Loss of which trained by VGG16 is shown in figure 4. The training accuracy of MNISR is 0.93359375, the result of CIFAR-10 is 89.790%.[7][9][10]

地图的截图

描述已自动生成

Fig. 4. The arccuracy in MNIST and CIFAR-10 with VGG16

## *C. ResNet*

*1). Introduction of ResNet*

The third model we used to train the two data set is ResNet. The Residual Network (ResNet) referred to the VGG19 and make some modifications on it. ResNet also added residual unit through the short circuit mechanism. The structure of a building block of residual learning is shown in figure 5. For an accumulation layer structure, when the input is x, the characteristics it learns are recorded as H(x). Now we hope that it can learn residual F(x) = H(x) - x, so that the original learning characteristic is F(x) + x. This is because residuals are easier to learn than primitive features. When the residual is 0, the accumulation layer only does identity mapping, and the network performance will not decline. In fact, the residual will not be 0, which will also make the accumulation layer learn new features based on the input characteristics, so as to have better performance. [5]

手机屏幕截图

描述已自动生成

Fig. 5. Residual learning: a building block [5]

The changes between VGG and ResNet are mainly reflected in the fact that ResNet directly uses the convolution of stride=2 to sample and replaces the full connection layer with the global average pool layer. An important design principle of ResNet is that when the feature map size is reduced by half, the number of feature maps is doubled, which keeps the complexity of the network layer. [5]

*2). The Result of Two Dataset with ResNet*

The results of training the MNIST and CIFAR-10 dataset and the curves of Train Loss of which trained by VGG16 is shown in figure 6. The training accuracy of MNISR is 0.906250, the result of CIFAR-10 is 79.460%.[7]

地图的截图

描述已自动生成

Fig. 6. The arccuracy in MNIST and CIFAR-10 with ResNet

IV. Evaluation

In this section, we attempt to compare and analyze the results for each of the CNN to identify which one has better accuracy in classification and identification. The result is shown in table I.

TABLE I. The Accuracy of Each Model With the Two Dataset

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy in MNIST** | **Accuracy in CIFAR-10** |
| AlexNet | 97.82% | 74.74% |
| ResNet | 93.35% | 79.46% |
| VGGNet | 90.62% | 89.79% |

As shown in the table, when the CIFAR-10 data set was used for testing, the best accuracy performance of VGGNet was the highest, reaching 89.790%. The highest accuracy demonstrates VGG-16 has better classification ability and eliminating unrelated information.VGG-16 makes improvement at the expense of longer training time. The possible reason for this result might be that AlexNet retains more irrelevant information in the final convolutional layer than VGG-16, which interferes with the final prediction. In addition, the overmuch layers in ResNet lead to functional complexity, which result in over-fitting problems and ultimately accuracy reduction.[6]

V. Conclusion

The aim of this work was to use Colab to train and evaluate the performances of three CNN models (AlexNet, ResNet and VGG16) to identify which one has better accuracy in classification and identification. From this work, we conclude that VGG-16 by far presents the best accuracy rate, as compared to the AlexNet and ResNet. Moreover, the train accuracy and the number of epoch maybe affected by the size of data set. Based on this result, VGG-16 should be the best choice to do the image classification and identification.

Reference

1. Fazla Rabbi Mashrur, Amit Dutta Roy, Dabasish Kumar Saha, “Automatic Identification of Arrhythmia from ECG Using AlexNet Convolutional Neural Network”, *2019 4th International Conference on Electrical Information and Communication Technology (EICT)*, 20-22 Dec. 2019.
2. A. Krizhevsky, I. Sutskever, G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, *International Conference on Neural Information Processing Systems*, vol. 2, pp. 1097-1105, 2012.
3. S. Gu, L. Ding, Y. Yang, X. Chen, “A new deep learning method based on AlexNet model and SSD model for tennis ball recognition”, *2017 IEEE 10th International Workshop on Computational Intelligence and Applications (IWCIA)*, 11-12 Nov. 2017.
4. Hussam Qassim, Abhishek Verma, David Feinzimer. Compressed residual-VGG16 CNN model for big data places image recognition[C]// IEEE Computing & Communication Workshop & Conference. IEEE, 2018.
5. He, Kaiming, et al. “Deep residual learning for image recognition.” *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
6. Zhang, Yue, "Evaluation of CNN Models with Fashion MNIST Data" (2019). Creative Components. 364.
7. <https://github.com/icpm/pytorch-cifar10.git>
8. <https://blog.csdn.net/u014281392/article/details/74938524>
9. <https://blog.csdn.net/qq_39751437/article/details/97366544>
10. <https://blog.csdn.net/xun__Meng/article/details/89194148>

Artifact Appendix

We trained three Convolutional Neural Network models with MNIST and cifar-10 dataset. for run the project, we need to load dataset and build model.

## *A. Dataset Preparation*

Cifar-10 dataset is downloaded from torchvision.datasets.CIFAR10. The MNIST dataset is loaded from torchvision.datasets.MNIST. The format of the images stored in these two data sets is different, the training results of the model will also have different effects.

## *B. Experiment Environment*

- google colab (GPU)

- python3.6

- numpy

- pytorch 0.4.0

- torchvision 0.2.0

## *C. Optional Arguments*

TABLE II. The Optional Arguments

|  |  |  |
| --- | --- | --- |
| **argument** | **Default value** | **notes** |
| lr | 0.001 | Learning rate |
| epoch | 200 | Training times |
| trainBatchSize | 100 | Train batch size |
| testBatchSize | 100 | Test batch size |

## *D. Configurantion*

Epoch:

200 epochs for each run-through

500 batches for each training epoch

100 batches for each validating epoch

100 images for each training and validating batch

Learning Rate:

0.001 for [1,74] epochs

0.0005 for [75,149] epochs

0.0002.5 for [150,200) epochs

## *E. Run Experiment Step*

python3 Alexnet.py

python3 VGG.py

python3 Resnet.py

We listed the code of building and training Alexnet model. We set the number of classes to 10 and design each convolution layers’ kernel and padding. The sample source code is as follow.

#build Alexmodel

import torch.nn as nn

NUM\_CLASSES = 10

class AlexNet(nn.Module):

def \_\_init\_\_(self, num\_classes=NUM\_CLASSES):

super(AlexNet, self).\_\_init\_\_()

self.features = nn.Sequential(

nn.Conv2d(3, 64, kernel\_size=3, stride=2, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2),

nn.Conv2d(64, 192, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2),

nn.Conv2d(192, 384, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.Conv2d(384, 256, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.Conv2d(256, 256, kernel\_size=3, padding=1),

nn.ReLU(inplace=True),

nn.MaxPool2d(kernel\_size=2),

)

self.classifier = nn.Sequential(

nn.Dropout(),

nn.Linear(256 \* 2 \* 2, 4096),

nn.ReLU(inplace=True),

nn.Dropout(),

nn.Linear(4096, 4096),

nn.ReLU(inplace=True),

nn.Linear(4096, num\_classes),)

def forward(self, x):

x = self.features(x)

x = x.view(x.size(0), 256 \* 2 \* 2)

x = self.classifier(x)

return x

We defined two arrays Accu[] and Cost[] to store the loss and accuracy. We directly display the training results in the form of the line chart. Setting specified arguments. The default values of learning rate and batch size are 0.001 and 100 respectively. We also use CUDA to train model. The training times is 200.

#train model Alexnet.py

import torch.optim as optim

import torch.utils.data

import torch.backends.cudnn as cudnn

import torchvision

from torchvision import transforms as transforms

import numpy as np

import matplotlib.pyplot as plt

import argparse

from models import \*

from misc import progress\_bar

CLASSES = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

Accu = []

Cost = []

def main():

parser = argparse.ArgumentParser(description="cifar-10 with PyTorch")

parser.add\_argument('--lr', default=0.001, type=float, help='learning rate')

parser.add\_argument('--epoch', default=200, type=int, help='number of epochs)

parser.add\_argument('--trainBatchSize', default=100, type=int, help='trbatchsize')

parser.add\_argument('--testBatchSize', default=100, type=int, help='testbatchsize')

parser.add\_argument('--cuda',default=torch.cuda.is\_available(),type=bool, help='')

args = parser.parse\_args()

solver = Solver(args)

solver.run()

class Solver(object):

def \_\_init\_\_(self, config):

self.model = None

self.lr = config.lr

self.epochs = config.epoch

self.train\_batch\_size = config.trainBatchSize

self.test\_batch\_size = config.testBatchSize

self.criterion = None

self.optimizer = None

self.scheduler = None

self.device = None

self.cuda = config.cuda

self.train\_loader = None

self.test\_loader = None

def load\_data(self):

train\_transform = transforms.Compose([transforms.RandomHorizontalFlip(), transforms.ToTensor()])

test\_transform = transforms.Compose([transforms.ToTensor()])

train\_set=torchvision.datasets.CIFAR10(root='./data',train=True,download =True, transform=train\_transform)

self.train\_loader=torch.utils.data.DataLoader(dataset=train\_set, batch\_size=self.train\_batch\_size, shuffle=True)

test\_set=torchvision.datasets.CIFAR10(root='./data',train=False, download=True, transform=test\_transform)

self.test\_loader = torch.utils.data.DataLoader(dataset=test\_set, batch\_size=self.test\_batch\_size, shuffle=False)

def load\_model(self):

if self.cuda:

self.device = torch.device('cuda')

cudnn.benchmark = True

else:

self.device = torch.device('cpu')

self.model = AlexNet(num\_classes=10).to(self.device)

self.optimizer = optim.Adam(self.model.parameters(), lr=self.lr)

self.scheduler = optim.lr\_scheduler.MultiStepLR(self.optimizer,milestones =[75, 150], gamma=0.5)

self.criterion = nn.CrossEntropyLoss().to(self.device)

def train(self):

print("train:")

self.model.train()

train\_loss = 0

train\_correct = 0

total = 0

for batch\_num, (data, target) in enumerate(self.train\_loader):

data, target = data.to(self.device), target.to(self.device)

self.optimizer.zero\_grad()

output = self.model(data)

loss = self.criterion(output, target)

loss.backward()

self.optimizer.step()

train\_loss += loss.item()

# second param "1" represents the dimension to be reduced

prediction = torch.max(output, 1)

total += target.size(0)

# train\_correct incremented by one if predicted right

train\_correct+=np.sum(prediction[1].cpu().numpy()==target.cpu()

.numpy())

progress\_bar(batch\_num,len(self.train\_loader),'Loss:%.4f | Acc: %.3f%% (%d/%d)' % (train\_loss / (batch\_num + 1), 100. \* train\_correct / total, train\_correct, total))

Cost.append(train\_loss / (batch\_num + 1))

Accu.append(train\_correct / total)

return train\_loss, train\_correct / total

def test(self):

print("test:")

self.model.eval()

test\_loss = 0

test\_correct = 0

total = 0

with torch.no\_grad():

for batch\_num, (data, target) in enumerate(self.test\_loader):

data, target = data.to(self.device), target.to(self.device)

output = self.model(data)

loss = self.criterion(output, target)

test\_loss += loss.item()

prediction = torch.max(output, 1)

total += target.size(0)

test\_correct+=np.sum(prediction[1].cpu().numpy()==target.cpu()

.numpy())

progress\_bar(batch\_num,len(self.test\_loader),'Loss:%.4f|Acc: %.

3f%% (%d/%d)' % (test\_loss / (batch\_num + 1), 100. \* test\_correct

/ total, test\_correct, total))

return test\_loss, test\_correct / total

def save(self):

model\_out\_path = "model.pth"

torch.save(self.model, model\_out\_path)

print("Checkpoint saved to {}".format(model\_out\_path))

def run(self):

self.load\_data()

self.load\_model()

accuracy = 0

for epoch in range(1, self.epochs + 1):

self.scheduler.step(epoch)

print("\n===> epoch: %d/200" % epoch)

train\_result = self.train()

print(train\_result)

test\_result = self.test()

accuracy = max(accuracy, test\_result[1])

if epoch == self.epochs:

print("===> BEST ACC.%.3f%%" % (accuracy \* 100)) self.save()

if \_\_name\_\_ == '\_\_main\_\_':

main()

地图的截图

描述已自动生成

Fig. 7. The training result of AlexNet

TABLE III. The Accuracy of Each Model With the Two Dataset

|  |  |  |
| --- | --- | --- |
| **Model** | **Dataset** | **Accuracy** |
| Alexnet | Cifar10 | 74.74% |
| MNIST | 97.82% |
| Resnet | Cifar10 | 79.46% |
| MINST | 93.35% |
| VGGNet | Cifar10 | 89.79% |
| MINST | 90.62% |