INVARIANCE IN CNN ON MNIST DATASET

A PREPRINT

Megan Jennings[*]
Department of Information Technology
University of Texas at San Antonio
San Antonio, TX 78256
`Megan.Jennings@my.utsa.edu`

March 30, 2019

ABSTRACT

The purpose of this research is to demonstrate the idea of rotational invariance in CNN. The well-known MNIST dataset is used. The paper compares the results of 2 models. One built on the original dataset. The second attempts to model on the data after it is rotated 90 degrees.IN this way a deeper understanding of rotational invariance in CNN can be obtained.

***K*eywords** CNN, Invariance, MNIST

1 Introduction

Convolutional Neural Networks, or CNN, are widely used to identify images within Deep Learning. CNNs compare images piece by piece. The pieces that it looks for are called features. By finding rough feature matches in roughly the same positions in two images, CNNs get a lot better at seeing similarity than whole-image matching schemes.[1] Invariance means that you can recognize an object as an object, even when its appearance varies in some way.[2] Unlike the human eye, CNN does not recognize images with significant translation applied as being the same as an original image. This means that rotating the data in a significant way can cause training a model with a good accuracy very difficult. In this paper, I demonstrate this concept using the MNIST dataset; a collection on handwritten digits between 0 and 9.

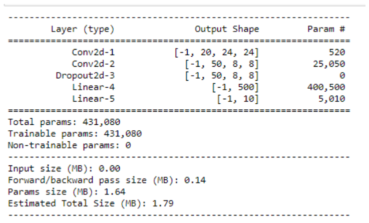
2 Related Work

There have been several works addressing the invariance problem in CNNs. CNN's work well, but have 2 very dangerous flaws: translation invariance and pooling layers.[3] Some interesting work has been done on the visualization and measurement of translational invariance using translation sensitivity maps and radial translation-sensitivity functions, for visualizing and quantifying the translation-invariance of classification algorithms [4].

3 Methods

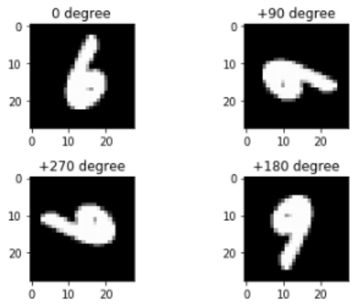
To demonstrate the invariance translation issue I trained a CNN on the original MNIST data set. I created an 80/20 split for test and train data. I utilized pytorch to train the model. The test dataset returned an accuracy of 98% after only 3 epochs and an average loss of 0.2639. The model summary is found in figure 1 below.

[*]https://github.com/msjennings/IS7033

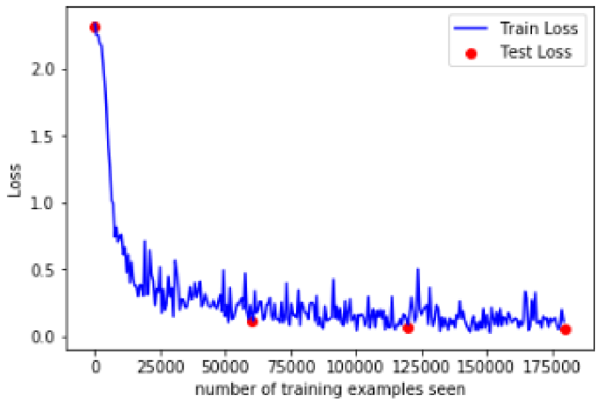


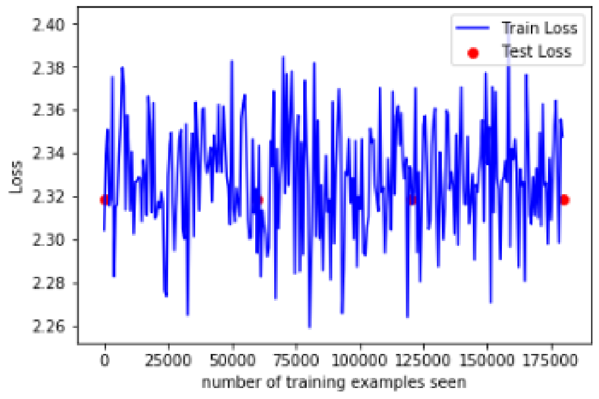
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 20, 24, 24]             520
            Conv2d-2             [-1, 50, 8, 8]          25,050
         Dropout2d-3             [-1, 50, 8, 8]               0
            Linear-4                  [-1, 500]         400,500
            Linear-5                   [-1, 10]           5,010
================================================================
Total params: 431,080
Trainable params: 431,080
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.14
Params size (MB): 1.64
Estimated Total Size (MB): 1.79
----------------------------------------------------------------
```

I then rotated the MNIST dataset 90 degrees, creating a new dataset. An example of rotated data is shown in figure 2 below.



Training a CNN on the rotated data set even with 50 epochs only provided 8% accuracy with an average loss of 2.3186. This is a very poor performance and confirms the current work around issues with invariance in CNN. In fact, visualizing the training Loss for each dataset in figure 3 & 4 below show just how difficult it is to get the CNN to converge on the rotated dataset.





4 Conclusion

The next steps of the research will be to investigate ways to address the invariance. Some emerging ideas on how to address this are Estimation of Manifold Structure (math theory in high dimensional data) and Hypernet (multi function optimization). I will research these ideas and determine if one can help address the invariance in this problem. Other options for attempting to correct the invariance is through data augmentation or model variation. Additional research into CapsNet & routing may also provide fresh insight into & innovative solutions for addressing the issue. This research will be investigated in an upcoming paper.

5 Appendix

Let's start coding right away. First goal is to define the models and train one with pure MNIST dataset -> Rotation = 0^o.

Import Libraries

```
import torch
import torchvision
from torchvision import datasets, transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchsummary import summary
import matplotlib.pyplot as plt
from torch.autograd import Variable
import numpy as np
from torchvision.utils import make_grid
import math
```

Define Hyperparameters

```
n_epochs = 3
n_epochs_90 = 5
batch_size_train = 64
batch_size_test = 1000
learning_rate = 0.01
momentum = 0.5
log_interval = 10

random_seed = 1
torch.backends.cudnn.enabled = False
torch.manual_seed(random_seed)
```

Create Rotation Function

```
class CustomRotation(object):
    """Rotate image by a fixed angle which is ready for tranform.Compose()
    """

    def __init__(self, degrees, resample=False, expand=False, center=None):
        self.degrees = degrees
        self.resample = resample
        self.expand = expand
        self.center = center

    def __call__(self, img):

        return transforms.ToTensor()(
            transforms.functional.rotate(
                transforms.ToPILImage()(img),
                self.degrees, self.resample, self.expand, self.center))
```

Create Test & Train Datasets

```python
rotation = 0 # Specifies the rotation of images.

# Define the train and test loader
# Here we are adding our CustomRotation function to the transformations

train_loader = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=True, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               CustomRotation(rotation),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_train, shuffle=True)

test_loader = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=False, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               CustomRotation(rotation),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_test, shuffle=True)

examples = enumerate(test_loader)
batch_idx, (example_data, example_targets) = next(examples)
```

Build Network

```python
class Net(nn.Module):
    #This defines the structure of the NN.
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, kernel_size=5)
        self.conv2 = nn.Conv2d(20, 50, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()  #Dropout
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2)) #Convolutional Layer/Pooling Layer/Activation
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2)) #Convolutional Layer/Dropout/Pooling Layer/Activation
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x)) #Fully Connected Layer/Activation
        x = F.dropout(x, training=self.training)
        x = self.fc2(x) #Fully Connected Layer/Activation
        return F.log_softmax(x, dim=1) #Softmax gets probabilities.
```

Initalize the network & optimzer

```python
network = Net()
optimizer = optim.SGD(network.parameters(), lr=learning_rate,
                      momentum=momentum)
```

Training

```python
def train(epoch):
    network.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = network(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))
            train_losses.append(loss.item())
            train_counter.append(
                (batch_idx*64) + ((epoch-1)*len(train_loader.dataset)))


def test():
    network.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = network(data)
            test_loss += F.nll_loss(output, target, size_average=False).item()
            pred = output.data.max(1, keepdim=True)[1]
            correct += pred.eq(target.data.view_as(pred)).sum()
    test_loss /= len(test_loader.dataset)
    test_losses.append(test_loss)
    print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

test()
for epoch in range(1, n_epochs + 1):
    train(epoch)
    test()
```

Create Test & Train Rotated Dataset

```

rotation = 90 # Specifies the rotation of images.

# Define the train and test loader
# Here we are adding our CustomRotation function to the transformations

train_loader_90 = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=True, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               CustomRotation(rotation),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_train, shuffle=True)

test_loader_90 = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=False, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               CustomRotation(rotation),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_test, shuffle=True)
```

Create Network

```
network_90 = Net()
```

Training Rotated

```
def train_90(epoch):
    network_90.train()
    for batch_idx, (data, target) in enumerate(train_loader_90):
        optimizer.zero_grad()
        output = network_90(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader_90.dataset),
            100. * batch_idx / len(train_loader_90), loss.item()))
            train_losses_90.append(loss.item())
            train_counter_90.append(
                (batch_idx*64) + ((epoch-1)*len(train_loader_90.dataset)))
            #torch.save(network.state_dict(), '/results/model.pth')
            #torch.save(optimizer.state_dict(), '/results/optimizer.pth')

def test_90():
    network_90.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader_90:
            output = network_90(data)
            test_loss += F.nll_loss(output, target, size_average=False).item()
            pred = output.data.max(1, keepdim=True)[1]
            correct += pred.eq(target.data.view_as(pred)).sum()
    test_loss /= len(test_loader_90.dataset)
    test_losses_90.append(test_loss)
    print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader_90.dataset),
        100. * correct / len(test_loader_90.dataset)))

test_90()
for epoch in range(1, n_epochs_90 + 1):
    train_90(epoch)
    test_90()
```

Graphing Loss

```
#create two lists for saving training and testing losses
#On the x-axis we want to display the number of training examples the network has seen
     during training
train_losses_90 = []
train_counter_90 = []
test_losses_90 = []
test_counter_90 = [i*len(train_loader_90.dataset) for i in range(n_epochs + 1)]

fig = plt.figure()
plt.plot(train_counter_90, train_losses_90, color='blue')
plt.scatter(test_counter_90, test_losses_90, color='red')
plt.legend(['Train Loss', 'Test Loss'], loc='upper right')
plt.xlabel('number of training examples seen')
plt.ylabel('Loss')
```

References

[1] Brandon Rohrer. How do convolutional neural networks work? `https://brohrer.github.io/how_convolutional_neural_networks_work.html`, 2013.

[2] Matt Krause (https://stats.stackexchange.com/users/7250/matt krause). What is translation invariance in computer vision and convolutional neural network? Cross Validated. URL:https://stats.stackexchange.com/q/208949 (version: 2018-10-02).

[3] Mahmoud Tarrasse. What is wrong with convolutional neural networks ? `https://towardsdatascience.com/what-is-wrong-with-convolutional-neural-networks-75c2ba8fbd6f`, 2018.

[4] Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *arXiv preprint arXiv:1801.01450*, 2017.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[1] [2] [3] [4] [5]