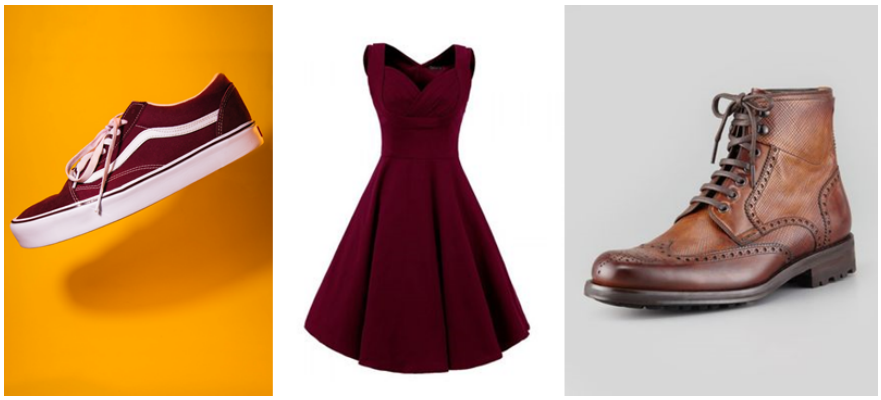
Training Convolutional Neural Networks to Categorize Clothing with PyTorch

A guide to coding a CNN with the Torch framework

Albert Lai

Nov 30, 2018 · 5 min read

Here are pictures of clothing. Can you tell me what they are?



If you guessed sneakers, a dress, and a boot, you're right! It wasn't too hard, was it? That's because humans are **really good at detecting patterns and classifying objects.** However, that's not always the case, especially with machines. A computer doesn't see that, it sees this:

```
[[  9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
 [  3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
 [  1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
 [  3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
 [  6   1  32  36  47  81  85  90 176 206 140 171 186  22   3  15  72  63]
 [  4   1  31  39  66  71  71  97 147 214 203 190 198  22   6  17  73  65]
 [  2   3  15  30  52  57  68 123 161 197 207 200 179   8   8  18  73  66]
 [  2   2  17  37  34  40  78 103 148 187 205 225 165   1   8  19  76  68]
 [  2   3  20  44  37  34  35  26  78 156 214 145 200  38   2  21  78  69]
 [  2   2  20  34  21  43  70  21  43 139 205  93 211  70   0  23  78  72]
 [  3   4  16  24  14  21 102 175 120 130 226 212 236  75   0  25  78  72]
 [  6   5  13  21  28  28  97 216 184  90 196 255 255  84   4  24  79  74]
 [  6   5  15  25  30  39  63 105 140  66 113 252 251  74   4  28  79  75]
 [  5   5  16  32  38  57  69  85  93 120 128 251 255 154  19  26  80  76]
 [  6   5  20  42  55  62  66  76  86 104 148 242 254 241  83  26  80  77]
 [  2   3  20  38  55  64  69  80  78 109 195 247 252 255 172  40  78  77]
 [ 10   8  23  34  44  64  88 104 119 173 234 247 253 254 227  66  74  74]
 [ 32   6  24  37  45  63  85 114 154 196 226 245 251 252 250 112  66  71]]
```

Images are stored in computers as matrices of numbers where each number represents the colour of a pixel. For black and white images, there will only be one number per pixel to represent its darkness, while for coloured images, there will be 3 numbers for the RGB channels (one for red, one for blue, one for green). Each number represents the intensity of that colour and can range between 0 and 255.

So, how can we teach a computer to tell us if an image is a shoe or a t-shirt if all it can "see" are numbers? The process of computers analyzing and understanding images is called **computer vision**, a subset of machine learning, and **convolutional neural networks** are the best algorithms for the task.

Creating the Convolutional Neural Network

I'll be showing you how I built my convolutional neural network in Pytorch. I trained it using the MNIST—Fashion dataset with 60,000 examples of 28x28 resolution black-and-white images of clothes. Let's jump right in!

> *Check out the entire code from GitHub here.*

Part 1: Imports, Initializations, and Dataset

First, we import pytorch, the deep learning library we'll be using, and torchvision, which provides our dataset and image transformations. We then explicitly import torch.transform and torch.datasets for convenience. We also import torch.nn (pytorch's neural network library), torch.nn.functional (includes non-linear functions like ReLu and sigmoid), and torch.autograd (computational graph for backpropagation).

```
#Importing libraries
import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torch.nn as nn
```

Then we initialize the hyperparameters, which include the number of epochs (training rounds), number of classes (there are 10: t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and boot), the batch-size for mini-batch gradient descent, and the learning rate for gradient descent.

```
#Initializing hyperparameters
num_epochs = 8
num_classes = 10
batch_size = 100
```

Time to retrieve the dataset! We load both the training set and the test set of the MNIST-Fashion and set the transform parameter to convert the datasets into tensors *(transforms.ToTensor())*. We also normalize them by setting the mean and standard deviation *(transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)))*.

```
#Loading dataset
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((

train_dataset = datasets.FashionMNIST(root='./data',
                            train=True,
                            download=True,
                            transform=transform)

```

As the dataset is loaded, you should see these messages:

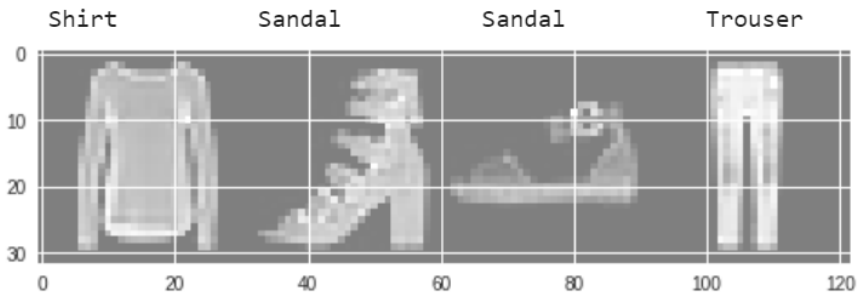
```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Processing...
Done!
```

The last step in this stage is to place the data into an object (data loader) to make it more easily accessible. We shuffle the training dataset so that there won't be any bias in the training.

```
#Loading dataset into dataloader
train_loader = torch.utils.data.DataLoader(dataset=trai
                                           batch_size=b
                                           shuffle=True

test_loader = torch.utils.data.DataLoader(dataset=test_
```

Here are some of the images from the dataset we'll be using to the train the network:

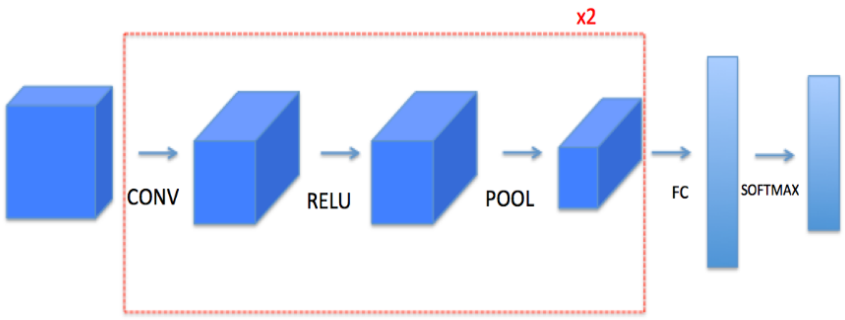


Part 2: Constructing the Convolutional Neural Network

We initialize our neural network using the nn.Module class, the base class of all layered neural network modules:

```
#Defining the network
class CNNModel(nn.Module):
```

Our convolutional network will two convolution layers, each followed by a non-linear function (ReLu) and a max-pooling layer, and finally a fully connected layer and softmax for linear regression.



Layers of the Convolutional Neural Network

We also use dropout for regularization before the fully connected layer to prevent against overfitting. Thus, we can initialize the layers in the network as follows:

```
    def __init__(self):
        super(CNNModel, self).__init__()

        #Convolution 1
        self.cnn1 = nn.Conv2d(in_channels=1, out_chann
        self.relu1 = nn.ReLU()

        #Max pool 1
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        #Convolution 2
        self.cnn2 = nn.Conv2d(in_channels=16, out_chan
        self.relu2 = nn.ReLU()

        #Max pool 2
```

In our forward function for forward propagation, we apply each layer to the input data, as well as the dropout. Additionally, before the regularization, we flatten the data to be one-dimensional for the linear regression (the first dimension is the batch size).

```
    def forward(self, x):
        #Convolution 1
        out = self.cnn1(x)
        out = self.relu1(out)

        #Max pool 1
        out = self.maxpool1(out)

        #Convolution 2
        out = self.cnn2(out)
        out = self.relu2(out)

        #Max pool 2
        out = self.maxpool2(out)

        #Resize
```

Part 3: Creating Instances

Now that we've created a class for our convolutional neural network, we need to create an instance of it (we've created a class to determine its layers and forward propagation, but we haven't actually created an actual neural net yet).

```
#Create instance of model
model = CNNModel()
```

We also use cross-entropy loss to determine the labels from the output of the neural net.

```
#Create instance of loss
criterion = nn.CrossEntropyLoss()
```

Finally, we initialize the linear regression/softmax function.

```
#Create instance of optimizer (Adam)
optimizer = torch.optim.Adam(model.parameters(), lr=lea
```

Part 4: Training the Model

After creating the network and the instances, we're ready to train it using the dataset! We iterate through the dataset and for each mini-batch, we perform forward propagation, calculate the cross-entropy loss, perform backward propagation and use the gradients to update the parameters.

```
#Train the model
iter = 0
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader)
        images = Variable(images)
        labels = Variable(labels)

        #Clear the gradients
        optimizer.zero_grad()

        #Forward propagation
        outputs = model(images)

        #Calculating loss with softmax to obtain cross
        loss = criterion(outputs, labels)

        #Backward propation
        loss.backward()

        #Updating gradients
        optimizer.step()

        iter += 1

        #Total number of labels
```

Also, for every 100 mini-batches, the loss and accuracy of the neural network. The final outputs should look something like this:

```
Epoch [8/8], Step [400/600], Loss: 0.4103, Accuracy: 86.00%
Epoch [8/8], Step [500/600], Loss: 0.2225, Accuracy: 95.00%
Epoch [8/8], Step [600/600], Loss: 0.1543, Accuracy: 93.00%
```

Part 5: Testing the Model

So our model is trained, all that's left to do is to test it! We run the neural network on the test dataset, compare our outputs to the correct labels, and determine our overall accuracy.

```
#Testing the model
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = Variable(images)
        labels = Variable(labels)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
```

Results

Here are the results!

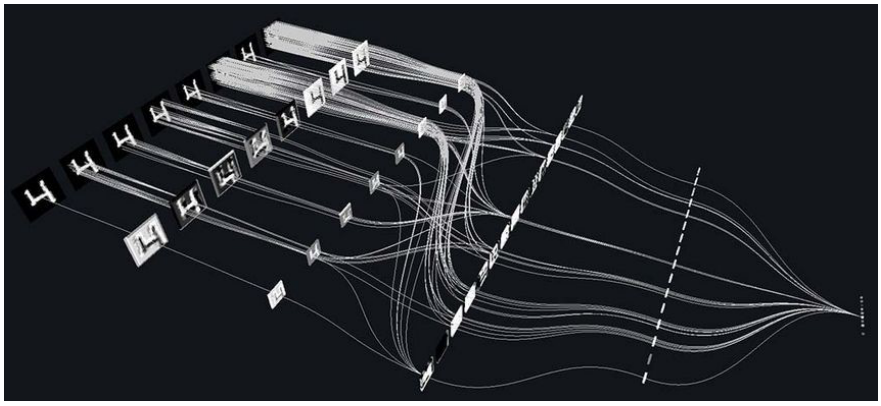
```
Test Accuracy of the model on the 10000 test images: 89.24 %
```

The neural network should achieve an accuracy from 88%–90%, which is quite good compared to the 91.6% benchmark of 2-layered CNNs on the MNIST-Fashion dataset. Note that the MNIST-Fashion dataset is much harder to train on than the original MNIST-digit dataset. If we want to achieve a higher accuracy, we would have to add more layers, preprocess the data more to normalize it better, and increase the number of epochs.

Here's some predictions made by the neural network:

Predictions made by the convolutional neural network

As you can see, our model can correctly classify these clothes! However, it's still not perfect as it incorrectly classified the last image (a sneaker) as a pullover. Nevertheless, the model is still pretty accurate and I'll be trying to improve it.



Convolutional Neural Network graphic

I hope you enjoyed coding up this convolutional neural network! If you liked this article, please connect with me on linkedin and follow me!