université de BORDEAUX

Collège Sciences et Technologies

Deep Learning

Author:

Martial BODET

Bordeaux
02-08-2019
Write in LaTeX

Outline

1 Material and Method **2**

1.1 Pytorch 2

1.2 TensorFlow 4

1.3 Algorithm 5

2 Results **6**

Introduction

During this project, we used Pytorch to implement a CNN neural network to classify images. These images are taken from a dataset given in the subject. This goal is to train the algorithm with data training, and finally to test of full dataset. The last part is to provide an image to the algorithm, and it determines what fruit it is. To do this we have used python, and most particularly *Pytorch*, and the library *TensorFlow*. This packages for python, are used for implement deep learning algorithm.

1 Material and Method

Different implementation have been tested empirically to chose bests parameters

1.1 Pytorch

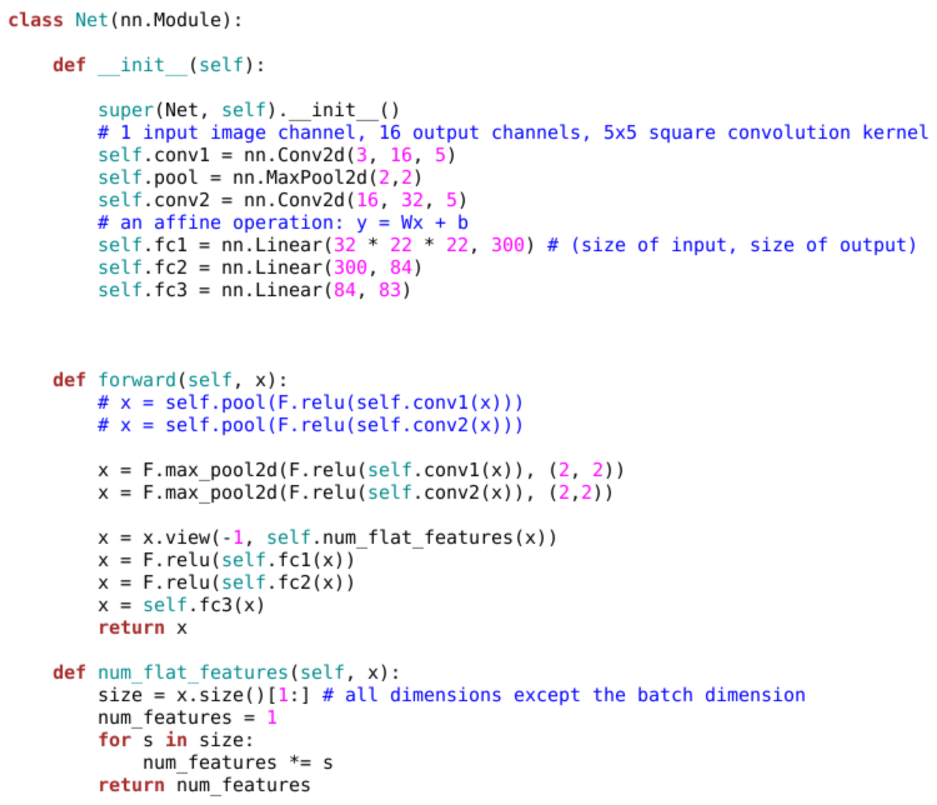
Pytorch It's a Python-based scientific computing package targeted at two sets of audiences:

- A replacement for NumPy to use the power of GPUs
- A deep learning research platform that provides maximum flexibility and speed

Thanks to this library, we can implement a simple feed-forward network. It takes the input, feeds it through several layers one after the other, and then finally gives the output.

A typical training procedure for a neural network is as follows:

- Define the neural network that has some learnable parameters (or weights)
- Iterate over a data-set of inputs
- Process input through the network
- Compute the loss (how far is the output from being correct)
- Propagate gradients back into the network's parameters
- Update the weights of the network.

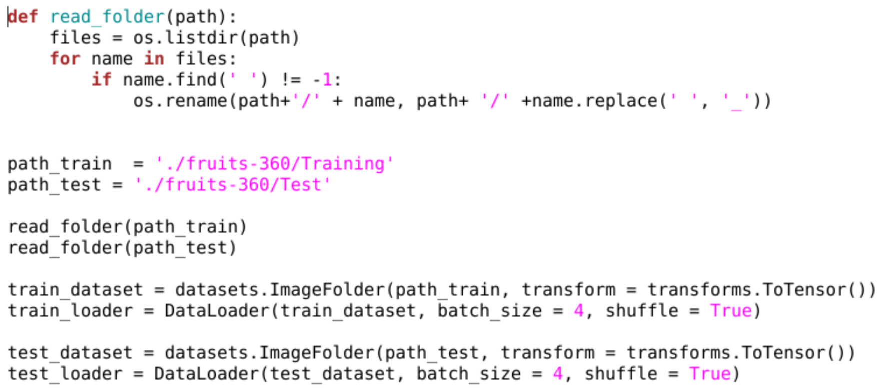


```python
class Net(nn.Module):

    def __init__(self):

        super(Net, self).__init__()
        # 1 input image channel, 16 output channels, 5x5 square convolution kernel
        self.conv1 = nn.Conv2d(3, 16, 5)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(16, 32, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(32 * 22 * 22, 300) # (size of input, size of output)
        self.fc2 = nn.Linear(300, 84)
        self.fc3 = nn.Linear(84, 83)



    def forward(self, x):
        # x = self.pool(F.relu(self.conv1(x)))
        # x = self.pool(F.relu(self.conv2(x)))

        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2,2))

        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:] # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

Figure 1: Implementation of neural network

After this implementation, we need to train the model with our data-sets. We will do the following steps in order:

- Load and normalizing the CIFAR10 training and test data-sets using torchvision
- Define a Convolutional Neural Network
- Define a loss function
- Train the network on the training data
- Test the network on the test data



```
def read_folder(path):
    files = os.listdir(path)
    for name in files:
        if name.find(' ') != -1:
            os.rename(path+'/' + name, path+ '/' +name.replace(' ', '_'))


path_train  = './fruits-360/Training'
path_test = './fruits-360/Test'

read_folder(path_train)
read_folder(path_test)

train_dataset = datasets.ImageFolder(path_train, transform = transforms.ToTensor())
train_loader = DataLoader(train_dataset, batch_size = 4, shuffle = True)

test_dataset = datasets.ImageFolder(path_test, transform = transforms.ToTensor())
test_loader = DataLoader(test_dataset, batch_size = 4, shuffle = True)
```

Figure 2: Loading of datasets for train and test

1.2 TensorFlow

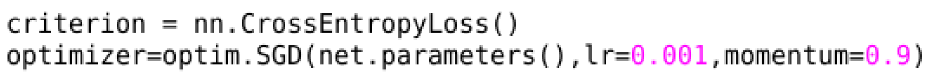
TensorFlow is an open source software library for numerical computation using data-flow graphs. It was originally developed by the Google Brain Team within Google's Machine Intelligence research organization for machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. This library used in python permitted us to create an deep learning algorithm.

1.3 Algorithm

In this section i will develop how does it work the algorithm. Firstly, like i said previously, we need to load datasets, like we can see in figure 2. This part of program load files, and delete spaces present in the name of file. batch size allows to create groups of x images. The choice of group size is *4*.

The neural network implemented, like we can see in figure 1 is composed of two convolutional layer, and tree fully connected layer. Thanks to *Pytorch* we have to implement only forward function, because backward function is auto-generated.

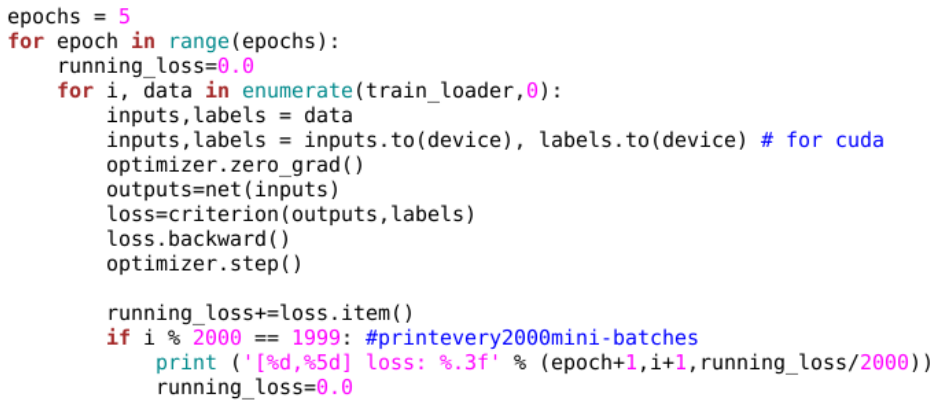
After, we have to define a Loss function and optimizer as we can see in figure 3 bellow.



```
criterion = nn.CrossEntropyLoss()
optimizer=optim.SGD(net.parameters(),lr=0.001,momentum=0.9)
```

Figure 3: Loss function and optimizer

Finally, we are able to train the network. To realize this, we have crate a *"for" loop* as we can see on figure 4 bellow. The number of epoch have been



```
epochs = 5
for epoch in range(epochs):
    running_loss=0.0
    for i, data in enumerate(train_loader,0):
        inputs,labels = data
        inputs,labels = inputs.to(device), labels.to(device) # for cuda
        optimizer.zero_grad()
        outputs=net(inputs)
        loss=criterion(outputs,labels)
        loss.backward()
        optimizer.step()

        running_loss+=loss.item()
        if i % 2000 == 1999: #printevery2000mini-batches
            print ('[%d,%5d] loss: %.3f' % (epoch+1,i+1,running_loss/2000))
            running_loss=0.0
```

Figure 4: Training loop of network

found empirically, and it's the best compromise between speed and efficiency.

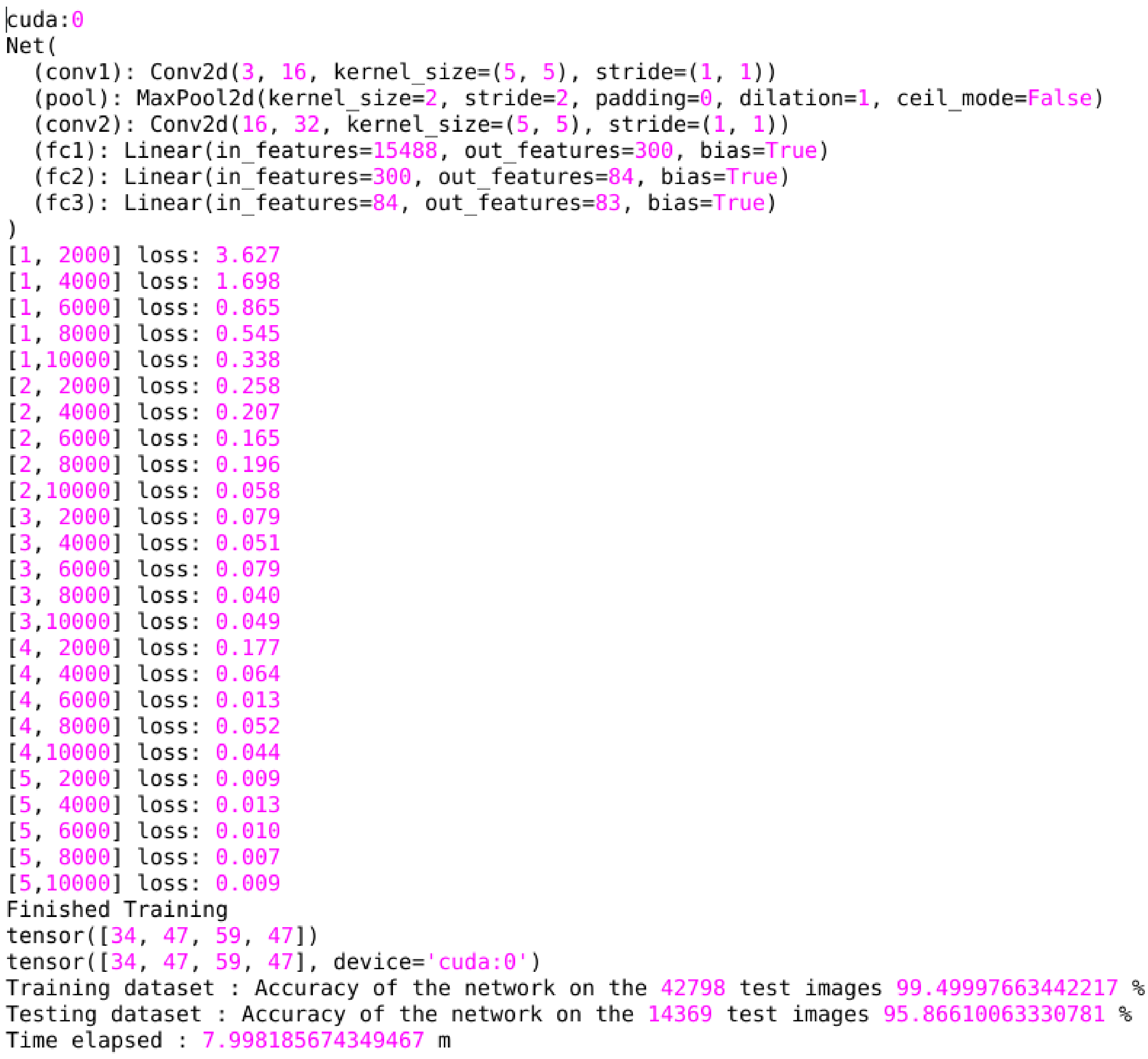
The algorithm is complete, but run for a long time, so in order to decrease the execution time, we run the program on GPU. Thereby, the execution time is under 8 minutes.

2 Results

Different parameters have been tested to increase the accuracy of the program. The most values of parameters found are presented bellow :

- Epochs = 5
- Learning rate = 0.001
- Momentum = 0.9
- Convolutional layer 1 (3, 16, 5)
- Convolutional layer 2 (16, 32, 5)
- Fully Connected layer 1 (32 * 22 * 22, 300)
- Fully Connected layer 2 (300, 84)
- Fully Connected layer 3 (84, 83)

With this parameters chosen empirically, the accuracy value is more than 94% The epoch is the number of training loop turns that we realize. it must not be too weak, otherwise the model is not trained enough. But it must also not be too high, otherwise the model is over trained and it will be too specific. Learning rate and momentum have not been modified because they offer a high accuracy value. the results obtained using these parameters are shown in the figure 5 below.



```
cuda:0
Net(
  (conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=15488, out_features=300, bias=True)
  (fc2): Linear(in_features=300, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=83, bias=True)
)
[1, 2000] loss: 3.627
[1, 4000] loss: 1.698
[1, 6000] loss: 0.865
[1, 8000] loss: 0.545
[1,10000] loss: 0.338
[2, 2000] loss: 0.258
[2, 4000] loss: 0.207
[2, 6000] loss: 0.165
[2, 8000] loss: 0.196
[2,10000] loss: 0.058
[3, 2000] loss: 0.079
[3, 4000] loss: 0.051
[3, 6000] loss: 0.079
[3, 8000] loss: 0.040
[3,10000] loss: 0.049
[4, 2000] loss: 0.177
[4, 4000] loss: 0.064
[4, 6000] loss: 0.013
[4, 8000] loss: 0.052
[4,10000] loss: 0.044
[5, 2000] loss: 0.009
[5, 4000] loss: 0.013
[5, 6000] loss: 0.010
[5, 8000] loss: 0.007
[5,10000] loss: 0.009
Finished Training
tensor([34, 47, 59, 47])
tensor([34, 47, 59, 47], device='cuda:0')
Training dataset : Accuracy of the network on the 42798 test images 99.49997663442217 %
Testing dataset : Accuracy of the network on the 14369 test images 95.86610063330781 %
Time elapsed : 7.998185674349467 m
```

Figure 5: Results of algorithm

Conclusion

To conclude it is possible to say that the algorithm works relatively well, with an accuracy value greater than 95% However, it would be interesting to test it on other datasets to see if it would be as efficient, or if it would be necessary to review the code.
However it would have been interesting to implement other layers of convolution, or fully connected, to see if it had a real impact on the accuracy.