Convolutional Neural Networks with PyTorch library

LE Van Linh

April 12, 2018

Abstract

In this work, we study a new Deep Learning framework, PyTorch. That is a python package that provides two high-level features (Tensor computation and Deep Networks). In this study, we firstly study the components of PyTorch,how to create a Deep Network with PyTorch. Then, we use this framework to create a network model that we had submitted to ICPRS-18. At the beginning of the model, we use one channel images with the size of 96×96 as the inputs. To compare with different frameworks (Lasagne), in the experiment part, we will compare the losses and time-consuming during training the model.

1 Pytorch

PyTorch [1] is a python package that provides two high-level features:

- Tensor computation (like numpy) with strong GPU acceleration,
- Deep Neural Networks built on a tape-based autodiff system.

In the core of library, PyTorch consists of the following packages:

- **torch**: a Tensor library with GPU support
- **torch.autograd**: supports all Tensor operation in torch
- **torch.nn**: a neural networks library. It deeply integrated with autograd.
- **torch.optim**: an optimization package such as SGD, RMSProp, ...which are used with *torch.nn* package
- **torch.multiprocessing**: provides the multiprocessing in python.
- **torch.utils**: provides the helper functions such as DataLoader, Trainer, ...
- **torch.legacy(.nn/.optim)**: legacy code that has been ported over from torch for backward compatibility reasons

Most frameworks such as TensorFlow, Theano, Caffe and CNTK have a static view of the world. One has to build a neural network, and reuse the same structure again and again. Changing the way the network behaves means that one has to start from scratch.

With PyTorch, they have a unique way of builiding neural networks: using and replaying a tape recoder. They use a technique called Reserse mode auto-differentiation, which allows the user to change the way the network behaves arbitrarily with zero lag or overhead

2 Create a neural network with PyTorch

Neural networks can be constructed using **torch.nn** package. A **nn.Module** contains layers, method (*forward*) that return the output. Let's define a network with 2 Convolutional layers followed by 2 Maximum Pooling layers and finish with 3 full-connected layers.

```
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels,
        #  5x5 square convolution kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square,
        # you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 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):
        # all dimensions except the batch dimension
        size = x.size()[1:]
        num_features = 1
        for s in size:
        num_features *= s
        return num_features

net = Net()
print(net)
```

In this network, we just need define the **forward** function, the **backward** function will automatically defined for us using **autograd**.

3 Landmark predicted model

In ICPRS-18 paper, we have proposed a network to predict the landmarks on pronotum images. It receives an image of $1 \times 256 \times 192$ as the input. The network was constructed from 3 *"elementary block"* following by 3 full-connected layers. An elementary block is defined as a sequence of convolution (C_i), pooling (P_i) and dropout (D_i) layers. The parameters for each layers are as below, the list of values follows the order of elementary blocks:

- CONV layres:
  - Number of filters: 32, 64 and 128,
  - Kernel filters size: $(3 \times 3), (2 \times 2)$, and (2×2)
  - Stride values: $1, 1, 1$
  - No padding is used for CONV layers
- POOL layers:
  - Kernel filters size: $(2 \times 2), (2 \times 2)$, and (2×2)
  - Stride values: $2, 2, 2$
  - No padding is used for CONV layers
- DROP layers:
  - Propabilities: $0.1, 0.2$ and 0.3.

In the last full-connected layers (FC), the parameters are: FC1 output: 1000, FC2 output: 1000, FC3 output: 16. As usual, a dropout layer is inserted between FC1 ond FC2 with a probability equal to 0.5. Fig.1 illustrate the order of the layers in the network.

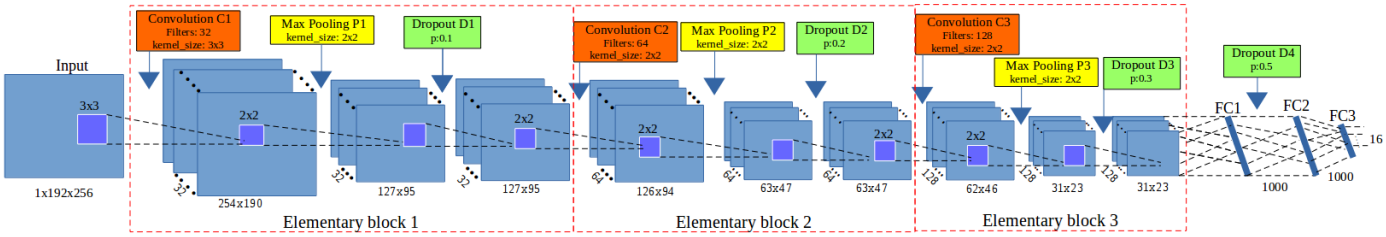


Figure 1: Network architecture using 3 *elementary blocks*. Convolution layer in red, pooling in yellow and dropout in green color.

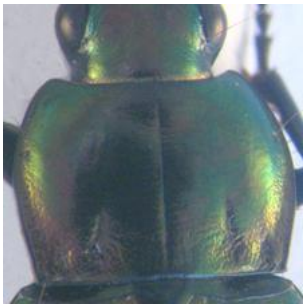
In our study, instead of using a *"square"* image, we have used a *"rectangle"* image as the input. The result shows that the network has the ability to work well on the input that size of width and height are different. Considering as a different of workflows, we would like to see how the network works on "square" input. In this study, we continue train the proposed network on a new dataset with different size of 96×96. The network is implemented on PyTorch.

4 Dataset

The dataset includes 293 RGB-images of beetle's pronotum. The images were taken by the same camera with the same conditions of resolution of 3264×2448. The images in the dataset were divided into two subsets: training (including 260 images) and testing (including 33 images). For each image, a set of 8 manual landmarks have been set by biologists. In this section, we introduce the process to down-sample the original images to the new size of images. To obtain the images size of 96×96, we have applied the procedure following:

1. Left crop image to obtain the new size of 2448×2448,
2. Down-sample the image to size of 96×96,
3. Scale the coordinates of the manual landmarks to adapt with the new size of images.

Then, we augment the images for training and validation (it have been presented in ICPRS-18).



(a)

Figure 2: An image example in dataset.

5 Experiments

We have trained the model on 2 different libraries: Lasagne [2] and PyTorch in 10000 epochs. During training, we have applied differents optimizers (with the same parameters) to compute backprogation. The number of images, which use to train and to validate, are split automatically followed the ratio of 0.8/0.2. Table.1 shows losses corresponding to the optimizer during training.

Optimizer	Lasagne			PyTorch		
	Train loss	Validation loss	Time(s)	Train loss	Validation loss	Time(s)
Adam	0.00011	0.00074	4,200	0.00002	0.00286	22,500
RMSprop	0.00839	0.00778	4,308	0.00818	0.00872	23,548
SGD Momentum	0.00041	0.00025	4,236	0.00034	0.00154	23,264
SGD Momentum nesterov	**0.00015**	**0.00004**	**4,258**	**0.00010**	**0.00012**	**24,538**

Table 1: A comparing of losses during training between the libraries.

Following Table.1, we can clearly see the differences between the losses of 2 libraries on the same network with the same data, especially, training time. In the case of **SGD Momentum nesterov** optimizer, we have obtained the same losses during training model in two libraries.

6 Conclusions

In this study, we have studied PyTorch, a new libraries which used to design a Convolutional Neural Network. We have shown the way to create a model with some basic layers. At the end of study, we have trained the model which we have submitted to **ICPRS-18** in PyTorch. We have trained the model with different optimizers. As the results, mostly the losses are different between two libraries. In which, the large distance between the training times is a worth to consider in the future works.

References

[1] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[2] Sander Dieleman, Jan Schlter, Colin Raffel, Eben Olson, Sren Kaae Snderby, Daniel Nouri, et al. Lasagne: First release., August 2015.