DEEP LEARNING : CONVOLUTIONAL NEURAL NETWORK

February 11, 2019

GONCALVES CLARO SÃl'bastien
University of Bordeaux
Bioinformatic Master

Contents

1 Material and Methods **3**
1.1 Material 3
1.1.1 Pytorch 3
1.1.2 Jupyter 3
1.1.3 Fruit 360 dataset 3
1.2 Method 4
1.2.1 Simple convolutional neural network 4
1.2.2 Improve accuracy and avoid overfitting 5

2 Results and Discussion **6**
2.1 Results 6
2.2 Discussion 8

Introduction

Current method like recognition tasks are essential methods of machine learning. With a huge amount of dataset, that could reach a million images, it is needed to use models with a large capacity of learning. Thus, the convolutional neural network (CNNs) was created to bring a solution and work efficiently.[2]

A neural network provides state of the accuracy, and the goal is to reach the better accuracy while avoiding typical errors like overfitting. They are really great in classification of visual inputs. But it's important to take into account some good practices in order to get the best from a neural network.[4]

The CNNs is a class of deep neural networks, that works greatly with images inputs. It is a variation of multilayer perceptrons and it is designed to require minimal preprocessing. The network then learn the filters. On the contrary, traditional image classification algorithms filters are hand-written. CNNs have attractive qualities, but they kind of expensive in a computing aspects. It is crucial to optimized the implementation to ease the training of a large scale CNNs. Nowaday, datasets like *ImageNet* can train such model without overfitting. But to work efficiently, we need high quality graphics processing unit (GPU).
[2]

In this report, we propose to design a CNNs architecture to classify images of fruits into different classes. The dataset called **"Fruit 360"** contains the pictures of 95 types of fruits. The goal is to find the parameters that allow to get the best possible accuracy, while avoiding errors.

Chapter 1

Material and Methods

1.1 Material

1.1.1 Pytorch

To design our model, we use the computing package Pytorch. It is a Python-based scientific tool for deep learning research. With Pytorch, we are able to use the power of GPUs to provide the maximum speed and flexibility. We use the version 0.4.1 for this project.[3]

1.1.2 Jupyter

Jupyter is an open-source project about a support interactive data science. The main part is the web application called Jupyter Notebook. It allows to create documents with live code, visualizations and narrative text. Jupyter Notebook support different programming languages, and more specifically Python.

1.1.3 Fruit 360 dataset

The dataset we work with contains 65 429 images, splitted into two sets : training and test sets. In each set, there are one fruit per image and their size are 100x100 pixels. As said previously in the introduction, there are 95 types of fruit and we are going to classify them with our CNNs.

1.2 Method

1.2.1 Simple convolutional neural network

In order to train efficiently our CNNs classifier, we need to define the best architecture. To define the network, we use Pytorch with its modules. To begin with, a simple neural network at two layer is enough to get the first idea about the accuracy and efficiency. But we need to tune the neural network, with its architecture and parameters, until we get the best results possible.

These layers apply a 2D convolution over the input signal, because the images are in a 2D dimension. Pytorch already has a function for that called *Conv2d*. Different parameters are defined to design these layers, that is to say the input channels, the output channels and the kernel size.

The input channels for this project is equal to three, because it is a collection of RGB (red,green,blue) images. Also, it is important that the input channel is equivalent of the output channel of the previous layer. Talking about the output channels, it is a parameter defined in function of the preferences of the user. A larger output will permit to potentially learn more useful features about the input data. And the kernel size concerns the size of the filter applied on the images. Features of each pixel are calculated locally.

The inputs provided initially are then propagated to the hidden units at each layer, to finally produce the output. The width and depth are the two variables defining the architecture of our CNN. Depth is the number of hidden layers. Width is the number of units on each hidden layer. Finetuning those variables with activation functions like *Rectified Linear Unit* (ReLU) can enhance the precision the forward propagation of the information, and therefore the strength of the CNN.

In a neural network, we should also apply a linear transformation at each output unit. With this process, the compute time of the neural network is faster and also gain more flexibility. We compute the affine operation like the following :

$$y = Wx + b$$

where W is the weight, b the bias.

When the model is well defined, the training phase can begin. It means we use the

training dataset to train the classifier. And then the test is ran with the test dataset, which permits to evaluate the accuracy.

1.2.2 Improve accuracy and avoid overfitting

In deep learning research, it is common to tweak parameters and also change the architecture of the neural network, to improve its confidence and accuracy. One of the regular problem that can be avoided is the overfitting. Overfitting is the case where our model is too well fit to the training dataset. In that case, the classifier is not able to classify correctly other data except for the training one. A simple method to test if the classifier is overfitting is to compare the gap between the accuracy on the training set and the test set. If the accuracy of the training set is much higher than the test one, then the model is overfitting.[1]

To avoid the overfitting, some techniques exist such as the **dropouts**. Dropout consists to randomly dropping connections between units. It forces the network to find new paths and generalise. It will help to enhance the confidence over our model. But if we want to improve the accuracy of our classifier, the hyperparameters tuning is mandatory.

The hyperparameters are values that must be initialise by hands to the network. The classifier can't learn those values. In CNN, we can tweak the number of layers, batch size, loss function, and/or the optimizer used. Numerous hyperparameters could be tuned but we need to take the best set of hyperparameters for the CNN.

Some best practices exist concerning the finetuning of hyperparameters. The tuning process, for this project, will focus on those parameters to get the maximum accuracy :

- Learning Rate : the lower the learning rate is, the more the network converge to the global minima.
- Number of layers : the more layers are implemented, the more accuracy we can get. But the computing process will be longer.
- Optimizers and Loss function
- Batch size and Number of Epochs

Chapter 2

Results and Discussion

2.1 Results

The first architecture of our CNN was simple. It was designed by two layers (input and output), a learning rate of 0.01, the cross entropy loss function, and the stochastic gradient descent (SGD) optimizer. Running the training and the test on the fruit dataset, with 2 epochs, we have reached very low accuracy (between 1-10%).

This is where the finetuning process begins. We tried in a first step to tweak the parameters without changing the architecture of the CNN. The learning rate was reduced to 0.001, the epochs was enhanced to 5. Keeping the other parameters at their first state, we have reached an accuracy of 77%. The result is satisfying and we could have stopped there. But the simple test of accuracy on he training set showed that the training set accuracy was higher (97%). We need to get rid of the overfitting by tuning other parameters (fig.2.1).

Entrée [11]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images,labels = data
        outputs = convNet(images)
        _,predicted = torch.max(outputs.data,1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print("Accuracy of the network on the 10000 test images : %d %% " %(100 * c
```

```
Accuracy of the network on the 10000 test images : 77 %
```

Entrée [12]:
```
# test overfitting
correct = 0
total = 0
with torch.no_grad():
    for data in train_loader:
        images,labels = data
        outputs = convNet(images)
        _,predicted = torch.max(outputs.data,1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print("Accuracy of the network on the training images : %d %% " %(100 * cor
```

```
Accuracy of the network on the training images : 95 %
```

Figure 2.1: Basic architecture CNN

We then applied dropouts and batch normalization to enhance the accuracy and we have reached the maximum we could get (94%). Tuning the architecture did not change the ending result by adding a hidden layer (fig.2.2).

Entrée [11]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images,labels = data
        outputs = convNet(images)
        _,predicted = torch.max(outputs.data,1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print("Accuracy of the network on the 10000 test images : %d %% " %(100 * c
```

```
Accuracy of the network on the 10000 test images : 93 %
```

Entrée [12]:
```
# test overfitting
correct = 0
total = 0
with torch.no_grad():
    for data in train_loader:
        images,labels = data
        outputs = convNet(images)
        _,predicted = torch.max(outputs.data,1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print("Accuracy of the network on the training images : %d %% " %(100 * cor
```

```
Accuracy of the network on the training images : 99 %
```

Figure 2.2: Tuned hyperparameters architecture CNN

2.2 Discussion

The results are quite promising, tweaking the hyperparameters permits to enhance the accuracy of our a CNN. But the lack of knowledge about the complex subject of deep learning let place to an important amount of potential errors. We can't have confidence about the architecture proposed to classify the fruit dataset. Therefore, we thought that using transfer learning was the best idea, because we could work with pre-trained models.

Sadly, the computing power of the machine was to low and we could not use the GPU for faster results. Hence, we tried to go with the CPU but the process time was too long (more than 5 hours), without any results. Further information are not available to be confident in our results.

We also changed optimizer (Adam) to test out the accuracy to check any improvement, but it was not interesting. For further analyses, it would interesting to design a more complex CNN and finetuning the loss function. And maybe working with a good GPU could allow us to work with transfer learning.

Conclusion

Our CNN architecture seems to classify with great accuracy the fruits in our dataset. Following the best practices to tweak the hyperparameters, we managed to reach high accuracy. The problem still remain in our lack of practices and knowledge on deep learning. We can not be sure of the confidence of our results.

But the goal was to get the best accuracy possible, and in our state we have reached the best possible solution. Pytorch, even with his level of complexity, remains a great tool for image recognition.

Bibliography

[1] F O R L Arge and Cale I Mage. V d c n l -s i r. pages 1–14, 2015.

[2] Alex Krizhevsky and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. pages 1–9.

[3] 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. 2017.

[4] Patrice Y Simard, Dave Steinkraus, John C Platt, One Microsoft Way, and Redmond Wa. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. 2000.