Master of Bio-Informatics

Project Deep Learning

Julie Blasquiz

Supervisor: LE Van Linh

DEA (Donnes : De l'Entrepot a l'Analyse)

February 14, 2019

Contents

1 Introduction

Deep learning is a sub-field of machine learning dealing with algorithms inspired by the structure and function of the brain called artificial neural networks. In other words, it mirrors the functioning of our brains. Deep learning algorithms are similar to how nervous system structured where each neuron connected each other and passing information[Horikawa et al. (2019)]. Because the human brain is very good at object recognition, deep learning, based on the same logical, is more and more used in this field. With the important amount of data, classify the data obtained is a first step in data analysis.
The goal of this project is to propose a CNN architecture to classify a series of fruit images[Mureşan & Oltean (2018)] into different classes using PyTorch framework, an open-source machine learning library for Python. The second objectives are to compare the result of the network on several hyper-parameters, and evaluate the accuracy of the model in predicting the labels for the images in the fruit data set.

2 Material and method

2.1 Material

The data set used in this project is named Fruits 360 dataset. Fruits 360 dataset contains a series of fruit pictures. The images were obtained by filming the fruits while they are rotated by a motor and then extracting frames[Mureşan & Oltean (2018)]. Fruits were scaled to fit a 100x100 pixels image. The resulted data set has 57167 images of fruits spread across 83 labels. These images are shared between two files : the training set file (with 42798 images) and the testing set file (with 14369 images). The Figure 1 proposes a recap of the organisation of the Fruits 360 dataset.

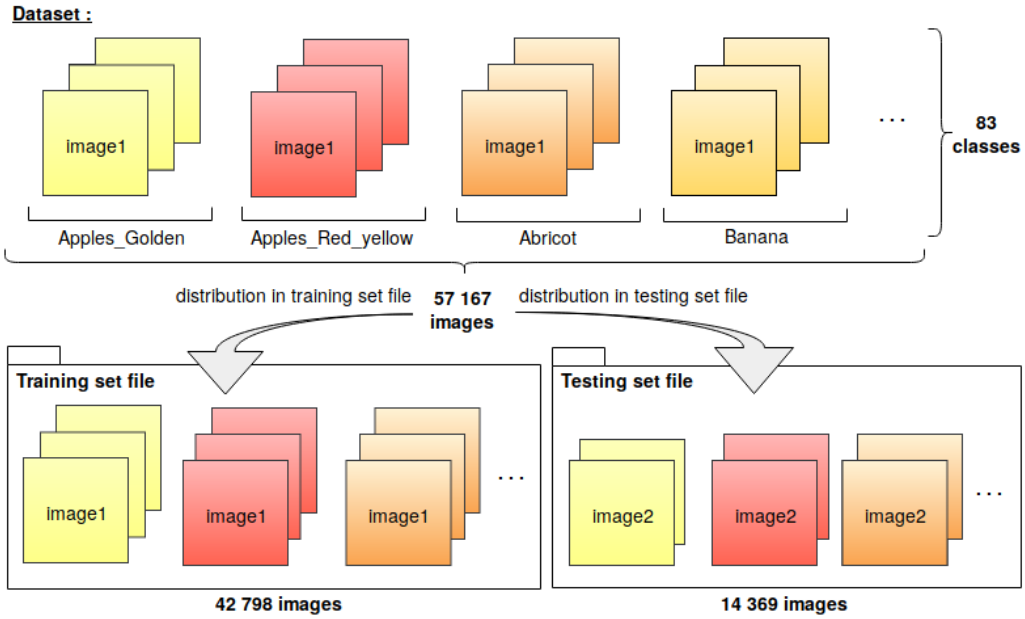


Figure 1: *Organisation of the data set*

The figure 2 shows some of the images with their classes as labels.

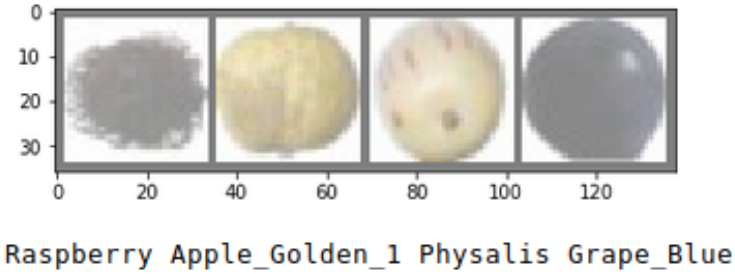


Figure 2: *Some images from the training dataset file*

2.2 Method

To create a convolutional neural network on this dataset, the network is implemented by using PyTorch framework. All the pictures of the dataset are first resized, from 100x100 pixels to 32x32 pixels, via a JavaScript script, in the Annexe A, executed with ImageJ, an image processing program[Girish & Vijayalakshmi (2004)]. The script allowing to create the convolutional neural network, in the Annexe B, is written in the programming language Python3.

A convolutional neural network is created. It makes use of two convolutional layers, one pooling layer (used two times), one Rectified Linear Unit layer (used two times), tree fully connected layers and one loss layer. The CNN architecture is illustrated in figure 3.

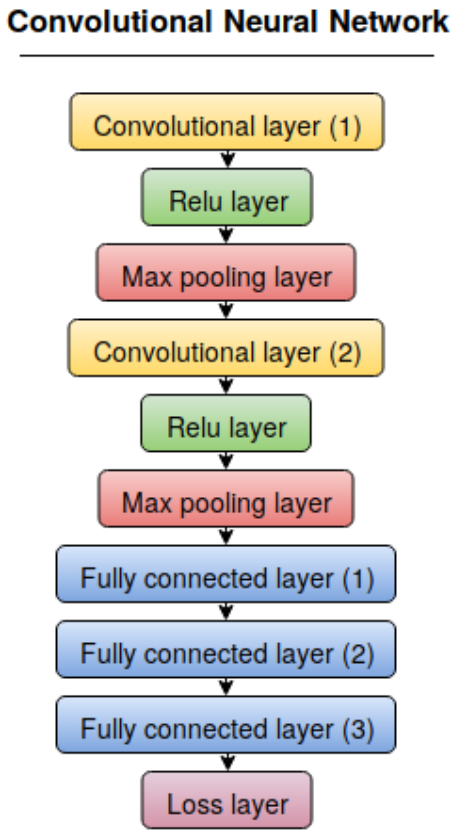


Figure 3: *CNN architecture*

Each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer, and then a Max pooling layer. A convolutional layer allows to do a convolution operation on the input. A Relu layer applies the activation function max(0, x). It increases its

nonlinear properties. The Max pooling layer is used on one hand to reduce the spatial dimensions of the representation and to reduce the amount of computation done in the network. The other use of pooling layers is to control over fitting. The filters of the Max pooling layer have a size of 2 x 2 with a stride 2. This effectively reduces the input to a quarter of its original size[Mureşan & Oltean (2018)].
The second convolutional layer is then followed by three fully connected layer where each neuron from a fully connected layers is linked to each output of the previous layer. The last layer is a loss layer used to penalize the network for deviating from the expected output[Mureşan & Oltean (2018)]. An optimizer is also defined to update rule. The Table 1 contains all the parameters applied in the convolutional neural network.

Table 1: *Convolutional neural network parameters.*

CNN elements	**Parameters**
Convolutional layer 1	(3,6,5)
Max pooling layer	(2,2)
Convolutional layer 2	(6,16,5)
Fully connected layer 1	(16*5*5, 120)
Fully connected layer 2	(120, 84)
Fully connected layer 3	(84, 83)

To evaluate the efficiency of the network, this one was tried with different hyper-parameters: two different optimizers were used, Adam and SGD, and the number of epochs was also tested. The results of the comparisons were then compared (loss and accuracy).

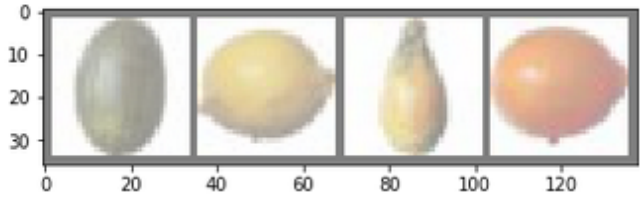
3 Results

The accuracy and loss result of the convolutional neural network depends on the hyper-parameters used. With the SGD optimizer and a number of epochs of one, the accuracy is off 73% and the loss result 0.562. With the SGD optimizer and a number of epochs of two, the accuracy switch to 87% and the loss result in 0.183. With the Adam optimizer and a number of epochs of one, the accuracy is 85% and the loss result in 0.246. And with the Adam optimizer and a number of epochs of two, the accuracy is 89% and the loss result in 0.183.

Table 2: *CNN hyper-parameters comparisons.*

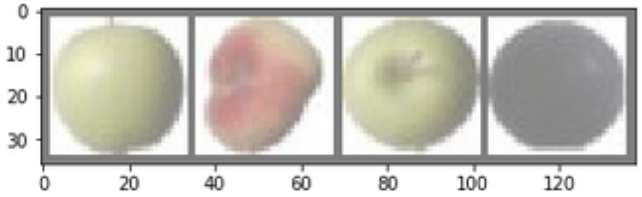
CNN hyper-parameters	**Loss result**	**Accuracy**
Optimizer : SGD + momentum (0.9), lr = 0.001 Nb Epoch : 1	0.562	73%
Optimizer : SGD + momentum (0.9), lr = 0.001 Nb Epoch : 2	0.183	87%
Optimizer : Adam, lr = 0.001 Nb Epoch : 1	0.246	85%
Optimizer : Adam, lr = 0.001 Nb Epoch : 2	0.183	89%

The figure 4 shows some examples of the prediction results on the testing set file images.




```
GroundTruth:  Mulberry Lemon_Meyer Cactus_fruit Tomato_1

Predicted:  Mulberry Lemon_Meyer Cactus_fruit Tomato_1
```




```
GroundTruth:  Apple_Granny_Smith  Pear Apple_Granny_Smith Grape_Blue

Predicted:  Apple_Granny_Smith  Plum Apple_Granny_Smith Grape_Blue
```

Figure 4: *Prediction results*

4 Discussion

As it figures in the Table 2, the Adam optimizer is more performing than the SGD optimizer on this CNN and dataset (for an epoch of one, 73% of accuracy with SGD against 85% with Adam).
More the number of epochs is important, more the loss resulting decrease. The results of the prediction when the epoch is two, compared with the result when the number of epochs is one, can be increased from 4% to 14%.
Thus, with an accuracy of 89% and a loss result of 0.183, the Adam optimizer and an epoch number of 2 is the more efficiency combination tested.

The 89% of CNN accuracy can be improved by several options : To increase the number of epochs is one of them. Here, the maximum epoch number tested is two because of the capacity of the computer (computer device : CPU). With another device like CUDA, the computer should support to increase the number of epoch more than two.
Another option is to improve the number of convolutional layers. The actual CNN comport two convolutional layers. This number can be improved followed for each convolutional layers added by a Relu layer and a pooling layer.
The last option to obtain a better accuracy is to replace the actual fully connected layers by convolutional layers. The result of these substitutes allows to improve the prediction of the CNN[Mureşan & Oltean (2018)].

5 Conclusion

In the purpose of classify a series of fruit images into different classes, a CNN architecture was proposed including : two convolutional layers, one pooling layer (used two times), one Rectified Linear Unit layer (used two times), tree fully connected layers and one loss layer. The CNN was tested using different hyper-parameters and the result compared to each other. Finally, the accuracy of the model was evaluated by predicting the labels for the images in the data set. The accuracy of the model turn around 89% of accuracy. This percentage can be higher in increasing the number of epochs, adding more convolutional layers, or replacing the fully connected layers by some convolutional layers.

References

Girish, V. & Vijayalakshmi, A. (2004), 'Affordable image analysis using nih image/imagej', *Indian J Cancer* **41**(1), 47.

Horikawa, T., Aoki, S. C., Tsukamoto, M. & Kamitani, Y. (2019), 'Characterization of deep neural network features by decodability from human brain activity', *Scientific Data* **6**, 190012.
URL: *https://doi.org/10.1038/sdata.2019.12*

Mureşan, H. & Oltean, M. (2018), 'Fruit recognition from images using deep learning', *Acta Universitatis Sapientiae, Informatica* **10**(1), 26–42.
URL: *https://doi.org/10.2478/ausi-2018-0002*

Appendices

A JavaScript script executed in ImageJ

```
function process(name){
var imp = IJ.openImage(name);

IJ.run(imp, "Size...", "width=32 height=32 constrain average interpolation=Bilinear");
IJ.saveAs(imp, "Jpeg", name);
}

var od=new OpenDialog("choose file");
var folder=od.getDirectory();

var dir=new java.io.File(folder);
var files=dir.listFiles();


for(var i=0;i<files.length ;i++){
  process(files[i]);
}
```

B Python3 notebook executed in Jupyter

```
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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

from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, datasets

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

testset = datasets.ImageFolder(path_test, transform = transforms.ToTensor())
testloader = DataLoader(testset, batch_size = 4, shuffle = True)

classes = ('Apple_Braeburn','Apple_Golden_1','Apple_Golden_2','Apple_Golden_3',
        'Apple_Granny_Smith','Apple_Red_1','Apple_Red_2','Apple_Red_3',
        'Apple_Red_Delicious','Apple_Red_Yellow','Apricot','Avocado',
        'Avocado_ripe','Banana','Banana_Red','Cactus_fruit','Cantaloupe_1',
        'Cantaloupe_2','Carambula','Cherry_1','Cherry_2','Cherry_Rainier',
        'Cherry_Wax_Black','Cherry_Wax_Red','Cherry_Wax_Yellow','Clementine',
        'Cocos','Dates','Granadilla','Grape_Blue','Grapefruit_Pink',
         'Grapefruit_White','Grape_Pink','Grape_White_2','Guava','Huckleberry',
         'Kaki','Kiwi','Kumquats','Lemon','Lemon_Meyer','Limes','Lychee','Mandarine',
         'Mango','Maracuja','Melon_Piel_de_Sapo','Mulberry','Nectarine','Orange',
         'Papaya','Passion_fruit','Peach','Peach_Flat','Pear','Pear_Abate',
         'Pear_Monster','Pear_Williams','Pepino','Physalis','Physalis_with_Husk',
         'Pineapple','Pineapple_Mini','Pitahaya_Red','Plum','Pomegranate','Quince',
         'Rambutan','Raspberry','Redcurrant','Salak','Strawberry','Strawberry_Wedge',
         'Tamarillo','Tangelo','Tomato_1','Tomato_2','Tomato_3','Tomato_4',
         'Tomato_Cherry_Red','Tomato_Maroon','Walnut')
```

```
# functions to show an image
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 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 = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()

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

for epoch in range(2):  # loop over the dataset multiple times
```

```
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

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

print('Finished Training')

dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))

outputs = net(images)

_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))

correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
```

```
        outputs = net(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 * correct / total))
```