Master 2 Image Mining Course
Lab 2 - Deep Learning with PyTorch: CIFAR10 object classification

Émilie THOMÉ

October 2020

Introduction

In this lab-work, we studied different structures of deep convolutional neural networks to work on image classification using the PyTorch library. We used the CIFAR10 database with ten categories of objects. The notebook guided us to construct an artificial neural network to classify them.
Github repository for the notebook

1 Default CNN

The default CNN was a simple one, defined as follow:

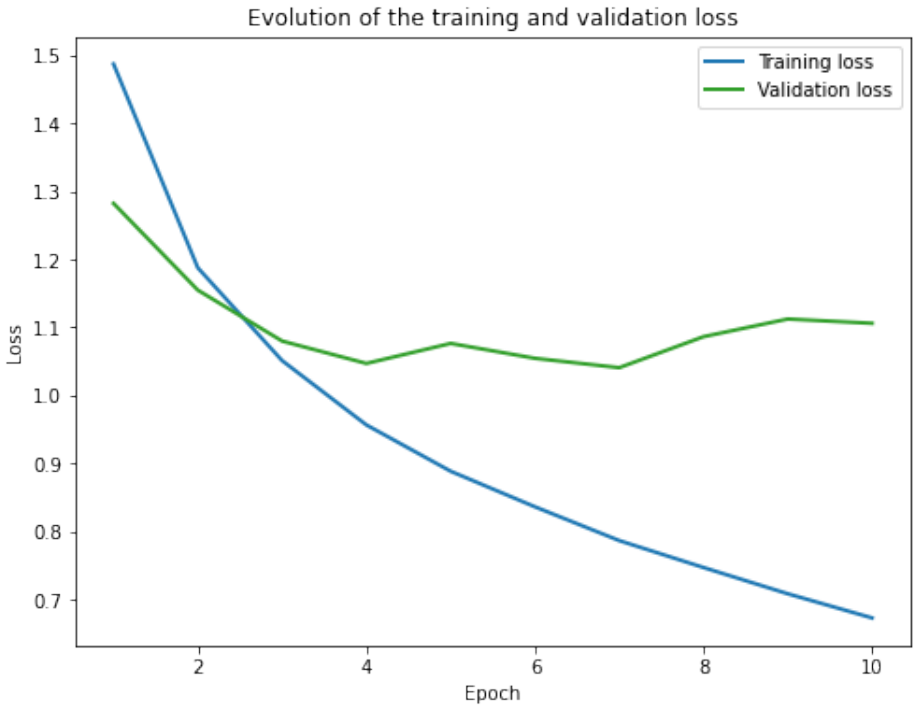
```
class SimpleConvolutionalNetwork(nn.Module):
    def __init__(self):
        super(SimpleConvolutionalNetwork, self).__init__()https://engmrk.com/len
        self.conv1 = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # cf comments in forward() to have step by step comments
        # on the shape (how we pass from a 3x32x32 input image
        # to a 18x16x16 volume)
        self.fc1 = nn.Linear(18 * 16 * 16, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        """
        Forward pass,
        x shape is (batch_size, 3, 32, 32)
        (color channel first)
        in the comments, we omit the batch_size in the shape
```

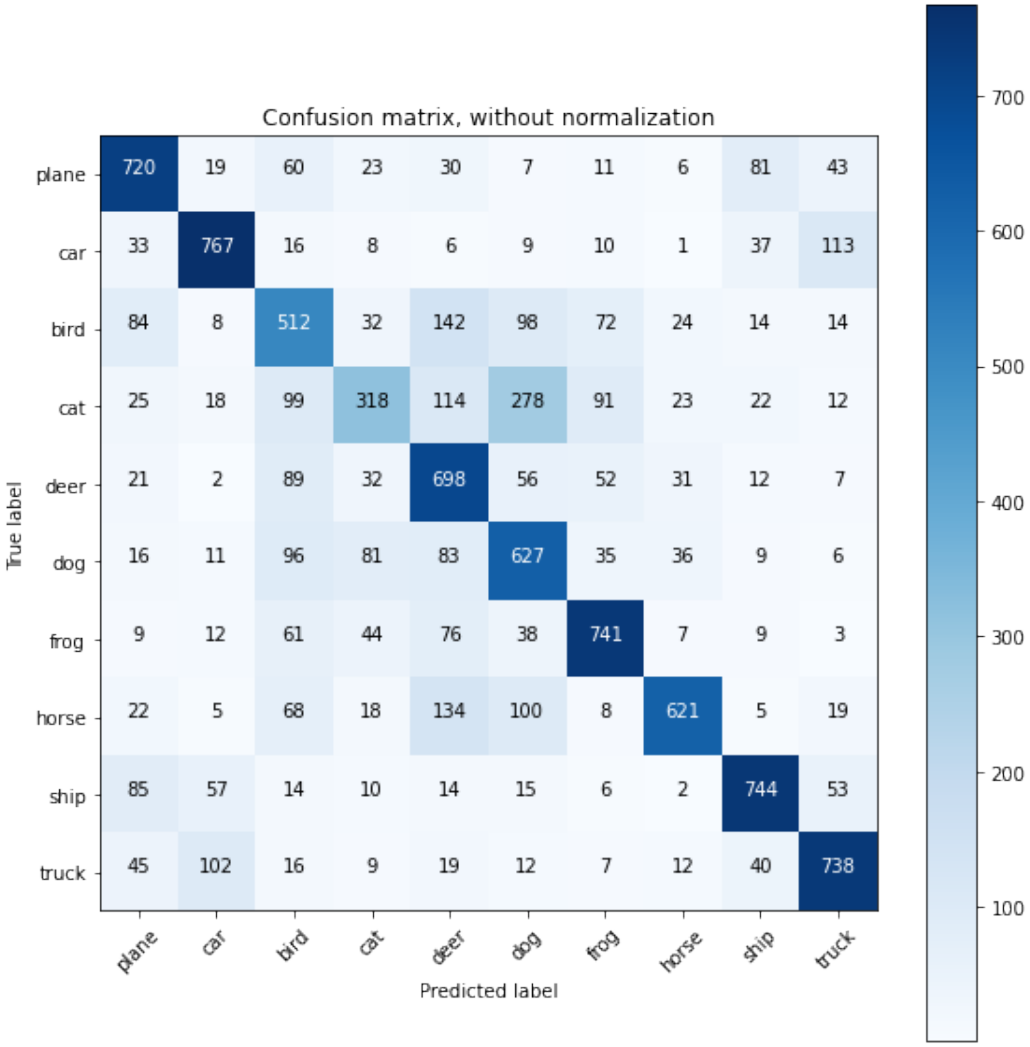
```
        """
        # shape : 3x32x32 -> 18x32x32
        x = F.relu(self.conv1(x))
        # 18x32x32 -> 18x16x16
        x = self.pool(x)
        # 18x16x16 -> 4608
        x = x.view(-1, 18 * 16 * 16)
        # 4608 -> 64
        x = F.relu(self.fc1(x))
        # 64 -> 10
        # The softmax non-linearity is applied later
        x = self.fc2(x)
        return x
```

The results were :




```
Accuracy of the network on the 40000 train images: 75.91 %
Accuracy of the network on the 10000 validation images: 64.82 %
Accuracy of the network on the 10000 test images: 64.86 %
```

Class	Accuracy (%)
plane	72.00
car	76.70
bird	51.20
cat	31.80
deer	69.80
dog	62.70
frog	74.10
horse	62.10
ship	74.40
truck	73.80



2 Trials

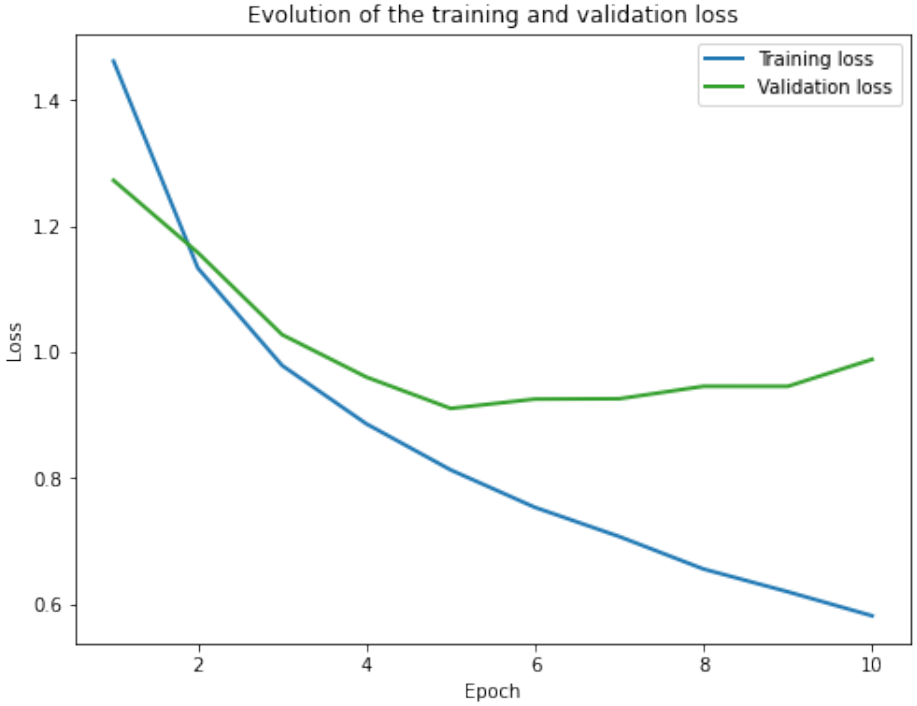
The default CNN was not efficient enough and in order to improve it I made some tests. It permitted me to be more comfortable with the CNN structure, the influence of some variables and the code was PyTorch.

2.1 New layers

I added the following layers in the previous CNN structure:

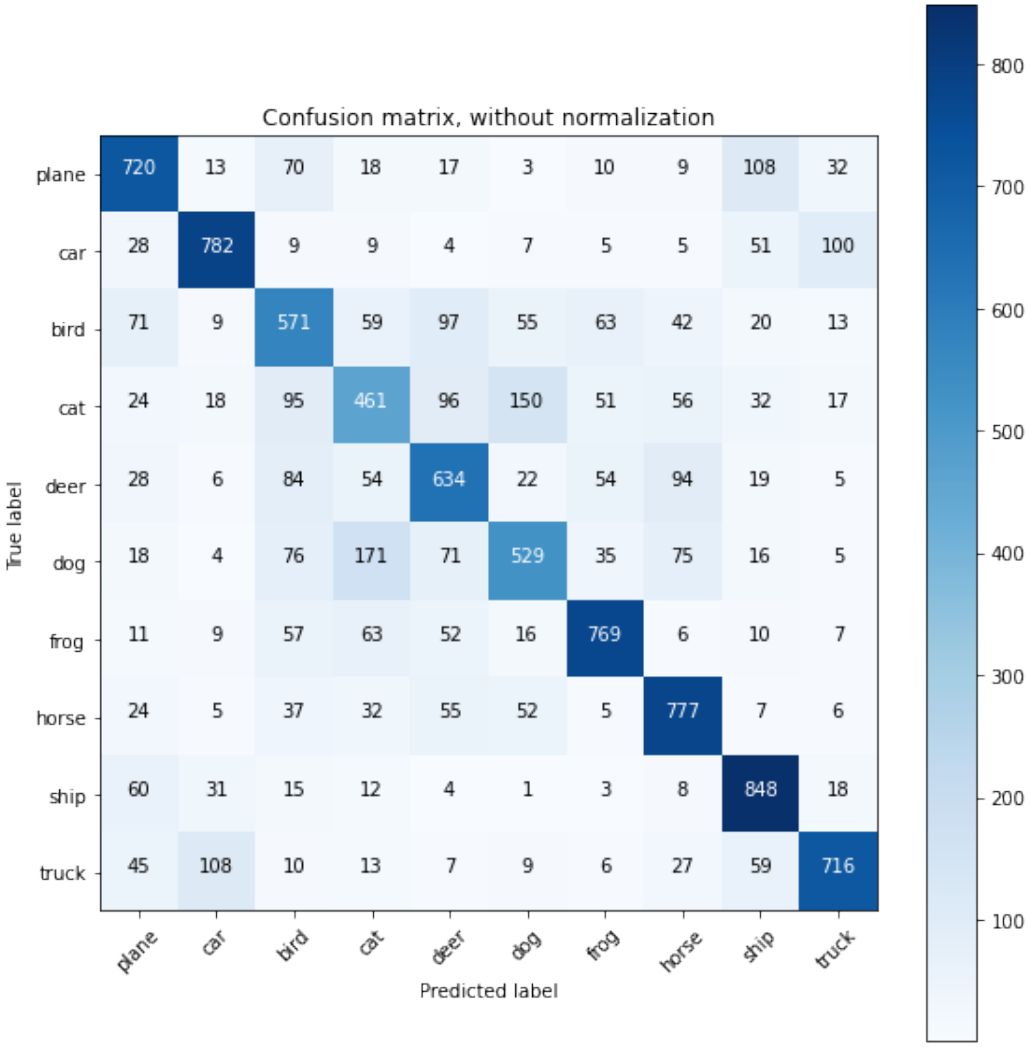
```
self.conv2 = nn.Conv2d(18, 25, kernel_size=4, stride=1, padding=1)
self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0) :
```

The new results were :




```
Accuracy of the network on the 40000 train images: 75.10 %
Accuracy of the network on the 10000 validation images: 68.41 %
Accuracy of the network on the 10000 test images: 68.07 %
```

Class	Accuracy (%)
plane	72.00
car	78.20
bird	57.10
cat	46.10
deer	63.40
dog	52.90
frog	76.90
horse	77.70
ship	84.80
truck	71.60



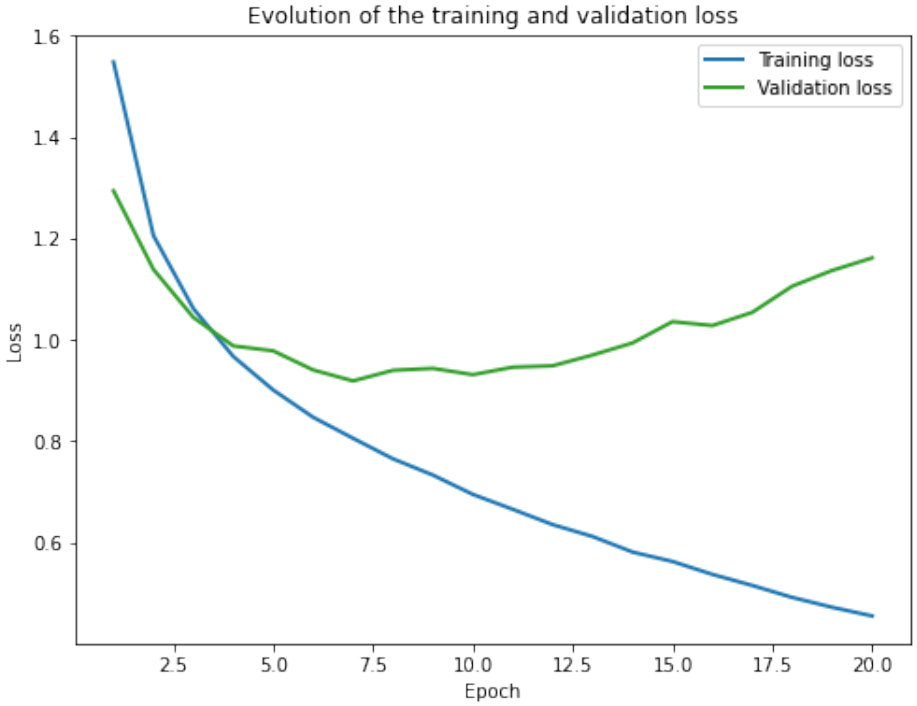
We can see that adding layers is a mean to obtain better results.

2.2 More epochs

I took the previous code with the added layer and I added epochs :

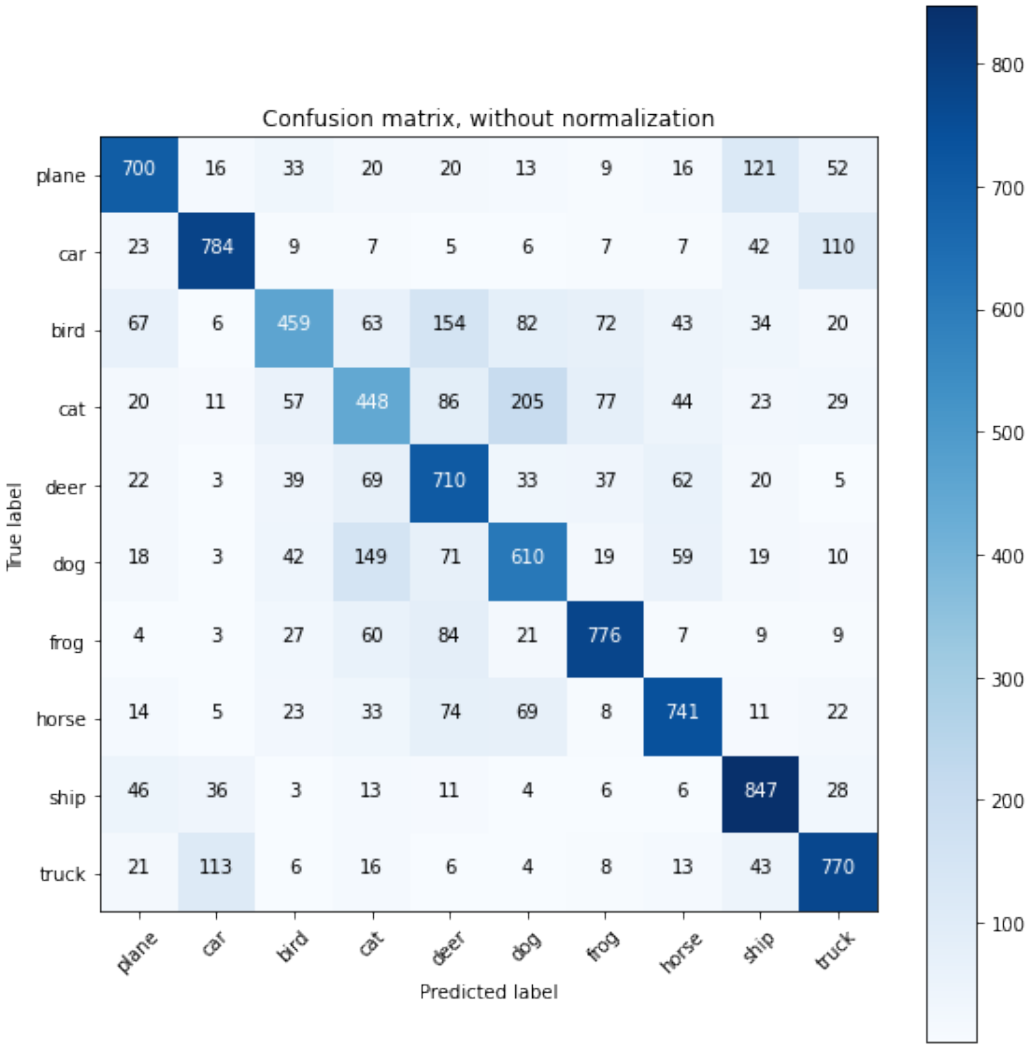
`n_epochs=20` :

The results :




```
Accuracy of the network on the 40000 train images: 74.95 %
Accuracy of the network on the 10000 validation images: 67.89 %
Accuracy of the network on the 10000 test images: 68.45 %
```

Class	Accuracy (%)
plane	70.00
car	78.40
bird	45.90
cat	44.80
deer	71.00
dog	61.00
frog	77.60
horse	74.10
ship	84.70
truck	77.00



Adding epochs can be a good idea, but not too much unless the CNN became specialised with the training set and can not adapt very well with the test set.

2.3 More channels

I tested to add more channels in the second convolutional layer.

30 Channels out

```
self.conv2 = nn.Conv2d(18, 30, kernel_size=4, stride=1, padding=1)
```

results :

```
Accuracy of the network on the 40000 train images: 77.88 %
Accuracy of the network on the 10000 validation images: 68.28 %
Accuracy of the network on the 10000 test images: 67.99 %
```

40 Channels out

```
self.conv2 = nn.Conv2d(18, 40, kernel_size=4, stride=1, padding=1)
```

results :

```
Accuracy of the network on the 40000 train images: 82.27 %
Accuracy of the network on the 10000 validation images: 70.39 %
Accuracy of the network on the 10000 test images: 69.67 %
```

Adding channels is great to make the CNN better.

2.4 Changing the size of train and test set

I changed the test and train set to 10000/40000 to 5000/45000.

70 Channels out

```
self.conv2 = nn.Conv2d(18, 70, kernel_size=4, stride=1, padding=1)
```

results :

```
Accuracy of the network on the 45000 train images: 78.11 %
Accuracy of the network on the 5000 validation images: 71.88 %
Accuracy of the network on the 10000 test images: 70.79 %
```

100 Channels out

```
self.conv2 = nn.Conv2d(18, 100, kernel_size=4, stride=1, padding=1)
```

results :

```
Accuracy of the network on the 45000 train images: 84.31 %
Accuracy of the network on the 5000 validation images: 73.04 %
Accuracy of the network on the 10000 test images: 71.63 %
```

200 Channels out

```
self.conv2 = nn.Conv2d(18, 200, kernel_size=4, stride=1, padding=1)
```

results :

```
Accuracy of the network on the 45000 train images: 87.78 %
Accuracy of the network on the 5000 validation images: 74.78 %
Accuracy of the network on the 10000 test images: 72.91 %
```

This is actually the best result I had.

300 Channels out

```
self.conv2 = nn.Conv2d(18, 300, kernel_size=4, stride=1, padding=1)
```

results :

```
Accuracy of the network on the 45000 train images: 83.45 %
Accuracy of the network on the 5000 validation images: 72.40 %
Accuracy of the network on the 10000 test images: 71.16 %
```

I think that increasing the train set and diminishing the test set was a good way to improve the flexibility of the CNN.

2.5 Batch size

I changed the batch size :

```
train_history, val_history = train(net,
                                   batch_size=64,
                                   n_epochs=10,
                                   learning_rate=0.001)
```

And it gave :

```
Accuracy of the network on the 40000 train images: 75.66 %
Accuracy of the network on the 10000 validation images: 67.13 %
Accuracy of the network on the 10000 test images: 67.78 %
```

I also doubled the number of epoch in another test but it was not more relevant than this.

2.6 Learning rate

I augmented the learning rate to 0.1 :

```
Accuracy of the network on the 40000 train images: 51.97 %
Accuracy of the network on the 10000 validation images: 51.31 %
Accuracy of the network on the 10000 test images: 49.65 %
```

And diminished it to 0.0001:

```
Accuracy of the network on the 40000 train images: 63.77 %
Accuracy of the network on the 10000 validation images: 62.74 %
Accuracy of the network on the 10000 test images: 61.26 %
```

There is nothing to touch here.

2.7 Adding more fully connected layers

I added to a code with a convolutional layer giving an out channels number of 300 more fully connected layers :

```
self.conv2 = nn.Conv2d(18, 300, kernel_size=4, stride=1, padding=1)
self.fc1 = nn.Linear(self.flattened_size, 200) <= RELU
self.fc2 = nn.Linear(200, 50) <= RELU
self.fc3 = nn.Linear(50, 10)

Accuracy of the network on the 45000 train images: 88.42 %
Accuracy of the network on the 5000 validation images: 73.08 %
Accuracy of the network on the 10000 test images: 72.34 %
```

This is also one of my best results for now on. Adding fully connected layers and convolutional layers is the key. I think that having only ten features in the end permit to distinguish the ten categories of objects.

2.8 LeNet-5

I tried to implement a **LeNet-5** CNN (adapting it to the case of colored images) :

```
class MyConvolutionalNetwork(nn.Module):
    def __init__(self):
        super(MyConvolutionalNetwork, self).__init__()

        #### START CODE: ADD NEW LAYERS ####
        # (do not forget to update `flattened_size`:
        # the input size of the first fully connected layer self.fc1)
        self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.avg_pool = nn.AvgPool2d(kernel_size=2, stride=2, padding=0)

        self.conv1 = nn.Conv2d(3, 18, kernel_size=5, stride=1, padding=1)
        self.conv2 = nn.Conv2d(18, 48, kernel_size=5, stride=1, padding=1)

        # Size of the output of the last convolution:
        self.flattened_size = 48 * 6 * 6
        ### END CODE ###

        self.fc1 = nn.Linear(self.flattened_size, 360)
        self.fc2 = nn.Linear(360, 252)
        self.fc3 = nn.Linear(252, 30)
        self.fc4 = nn.Linear(30, 10)

    def forward(self, x):
        """
        Forward pass,
```

```
        x shape is (batch_size, 3, 32, 32)
        (color channel first)
        in the comments, we omit the batch_size in the shape
        """

        #### START CODE: USE YOUR NEW LAYERS HERE ####
        x = F.relu(self.conv1(x))
        x = F.relu(self.avg_pool(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.avg_pool(x))

        # Check the output size
        output_size = np.prod(x.size()[1:])
        assert output_size == self.flattened_size,\
                "self.flattened_size is invalid {} !=
                {}".format(output_size, self.flattened_size)

        x = x.view(-1, self.flattened_size)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        #### END CODE ####

        return x
```

I first used TANH and SOFTMAX as used by LeNet-5 but I replaced them by RELU because it gave me better results (with a batch size of 64 and a number of epochs of 30) :

```
Accuracy of the network on the 45000 train images: 80.99 %
Accuracy of the network on the 5000 validation images: 70.78 %
Accuracy of the network on the 10000 test images: 69.44 %
```

2.9 Edit LeNet-5

I editted my previous code by adding convolutional and fully connected layers and by only using MAXPOOL instead of AVGPOOL :

```
class MyConvolutionalNetwork(nn.Module):
    def __init__(self):
        super(MyConvolutionalNetwork, self).__init__()

        #### START CODE: ADD NEW LAYERS ####
        # (do not forget to update 'flattened_size':
        # the input size of the first fully connected layer self.fc1)
        self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
```

```
        self.avg_pool = nn.AvgPool2d(kernel_size=2, stride=2, padding=0)

        self.conv1 = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(18, 54, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(54, 108, kernel_size=3, stride=1, padding=1)

        # Size of the output of the last convolution:
        self.flattened_size = 108 * 4 * 4
        ### END CODE ###

        self.fc1 = nn.Linear(self.flattened_size, 360)
        self.fc2 = nn.Linear(360, 252)
        self.fc3 = nn.Linear(252, 99)
        self.fc4 = nn.Linear(99, 30)
        self.fc5 = nn.Linear(30, 10)

    def forward(self, x):
        """
        Forward pass,
        x shape is (batch_size, 3, 32, 32)
        (color channel first)
        in the comments, we omit the batch_size in the shape
        """

        #### START CODE: USE YOUR NEW LAYERS HERE ####
        x = F.relu(self.conv1(x))
        x = F.relu(self.max_pool(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.max_pool(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.max_pool(x))

        # Check the output size
        output_size = np.prod(x.size()[1:])
        assert output_size == self.flattened_size,\
                "self.flattened_size is invalid {} !=
                {}".format(output_size, self.flattened_size)

        x = x.view(-1, self.flattened_size)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        #### END CODE ####
```

```
        return x
```

Giving :

```
Accuracy of the network on the 45000 train images: 84.61 %
Accuracy of the network on the 5000 validation images: 73.86 %
Accuracy of the network on the 10000 test images: 72.82 %
```

More layers :

```
class MyConvolutionalNetwork(nn.Module):
    def __init__(self):
        super(MyConvolutionalNetwork, self).__init__()

        #### START CODE: ADD NEW LAYERS ####
        # (do not forget to update `flattened_size`:
        # the input size of the first fully connected layer self.fc1)
        self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.avg_pool = nn.AvgPool2d(kernel_size=2, stride=2, padding=0)

        self.conv1 = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(18, 54, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(54, 108, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(108, 216, kernel_size=3, stride=1, padding=1)

        # Size of the output of the last convolution:
        self.flattened_size = 216 * 2 * 2
        ### END CODE ###

        self.fc1 = nn.Linear(self.flattened_size, 360)
        self.fc2 = nn.Linear(360, 252)
        self.fc3 = nn.Linear(252, 99)
        self.fc4 = nn.Linear(99, 30)
        self.fc5 = nn.Linear(30, 10)

    def forward(self, x):
        """
        Forward pass,
        x shape is (batch_size, 3, 32, 32)
        (color channel first)
        in the comments, we omit the batch_size in the shape
        """

        #### START CODE: USE YOUR NEW LAYERS HERE ####
        x = F.relu(self.conv1(x))
        x = F.relu(self.max_pool(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.max_pool(x))
```

```
        x = F.relu(self.conv3(x))
        x = F.relu(self.max_pool(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.max_pool(x))

        # Check the output size
        output_size = np.prod(x.size()[1:])
        assert output_size == self.flattened_size,\
                "self.flattened_size is invalid {} !=
                {}".format(output_size, self.flattened_size)

        x = x.view(-1, self.flattened_size)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        #### END CODE ####

        return x
```

Giving :

```
Accuracy of the network on the 45000 train images: 87.36 %
Accuracy of the network on the 5000 validation images: 75.46 %
Accuracy of the network on the 10000 test images: 73.54 %
```

Remove the first MAXPOOL :

```
class MyConvolutionalNetwork(nn.Module):
    def __init__(self):
        super(MyConvolutionalNetwork, self).__init__()

        #### START CODE: ADD NEW LAYERS ####
        # (do not forget to update `flattened_size`:
        # the input size of the first fully connected layer self.fc1)
        self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.avg_pool = nn.AvgPool2d(kernel_size=2, stride=2, padding=0)

        self.conv1 = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(18, 54, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(54, 108, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(108, 216, kernel_size=3, stride=1, padding=1)

        # Size of the output of the last convolution:
        self.flattened_size = 216 * 4 * 4
        ### END CODE ###
```

```
        self.fc1 = nn.Linear(self.flattened_size, 360)
        self.fc2 = nn.Linear(360, 252)
        self.fc3 = nn.Linear(252, 99)
        self.fc4 = nn.Linear(99, 30)
        self.fc5 = nn.Linear(30, 10)

    def forward(self, x):
        """
        Forward pass,
        x shape is (batch_size, 3, 32, 32)
        (color channel first)
        in the comments, we omit the batch_size in the shape
        """

        #### START CODE: USE YOUR NEW LAYERS HERE ####
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.max_pool(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.max_pool(x))
        x = F.relu(self.conv4(x))
        x = F.relu(self.max_pool(x))

        # Check the output size
        output_size = np.prod(x.size()[1:])
        assert output_size == self.flattened_size,\
                "self.flattened_size is invalid {} !=
                {}".format(output_size, self.flattened_size)

        x = x.view(-1, self.flattened_size)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        #### END CODE ####

        return x
```

Results :

```
Accuracy of the network on the 45000 train images: 91.36 %
Accuracy of the network on the 5000 validation images: 78.98 %
Accuracy of the network on the 10000 test images: 77.96 %
```

I remarked that the validation loss was increasing after six epochs (with a batch size of 64) so I changed :

```
train_history, val_history = train(net,
                                   batch_size=64,
                                   n_epochs=6,
                                   learning_rate=0.001)
```

And had the following result which is my best attempt :

```
Accuracy of the network on the 45000 train images: 90.75 %
Accuracy of the network on the 5000 validation images: 78.28 %
Accuracy of the network on the 10000 test images: 78.03 %
```