RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

La libraire PyTorch

DSM/LabIA
22/11/2024

La Libraire PyTorch : la librairie du deep learning

Quand utiliser PyTorch ?

- Différences entre PyTorch et scikit-learn
  - Pytorch
    - Flexibilité dans la création des architectures de réseaux de neurones
    - Utilisation du GPU
  - Quand utiliser Pytorch à la place de scikit-learn ?
    - Dès qu'on veut faire du deep learning (réseaux de neurones profond)

La régression linéaire sur Pytorch

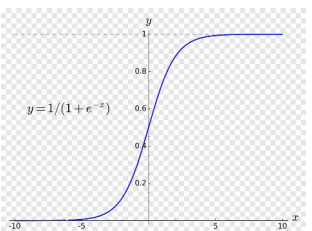
```python
import torch
import torch.nn as nn
```

```python
# Create class
class LinearRegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        out = self.linear(x)
        return out
```

La régression logistique sur Pytorch

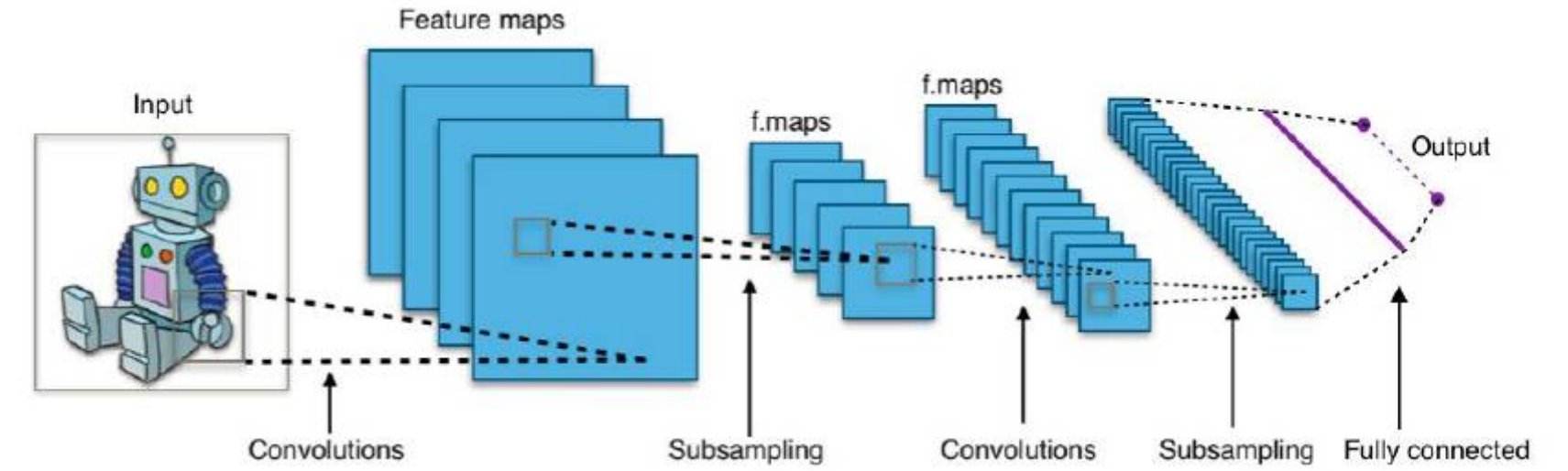
```python
# build custom module for logistic regression
class LogisticRegression(torch.nn.Module):
    # build the constructor
    def __init__(self, n_inputs, n_outputs):
        super(LogisticRegression, self).__init__()
        self.linear = torch.nn.Linear(n_inputs, n_outputs)
    # make predictions
    def forward(self, x):
        y_pred = torch.sigmoid(self.linear(x))
        return y_pred
```



Exemple de couches Pytorch

- Couche dense ou linéaire : Linear
- Convolution : Conv2D
- Fonction d’activation : ReLU
- Max Pooling : MaxPool2D
- Dropout : Dropout

Schéma d’un réseau convolutionnel



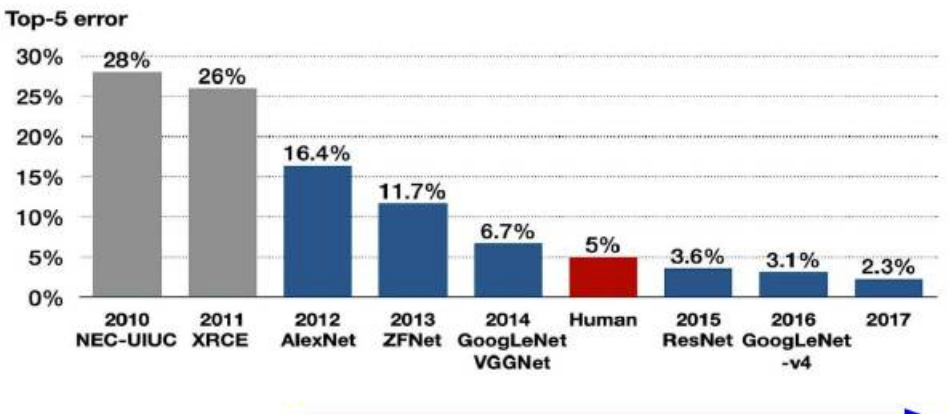
ImageNet : présentation

ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

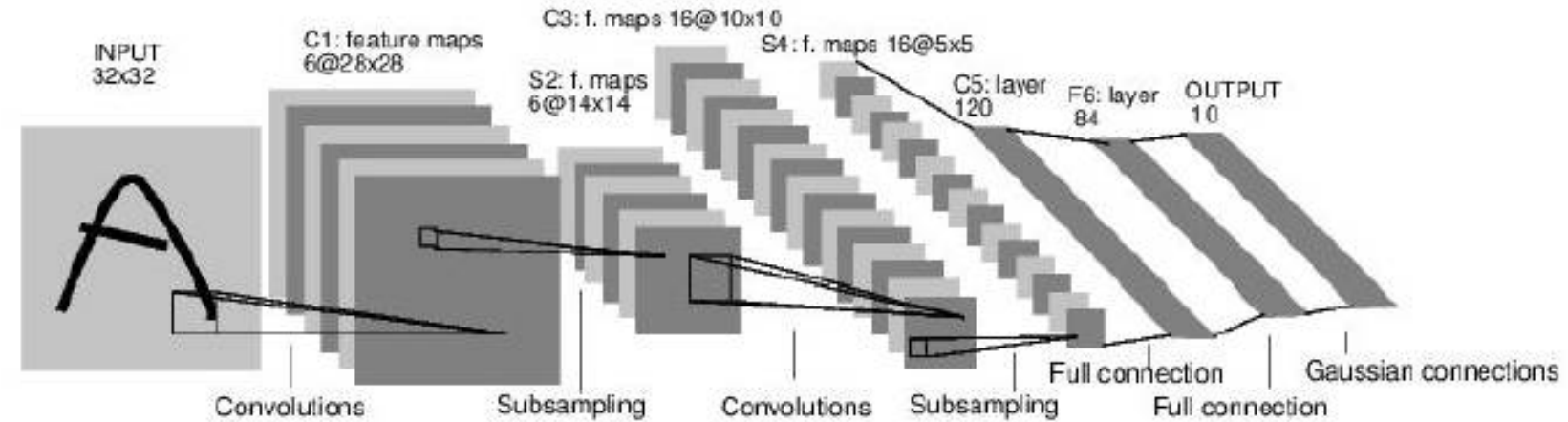
4



en bleu: les méthodes à base de réseaux de neurones

LeNet le pionnier

- Peu de couches
- Peu de convolutions



LeNet le pionnier

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

    self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 6,
                           kernel_size = 5, stride = 1, padding = 0)
    self.conv2 = nn.Conv2d(in_channels = 6, out_channels = 16,
                           kernel_size = 5, stride = 1, padding = 0)
    self.conv3 = nn.Conv2d(in_channels = 16, out_channels = 120,
                           kernel_size = 5, stride = 1, padding = 0)
    self.linear1 = nn.Linear(120, 84)
    self.linear2 = nn.Linear(84, 10)
    self.tanh = nn.Tanh()
    self.avgpool = nn.AvgPool2d(kernel_size = 2, stride = 2)

  def forward(self, x):
    x = self.conv1(x)
    x = self.tanh(x)
    x = self.avgpool(x)
    x = self.conv2(x)
    x = self.tanh(x)
    x = self.avgpool(x)
    x = self.conv3(x)
    x = self.tanh(x)

    x = x.reshape(x.shape[0], -1)
    x = self.linear1(x)
    x = self.tanh(x)
    x = self.linear2(x)
    return x
```

On définit les différentes couches dans le __init__

On définit les étapes à l'intérieur du réseau dans le forward