RESEAUX DE NEURONES CONVOLUTIFS

Qu'est ce qu'un CNN ?

Type de reseaux de neurones artificiels conçue pour traiter des données structurés en grille.

Ils s'inspirent du cortex visuel du cerveau humain.

Domaine d'Application des CNNs :

Les Convolutional Neural Networks (CNN) sont omniprésents, utilisés dans la reconnaissance faciale, les filtres intelligents, les voitures autonomes, la médecine et l'agriculture. Leur capacité à extraire des caractéristiques complexes en fait un outil indispensable dans de nombreux domaines.

ARCHITECTURES DES CNN :

01

Couche Convolution

Elle utilise des filtres pour balayer l'image et détecter des motifs locaux, comme des bords, des textures ou des formes.

02

Couche d’Activation

on applique une fonction d'activation (ReLu) pour introduire de la non-linéarité dans le modèle

03

Couche Pooling

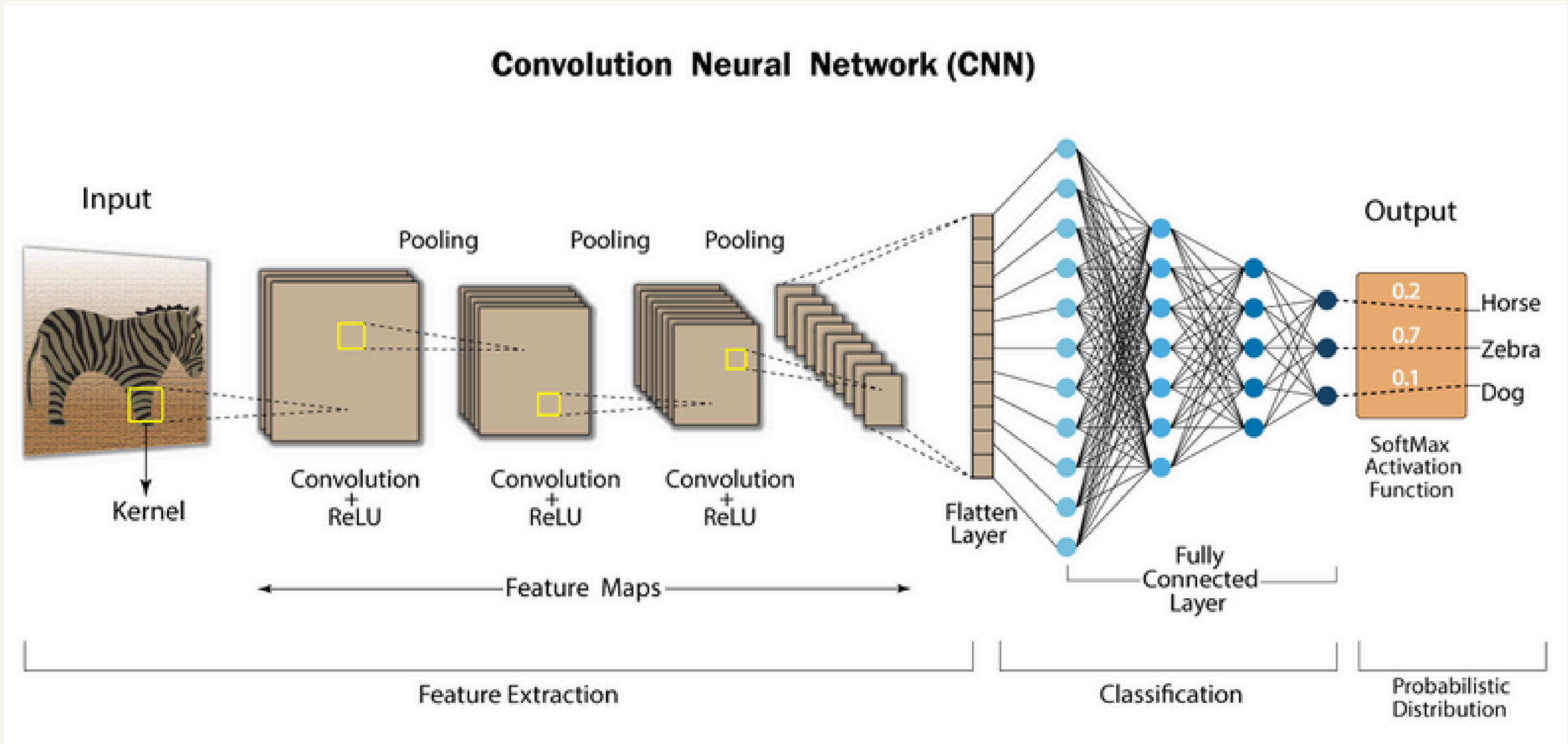
Cette couche réduit la taille des cartes de caractéristiques tout en conservant l'information importante

04

Couche Fully-Connected

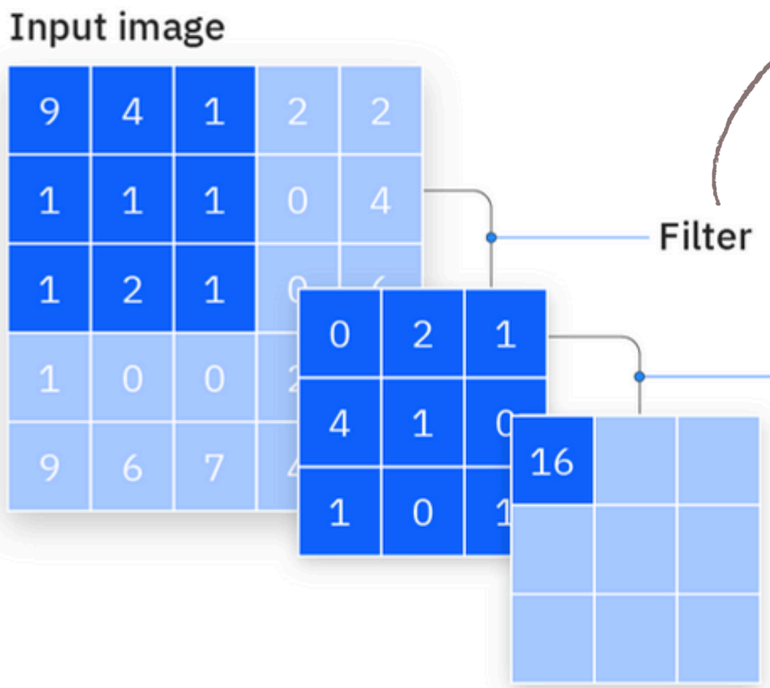
on aplatit les feature-maps en un vecteur, qui ensuite passe par une ou plusieurs couches fully-connected pour effectuer la classification ou la régression.

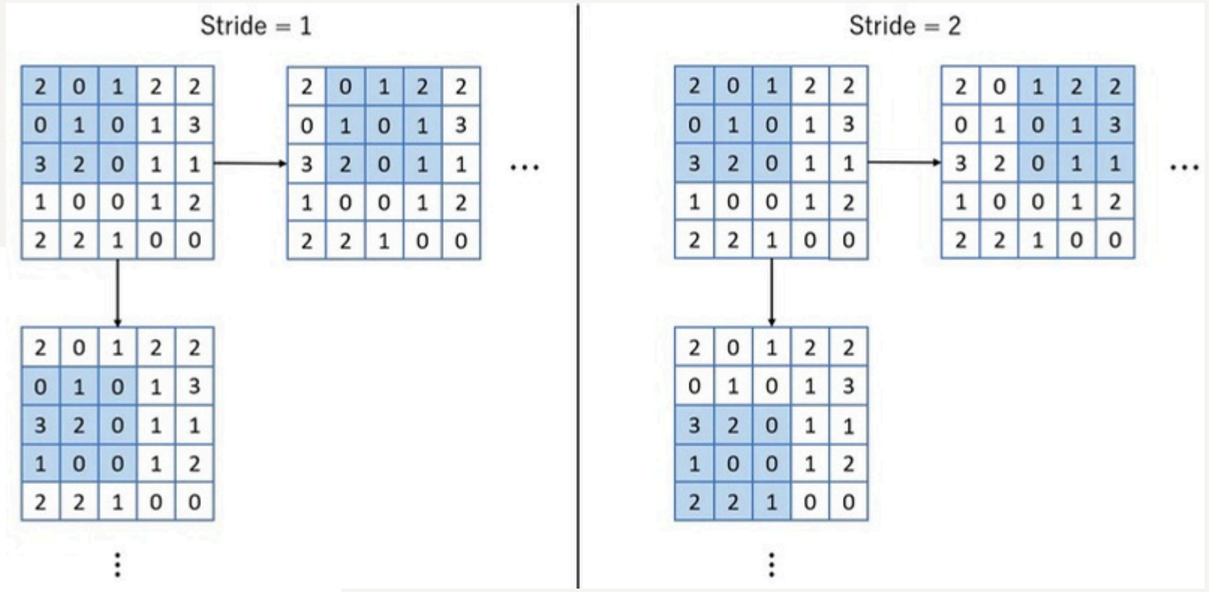
Exemple :

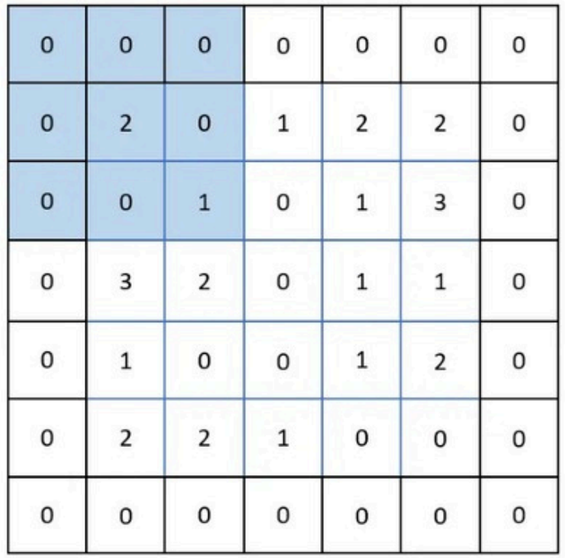

Convolution Neural Network (CNN)
Input
Pooling
Pooling
Pooling
Output
0.2
Horse
0.7
Zebra
0.1
Dog
Kernel
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Flatten Layer
SoftMax Activation Function
Feature Maps
Fully Connected Layer
Feature Extraction
Classification
Probabilistic Distribution

Methode de calculs :

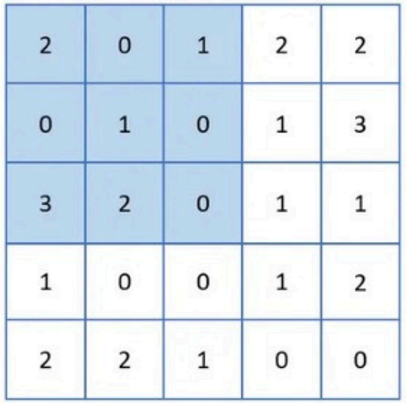
convolutional layer







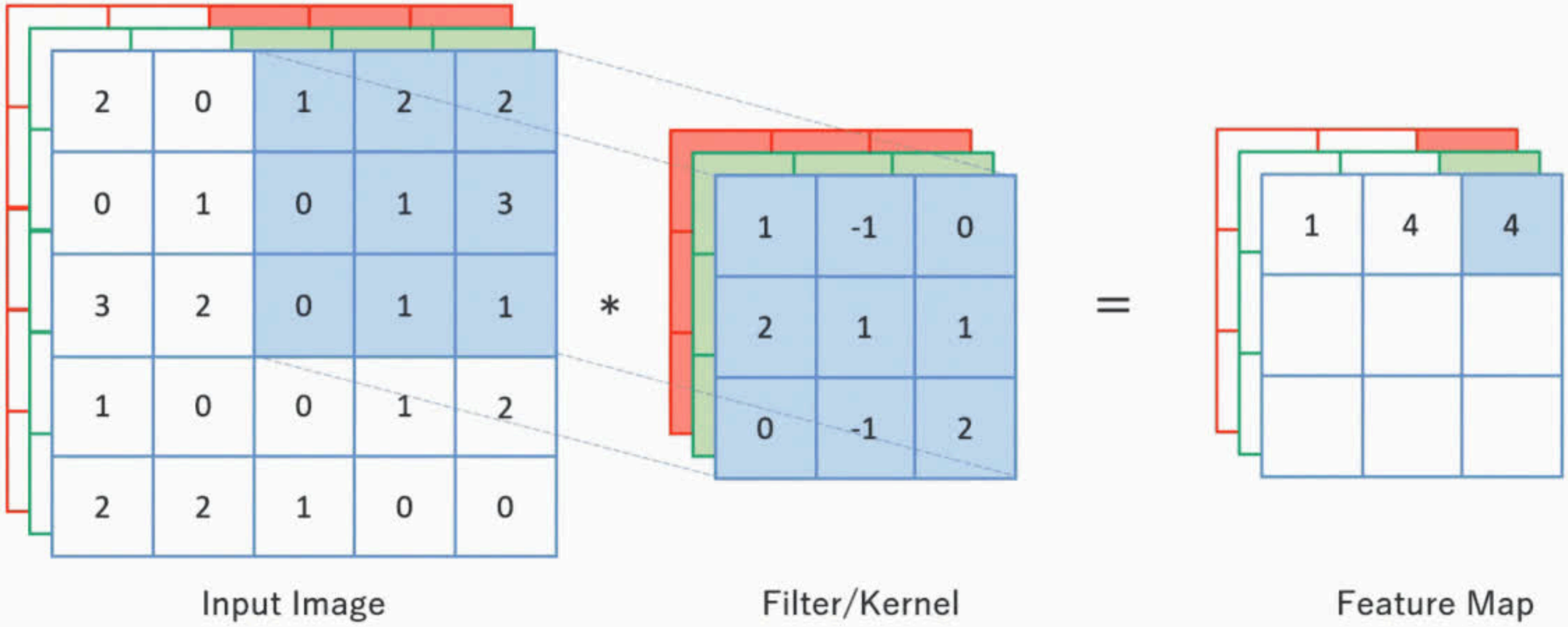
With Padding



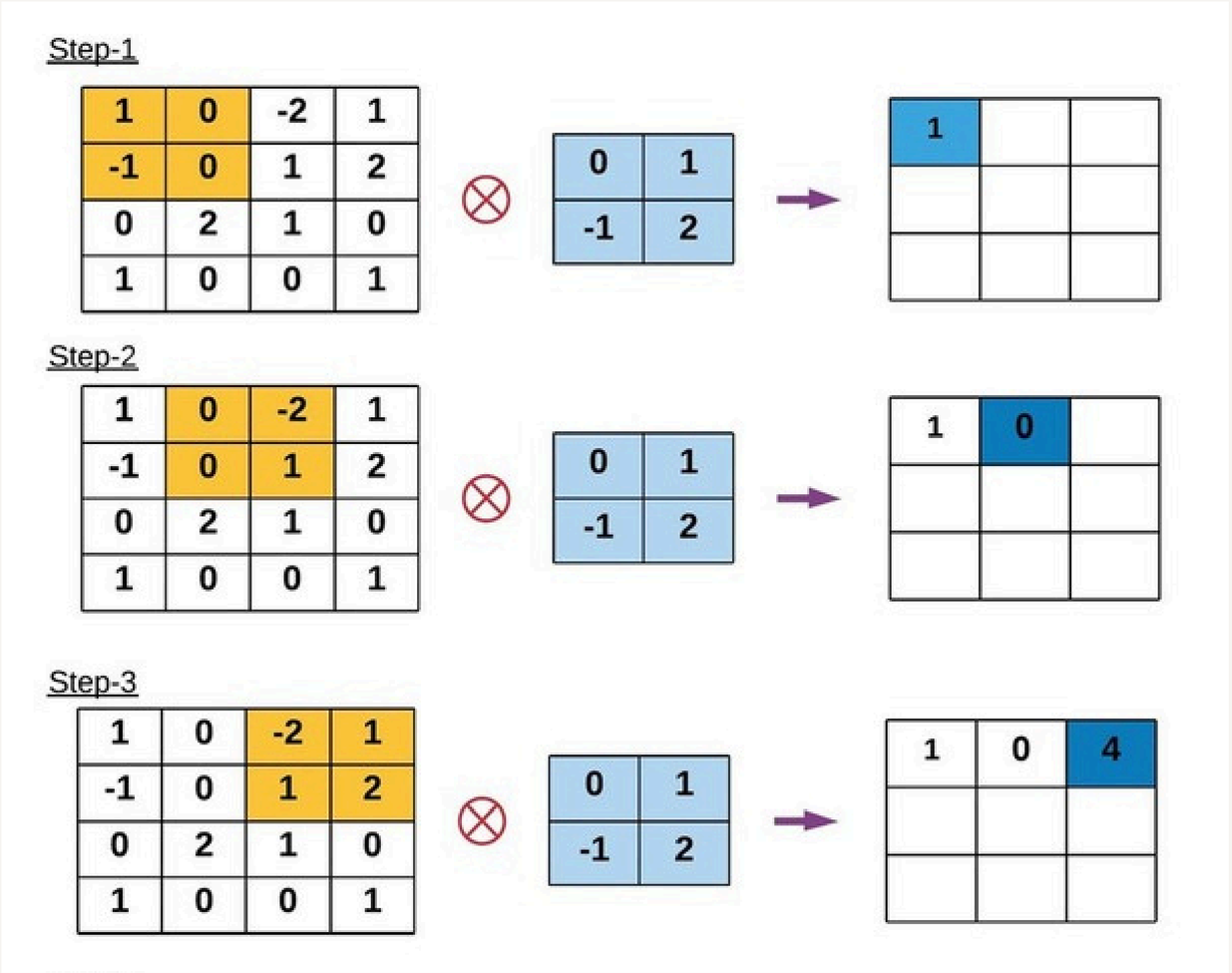
Without Padding

Methode de calculs :

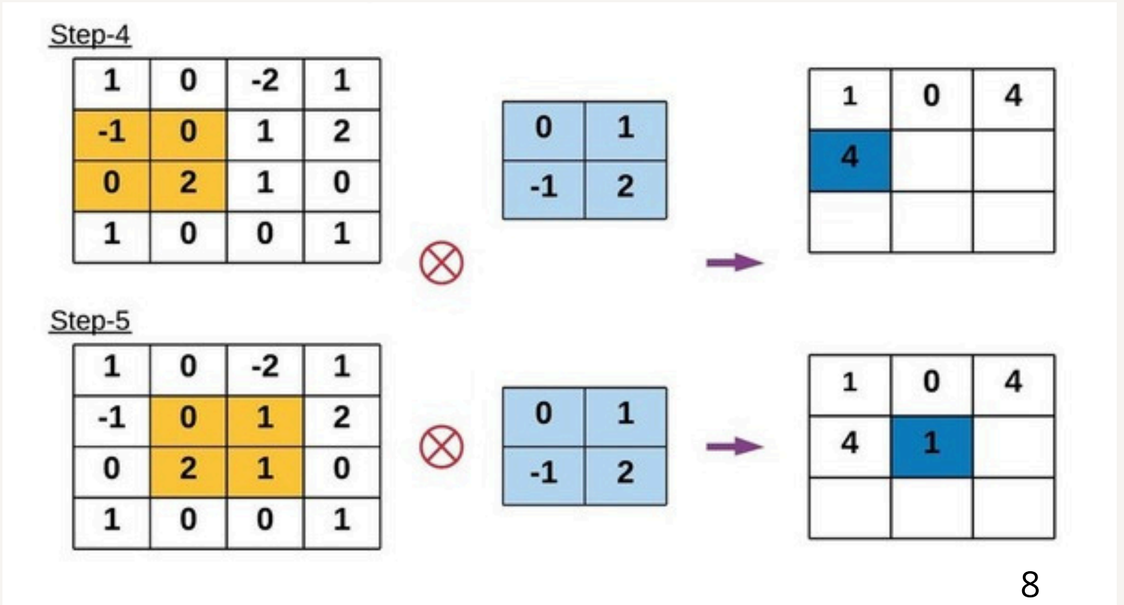
convolutional layer



Methode de calculs :

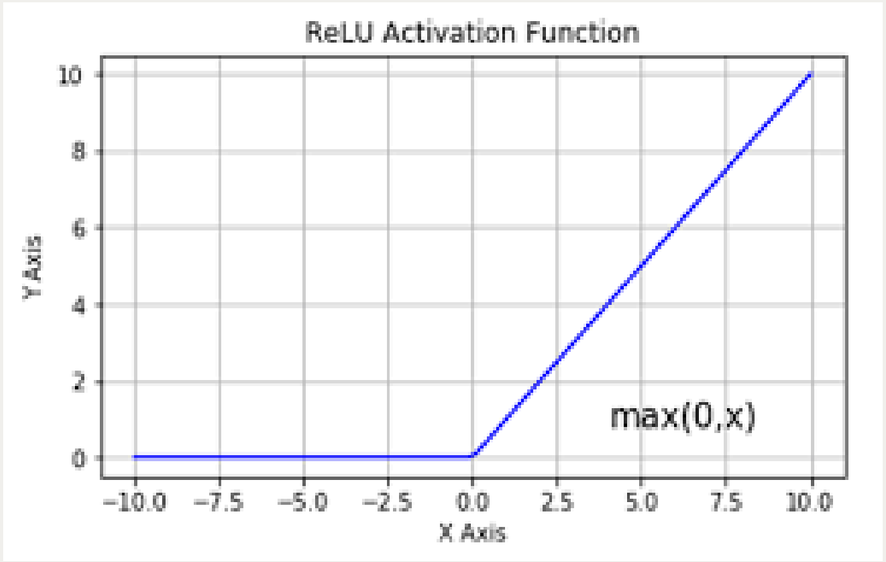


convolutional layer



Methode de calculs :

Activation layer



1	14	-9	4
-2	-20	10	6
-3	3	11	1
2	54	-2	80

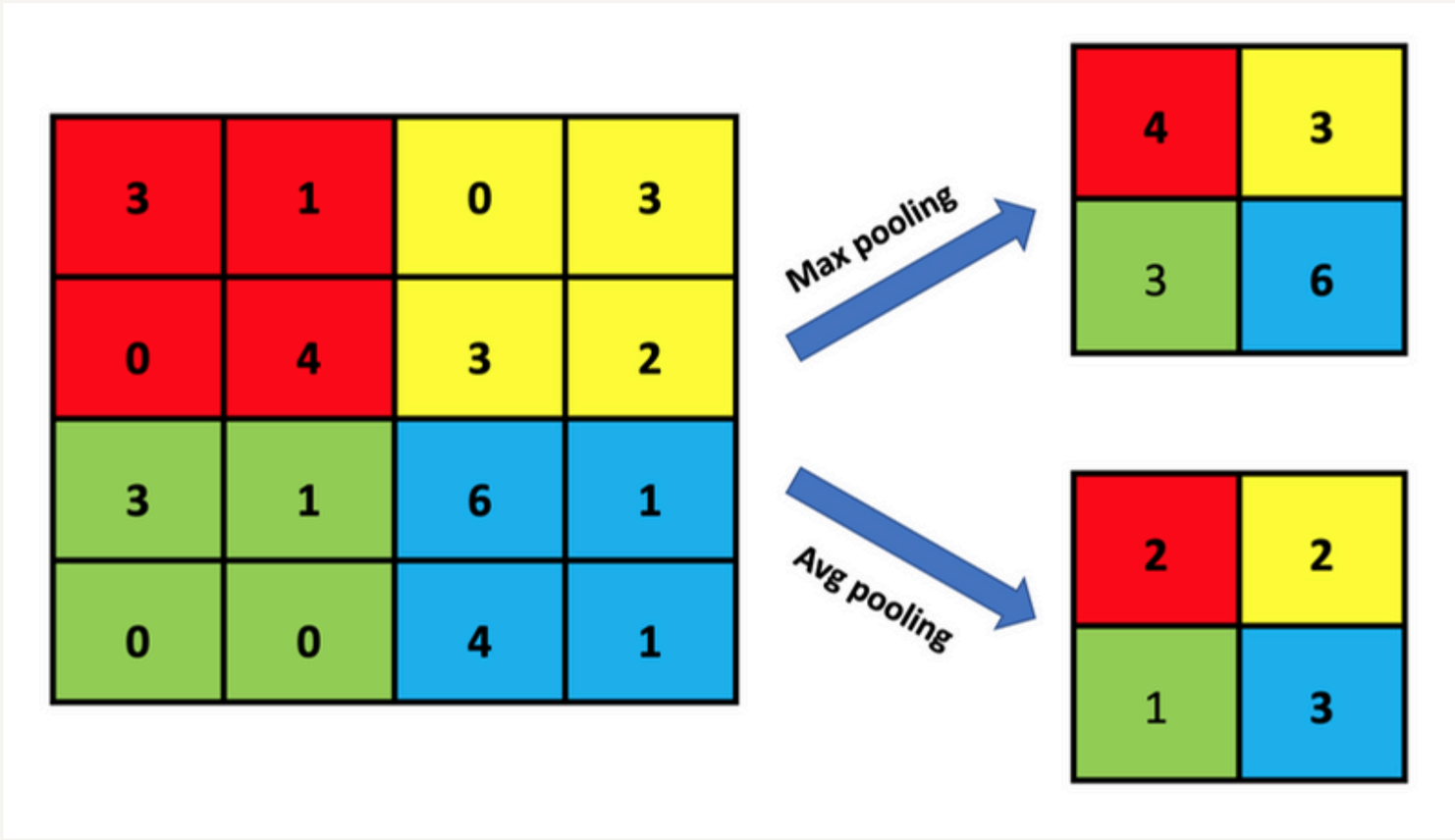
ReLU

1	14	0	4
0	0	10	6
0	3	11	1
2	54	0	80

Methode de calculs :

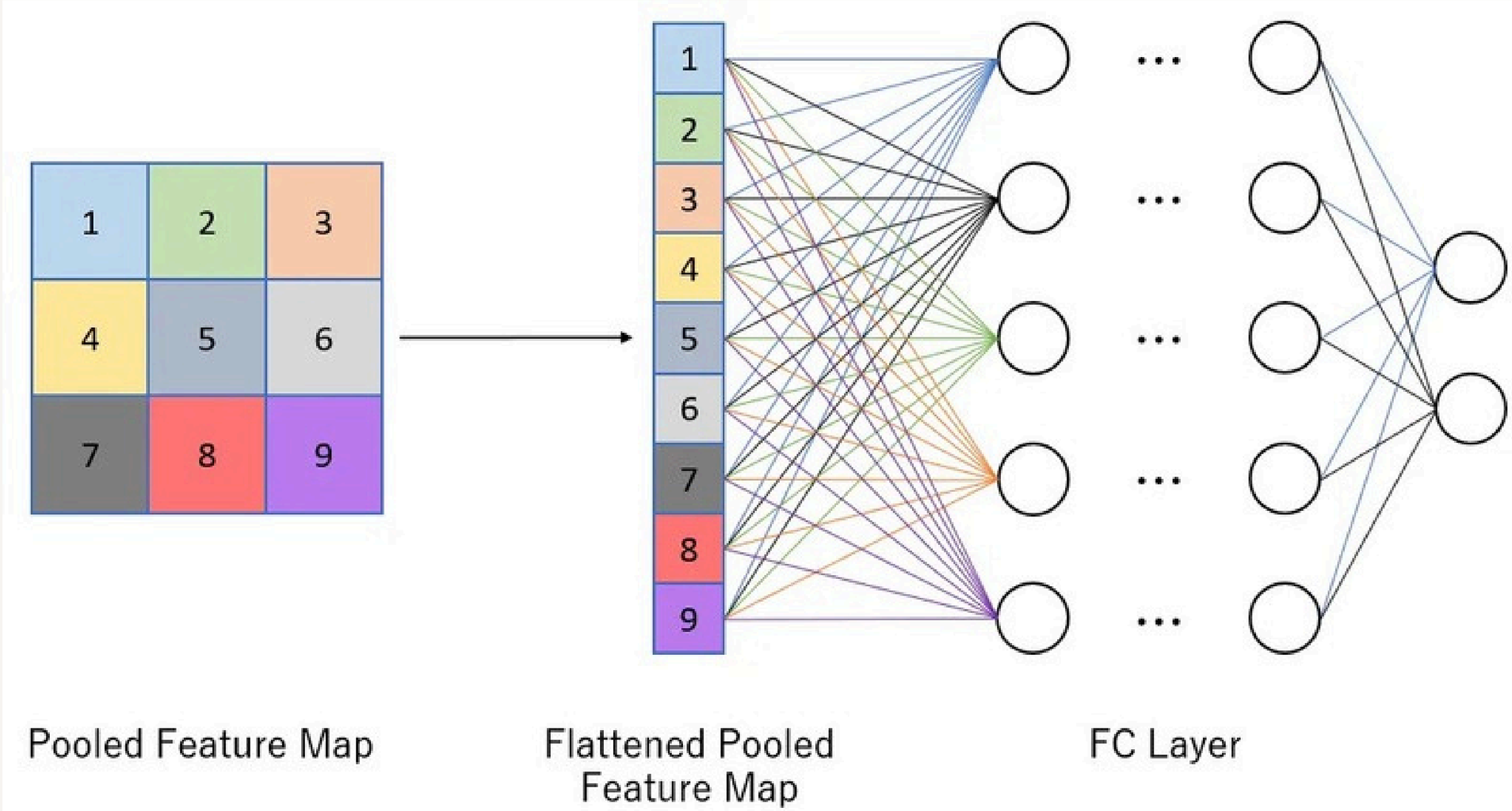
Pooling layer

Types



Methode de calculs :

Fully-Connected layer



Exemple pratique :

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
import numpy as np
import time

# Charger les données MNIST
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normaliser les données (mise à l'échelle entre 0 et 1)
x_train, x_test = x_train / 255.0, x_test / 255.0

# Ajouter une dimension pour les canaux (nécessaire pour les CNN)
x_train_cnn = x_train[..., tf.newaxis]
x_test_cnn = x_test[..., tf.newaxis]

# Aplatir les données pour les modèles sans CNN
x_train_flat = x_train.reshape((x_train.shape[0], -1))
x_test_flat = x_test.reshape((x_test.shape[0], -1))
```

```python
def create_cnn_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),#32 filtres de taille 3x3
        layers.MaxPooling2D((2, 2)),#réduit la taille de l'image par un facteur de 2
        layers.Conv2D(64, (3, 3), activation='relu'),#64 filtres de taille 3x3
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),# Aplatissement des caractéristiques extraites pour les connecter aux couches dense
        layers.Dense(128, activation='relu'), #Couche dense entièrement connectée avec 128 neurones
        layers.Dense(10, activation='softmax') # Couche de sortie avec 10 neurones (correspondant aux 10 classes)
    ])
    model.compile(optimizer='adam',#L'optimiseur Adam pour ajuster les poids
                  loss='sparse_categorical_crossentropy',#pour la classification multi-classes
                  metrics=['accuracy'])#pour évaluer la performance
    return model


  # Modèle sans CNN (MLP)
def create_mlp_model():
    model = models.Sequential([
        layers.Flatten(input_shape=(28, 28)),  # Aplatir l'image en un vecteur de 784 valeurs
        layers.Dense(256, activation='relu'),  # Couche cachée avec 256 neurones
        layers.Dense(128, activation='relu'),  # Couche cachée avec 128 neurones
        layers.Dense(10, activation='softmax') # Couche de sortie avec 10 neurones (un par classe)
    ])
    model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
    return model
# Modèle SVM (sans CNN)
def train_svm(x_train, y_train, x_test, y_test):
    svm_model = SVC(kernel='rbf')
    start_time = time.time()
    svm_model.fit(x_train, y_train)
    training_time = time.time() - start_time
    y_pred = svm_model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy, training_time
```

```python
# Entrainer et evaluer le modele CNN
cnn_model = create_cnn_model()
start_time = time.time()
cnn_model.fit(x_train_cnn, y_train, epochs=5, validation_data=(x_test_cnn, y_test))
cnn_training_time = time.time() - start_time
cnn_loss, cnn_accuracy = cnn_model.evaluate(x_test_cnn, y_test, verbose=2)

# Entraîner et évaluer le modèle MLP
mlp_model = create_mlp_model()
start_time = time.time()
mlp_model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
mlp_training_time = time.time() - start_time
mlp_loss, mlp_accuracy = mlp_model.evaluate(x_test, y_test, verbose=2)

# Entraîner et évaluer le modèle SVM
svm_accuracy, svm_training_time = train_svm(x_train_flat, y_train, x_test_flat, y_test)

# Afficher les résultats
print("\nRésultats :")
print(f"CNN - Précision : {cnn_accuracy * 100:.2f}%, Temps d'entraînement : {cnn_training_time:.2f} s"
print(f"MLP - Précision : {mlp_accuracy * 100:.2f}%, Temps d'entraînement : {mlp_training_time:.2f}
print(f"SVM - Précision : {svm_accuracy * 100:.2f}%, Temps d'entraînement : {svm_training_time:.2f}
s")
```

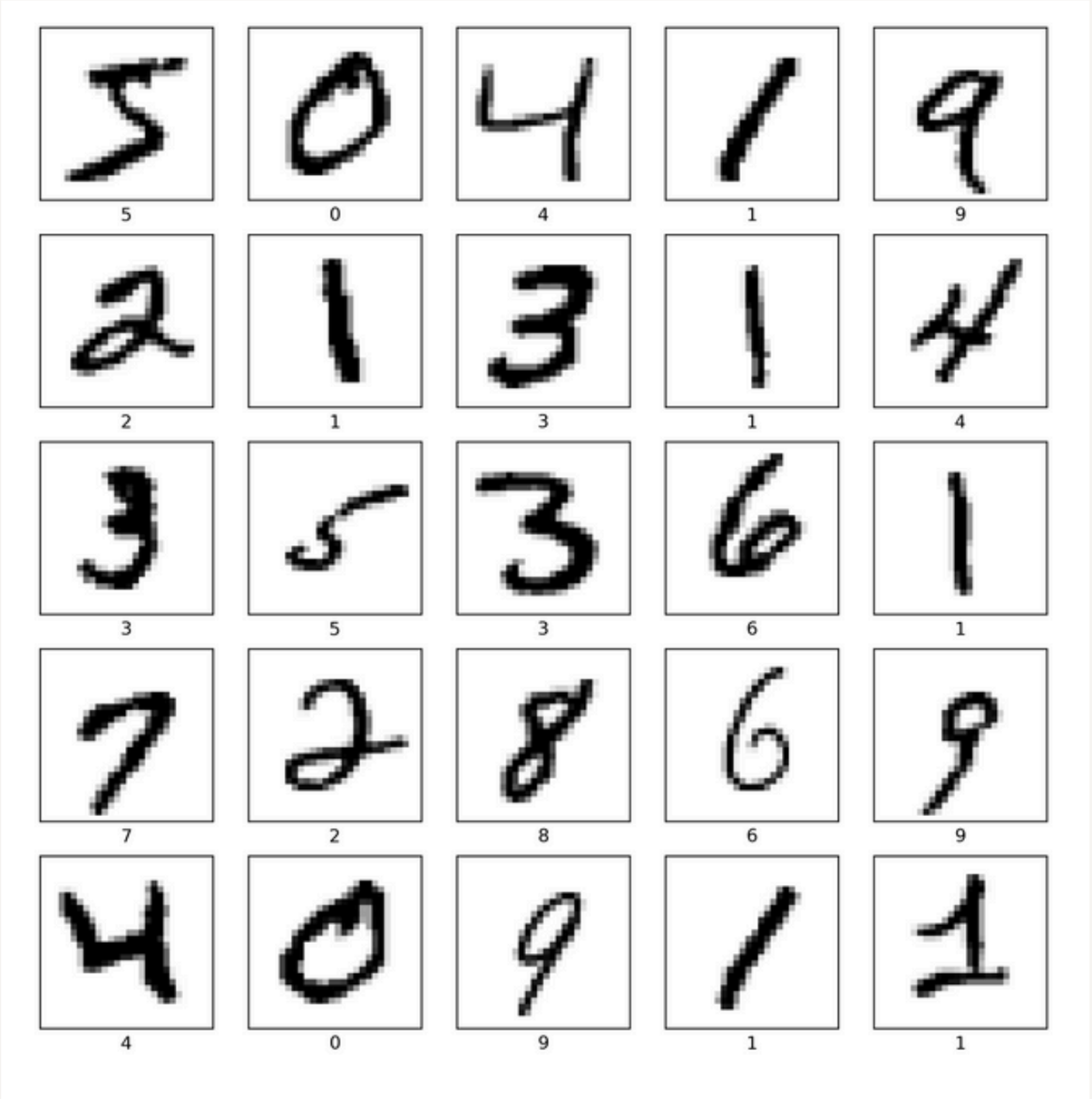
Resultats :

***CNN* - Précision : 99.00%, Temps d'entraînement : 116.20secondes**

***MLP* - Précision : 97.40%, Temps d'entraînement : 37.60 secondes**

***SVM* - Précision : 97.92%, Temps d'entraînement : 294.91secondes**

Resultats :



Avantages :

1. Extraction automatique de caractéristiques : Pas besoin de définir manuellement des caractéristiques, le modèle les apprend tout seul.
2. Invariance translationnelle : Reconnaît les objets même s'ils sont déplacés dans l'image.
3. Efficacité avec les données structurées en grille.
4. Performances élevées.
5. Réduction de la dimensionnalité.

Limites :

1. Besoin de grandes quantités de données : Performances optimales uniquement avec des jeux de données volumineux.
2. Coût computationnel élevé : Entraînement long et nécessite des ressources matérielles importantes (GPU/TPU).
3. Moins adapté aux données non structurées ou sans relation spatiale.
4. Risque de Surapprentissage

Les CNN sont utilisés pour leur capacité à extraire automatiquement des caractéristiques pertinentes à partir de données structurées, comme les images.Ils sont capables de transformer des données brutes en connaissances utiles, ils capturent des motifs locaux tout en préservant les relations spatiales, ce qui les rend efficaces pour des tâches de vision par ordinateur (classification, détection d'objets).

Thank You