Reconnaissance d'images par un réseau de neurones convolutifs (CNN)

Objectifs [a] *:*

- *Réseaux convolutifs associés à la reconnaissance d'images*
- *Applications aux bases MNIST et CIFAR-10*

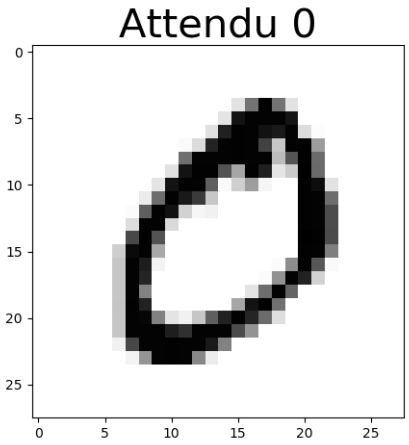
a. Version 2023 adapté BUT 3 Informatique (Orléans), inspiré du livre d'Arnaud Bodin et François Recher

1. Reconnaissance de chiffres

On souhaite reconnaître des chiffres écrits à la main.

1.1. Données

Les données sont celles de la base MNIST déjà rencontrée dans le chapitre « Python : tensorflow avec keras - partie 2 ». On rappelle brièvement que cette base contient 60 000 données d'apprentissage (et 10 000 données de test) sous la forme d'images 28×28 en niveau de gris, étiquetées par le chiffre attendu.

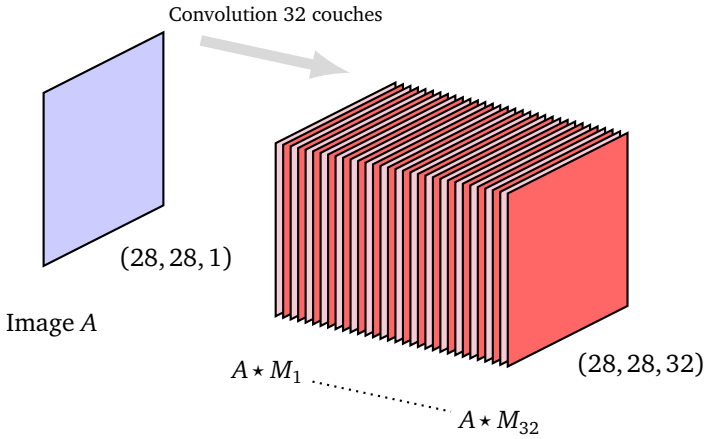


Pour le besoin de la convolution, on conserve chaque image sous la forme d'un tableau 28×28 ; afin de préciser que l'image est en niveau de gris, la taille du tableau *numpy* est en fait $(28, 28, 1)$ (à comparer à la représentation d'une image couleur de taille $(28, 28, 3)$).

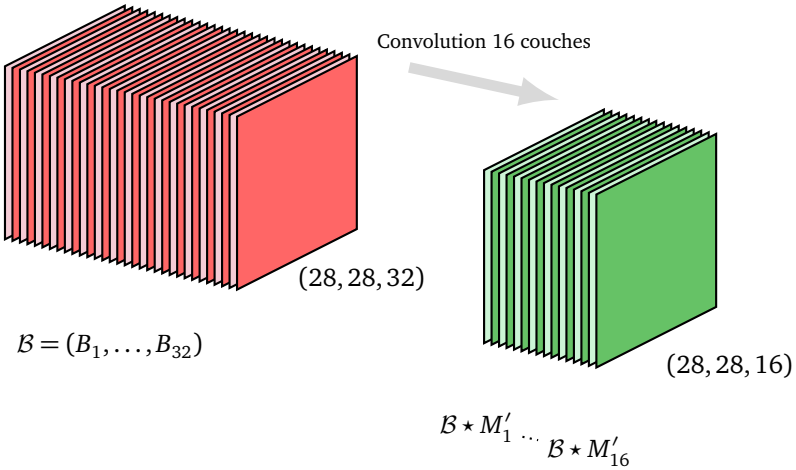
1.2. Modèle

On crée un réseau composé de la façon suivante :

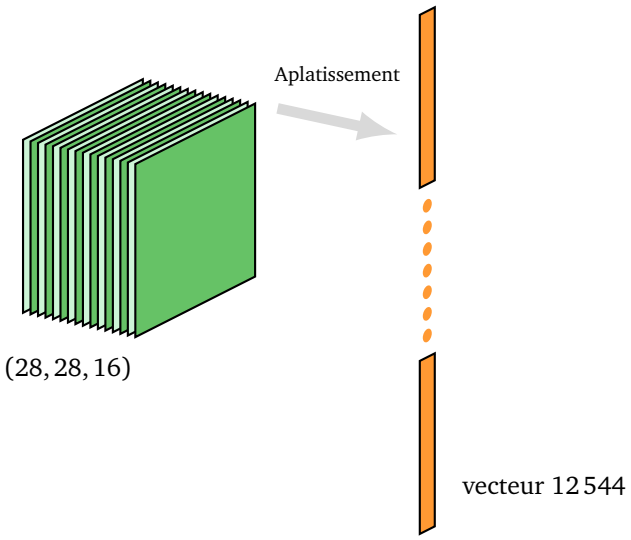
- **Première couche de convolution.** Une couche formée de 32 sous-couches de convolution. Si on note A l'image de départ, chaque sous-couche est la convolution $A \star M_i$ de A par un motif M_i de taille 3×3 ($i = 1, \dots, 32$). Une entrée de taille $(28, 28, 1)$ est transformée en une sortie de taille $(28, 28, 32)$.
  Chaque sous-couche correspond à un neurone de convolution avec 3×3 arêtes plus un biais, c'est-à-dire 10 poids par sous-couche. Avec nos 32 sous-couches, cela fait au total 320 poids pour cette première couche.



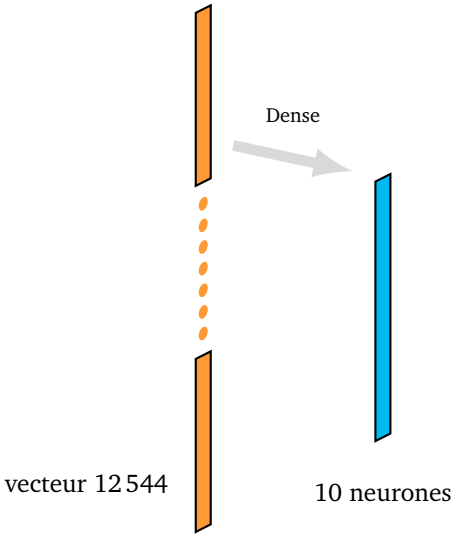
- **Seconde couche de convolution.** Une couche formée de 16 sous-couches de convolution. Si on note $\mathcal{B} = (B_1, \dots, B_{32})$ les sorties de la couche précédente, qui deviennent maintenant les entrées, alors chaque sous-couche de sortie est une convolution $\mathcal{B} \star M'_i$ par un motif M'_i de taille $3 \times 3 \times 32$. Une entrée de taille $(28, 28, 32)$ est transformée en une sortie de taille $(28, 28, 16)$.
  Chaque sous-couche correspond à un neurone de convolution avec $3 \times 3 \times 32$ arêtes plus un biais, soit 289 poids par sous-couche. Avec nos 16 sous-couches, cela fait un total de 4624 poids pour cette seconde couche.



- **Aplatissement.** Chaque sortie de taille $(28, 28, 16)$ est reformatée en un (grand) vecteur de taille 12 544 ($= 28 \times 28 \times 16$). Il n'y a aucun poids associé à cette transformation : ce n'est pas une opération mathématique, c'est juste un changement du format des données.

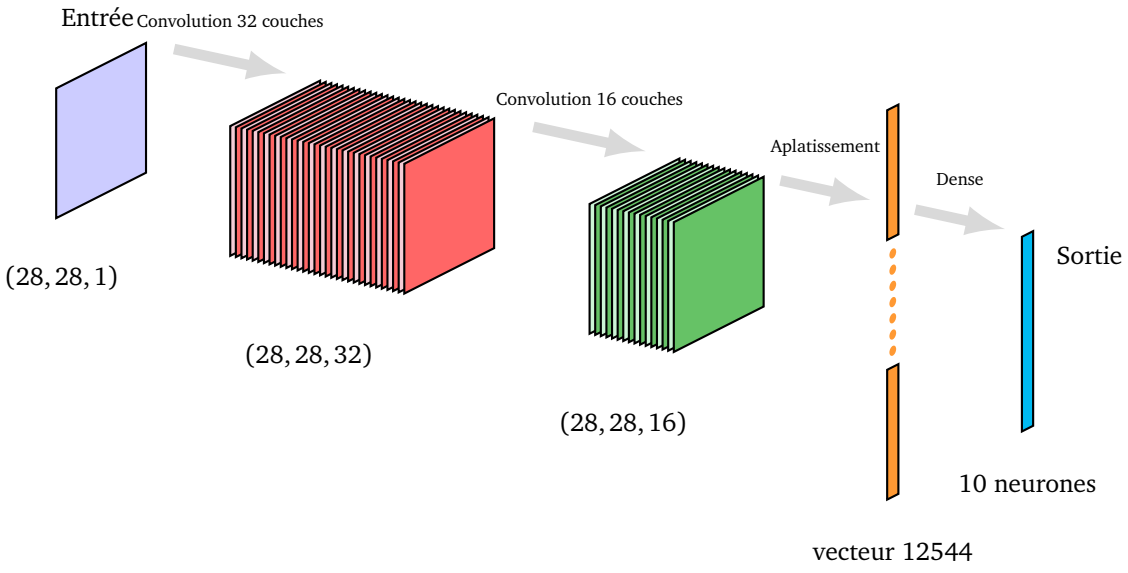


- **Couche dense.** C'est une couche « à l'ancienne » composée de 10 neurones, chaque neurone étant relié aux 12 544 entrées. En tenant compte d'un biais par neurone cela fait 125 450 ($= (12\,544 + 1) \times 10$) poids pour cette couche.

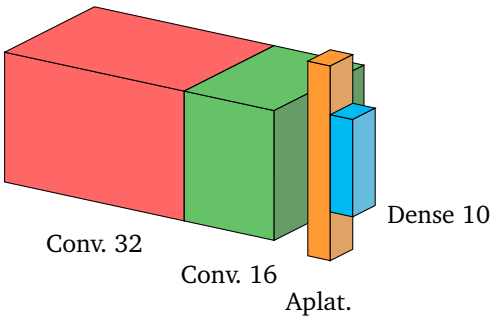


- **Bilan.** Il y a en tout 130 394 poids à calculer.

Voici l'architecture complète du réseau.



On résume l'architecture du réseau par des blocs, chaque bloc représentant une transformation.



1.3. Programme

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten

### Partie A - Création des données
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

(X_train_data,Y_train_data),(X_test_data,Y_test_data) = mnist.load_data()

N = X_train_data.shape[0]  # 60 000 données

X_train = np.reshape(X_train_data, (N,28,28,1))
X_test = np.reshape(X_test_data, (X_test_data.shape[0],28,28,1))

X_train = X_train/255  # normalisation
X_test = X_test/255

Y_train = to_categorical(Y_train_data, num_classes=10)
Y_test = to_categorical(Y_test_data, num_classes=10)

### Partie B - Réseau de neurones
modele = Sequential()

# Première couche de convolution : 32 neurones, motif 3x3, activ. relu
modele.add(Conv2D(32, kernel_size=3, padding='same', activation='relu',
                  input_shape=(28,28,1)))

# Deuxième couche de convolution : 16 neurones
modele.add(Conv2D(16, kernel_size=3, padding='same', activation='relu'))

# Aplatissage
modele.add(Flatten())
```

```
# Couche de sortie : 10 neurones
modele.add(Dense(10, activation='softmax'))

# Descente de gradient
modele.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])

print(modele.summary())

# Calcul des poids
modele.fit(X_train, Y_train, batch_size=32, epochs=5)

### Partie C - Résultats
score = modele.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

1.4. Explications

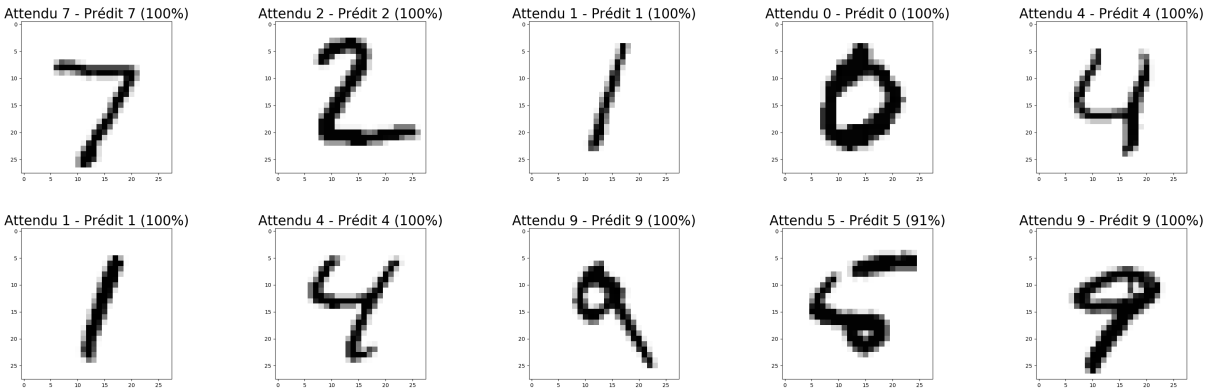
Une couche de convolution est ajoutée avec *tensorflow/keras* par une commande du type :

```
modele.add(Conv2D(16, kernel_size=3, padding='same', activation='relu'))
```

qui correspond à une couche composée de 16 sous-couches de convolution. Chacune de ces sous-couches correspond à une convolution par un motif de taille 3×3 (option `kernel_size=3`) et conserve la taille de l'image (option `padding='same'`). La fonction d'activation pour chaque neurone est « ReLU ».
Note : en fait le calcul d'une « convolution » avec *tensorflow/keras* est un calcul de corrélation, c'est-à-dire que les coefficients du motif ne sont pas retournés. Cela n'a pas vraiment d'importance car ces coefficients sont déterminés par rétropropagation et cela reste transparent à l'usage.

1.5. Résultats

Après 5 époques, on obtient une précision de 99% sur les données de test, ce qui est un gros progrès par rapport au 95% obtenus dans le TP sur les réseaux denses (TP2). Il faut bien avoir conscience que les derniers pourcentages de précision sont de plus en plus difficiles à acquérir !



Sur les exemples ci-dessus toutes les prédictions sont correctes avec en plus une quasi-certitude de 100%, à l'exception de l'avant-dernière image qui conduit à une bonne prédiction mais avec une certitude plus faible. En effet le vecteur de sortie pour cette image est :

$$Y = (0, 0, 0, 0, 0, 0.911, 0.089, 0, 0, 0).$$

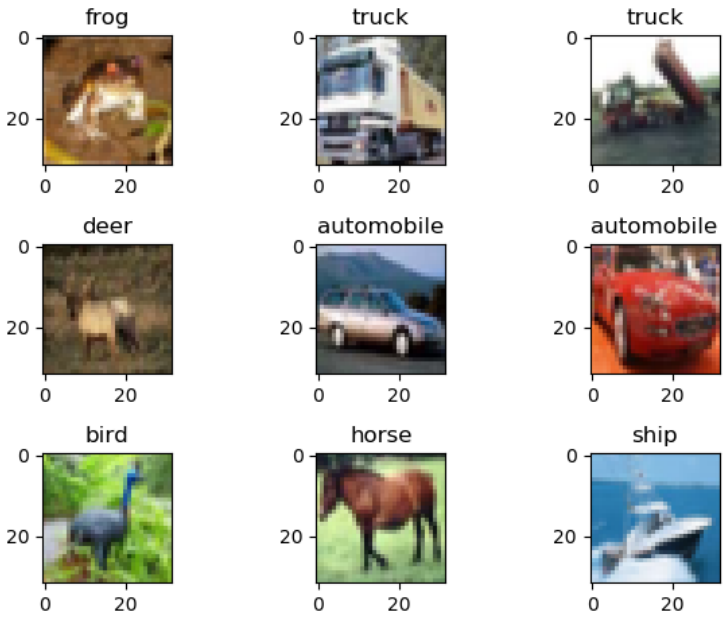
Le chiffre 5 est donc prédit à 91% avec un léger doute avec le chiffre 6 à 9%, ce qui est tout à fait rassurant vu que ce chiffre est très mal dessiné !

2. Reconnaissance d'images

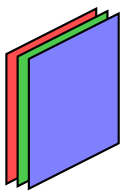
On souhaite reconnaître des objets ou des animaux sur de petites photos.

2.1. Données

On rappelle que la base CIFAR-10, déjà rencontrée dans le chapitre « Python : tensorflow avec keras - partie 2 » contient 50 000 images d'apprentissage, réparties en 10 catégories.



Chaque image possède 32×32 pixels de couleurs. Une entrée correspond donc à un tableau $(32, 32, 3)$.

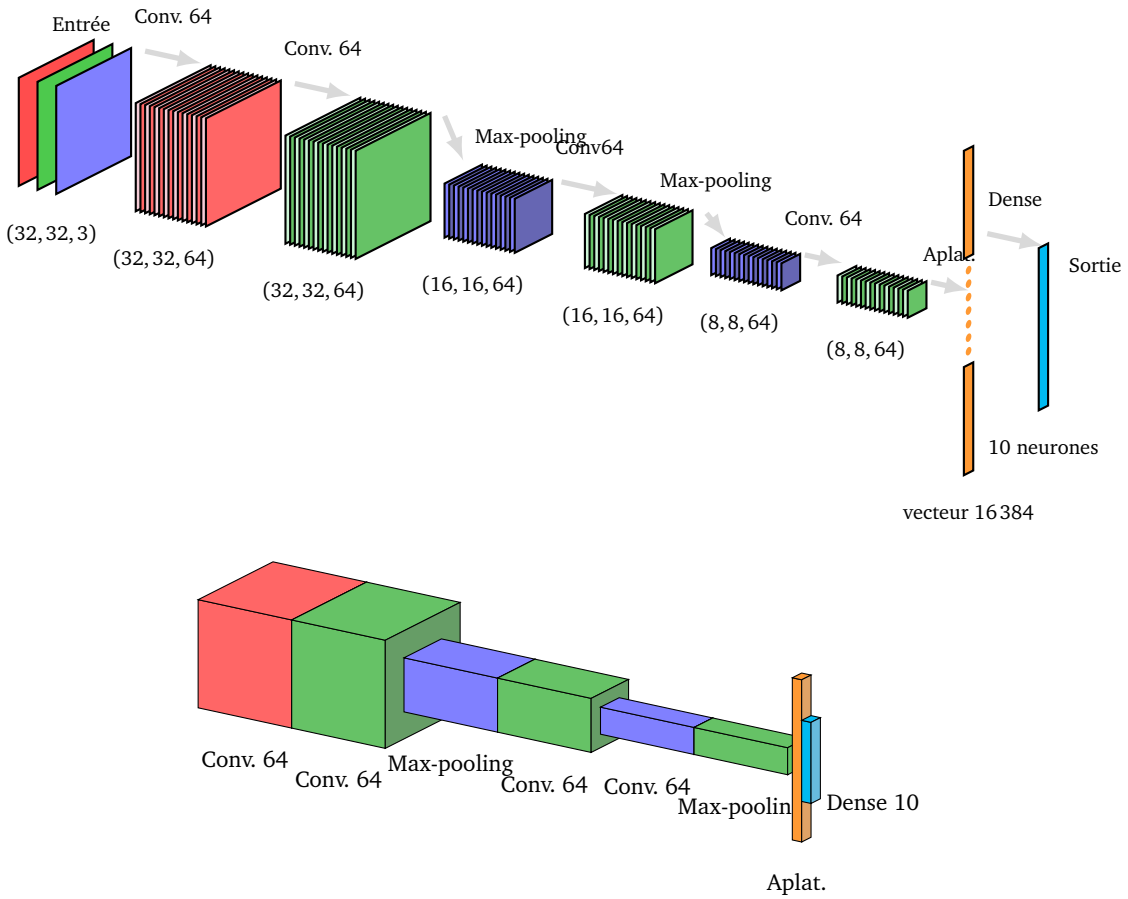


$(32, 32, 3)$

Image

2.2. Modèle

L'architecture du réseau utilisé est composée de plusieurs couches de convolution. Pour diminuer le nombre de poids à calculer, on intercale des couches de pooling (regroupement de termes). Ces pooling sont des max-pooling de taille 2×2. De tels regroupements divisent par 4 la taille des données en conservant les principales caractéristiques, ce qui fait que la couche de neurones suivante possèdera 4 fois moins de poids à calculer.



Il y a en tout 153 546 poids à calculer. Sans les deux couches de pooling, il y aurait 767 946 poids.

2.3. Programme

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D


# Partie A. Données

from tensorflow.keras.datasets import cifar10

(X_train_data, Y_train_data), (X_test_data, Y_test_data) = cifar10.load_data()

num_classes = 10
labels =  ['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']

Y_train = keras.utils.to_categorical(Y_train_data, num_classes)
Y_test = keras.utils.to_categorical(Y_test_data, num_classes)

X_train = X_train_data.astype('float32')
```

```
X_test = X_test_data.astype('float32')

X_train = X_train/255
X_test = X_test/255

print(X_train.shape)


# Partie B. Réseau

modele = Sequential()

# Première couche de convolution : 64 neurones, convolution 3x3, activation relu
modele.add(Conv2D(64, kernel_size=3, padding='same', activation='relu', input_shape=(32,32

# Deuxième couche de convolution : 64 neurones
modele.add(Conv2D(64, kernel_size=3, padding='same', activation='relu'))

# Mise en commun (pomodele.add(MaxPooling2D(pool_size=(2, 2)))

# Quatrième couche de convolution : 64 neurones
modele.add(Conv2D(64, kernel_size=3, padding='same', activation='relu'))

# Mise en commun (pooling)
# modele.add(MaxPooling2D(pool_size=(2, 2)))

# Aplatissage
modele.add(Flatten())

# Couche de sortie : 10 neurones
modele.add(Dense(10, activation='softmax'))

# Méthode de gradient
modele.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Affiche un résumé
modele.summary()


# Partie C. Apprentissage
modele.fit(X_train, Y_train, epochs=5, batch_size=32)


# Partie D. Résultats et visualisation

score = modele.evaluate(X_test, Y_test, verbose=0)
print('Test erreur (loss) :', score[0])
print('Test précision (accuracy) :', score[1])
```

```
Y_predict = modele.predict(X_test)

def affiche_images_test(debut):
    plt.axis('off')
    for i in range(9):
        plt.subplot(330 + 1 + i)
        image_predite = Y_predict[i]
        perc_max = int(round(100*np.max(image_predite)))
        rang_max = np.argmax(image_predite)
        titre = 'Attendu ' + labels[Y_test_data[i][0]] + ' \n Prédit ' + labels[rang_max]
        plt.title('Attendu %d - Prédit %d (%d%%)' % (Y_test_data[i],rang_max,perc_max))
        plt.title(titre)
        plt.imshow(X_test_data[i], interpolation='nearest')
    plt.tight_layout()
    # plt.savefig('tfconv-images-test.png')
    plt.show()

    return

affiche_images_test(0)
```

2.4. Résultats

Après une dizaine d'époques, on obtient plus de 80% de précision sur les données d'entraînement et un peu moins de 75% sur les données de test. Ainsi les trois quarts des images sont correctement prédites.