# Rapport de Projet: Traitement d'images

VO Van Nghia, YU Duan, and PHAM Tuan Kiet

Etudians 2ème année MIC, Institut national des sciences appliquées de Toulouse

27 avril 2020

## Table des matières

# 1 Introduction

## 1.1 Réseaux de neurones

### 1.1.1 Définition

Un réseau neuronal est l'association, en un graphe plus ou moins complexe, d'objets élémentaires, les neurones formels. Les principaux réseaux se distinguent par l'organisation du graphe (en couches, complets, ...), c'est-à-dire leur architecture, son niveau de complexité (le nombre de neurones, présence ou non de boucles de rétroaction dans le réseau) par le type des neurones (leurs fonctions de transition ou d'activation) et enfin par l'objectif visé : apprentissage supervisé ou non, optimisation, systèmes dynamiques, ...[1]

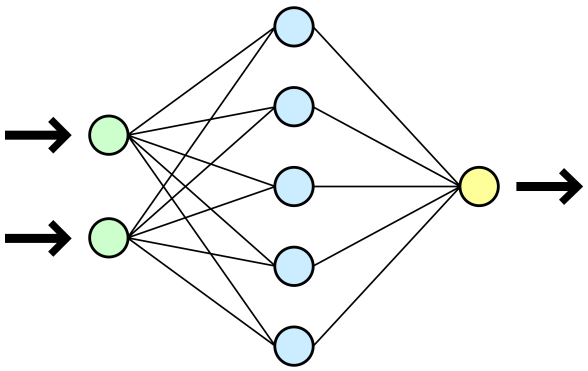

FIGURE 1 – Vue simplifiée d'un réseau artificiel de neurones

### 1.1.2 Neurone formel

Le neurone formel est un modèle qui se caractérise par un état interne $s$ , des signaux d'entrée $x_1$, ..., $x_p$ et une fonction d'activation

$$s = h(x_1, \ldots, x_p) = g(\alpha_0 + \sum_{j=1}^{p} \alpha_j x_j) = g(\alpha_0 + \alpha' x)$$

La fonction d'activation opère une transformation d'une combinaison affine des signaux d'entrée, $\alpha_0$, terme constant, étant appelé le biais du neurone. Cette combinaison affine est déterminée par un vecteur de poids $[\alpha_0, \ldots, \alpha_p]$ associé à chaque neurone et dont les valeurs sont estimées dans la phase d'apprentissage. Ils constituent la mémoire ou connaissance répartie du réseau. [1]

## 1.2 Réseau neuronal convolutif

Un réseau de neurones convolutifs ou réseau de neurones à convolution est un type de réseau de neurones artificiels, dont architecture est formée par un empilement de couches (les combinaisons des neurones formels) de traitement :[2]

- La couche de convolution (CONV)
- La couche de pooling (POOL)
- La couche d'activation
- La couche "entièrement connectée" (FC)
- La couche de perte (LOSS)

Dans notre projet, nous utilisons les couches suivantes : CONV, POOL et d'activation.

# 2 Réalisation d'un réseau neuronal convolutif dans Python

## 2.1 Objectif

## 2.2 Récupérer et préparer les images

Nous utilisons la base de donnée MNIST (Modified National Institute of Standards and Technology database) pour ce projet.

FIGURE 2 – Un exemple de chiffres de la base MNIST

Après charger la base de donnée, on ajoute le bruit gaussien. Finallement, on met données en forme pour passer dans le réseau.

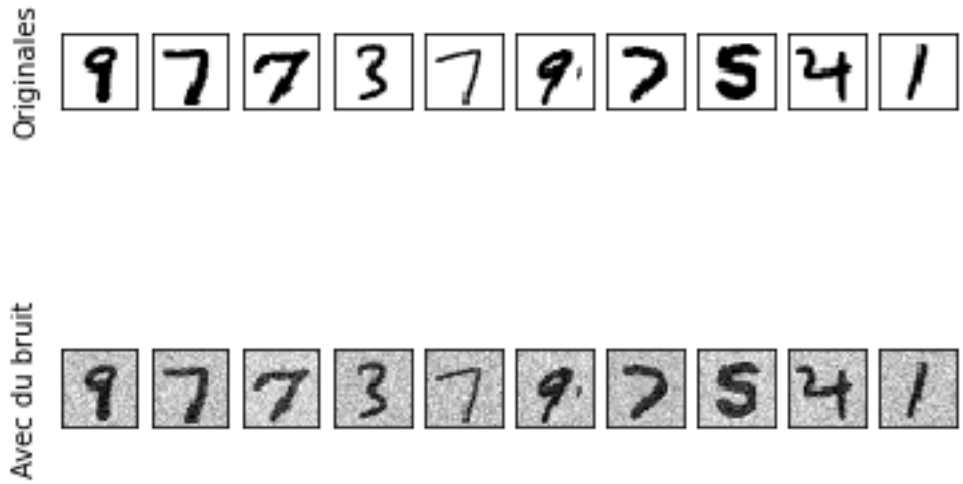


FIGURE 3 – Les images avec du bruit gaussien ajouté

## 2.3 Architecture du réseau neuronal convolutif

```
init = Input(shape=(None, None,1))
x = Convolution2D(16, (3, 3), activation='relu', padding='same')(
                                    init)
x = MaxPooling2D((2, 2))(x)
x = Convolution2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)
x = Convolution2D(64, (3, 3), activation='relu', padding='same')(x)
x = Convolution2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D()(x)
x = Convolution2D(16, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D()(x)
x = Convolution2D(1, (3, 3), activation='relu', padding='same')(x)
```

Ci-dessus est notre code pour créer un modèle simple de notre réseau de neurones. Chaque fonction va créer une couche comme dans la définition d'un réseau neuronal convolutif. On va voir chaque fonction dans les parties suivantes.

### 2.3.1 La couche de convolution (CONV)

La convolution est le processus consistant à ajouter chaque élément de l'image à ses voisins immédiats, pondéré par les éléments du noyau. C'est une forme de produit de convolution. Supposons que, on a un image en niveaux de gris : $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ et un noyau : $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$. L'élément au coordonnée $[1, 1]$ de l'image de sortie devrait être pondéré par la combinaison de les elements entrées de la matrice de l'image, avec les poids données par le noyau comme suit :

$$\left( \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \right) [1, 1] = 0 \cdot a + 0 \cdot b + 0 \cdot c + 0 \cdot d + 1 \cdot e + 2 \cdot f + 0 \cdot g + 4 \cdot h + 5 \cdot i$$

Les autres entrées devraient être pondérées de manière similaire, appliquée à tous les pixels de l'image. De manière analogie, la fonction suivante crée deux couches successives : CONV et d'activation (ReLU). La couche CONV est composée de $filters$ neurones formels. Chaque neurone est un noyau dont la taille est $size$.

```
Convolution2D(filters, size, activation='relu', padding='same')
```

### 2.3.2 La couche d'activation

La fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel. Le but d'une fonction d'activation est d'ajouter une sorte de propriété non linéaire au réseau neuronal. Sans les fonctions d'activation, la seule opération mathématique pendant la propagation directe serait des produits scalaires entre un vecteur d'entrée et une matrice de poids. Puisqu'un seul produit scalaire est une opération linéaire, les produits scalaires successifs ne seraient rien de plus que de multiples opérations linéaires répétées les unes après les autres. Afin de pouvoir calculer des choses vraiment intéressantes, les réseaux de neurones doivent être capables d'approximer les relations non linéaires des entrées aux sorties. [3]

Dans notre projet, on utilise la fonction d'activation $ReLU(x) = \max(x, 0)$

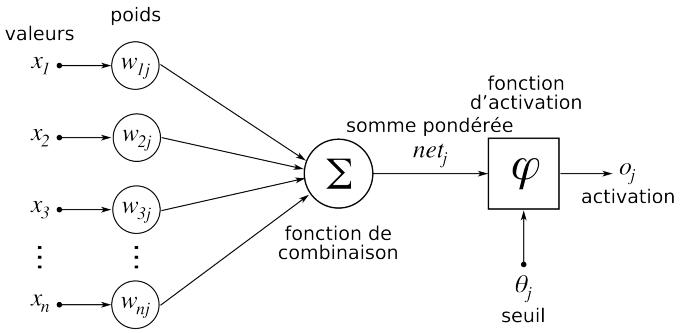


FIGURE 4 – Structure d'un neurone artificiel. Le neurone calcule la somme de ses entrées puis cette valeur passe à travers la fonction d'activation pour produire sa sortie.

### 2.3.3 La couche de pooling (POOL)

Pooling est une forme de sous-échantillonnage de l'image. L'image d'entrée est découpée en une série de rectangles de n pixels de côté ne se chevauchant pas. Chaque rectangle peut être vu comme une tuile. Le signal en sortie de tuile est défini en fonction des valeurs prises par les différents pixels de la tuile. Le pooling réduit la taille spatiale d'une image intermédiaire, réduisant ainsi la quantité de paramètres et de calcul dans le réseau [2]

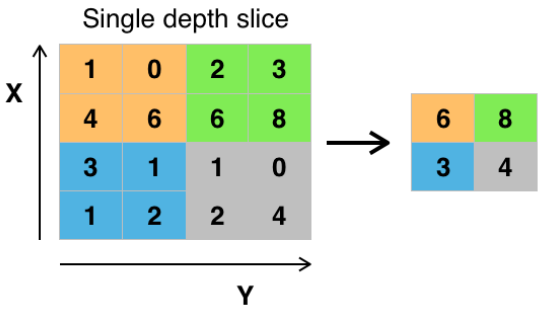


FIGURE 5 – Max pooling avec un filtre $2 \times 2$

Enfin, afin d'obtenir un image de sortie ayant le même taille avec l'image d'entrée, on utilise la fonction UpSampling2D qui répète les lignes et les colonnes des données respectivement par size [0] et size [1] :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 4 & 4 & 5 & 5 & 6 & 6 \\ 4 & 4 & 5 & 5 & 6 & 6 \\ 7 & 7 & 8 & 8 & 9 & 9 \\ 7 & 7 & 8 & 8 & 9 & 9 \end{bmatrix}$$

### 2.3.4 Fonction de coût et optimisation

Après ajouter des couches au réseau. On va choisir la fonction de coût et le technique d'optimisation. On utilisera MSE ( Mean squared error ) comme la fonction de coût et Adam ( Adaptive moment estimation ) comme le technique d'optimisation. Finallement, on compile le model. L'architecture de ce réseau est le diagramme suivant.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 1)]   0
_________________________________________________________________
conv2d (Conv2D)              (None, None, None, 16)    160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, None, None, 16)    0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, None, None, 32)    4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, None, None, 32)    0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, None, None, 64)    18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, None, None, 32)    18464
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, None, None, 32)    0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, None, None, 16)    4624
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, None, None, 16)    0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, None, None, 1)     145
=================================================================
Total params: 46,529
Trainable params: 46,529
Non-trainable params: 0
```

FIGURE 6 – Architecture d'un réseau neuronal convolutif. Les "Trainable params" sont les poids que on doit calculer dans le phase d'entrainement

## 2.4 Entrainement

### 2.4.1 Algorithm

On notera $f : \theta \mapsto \mathbb{R}$ la fonction de coût, avec $\theta = [\alpha_0, \ldots, \alpha_p]$ le vecteur de poids. Dans cette partie, on va préciser comment Adam fonctionner pour calculer les poids de réseau pour minimiser la fonction de coût. D'abord, on regarde GD ( Gradient Descent ) qui est la fondation d'Adam.

On rappelle que dans un repère orthonormé, le vecteur gradient $\nabla f$ pointe dans la direction où la fonction croît le plus rapidement. Donc, dans le voisinage d'un point $\theta$, la fonction $f$ décroît le plus fortement dans la direction opposée à celle du $\nabla f$ en $\theta$. L'algorithm GD peut être décrire dans la manière suivante : [4]

1. Calcul de $\nabla f(\theta_k)$
2. Si $\|\nabla f(\theta_k)\| \leqslant \epsilon$, arrêt.
3. $\theta_{k+1} = \theta_k - \gamma \nabla f(\theta_k)$

$\gamma$ est une valeur scalaire appelée taux d'apprentissage pour déterminer le point suivant. Mais, le problème peut survenir si le taux d'apprentissage est mal choisi.

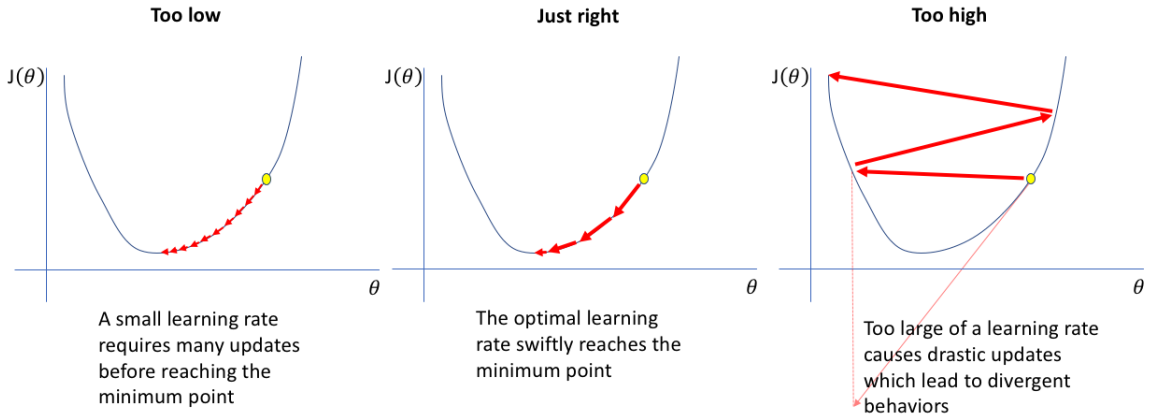


FIGURE 7 – 3 graphes montrant 3 taux d'apprentissage différents : trop petit (gauche), optimal (milieu), trop élevé (droite)

Adam résout ce problème par calcule des taux d'apprentissage adaptatifs individuels pour différents poids.

### 2.4.2 Réalisation

```
model.fit(y_train_ext, x_train_ext,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(y_test_ext, x_test_ext))
```

y_train_ext est les images d'entrées avec du bruit gaussien, x_train_ext est les image originals. On a besion de validation_data pour éviter les problèmes concernant "overfiting". Epochs est le nombre de pas de descente dans l'optimisation, et batch_size est un nombre d'échantillons traités avant la mise à jour du modèle.

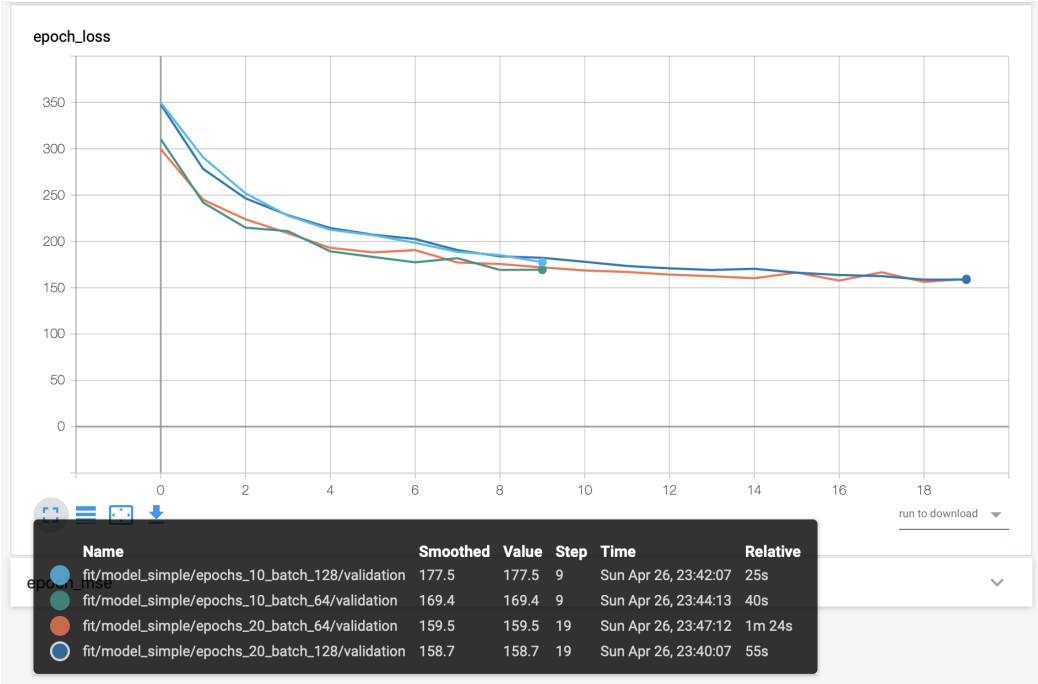


FIGURE 8 – L'impact d'epochs et de batch_size sur la fonction de coût

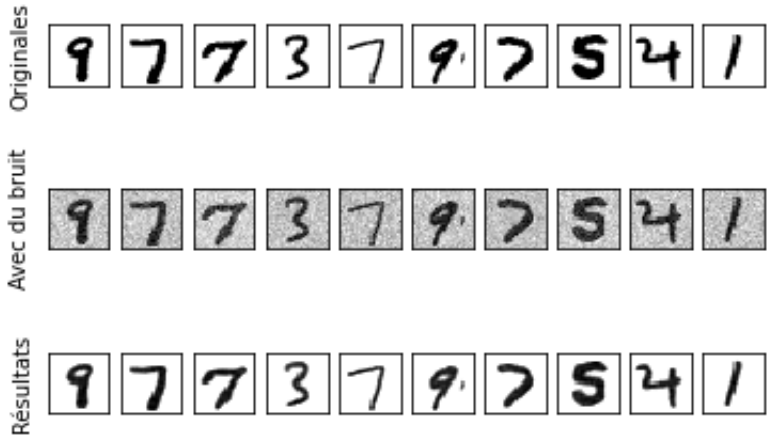


FIGURE 9 – Résultat de réseau simple avec epochs = 20 et batch_size = 64

# 3 Autre architecture pour le réseau neuronal convolutif

## 3.1 Architecture linéaire

# Références

[1] https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-rn.pdf

[2] https://fr.wikipedia.org/wiki/Réseau_neuronal_convolutif/

[3] https://www.deeplearning-academy.com/p/ai-wiki-activation-functions

[4] https://fr.wikipedia.org/wiki/Algorithme_du_gradient