ROB313 - PERCEPTION POUR LES SYSTÈMES AUTONOMES

TP 4 : Deep learning with PyTorch : CIFAR10 object classification

14 février 2020

Gabriel Henrique Riqueti

Victor Kenichi Nascimento Kobayashi

ENSTA IP Paris

Ce travail peut être accéder par le lien https://github.com/gabrielriqu3ti/ROB313_Perception_pour_les_systemes_autonomes/tree/master/TP4

Importation des données

D'abord on a noté que l'algorithme originale n'utilisait pas toutes les données disponibles de la base. Alors, on a augmenté le nombre des images d'entraînement jusqu'à 47500 et de validation jusqu'à 2500. C'est-à-dire qu'on a doublée la base de données totale et on a réduit la base de validation par la moitié. L'usage de d'une base de données plus importante nous permet d'entraîner la réseau de neurones dans plusieurs cas différents, en favorisent la généralisation.

Création du Modèle

Le modèle final utilisé est montré dans la figure 1. Le modèle est un CNN avec 4 couches de convolution et avec 3 chemin différents On a créé plusieurs couches afin de détecter d'abstractions et des caractéristiques plus complexes et on a définit plusieurs chemins pour obtenir des caractéristiques à plusieurs échelles différentes. On remarque que cette complexité peut augmenter trop les nombre de paramètres du modèle et rendre la base de données insuffisante par rapport au nombre de paramètres du modèle enchaînant *overfitting*.

Dans le premier, on n'utilise que 3 couches de *MaxPooling2D* pour retenir l'information de l'image originale. On signale que toutes les couches de *MaxPooling2D* on un noyeau de taille 2 et un *stride* égal à 2, alors il réduit l'échelle de l'entrée par 4 en retenant les valeur maximales dans la voisinage 2x2. Dans le deuxième, on réalise des convolutions avec 6 noyaux de taille 3, puis on réalise un opération de *MaxPooling2D*. On repète ces deux couches 3 fois dont la dernière, on n'utilise pas *MaxPooling2D*, et dont les deux dernières, on utilise 3 noyaux de taille 3. Dans le dernier chemin, on a copié le deuxième chemin sauf qu'on utilise

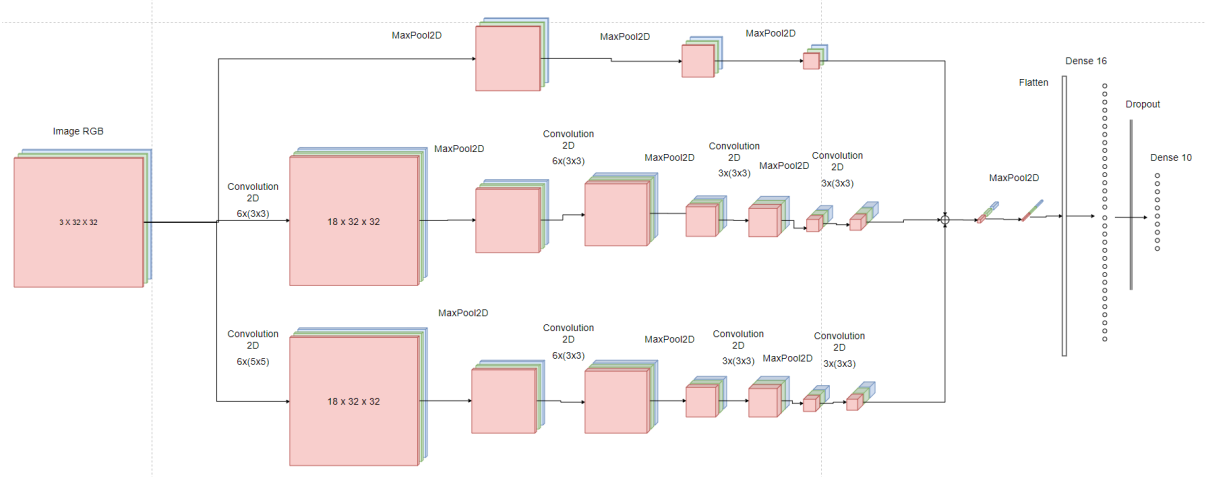

FIGURE 1: Architecture de la réseau de neurone convolutionnel.

6 noyaux de taille 5 pour les deux premières convolutions.

Après ces étapes, on concatène le résultat de tous ces chemin et on applique une couche de *MaxPooling2D* et une couche qui rend les informations linéaires, parfaites pour introduire dans une réseau de neurones artificielle classique. La notre a un couche cachée dense avec 16 neurones. Afin de réduire l'*overfitting*, on ajoute une couche de Dropout qui supprime la sortie d'un neurone de la couche précédente avec une probabilité de 20%. Enfin, on a une couche de 10 perceptrons qui représentent les 10 classes de la base de données.

Concernant les fonctions d'activation, on utilise toujours la fonction PReLu après les convolutions et la couche cachée dense. Cette fonction est intéressant parce qu'elle ajoute seulement une paramètre à chaque opération en permettant une flexibilité plus importante que la fonction classique ReLu.

Entraînement du modèle

Pour l'entraînement, on a utilisé $batch\_size = 128$ et $learning\_rate = 0,001$ pour réaliser un entraînement plus stable et moins vite et $n\_epochs = 4$ car l'algorithme commence à souffrir d'*overfitting* avant de la quatrième époque. Les résultats sont les suivants :

Accuracy of the network on the 47500 train images : 88.22% Accuracy of the network on the 2500 validation images : 77.32% Accuracy of the network on the 5000 test images : 75.60%

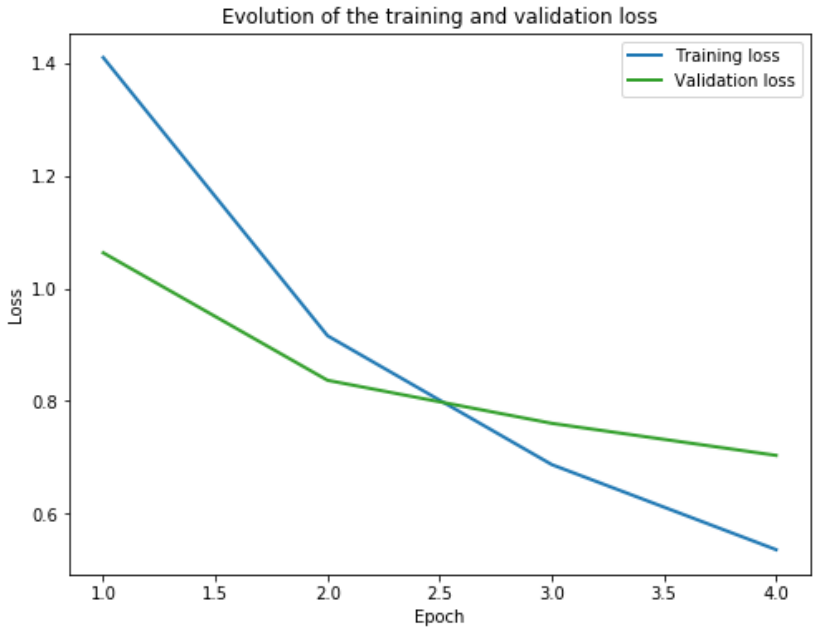

FIGURE 2: Graphique de la perte des bases d'entraînement et de validation.

Dans la Figure 2, on voit que à partir de 3 epochs, la perte d'entraînement est plus petit que la perte de validation. Ça veut dire que le modèle commence à être surentraîné et il commence à apprendre le bruit des images de la base de l'entraînement au lieu des caractéristiques communes à classes. Dans le Tableau 1, on voit que l'algorithme a des difficultés pour différencier les chiens des chats cependant le résultat est très supérieure à la choix par chance (

Classe	**Éxactitude** (%)
plane	80,74
car	87,13
bird	60,94
cat	53,92
deer	69,63
dog	76,23
frog	80,65
horse	76,36
ship	86,71
truck	83,63

Tableau 1: Éxactitude par classe

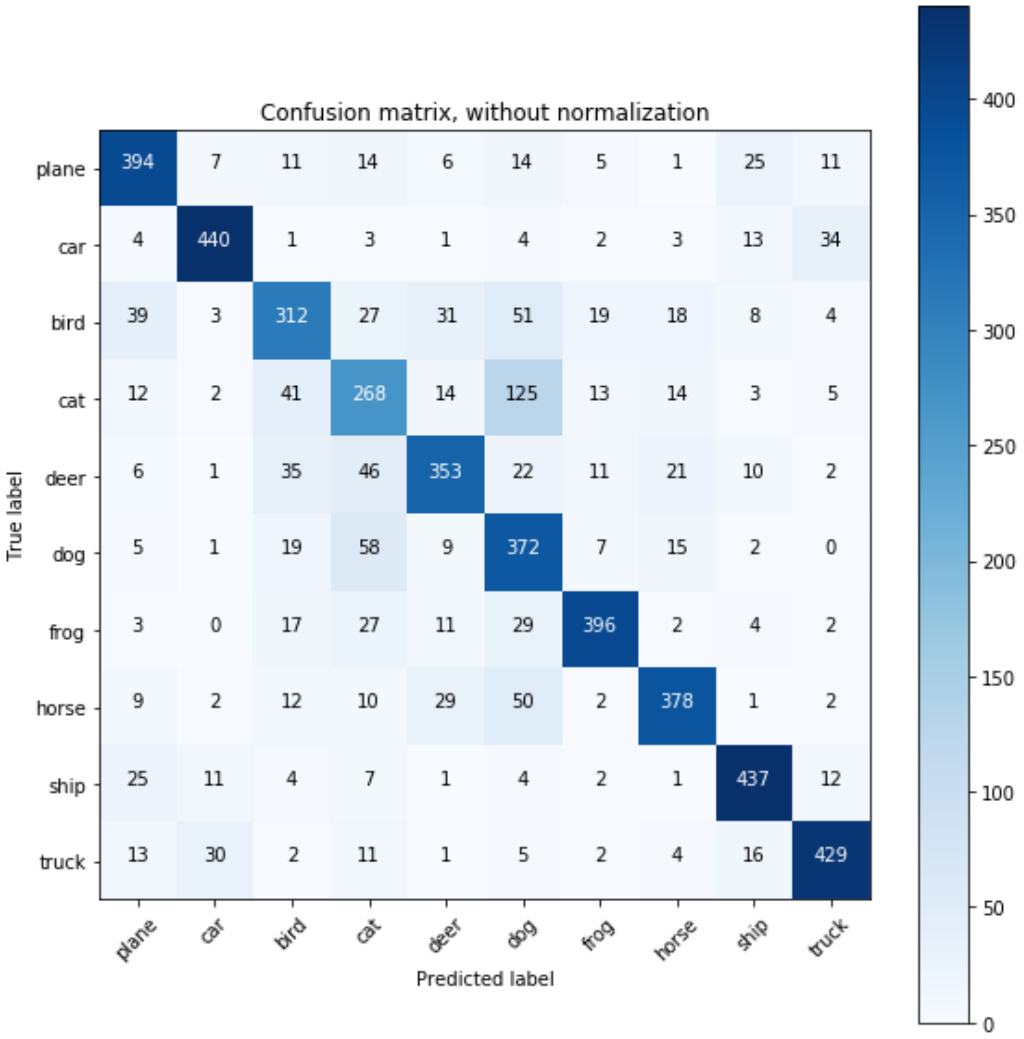

FIGURE 3: Matrice de confusion.