Rapport compétition Kaggle : classification de MNIST par CNN

Sommaire :

Description du projet:

L'objectif de ce projet est de comparer différentes architectures Deep Learning sur une problématique de classification d'images. Le dataset utilisé est MNIST; un dataset constitué d'images de chiffres manuscrits.
L'objectif est de prédire le chiffre associé à chaque images.
Les résultats seront finalement publiés sur la compétition Kaggle dédiée.
Résultats finaux :

- Accuracy obtenue sur les données test : 0.99757 pour un total de 138,682 parametres (pour l'architecture la plus efficace). **TOP 4% Kaggle**
- Temps d'apprentissage: 8mn. Machine: CPU 2 coeurs Intel Core i5 cadencé à 2.7 GHz // GPU Intel Iris Graphics 6100 1536 Mo

Lien vers le leaderboard: https://www.kaggle.com/c/digit-recognizer/leaderboard (https://www.kaggle.com/c/digit-recognizer/leaderboard)

Récupération et traitement initial des données:

Après avoir sauvé le dataset MNIST dans un DataFrame dédié, nous normalisons les données.
Puis afin de tester la qualité de nos modéles, nous allons séparer aléatoirement le dataset en un jeu d'apprentissage et un jeu de données test: X_train, y_train et X_test, y_test.
On fixe également une taille de batch de 20 pour tout les apprentissages (fit) des modéles testés ici et nous utiliserons un **optimizer** de type **Adam**.

Premier modéle : Un perceptron multicouches

Le premier modéle testé ici est un perceptron multicouches simple. L'architecture choisie est la suivante:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 50)                1650
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
=================================================================
Total params: 27,280
Trainable params: 27,280
Non-trainable params: 0
_________________________________________________________________
None
```

Le choix de 3 couches repose sur une étude préalable semblant indiquer l'apparition d'overfitting (pour ce nombre de neuronnes par couche) au dela de 3. En revanche le nombre relativement faible de neuronnes par couche cachée est le résultat d'une volonté de limiter le temps d'apprentissage. (L'objectif de l'étude étant de se focaliser sur des CNN.)

Résultats:
Après un apprentissage sur 10 epochs on obtient:

- sur les données d'apprentissage:
  - une loss value de 0.0502
  - une accuracy de 0.9841
- sur les données test:
  - une loss value de : 0.1060
  - une accuracy de 0.9702

Pour une durée d'apprentissage d'approximativement 1mn.

Modéles à convolutions

Afin d'améliorer la qualité dans la prédiction de classes, nous allons ajouter des couches de convolutions à notre modéle.
Etant donné qu'une couche de convolution reçoit en entré un tenseur, il est d'abord nécessaire de modifier le format des différents dataset (**X_train** et **X_test**) vers un format du type : **(Nombre d'images)**×**(Longeur images)**×**(Largueur images)**×**(Profondeur images)**

Second modéle: CNN

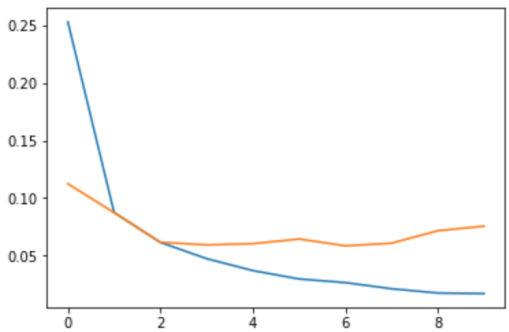
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 10)        100
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 10)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 10)        910
_________________________________________________________________
flatten_1 (Flatten)          (None, 1210)              0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                38752
_________________________________________________________________
dense_2 (Dense)              (None, 50)                1650
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
=================================================================
Total params: 41,922
Trainable params: 41,922
Non-trainable params: 0
_________________________________________________________________
None
```

Cette seconde architecture consiste à ajouter 2 couches de convolutions (séparées par une couche de max pooling) au réseau précédent.
Pour réduire le temps de calcule et améliorer la qualité du modéle on peut également jouer sur le padding et le strides des convolutions ou encore utiliser un noyau dilaté (peu utile au regard de la résolution des images).
L'ajout d'une couche de pooling permet de réduire la variance, réduire la compléxité (et donc le temps de calcule) et limite l'overfitting.
Résultats:
Après un apprentissage sur 10 epochs on obtient:

- sur les données d'apprentissage:
  - une loss value de 0.0502
  - une accuracy de 0.9841
- sur les données test:
  - une loss value de : 0.1060
  - une accuracy de 0.9702

Pour une durée d'apprentissage d'approximativement 10 mn.
On observe ainsi qu'avec cette architecture, l'accuracy sur les données d'apprentissage est de 99.9%, contre 99.3114% sur les données test (sans le maxpool et le second cnn), et 98.8871% avec. Nous pouvons donc soupçonner l'apparition d'overfitting.

Plusieurs solution s'offrent alors à nous:
Nous pouvons avant tout commencer par afficher la courbe de la validation-loss selon le nombre d'epochs effectués afin d'avoir une idée de l'évolution du modéle au cours de la phase d'apprentissage:



On observe que rapidement (après 4 à 5 epochs) la valeur de l'entropie croisée sur les données test stagne et ré-augmente. Il y a donc bien overfiting.
Afin d'éviter celà plusieurs solutions éxisent:

- L'ajout d'un **callback** lors de l'apprentissage du réseau: Cela permet notamment de stoper l'apprentissage (**EarlyStopping**) lorsque la valeur monitorée (ici la val_loss associée à la cross validation) n'augmente plus après $n_{Patience}$ epochs.
- L'ajout d'une **liste callback** grâce à un **checkpoint**: L'objectif de cette méthode est de stocker après chaque époch la valeur des différents paramètres du réseau, afin de pouvoir à postériori conserver ceux qui minimisent l'erreur sur les données test.
- Modifier l'architecture du réseau: par ajout de couches de **Dropout** et de **BatchNormalization**.

Modéle final de CNN:

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 16)	160
conv2d_4 (Conv2D)	(None, 28, 28, 16)	2320
max_pooling2d_2 (MaxPooling2	(None, 14, 14, 16)	0
dropout_1 (Dropout)	(None, 14, 14, 16)	0
batch_normalization_1 (Batch	(None, 14, 14, 16)	64
conv2d_5 (Conv2D)	(None, 14, 14, 32)	4640
conv2d_6 (Conv2D)	(None, 14, 14, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 7, 7, 32)	0
dropout_2 (Dropout)	(None, 7, 7, 32)	0
batch_normalization_2 (Batch	(None, 7, 7, 32)	128
conv2d_7 (Conv2D)	(None, 5, 5, 64)	18496
batch_normalization_3 (Batch	(None, 5, 5, 64)	256
flatten_2 (Flatten)	(None, 1600)	0
dense_4 (Dense)	(None, 64)	102464
batch_normalization_4 (Batch	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 10)	650

Total params: 138,682
Trainable params: 138,330
Non-trainable params: 352

L'ajout de couches de pooling diminuant le nombre de parametres on peut augmenter le nombre de noyaux sur les couches de convolutions et créer une architecture plus profonde.
De fait, nous avons ajouté des couches de **dropout**, mais aussi de **batch_normalization** dans le but de réduire l'overfitting.

L'apprentissage est alors exécuté sur 30 epochs avec une **liste callback**.

Résultats:
Après un apprentissage sur 30 epochs on obtient une **accuracy sur les données test de 0.99757** ; Pour une durée d'apprentissage d'approximativement 10 mn.

In [ ]: