Rapport du projet de Machine Learning :

Construction d'un convolutional Neural Network pour la reconnaissance de nombres écrits à la main

Réalisé par:

AZDAD Mohamed

ALLA Mina

SAHRAOUI Wiaam

Encadré par :

Mr. MAHMOUDI Abdelhak

Année scolaire : 2019/2020

Dans ce projet, nous avons essayé d'implémenter un réseau de neurones convolutifs sur Python. Ce réseau a pour rôle de recevoir en entrée une image d'un nombre de 0 à 9 écrit à la main et de prédire ce nombre en sortie.

Pour ce faire : nous utiliserons **la base donnée MNIST** qui contient des images de dimensions 28x28 pixels chacune.

Commençons tout d'abord par importer toutes les bibliothèques dont nous aurons besoin, pour construire ce modèle, pour manipuler les données et aussi pour construire des graphes.

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

L'étape suivante consiste à charger notre base de données et de la diviser en deux. **Une training set** sur laquelle va s'entraîner le modèle et une **test set** sur laquelle il va tester.

```
mnist = tf.keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Nous aimerions bien savoir **les dimensions des deux data set** produites à l'étape précédente.

```
[ ] X_train.shape
```

```
(60000, 28, 28)
```

```
[ ] X_test.shape
```

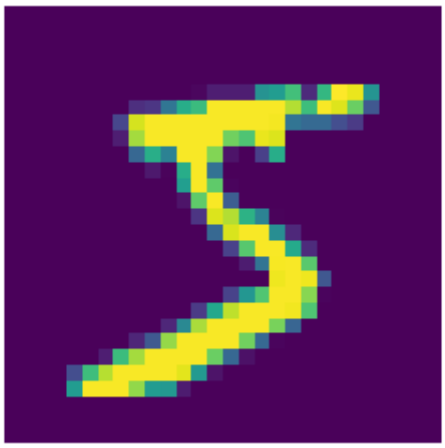
```
(10000, 28, 28)
```

Donc **la base de données d'entraînement** contient 60,000 images de dimensions 28x28 pixels et **la base de données de test** contient 10,000 images aussi de dimensions 28x28 pixels chacune.

A ce niveau, nous pouvons bien jeter un coup d'œil sur ces images, prenons l'exemple de la première image de notre base d'entraînement. Le code à exécuter est le suivant :

```
[ ]  plt.imshow(X_train[0])
     plt.axis('off')
     plt.show()
```

Comme sortie de ce code, nous obtenons l'image suivante :



Nous voyons bien que le nombre écrit dans cette image est 5. Pour vérifier cela nous devons tout simplement voir le label correspondant dans la base de données y_train :

```
y_train[0]
```

```
5
```

```
[ ]  y_train
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

Comme on peut le voir, le nombre est un 5.

Nous remarquons aussi dans l'image ci-dessus que les labels dans la base y_train sont sous forme des nombre allons de 0 à 9 stockés dans un tableau.

```
print(X_train.dtype)
print(X_test.dtype)
```

```
uint8
uint8
```

Ainsi dans l'image ci-dessus, la densité de pixel de chaque image est représenté par **un byte de 0 à 255.** Cela conduit au besoin de normalisation des images à fin que les intensités des pixels varient entre 0 et 1:

```
[ ] X_train=X_train.reshape(60000, 28, 28, 1)
    X_test=X_test.reshape(10000, 28, 28, 1)
```

Maintenant, nous voulons changer la forme des labels des deux bases y_train et y_test. Nous remplaçons chaque nombre de ces bases par une table contenant 10 cases, toutes les cases ont la valeur 0 à part la case correspondante au nombre que représente ce label est mise à 1. Cela se fait par ce qu'on appelle le one-hot encoding :

```
y_train_one_hot=to_categorical(y_train)
y_test_one_hot=to_categorical(y_test)

## print the new label
print(y_train_one_hot[0])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Comme nous pouvons le voir : **la case numéro 5 du premier élément est mise à 1.**

Maintenant nous pouvons passer à la construction de notre modèle et ajouter des couches à celui-ci.

Dans notre cas, nous utilisons *deux couches convolutives* : la première contient 64 nœuds et la deuxième contient 32 nœuds. Ainsi la dernière couche contient 10 nœuds (*le même nombre de sorties possibles*).

```
model = keras.models.Sequential()
#add model layers
model.add(keras.layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=[28,28,1]))
model.add(keras.layers.Conv2D(32, kernel_size=3, activation="relu"))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation="softmax"))
```

Nous pouvons voir les couches de notre modèle comme suit :

```
[ ]  model.layers
```

```
[<tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fb8989b8c88>,
 <tensorflow.python.keras.layers.convolutional.Conv2D at 0x7fb8989b5198>,
 <tensorflow.python.keras.layers.core.Flatten at 0x7fb8989b8a20>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fb8989b8a58>]
```

Ainsi nous pouvons explorer un résumé de notre modèle :

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 32)        18464
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0
_________________________________________________________________
dense (Dense)                (None, 10)                184330
=================================================================
Total params: 203,434
Trainable params: 203,434
Non-trainable params: 0
_________________________________________________________________
```

A ce niveau, nous sommes menés à compiler le modèle que nous avons construit :

```
[ ]  #compile the model
     model.compile(loss="categorical_crossentropy",
                   optimizer="adam",
                   metrics=["accuracy"])
```

Puis nous entraînons le réseau en utilisant nos données d'entraînement pour l'entraînement et les données de test pour la validation et la base va passer 30 fois au réseau (epochs = 30).

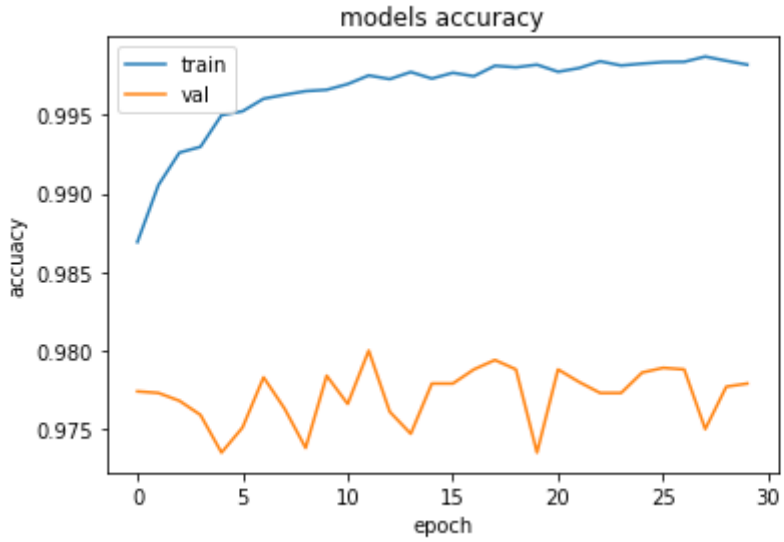
```
hist = model.fit(X_train, y_train_one_hot, validation_data=(X_test, y_test_one_hot), epochs=30)
```

```
Epoch 1/30
1875/1875 [==============================] - 16s 9ms/step - loss: 0.0392 - accuracy: 0.9869 - val_loss: 0.1043 - val_accuracy: 0.9774
Epoch 2/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0300 - accuracy: 0.9905 - val_loss: 0.1077 - val_accuracy: 0.9773
Epoch 3/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0235 - accuracy: 0.9926 - val_loss: 0.1358 - val_accuracy: 0.9768
Epoch 4/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0248 - accuracy: 0.9930 - val_loss: 0.1551 - val_accuracy: 0.9759
Epoch 5/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0159 - accuracy: 0.9950 - val_loss: 0.1839 - val_accuracy: 0.9735
Epoch 6/30
1875/1875 [==============================] - 16s 8ms/step - loss: 0.0170 - accuracy: 0.9952 - val_loss: 0.1497 - val_accuracy: 0.9751
Epoch 7/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0156 - accuracy: 0.9961 - val_loss: 0.1935 - val_accuracy: 0.9783
Epoch 8/30
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0151 - accuracy: 0.9963 - val_loss: 0.2208 - val_accuracy: 0.9763
Epoch 9/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0155 - accuracy: 0.9965 - val_loss: 0.2575 - val_accuracy: 0.9738
Epoch 10/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0140 - accuracy: 0.9966 - val_loss: 0.2797 - val_accuracy: 0.9784
Epoch 11/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0154 - accuracy: 0.9970 - val_loss: 0.2833 - val_accuracy: 0.9766
Epoch 12/30
1875/1875 [==============================] - 16s 8ms/step - loss: 0.0122 - accuracy: 0.9975 - val_loss: 0.2720 - val_accuracy: 0.9800
Epoch 13/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0152 - accuracy: 0.9973 - val_loss: 0.3053 - val_accuracy: 0.9761
Epoch 14/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0137 - accuracy: 0.9977 - val_loss: 0.3424 - val_accuracy: 0.9747
Epoch 15/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0171 - accuracy: 0.9973 - val_loss: 0.3438 - val_accuracy: 0.9779
Epoch 16/30
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0142 - accuracy: 0.9977 - val_loss: 0.3292 - val_accuracy: 0.9779
Epoch 17/30
1875/1875 [==============================] - 16s 9ms/step - loss: 0.0158 - accuracy: 0.9975 - val_loss: 0.4005 - val_accuracy: 0.9788
Epoch 18/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0123 - accuracy: 0.9981 - val_loss: 0.3780 - val_accuracy: 0.9794
Epoch 19/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0147 - accuracy: 0.9980 - val_loss: 0.3884 - val_accuracy: 0.9788
Epoch 20/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0128 - accuracy: 0.9982 - val_loss: 0.4838 - val_accuracy: 0.9735
Epoch 21/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0194 - accuracy: 0.9978 - val_loss: 0.4704 - val_accuracy: 0.9788
Epoch 22/30
1875/1875 [==============================] - 16s 9ms/step - loss: 0.0148 - accuracy: 0.9980 - val_loss: 0.5364 - val_accuracy: 0.9780
Epoch 23/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0119 - accuracy: 0.9984 - val_loss: 0.5987 - val_accuracy: 0.9773
Epoch 24/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0185 - accuracy: 0.9982 - val_loss: 0.6665 - val_accuracy: 0.9773
Epoch 25/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0156 - accuracy: 0.9983 - val_loss: 0.5333 - val_accuracy: 0.9786
Epoch 26/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0149 - accuracy: 0.9984 - val_loss: 0.6123 - val_accuracy: 0.9789
Epoch 27/30
1875/1875 [==============================] - 16s 8ms/step - loss: 0.0178 - accuracy: 0.9984 - val_loss: 0.6333 - val_accuracy: 0.9788
Epoch 28/30
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0140 - accuracy: 0.9987 - val_loss: 0.7550 - val_accuracy: 0.9750
Epoch 29/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0166 - accuracy: 0.9985 - val_loss: 0.7564 - val_accuracy: 0.9777
Epoch 30/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0237 - accuracy: 0.9982 - val_loss: 0.8366 - val_accuracy: 0.9779
```

Pour **visualiser la variation de l'occurrence de notre modèle** (l'occurrence sur le training set et l'occurrence sur le test se), nous exécutons le code suivant :

```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('models accuracy')
plt.ylabel('accuacy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

Le résultat de ce code est le graph suivant :



Nous passons à **l'évaluation de notre modèle :**

```
[ ]  model.evaluate(X_test, y_test_one_hot)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.8366 - accuracy: 0.9779
[0.8365880250930786, 0.9779000282287598]
```

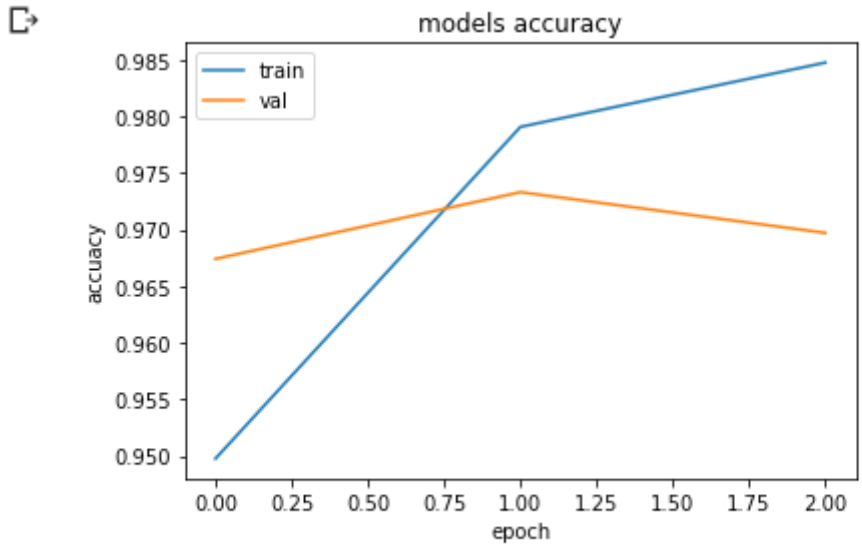
Donc notre modèle a **une occurrence de 97,79 %** et **un coût de 0,8366.**

Nous ré-entraînons le modèle mais en utilisant cette fois-ci seulement **3 «epoch»**, ce qui veut dire que nos données vont être passées au modèle **seulement 3 fois**. Nous faisons cela pour effectuer une comparaison entre les deux modèle **(comparer les occurrences et les coûts des deux modèles).**

```
[ ]  #train the model
     hist = model.fit(X_train, y_train_one_hot, validation_data=(X_test, y_test_one_hot), epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2748 - accuracy: 0.9498 - val_loss: 0.1052 - val_accuracy: 0.9674
Epoch 2/3
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0674 - accuracy: 0.9791 - val_loss: 0.0871 - val_accuracy: 0.9733
Epoch 3/3
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0507 - accuracy: 0.9847 - val_loss: 0.1174 - val_accuracy: 0.9697
```

La visualisation de la variance des occurrences du nouveau modèle conduit au résultat suivant :



Et son évaluation conduit à cela :

```
#model evaluation
model.evaluate(X_test, y_test_one_hot)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1174 - accuracy: 0.9697
[0.11735222488641739, 0.9696999788284302]
```

Donc notre modèle a **une occurrence de 96,97 %** et **un coût de 0,1174.**

Nous remarquons que quand epoch=30, nous obtenons une occurrence supérieure à celle avec epoch=3, mais cette différence est négligeable si nous jetons un coup d'œil sur **la variable loss (le coût)** qui a beaucoup augmenté quand nous avons augmenté la variable epoch.

Et donc i**l vaut mieux utiliser la valeur 3 pour la variable epoch et cela n'affectera pas notre modèle.**

Nous aimerions bien savoir ce que reçoit la dernière couche. En vérité , ce sont des tables de probabilités qu'une entrée peut correspondre au nombre de cette case. *Jetons un coup d'œil sur les probabilités des prédictions pour les 4 premières images de la base test set :*

```
predictions = model.predict(X_test[:4])
predictions
```

```
array([[3.4993758e-11, 8.4847489e-17, 1.8387785e-09, 1.7651692e-08,
        4.1930871e-15, 4.4109906e-17, 2.6763274e-19, 1.0000000e+00,
        4.7807994e-11, 4.8261319e-11],
       [1.8425417e-09, 5.2083539e-07, 9.9999797e-01, 9.5549308e-07,
        1.2018931e-13, 3.2929388e-14, 3.0872854e-08, 2.5675142e-14,
        6.1768623e-07, 2.4027787e-17],
       [3.3947789e-10, 9.9626428e-01, 3.7854170e-06, 8.0757516e-09,
        6.1057566e-04, 2.9027674e-06, 3.2507124e-08, 3.8896200e-05,
        3.0795720e-03, 2.8716161e-09],
       [9.9994600e-01, 1.7275079e-15, 7.4413798e-08, 1.3865650e-11,
        5.6386987e-12, 1.5981205e-10, 5.3929914e-05, 3.5005616e-11,
        8.0440978e-09, 2.3779609e-10]], dtype=float32)
```

Dans la première table, la probabilité la plus grande est à la position 7 de la table donc nous pouvons dire *que la première image de la base de test contient un 7.*

De même pour les trois autres images (2 puis 1 puis 0).

Pour vérifier cela, nous pouvons **extraire les 4 premiers labels** estimés sous forme de nombres **ainsi que les 4 premiers nombres qu'on a dans y_test :**

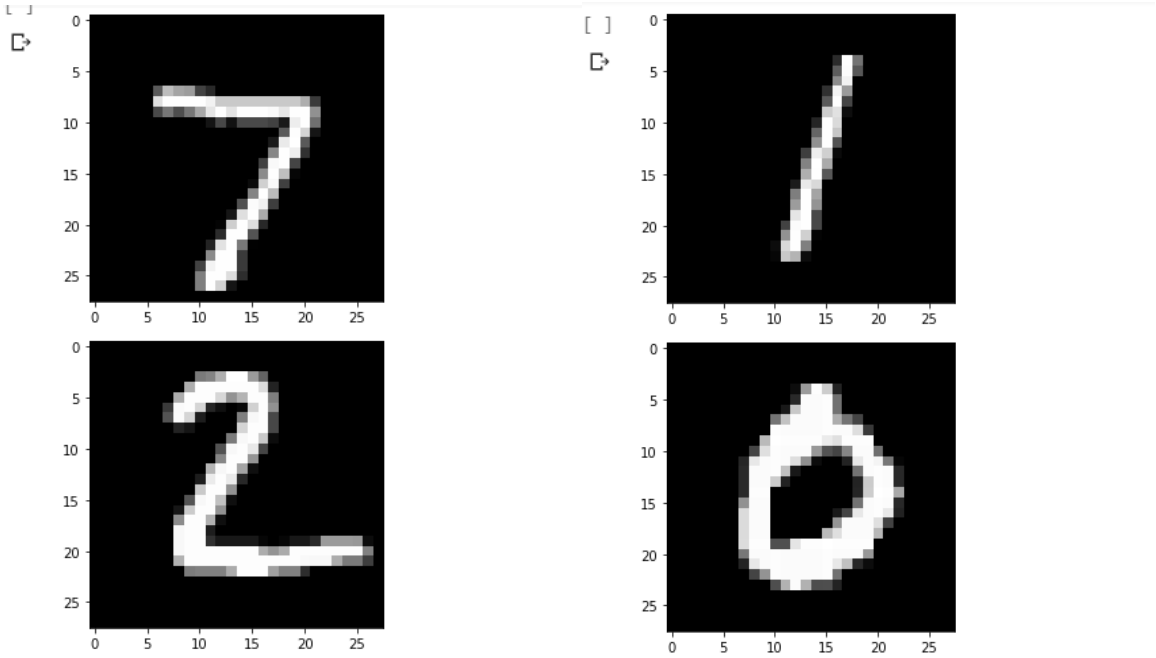
```
[ ]  #print out predictions as number labels for the first 4 images
     print(np.argmax(predictions, axis=1))
     #print out the actual labels
     print(y_test[:4])
```

```
[7 2 1 0]
[7 2 1 0]
```

Et si nous voulons les voir sous forme d'images originales, nous procédons comme suit:

```
for i in range(0,4):
    image = X_test[i]
    image = np.array(image, dtype= 'float')
    pixels = image.reshape((28,28))
    plt.imshow(pixels, cmap='gray')
    plt.show()
```

Et voici le résultat affiché :



Conclusion :

Dans ce projet, nous avons construit un réseau de neurones convolutives pour la reconnaissance des nombres de 0 à 9 et nous l'avons entraîné avec deux différentes valeurs de **la variable epoch** (le nombre de fois où nos données passent dans le modèle pour l'entraîner) pour effectuer une comparaison et choisir le modèle le plus intéressant.