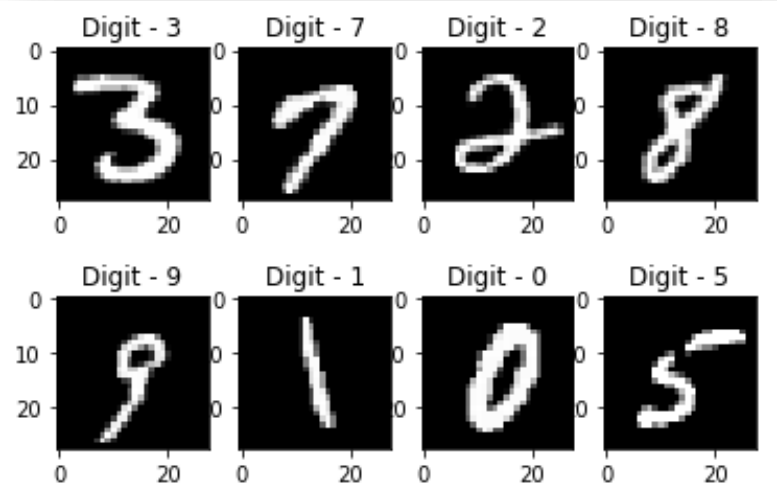


HANDWRITTEN NUMBER RECOGNITION

With PYTHON

iliasoo110@gmail.com

By : ILIAS TAHRI

Filière : IA&GI

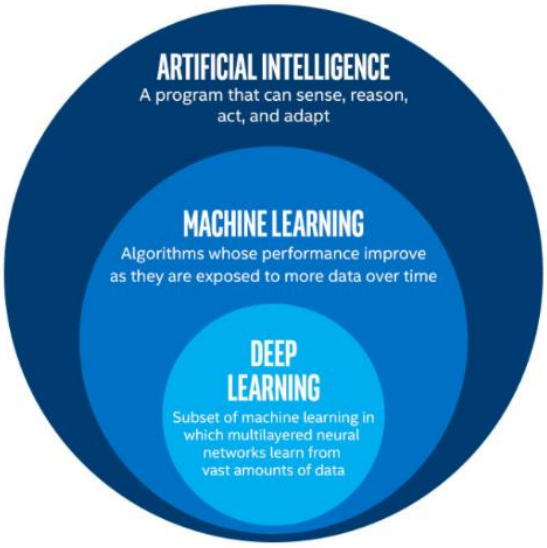
SOMMAIRE :

Introduction

Pour rendre les machines plus intelligentes, les développeurs plongent dans l'apprentissage des machines et dans les techniques d'apprentissage en profondeur. Un humain apprend à exécuter une tâche en s'exerçant et en la répétant encore et encore afin de mémoriser la façon dont il doit l'exécuter. Ensuite, les neurones de son cerveau se déclenchent automatiquement et ils peuvent rapidement exécuter la tâche qu'ils ont apprise. L'apprentissage profond est également très similaire à cela. Il utilise différents types d'architectures de réseaux de neurones pour différents types de problèmes. Par exemple - reconnaissance d'objets, classification d'images et de sons, détection d'objets, segmentation d'images, etc.

C'est quoi le Deep Learning ?

L'apprentissage approfondi est un sous-ensemble de l'apprentissage automatique, qui est par ailleurs un sous-ensemble de l'intelligence artificielle. L'intelligence artificielle est un terme général qui fait référence aux techniques permettant aux ordinateurs d'imiter le comportement humain. L'apprentissage machine représente un ensemble d'algorithmes formés sur des données qui rendent tout cela possible.



L'apprentissage profond, en revanche, n'est qu'une sorte d'apprentissage automatique, inspiré par la structure du cerveau humain. Les algorithmes d'apprentissage profond tentent de tirer des conclusions similaires à celles des humains en analysant continuellement les données avec une structure logique donnée. Pour y parvenir, l'apprentissage profond utilise une structure d'algorithmes à plusieurs niveaux appelée réseaux de neurones.

MNIST dataset :

Le MNIST (Modified National Institute of Standards and Technology database) est probablement l'un des datasets les plus populaires parmi les amateurs de Machine Learning et Deep Learning . L'ensemble de données du MNIST contient 60 000 petites images d'entraînement carrées de 28×28 pixels en mode Grayscale contenants des chiffres manuscrits de 0 à 9 et 10 000 images pour les tests. L'ensemble de données du MNIST comprend donc 10 classes différentes.

Steps :

1. Importer les Bibliotheques et charger le dataset :

Avant de commencer quoi que ce soit, assurez-vous que Tensorflow, Keras, numpy et pillow sont installés sur votre ordinateur. Nous devons importer tous les modules dont nous aurons besoin pour former notre modèle. La bibliothèque de Keras contient déjà quelques ensembles de données et le MNIST est l'un d'entre eux. Nous pouvons donc facilement importer l'ensemble de données par le biais de Keras. La méthode mnist.load_data() renvoie les données de formation, leurs étiquettes ainsi que les données de test et leurs étiquettes.

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K


# the data, split between train and test sets
from keras.utils import np_utils
from sklearn.model_selection import KFold
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.optimizers import SGD


# the MNIST data is split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

2. Prétraitement des données :

La dimension des données d'entrainement est (60000, 28, 28). Les données d'image ne peuvent pas être introduites directement dans le modèle, nous devons donc effectuer certaines opérations et traiter les données pour les préparer à notre réseau neuronal. Dans notre cas, CNN accepte quatre dimensions. Nous devons donc remodeler les images pour qu'elles aient des dimensions (échantillons*largeur*hauteur*pixels) .

```
# Reshape to be samples*pixels*width*height
X_train = X_train.reshape(X_train.shape[0], 28, 28,
1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

De nombreux algorithmes d'apprentissage machine ne peuvent pas fonctionner directement sur les données des étiquettes. Ils exigent que toutes les variables d'entrée et de sortie soient numériques. La technique pour y parvenir est appelée One-Hot Encode.

```
# One hot Encode
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

Nous devons normaliser les entrées de 0-255 à 0-1 afin de modifier les valeurs des colonnes numériques dans l'ensemble de données à une échelle commune, sans fausser les différences dans les plages de valeurs. Cela implique d'abord de convertir le type de données des entiers non signés en nombres flottants, puis de diviser les valeurs des pixels par la valeur maximale.

```
# convert from integers to floats
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')


# normalize to range [0,1]
X_train = X_train / 255.0
X_test = X_test / 255.0
```

3. Créer le Model :

Ensuite, nous devons définir un modèle de base de Convulotional neural network (CNN). Donc c'est quoi d'abord un CNN ?

```
# Create model
# Building CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform'))
model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu',
kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
```

4. Entrainement et évaluation du modèle :

La fonction model.fit() de Keras lancera l'entrainement du modèle. Elle prend les données d'entrainement, les données de validation, epochs et Batch size.
Une fois le modèle défini, nous devons l'évaluer. Nous évaluerons le modèle à l'aide du Cross Validation.

Avant d'aller plus loin, comprenons ce qu'est la validation croisée. Supposons que vous ayez n images de stylos et de crayons. Vous voulez entraîner un algorithme d'apprentissage profond pour qu'il puisse faire la différence entre les deux. L'idée qui sous-tend l'entrainement et le test de tout modèle de données est d'obtenir un taux d'apprentissage et une validation maximum. Un meilleur taux d'apprentissage et une meilleure validation peuvent être obtenus en augmentant les données d'entrainement et du test respectivement. Comme nos données sont limitées, il existe un point idéal où nous pouvons obtenir un taux d'apprentissage et une validation optimaux. Pour trouver ce point d'équilibre, nous utilisons la validation croisée qui divise l'ensemble de données en k sous-ensembles et recherche le meilleur rapport entre les données de test et les données d'entrainement
Dans notre cas, on choisit la valeur de k = 5. Ainsi, chaque ensemble de test représentera 20 % de l'ensemble de données d'entrainement, soit environ 12 000 exemples.

```
def evaluate_model(X_train, y_Train, n_folds=5):


    accuracy, data = list(), list()
    # prepare 5-cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)

    for x_train, x_test in kfold.split(X_train):
        # create model
        model = create_model()


        # select rows for train and test
        trainX, trainY, testX, testY = X_train[x_train],
y_Train[x_train], X_train[x_test], y_Train[x_test]


        # fit model
        data_fit = model.fit(trainX, trainY, validation_data=(testX,
testY), epochs=10, batch_size=32)


        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)


        # stores Accuracy
        accuracy.append(acc)
        data.append(data_fit)
    return accuracy, data
```

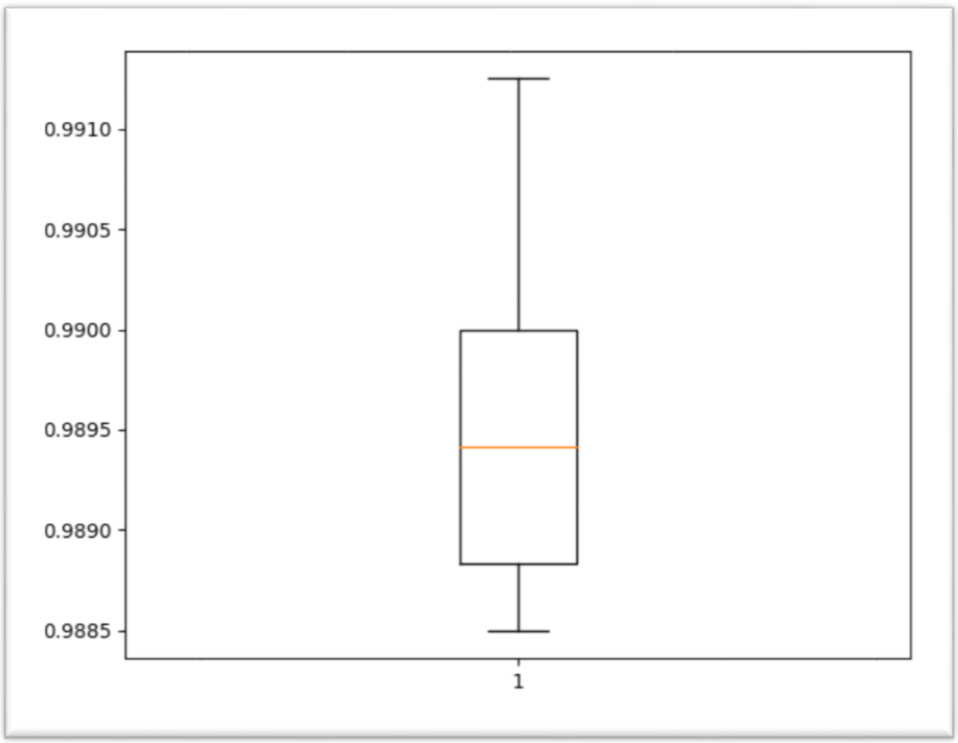
On peut apporter des modifications à notre modèle jusqu'à ce que nous soyons satisfait de son évaluation. On peut voir une représentation visuelle des précisions obtenues lors de l'évaluation à l'aide du MatPlotLib.pyplot.

```
# summarize model performance
def summarize_performance(acc):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (numpy.mean(acc) *
100, numpy.std(acc) * 100, len(acc)))

    # box and whisker plots of results
    pyplot.boxplot(acc)
    pyplot.show()
```

Résultat :

```
Accuracy: mean=98.960 std=0.097, n=5
```



Note : Vous pouvez itérer par la formation et l'évaluation de votre modèle à l'aide de l'étape 4 ou utiliser directement cette étape. La fonction model.fit() de Keras entraîne le modèle dont les données d'entraînement, les données de validation, Epochs et Batch_size sont les paramètres. Dans ce cas, Epochs = 10 alors que Batch_size = 200. Nous allons compiler notre modèle en utilisant Adam (un algorithme d'optimisation du taux d'apprentissage adaptatif)

```
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, batch_size=200)
```

```
300/300 [==============================] - 32s 107ms/step - loss: 0.0089 -
  accuracy: 0.9973 - val_loss: 0.0345 - val_accuracy: 0.9906
Large CNN Error: 0.94%
```

5. Sauvegarder le Model :

Si On est satisfait de la performance de notre model et de sa précision on peut le sauvegarder En utilisant *model.save("Nom_Modele.h5")* .

```
# serialize model to JSON and save the model
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights("final_model.h5")
```

6. Tester le Modèle :

Maintenant on va tester notre model sur une Image dans laquelle est inscrit le nombre 2.

Nous devons redimensionner et remodeler l'image pour (1, 28, 28, 1). (Note : l'image doit être en Grayscale) Nous devons charger le modèle sauvegardé en utilisant load_model.En utilisant l'image de test on prédit qu'elle contient le nombre 2.

```
def predict(img):
    image = img.copy()
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # image = cv2.threshold(image, 140, 255, cv2.THRESH_BINARY)[1]
    image = cv2.resize(image, (28, 28))
    # display_image(image)
    image = image.astype('float32')
    image = image.reshape(1, 28, 28, 1)
    image /= 255

    # plt.imshow(image.reshape(28, 28), cmap='Greys')
    # plt.show()
    model = load_model('cnn.hdf5')
    pred = model.predict(image.reshape(1, 28, 28, 1), batch_size=1)

    print("Predicted Number: ", pred.argmax())

    # return pred.argmax()
```

```
Predicted Number:  2
```

Conclusion :

En guise de conclusion, on a pu utiliser avec succès les techniques de deep learning pour aboutir a notre but qui est la reconnaissance des nombre manuscrits à partir du réseau neuronal convolutif et les bibliothèques de recherche scientifique et Data science : Keras , Numpy , Tenserflow …ect .

On a tout d'abord préparer les données d'entrée au préalable avant de les présenter au modèle. Ensuite, on a passé à la création du modèle et début d'entrainement après la division du Dataset en 2 parties , une partie pour l'entrainement et l'autre pour le test. On calcule, d'après les données de test, la précision de notre modèle. Et finalement , on utilise quelque exemples d'images de test pour vérifier la prédiction de notre modèle .

Ressources et Webographie :

- ✓ https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac
- ✓ https://www.tensorflow.org/resources/learn-ml
- ✓ https://en.wikipedia.org/wiki/Convolutional_neural_network