Handwritten Recognition Digits

I. Introduction

La reconnaissance d'écriture manuscrite est un domaine de recherche en croissance dans le domaine de l'intelligence artificielle et de l'apprentissage automatique. Il vise à développer des systèmes informatiques capables de reconnaître et de convertir des écritures manuscrites en caractères numérisés. Cette technologie a de nombreuses applications pratiques, notamment dans les systèmes de saisie de données, les systèmes de reconnaissance de caractères, les systèmes de reconnaissance de formulaires, les systèmes de reconnaissance de signatures électroniques, et bien d'autres. Il existe différentes techniques pour la reconnaissance d'écriture manuscrite, notamment l'analyse de formes, l'analyse de contours, l'analyse de mouvements, l'analyse de caractéristiques, et l'analyse de caractères. Chacune de ces techniques a ses propres avantages et inconvénients, et il est souvent nécessaire de combiner plusieurs techniques pour obtenir les meilleures performances.

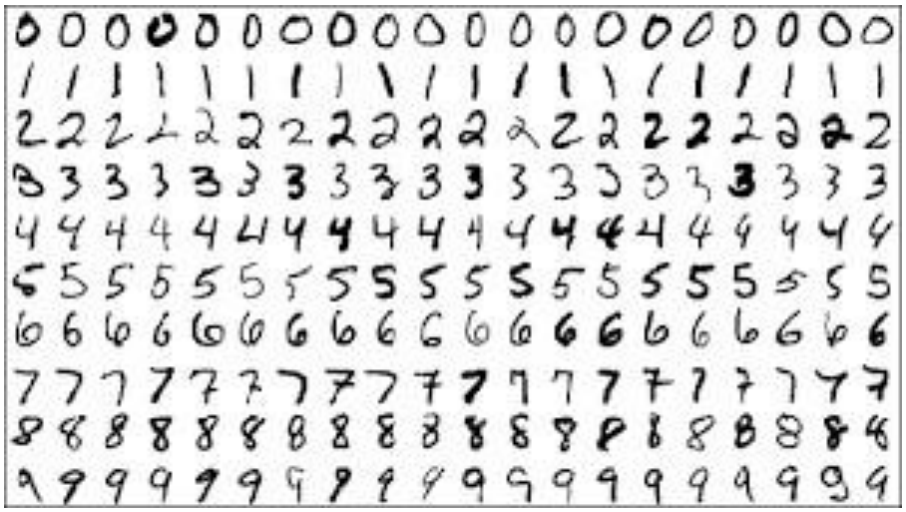
II. Objectifs :

Dans ce projet, nous allons mettre en place un réseau de neurones à convolution pour classer les images. Pour commencer, nous allons entraîner notre modèle avec la base de données MNIST qui contient des images manuscrites de chiffres. Une fois l'entraînement terminé, nous allons tester notre modèle en lui soumettant des images de chiffres qu'il n'a pas vues lors de l'apprentissage. Le but est que le modèle soit capable de reconnaître et de produire en sortie le chiffre présent sur l'image testée

III. La base de données MNIST :

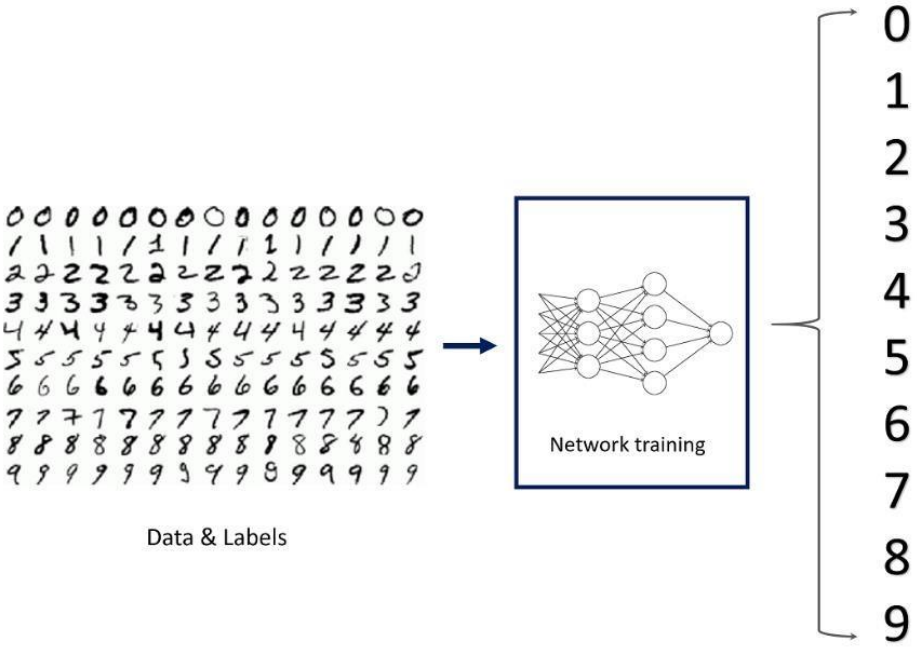
MNIST (Modified National Institute of Standards and Technology) est une base de données populaire dans le domaine de l'apprentissage automatique. Il contient des images de chiffres manuscrits, avec 60 000 images d'entraînement et 10 000 images de test. Les images ont une résolution de 28x28 pixels et sont en niveau de gris. Cette base de données est largement utilisée comme ensemble de données de test pour

les systèmes de reconnaissance de caractères manuscrits, car elle est facilement accessible, de petite taille et avec une bonne qualité d'image. Elle est considérée comme un bon point de départ pour les débutants dans le domaine de la reconnaissance d'images. Les données MNIST sont largement utilisées pour l'apprentissage et l'évaluation des différents modèles de reconnaissance de caractères manuscrits, de sorte qu'il est facile de comparer les résultats avec d'autres travaux de recherche. C'est également une base de données populaire pour les développeurs pour tester les performances de leurs modèles avant de les utiliser sur des données réelles.



IV. Model utilisé :

Le



modèle utilisé est un réseau de neurones convolutif (Convolutional Neural Network : CNN) pouvant être représenté comme ceci :

V. Réalisation :

- **Outils utilisés**

- **COLAB :**

Colab (ou Google Colaboratory) est un outil en ligne gratuit développé par Google qui permet de créer et d'exécuter des notebooks Jupyter en ligne. Il est conçu pour faciliter la collaboration et la recherche en intelligence artificielle et en apprentissage automatique. Il offre un environnement de développement intégré (IDE) complet pour écrire, exécuter et partager du code, et il est équipé de nombreux outils populaires pour le développement de modèles d'IA, tels que TensorFlow, Keras, PyTorch, etc. Il permet également de charger et de sauvegarder des fichiers à partir de Google Drive, ainsi que d'accéder à des GPU et des TPU pour accélérer les calculs.

➢ **Framework et bibliothèques :**

- **TensorFlow :**

TensorFlow est un logiciel libre de bibliothèque de calcul en deuxième génération pour les systèmes d'apprentissage automatique développé par Google Brain Team. Il a été rendu public en novembre 2015. Il permet de définir, optimiser et exécuter des calculs à l'aide de graphes de calculs. Il est utilisé pour la reconnaissance de la parole, la reconnaissance d'image, les systèmes de recommandation et les systèmes de traduction automatique.

- **Keras :**

Keras est une bibliothèque de haut niveau open-source écrite en Python pour la création de réseaux de neurones. Il a été développé pour faciliter la création rapide et efficace de modèles d'apprentissage automatique. Il permet de construire des architectures de réseaux de neurones de manière simple et modulaire. Il se concentre sur l'expérimentation rapide, en permettant de rapidement prototyper des idées d'architecture de réseau de neurones.

- **OpenCV :**

OpenCV (pour Open Computer Vision) est une bibliothèque libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage, puis la société ItSeez se sont succédé au support de cette bibliothèque. Depuis 2016 et le rachat de ItSeez par Intel, le support est de nouveau assuré par Intel.

- **Tkinter :**

Tkinter (Tk interface) est un module intégré à la bibliothèque standard de Python, permettant de créer des **interfaces graphiques** :

- des fenêtres,
- des widgets (boutons, zones de texte, cases à cocher, ...),
- des évènements (clavier, souris, ...).

VI. Application :

1. Importer les librairies :

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.datasets import mnist
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.utils import plot_model
from importlib.resources import path
import numpy as np
import matplotlib.pyplot as plt
```

2. Charger la base de données MNIST :

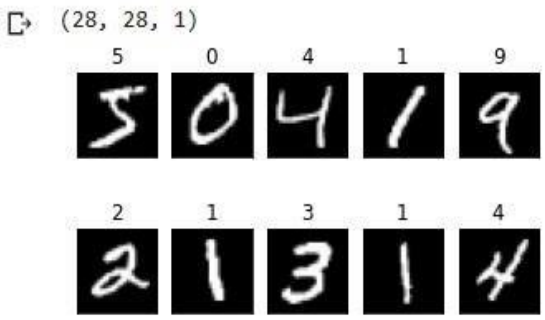
```
[ ]  (X_train, Y_train), (x_test, y_test) = mnist.load_data()
```

3. Prétraitement et à la visualisation des données d'images

```
[ ]  X_train, x_test = X_train/255.0, x_test/255.0

     X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
     x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

     fig = plt.figure(figsize=(5, 3))
     for i in range(10):
         ax = fig.add_subplot(2, 5, i+1, xticks=[], yticks=[])
         ax.imshow(np.squeeze(X_train[i]), cmap='gray')
         ax.set_title(Y_train[i])
     input_shape = X_train.shape[1:]
     print(input_shape)
```



Ce code effectue plusieurs tâches liées au prétraitement et à la visualisation des données d'images. Il utiliser la base de données MNIST.

Tout d'abord, il normalise les données X_train et x_test en divisant chaque élément par 255.0. Il s'agit d'une étape de prétraitement courante pour mettre à l'échelle les données de sorte que les valeurs de pixels se situent dans l'intervalle de 0 à 1

Ensuite, les données sont reshape pour être utilisées dans un réseau de neurones, X_train est reshape en (X_train.shape[0], 28, 28, 1) et x_test est reshape en (x_test.shape [0], 28, 28, 1).

Enfin, il y a une boucle for qui permet de visualiser les 10 premières images en utilisant matplotlib, avec la fonction imshow pour afficher l'image et la fonction set_title pour afficher le label de l'image. La variable input_shape est définie comme étant la forme des données d'entrée, qui est utilisée pour définir la forme de l'entrée du réseau de neurones.

4. Création du modèle :

```
[ ]  model = Sequential([
        Conv2D(32, (5,5), padding="Same", activation='relu', input_shape=(28, 28, 1)),
        Conv2D(32, (5,5), padding="Same", activation='relu'),
        MaxPool2D((2, 2)),
        Dropout(0.25),
        Conv2D(64, (3,3), padding='Same', activation='relu', input_shape=(28,28,1)),
        Conv2D(64, (3,3), padding='Same', activation='relu'),
        MaxPool2D((2,2),(2,2)),
        Dropout(0.25),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.25),
        Dense(10, activation='softmax')
    ])

    model.summary()
```

Les couches utilisées :

Ce code crée un modèle d'apprentissage profond en utilisant l'API Sequential de la bibliothèque Keras. Le modèle est composé de plusieurs couches, notamment des couches de convolution, des couches de pooling max, des couches de dropout et des couches entièrement connectées.

L'entrée du modèle est une image de forme (28, 28, 1) qui passe à travers la première couche de convolution (Conv2D) avec 32 filtres de taille (5, 5) et un remplissage défini sur "Same". Cette couche applique 32 filtres différents à l'image d'entrée, chaque filtre recherche une caractéristique spécifique dans l'image. La fonction d'activation utilisée est ReLU, qui est couramment utilisée dans les réseaux de neurones pour introduire une non-linéarité.

La prochaine couche est également une couche de convolution avec 32 filtres de taille (5, 5) et un remplissage défini sur "Same". Cette couche applique les mêmes filtres que la première couche à la sortie de la première couche, ce qui augmente la profondeur des cartes de caractéristiques.

Ensuite, il y a une couche MaxPool2D qui effectue une opération de pooling max sur la sortie de la couche précédente avec une taille de pooling de (2, 2). Cette opération réduit la taille des cartes de caractéristiques en conservant la valeur maximale de chaque fenêtre de taille (2, 2) et en jetant le reste.

Ensuite, il y a une couche Dropout avec un taux de 0,25, qui désactive aléatoirement 25% des neurones pendant l'entraînement. Il s'agit d'une technique de régularisation utilisée pour éviter le sur-apprentissage.

Les deux prochaines couches sont similaires aux deux premières, une couche de convolution avec 64 filtres de taille (3, 3) et un remplissage défini sur 'Same' et une fonction d'activation relu, suivie d'une couche MaxPool2D avec une taille de pooling de (2, 2) et un pas de (2, 2).

Ensuite, il y a une autre couche Dropout avec un taux de 0,25 pour éviter le surapprentissage.

Ensuite, il y a une couche Flatten qui aplatit la sortie des couches précédentes en un vecteur unique. Cela est nécessaire avant de passer les données aux couches entièrement connectées.

La prochaine couche est une couche Dense avec 256 neurones et une fonction d'activation de 'relu'. Il s'agit d'une couche entièrement connectée qui prend la sortie aplatie de la couche précédente et applique une fonction d'activation ReLU à la sortie de chaque neurone.

Ensuite, il y a une couche Dropout avec un taux de 0,25 pour éviter le surapprentissage.

Enfin, il y a une couche de sortie avec 10 neurones et une fonction d'activation de 'softmax', qui est utilisée pour les problèmes de classification multi-classes comme MNIST.

La méthode summary() montre un résumé de l'architecture du modèle, y compris le nombre de paramètres dans chaque couche et le nombre total de paramètres dans le modèle.

5. Optimisation et entrainement :

```
model.compile(
    optimizer=RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    print(logs)
    if(logs.get('accuracy') > 0.999):
      print("\nIt's accurate enough so let's stop!\n")
      self.model.stop_training = True

callbacks = myCallback()


learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(
  monitor='val_accuracy',
  patience=3,
  factor=0.5,
  min_lr=0.00001
  )

model.fit(
    X_train,
    Y_train,
    epochs=30,
    batch_size=112,
    callbacks=[callbacks, learning_rate_reduction],
    validation_data=(x_test, y_test)
)
```

Ce code configure et entraîne le modèle d'apprentissage profond qui a été créé précédemment en utilisant la méthode fit() de la bibliothèque Keras.

La première étape consiste à compiler le modèle en utilisant la méthode compile(). Cette méthode configure le modèle pour l'entraînement en définissant l'optimiseur, la fonction de perte et les métriques d'évaluation.

Dans ce cas, l'optimiseur utilisé est RMSprop (Root Mean Square Propagation) avec un taux d'apprentissage de 0,001, un taux de décroissance de 0,9, une valeur epsilon de 1e-08 et aucune décroissance. La fonction de perte utilisée est 'sparse_categorical_crossentropy' qui est utilisée pour les problèmes de classification multi-classes comme MNIST et la métrique d'évaluation utilisée est 'accuracy'.

Ensuite, il crée une fonction de rappel, myCallback, qui imprimera les journaux à la fin de chaque époque et arrêtera l'entraînement si la précision est supérieure à 0,999.

Ensuite, il ajoute une autre fonction de rappel, learning_rate_reduction, qui est utilisée pour réduire le taux d'apprentissage lorsque la précision de validation est plate, cela est fait en surveillant la 'val_accuracy' et en réduisant le taux d'apprentissage d'un facteur de 0,5 après 3 époques sans amélioration.

Enfin, le modèle est adapté aux données d'entraînement (X_train et Y_train) en utilisant la méthode fit(). Le nombre d'époques est défini sur 30, la taille du lot est définie sur 112 et les données de validation sont définies sur (x_test et y_test). Les rappels utilisés sont myCallback et learning_rate_reduction.

VII. Résultats :