Intelligence artificielle

DEEP LEARNING

Construire un réseau neuronal pour reconnaître Chiffres manuscrits avec TensorFlow

realized by
Mohamed ait aamer

Objectif de l’application

Dans cette presentation, je vais mettre en œuvre une petite sous-section de la reconnaissance d'objets : la reconnaissance des chiffres. En utilisant TensorFlow, une bibliothèque Python open source développée par les laboratoires Google Brain pour la recherche en apprentissage profond, je prends des images dessinées à la main des chiffres 0 à 9, puis je construis et entraîne un réseau neuronal pour reconnaître et prédire l'étiquette correcte du chiffre affiché.

Comment construire cette application ?

Etape 1 -- Configuration du projet

Etape 2 -- Chargement de DATA MNIST

Etape 3 -- Prétraitement des données

Etape 4 -- Creation du modele du réseau neuronal

Etape 5 -- Compilation, Entrainement du modele

Etape 6 -- Test de prediction sur image

Etape 1 -- Configuration du projet

Avant de pouvoir développer le programme de reconnaissance, vous devrez installer quelques dépendances et créer un espace de travail pour stocker vos fichiers. J'utilise un environnement virtuel Python 3 pour gérer les dépendances de mon projet. Créez un nouveau répertoire pour votre projet et naviguez jusqu'au nouveau répertoire :

```
mkdir tensorflow-demo
cd tensorflow-demo                  #creer et acceder au dessier tensorflow-demo

python3 -m venv tensorflow-demo
source tensorflow-demo/bin/activate     #creer et activer l'environnement virtuel

touch requirements.txt       #creer le fichier contient les modules python necessaires
pip install -r requirements.txt       #installer les modules
```

Etape 2 -- Chargement de DATA MNIST

1- Creation du fichier.py et importer les modules necessaires pour construire l'application

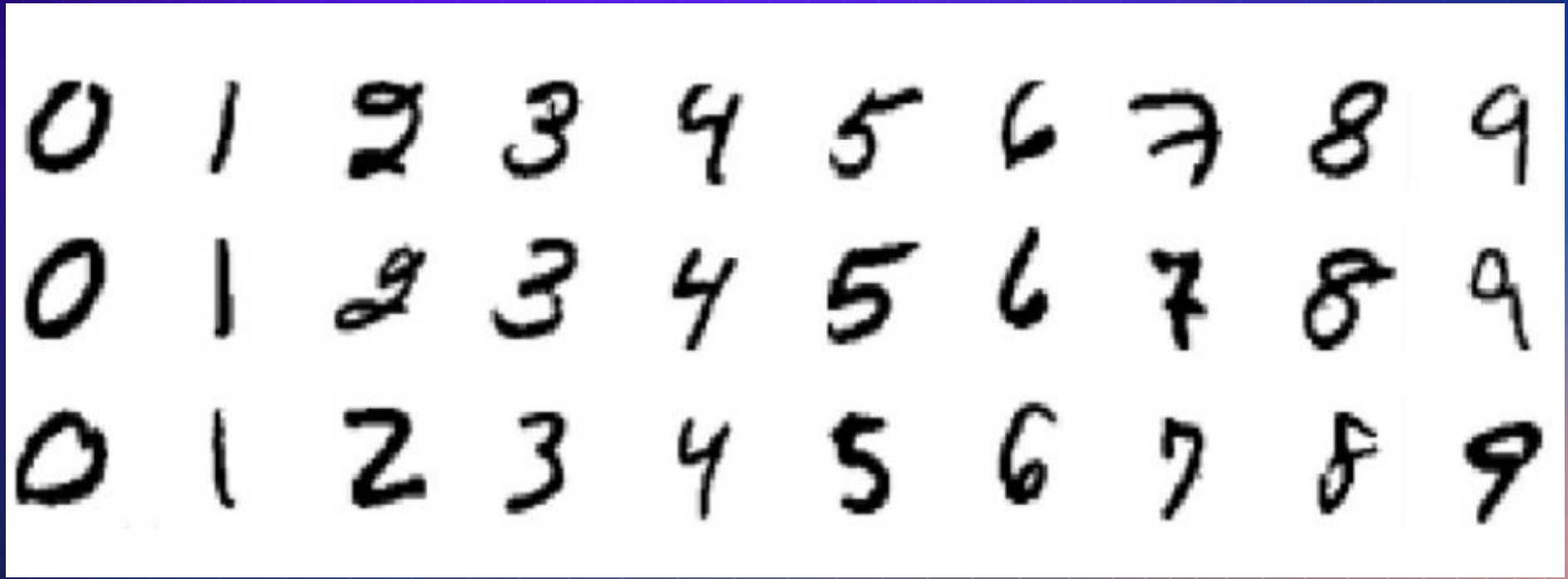
```
import tensorflow as tf
from tensorflow.keras import      datasets, layers, models
from tensorflow.keras.preprocessing.image import     load_img, img_to_array
import numpy as np
import    cv2
import    numpy as np
```

2- Chargement de data MNIST

```
mnist = datasets.mnist.load_data()
(x_train, y_train), (x_test, y_test) = mnist
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Les données MNIST sont téléchargées et divisées en ensembles d'entraînement et de test. Les images sont normalisées pour avoir des valeurs de pixels entre 0 et 1.

exemple des images MNIST



Etape 3 -- Prétraitement des données

```
x_train, x_test = x_train.reshape(-1, 784),  x_test.reshape(-1, 784)

n_input = 784                n_hidden1 = 512
n_hidden2 = 256              n_hidden3 = 128
n_hidden4 = 64               n_hidden5 = 32
n_hidden6 = 16               n_output = 10

learning_rate = 1e-4         n_iterations = 100
batch_size = 128             dropout = 0
```

Les images sont aplaties et remodelées pour être compatibles avec le modèle. Les paramètres du réseau sont définis, tels que le nombre de couches cachées, les tailles des couches, le taux d'apprentissage, etc.

Etape 4 -- Creation du modele du réseau neuronal

```
model = models.Sequential()
model.add(layers.Dense(n_hidden1, activation='relu', input_shape=(n_input,)))
model.add(layers.Dense(n_hidden2, activation='relu'))
model.add(layers.Dense(n_hidden3, activation='relu'))
model.add(layers.Dense(n_hidden4, activation='relu'))
model.add(layers.Dense(n_hidden5, activation='relu'))
model.add(layers.Dense(n_hidden6, activation='relu'))
model.add(layers.Dropout(dropout))
model.add(layers.Dense(n_output, activation='softmax'))
```

Le modèle séquentiel est créé en ajoutant des couches **dense** avec des activations **ReLU**, une couche de **dropout**, et une couche de sortie avec une activation **softmax**.

ReLU: remplaçant toutes les valeurs négatives par zéro.
Softmax: convertit les scores bruts (logits) de la couche de sortie en probabilités.
Dense: lier les poids de neuron de chaque couche avec les couches voisins.

Etape 5 -- Compilation, Entrainement du modele

```
model.compile(optimizer=optimizer, loss=loss_fn, metrics=['accuracy'])

model.fit(x_train, y_train, epochs=n_iterations, batch_size = batch_size)
```

Le modèle est entraîné sur les données d'entraînement avec le nombre d'époques spécifié et une taille de lot (batch size).

Optimizer: minimiser la fonction de perte sur les poids lors d'entrainement
Loss: mesure la différence entre les prédictions du modèle et les valeurs réelles.
Metrics: évaluer la performance du modèle pendant et après l'entraînement.
Batch_size: le nombre d'échantillons de data d'entraînement utilisés dans une seule itération

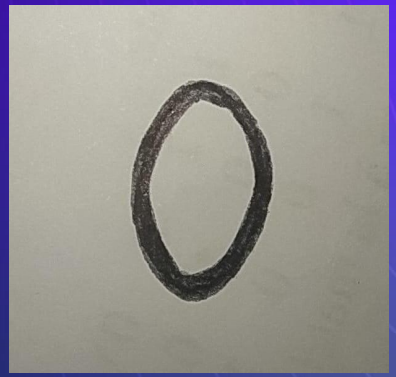
Etape 6 -- Test de prediction sur image

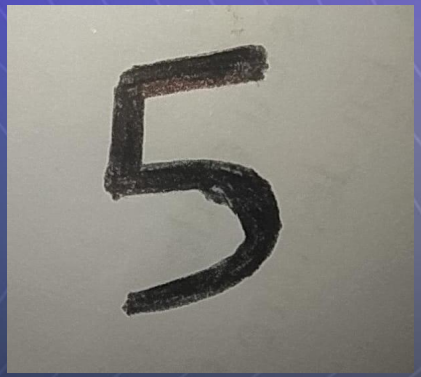
```
img_path = "/home/med/Desktop/MASTER_STRI/S3/IA/tensorflow-
demo/numeros/image.jpeg"
img = load_img(img_path, target_size=(28, 28), color_mode="grayscale")
img_array = img_to_array(img)
img_array1 = img_array.reshape(1, 784)
img_array1 /= 255.0
predictions = model.predict(img_array1)
predicted_digit = np.argmax(predictions[0])
print(predictions)
print('Predicted digit:', predicted_digit)

cv2.imshow('Image originale', img_array)
cv2.waitKey(0)
```

Dans cette partie du code, je prend une image d'un numero 0 a 9 et je la redimensionne en taille de 28x28 pixels, reformer l'image en suite de 1x784 pixels et faire la prediction.

Etape 6 -- Test de prediction sur image

Exemples des images utilisées dans le test de prédiction





Etape 6 -- Test de prediction sur image

```
45  # Define loss and optimizer
46  loss_fn = tf.keras.losses.S                                    se)
47  optimizer = tf.keras.optim
48
49  # Compile the model
50  model.compile(optimizer=opt
51
52  # Train the model
53  model.fit(x_train, y_train,                                    ze)
54
55  img_path = "/home/med/Desktop/MASTER_STRI/S3/IA/tensorflow-demo/numeros/deux.jpeg"
56  img = load_img(img_path, target_size=(28, 28), color_mode="grayscale")
57  img_array = img_to_array(img)
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
469/469 [==============================] - 6s 13ms/step - loss: 0.0382 - accuracy: 0.9889
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0362 - accuracy: 0.9897
1/1 [==============================] - 0s 105ms/step
[[2.2416137e-16 3.6470225e-09 9.9829143e-01 1.4616802e-03 1.6820600e-21
  1.9122651e-21 4.0556321e-07 2.4624908e-04 2.5241309e-07 5.5239905e-17]]
Predicted digit: 2
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in "/home/med/Desktop/MASTER_STRI/S3/IA/tensorflow-de
ages/cv2/qt/plugins"
```

Etape 6 -- Test de prediction sur image

```
45  # Define loss and optimizer
46  loss_fn = tf                                              from_logits=False)
47  optimizer =
48
49  # Compile th
50  model.compil                                              =['accuracy'])
51
52  # Train the
53  model.fit(x_                                              ize = batch_size)
54
55  img_path = "/home/med/Desktop/MASTER_STRI/S3/IA/tensorflow-demo/numeros/3.jpeg"
56  img = load_img(img_path, target_size=(28, 28), color_mode="grayscale")
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
469/469 [==============================] - 5s 12ms/step - loss: 0.0214 - accuracy: 0.9941
1/1 [==============================] - 0s 110ms/step
[[5.1624691e-05 1.3292996e-14 1.3434118e-08 9.9993992e-01 8.6995320e-24
  1.2417438e-09 7.4299756e-19 3.8291144e-22 8.4287685e-06 1.0532237e-12]]
Predicted digit: 3
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in "/home/med/Desktop/MASTER_STRI/S
ages/cv2/qt/plugins"
```

Etape 6 -- Test de prediction sur image

```
46  loss_fn = tf.kera                                    logits=False)
47  optimizer = tf.ke
48
49  # Compile the mod
50  model.compile(opt                                    curacy'])
51
52  # Train the model
53  model.fit(x_train                                    batch_size)
54
55  img_path = "/home/med/Desktop/MASTER_STRI/S3/IA/tensorflow-demo/numeros/5.jpeg"
56  img = load_img(img_path, target_size=(28, 28), color_mode="grayscale")
57  img_array = img_to_array(img)
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
1/1 [==============================] - 0s 108ms/step
[[7.67653519e-13 9.25074080e-12 7.90438557e-04 9.42702964e-02
  1.57266993e-16 8.93391192e-01 1.10221645e-02 4.93341940e-04
  3.25664041e-05 2.71913812e-13]]
Predicted digit: 5
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in "/home/med/Desktop/MASTER_STRI/S
n3.10/site-packages/cv2/qt/plugins"
```

Les probleme rencontrés lors de la réalisation de cette application sont:

- Prédiction n'est pas precises >> Utiliser la fonction Softmax.

- La forme des images utilisées n'est pas compatible >> Utiliser les images carrées.

- Erreur de predire le numero exacte >> augmenter le nombre d'itérations , Ajouter des couches cachées.

Conclusion

En conclusion, la construction d'un réseau neuronal pour la reconnaissance de chiffres manuscrits à l'aide de TensorFlow offre une solution puissante et efficace pour résoudre ce problème complexe. Ce modèle, basé sur des techniques avancées telles que les réseaux convolutionnels, a démontré une capacité notable à généraliser à partir du jeu de données MNIST. La précision obtenue lors de l'évaluation sur le jeu de test souligne l'efficacité de l'approche adoptée.