PROJET DIGIT RECOGNITION

GROUPE

Mohamed MAHMOUD

Nicolas DESFORGES

Présentation

SOMMAIRE

1 - Exploration des données

2 - Intégration de MongoDB

3 - Entraînement et sérialisation du modèle

4 - API

5 - Front-end

OBJECTIF

Développer une WebApp qui permet de détecter les chiffres manuscrits dessiné par un utilisateur

Data : MNIST

Apprentissage supervisé

Classification

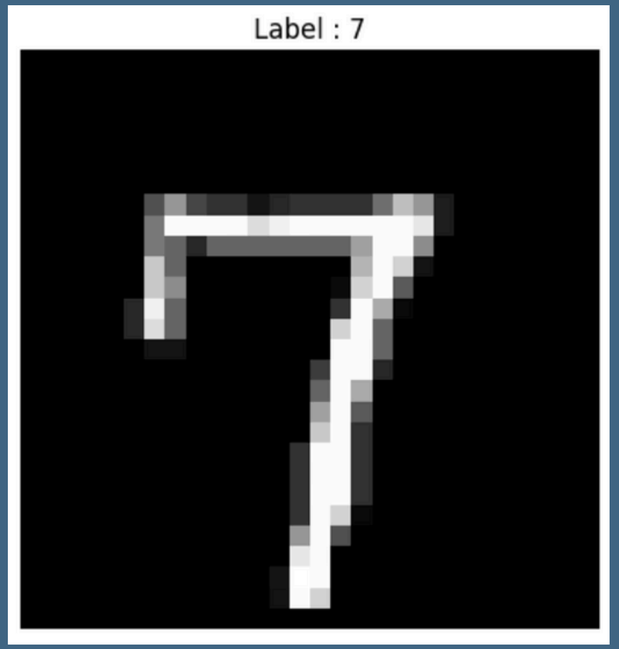
1 - Exploration des données

Présentation des données

Nous avons 3 fichiers
(test, train, et sample_submission)

Les images sont en pixels allant de 0 à 255

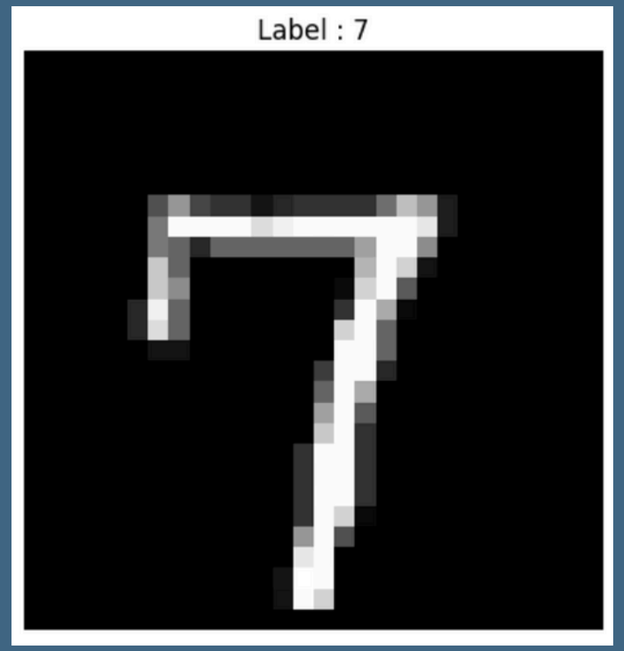
C’est du supervisé donc avec des labels



Afficher une image

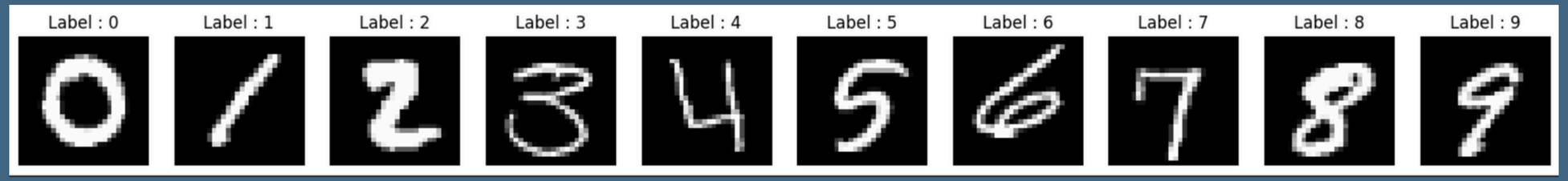
```
df_7 = df[df['label'] == 7]
image_data = df_7.iloc[0, 1:].values.reshape(28, 28)

plt.imshow(image_data, cmap='gray')
plt.title("Label : 7")
plt.axis('off')
plt.show()
```

```
labels = df['label']
pixels = df.drop('label', axis=1)

random_index = np.random.randint(0, len(df))

image_array = np.array(pixels.iloc[random_index]).reshape(28, 28)
plt.imshow(image_array, cmap='gray')
plt.title(f"Label : {labels.iloc[random_index]}")
plt.axis('off')
plt.show()
```



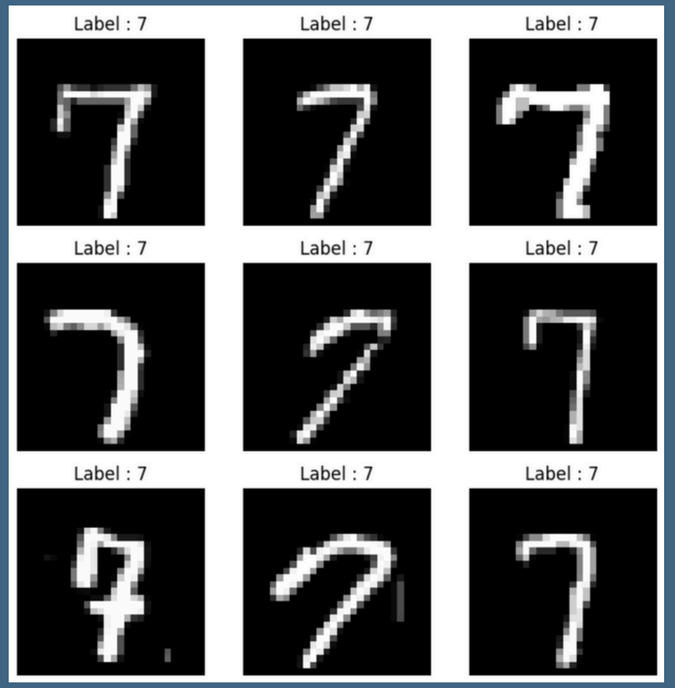
Afficher dans une même figure les chiffres de 0 à 9

```
digits_data = []

for digit in range(10):
    df_digit = df[df['label'] == digit]
    image_data = df_digit.iloc[0, 1:].values.reshape(28, 28)
    digits_data.append(image_data)

num_digits = len(digits_data)
fig, axes = plt.subplots(1, num_digits, figsize=(num_digits * 2, 2))

for i in range(num_digits):
    axes[i].imshow(digits_data[i], cmap='gray')
    axes[i].set_title(f"Label : {i}")
    axes[i].axis('off')

plt.show()
```



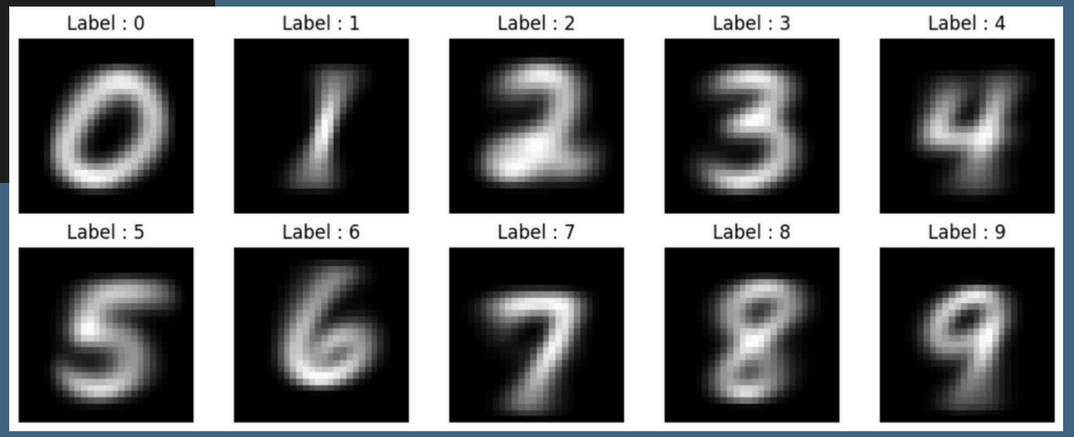
Afficher les 9 1ères images qui correspondent au chiffre 7

```
img_7 = df_7.iloc[:9, 1:].values.reshape(-1, 28, 28)

fig, axes = plt.subplots(3, 3, figsize=(8, 8))
for i, ax in enumerate(axes.flat):
    ax.imshow(img_7[i], cmap='gray')
    ax.set_title("Label : 7")
    ax.axis('off')
plt.show()
```



Afficher le représentant "moyen" de chaque chiffre

```python
mean_df = df.groupby('label').mean()

fig, axes = plt.subplots(2, 5, figsize=(10, 4))

for digit, ax in enumerate(axes.flat):
    ax.imshow(mean_df.loc[digit].values.reshape(28, 28), cmap='gray')
    ax.set_title(f"Label : {digit}")
    ax.axis('off')

plt.tight_layout()
plt.show()
```



2 - Intégration de MongoDB

Connexion à la base de données

```
!pip install pymongo[srv]

import pymongo

uri = "mongodb+srv://root:0139220337mM?@digit-recognition.splfod5.mongodb.net/?retryWrites=true&w=majority&appName=digit-recognition"

client = pymongo.MongoClient(uri)

db = client['digit-recognition']


try:
  client = pymongo.MongoClient(uri)
  db = client['digit-recognition']
  print("Connecté à MongoDB Atlas")
except Exception as e:
  print("Erreur lors de la connexion à MongoDB:", e)
```

```
Requirement already satisfied: pymongo[srv] in /usr/local/lib/python3.10/dist-packages (4.7.2)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from pymongo[srv]) (2.6.1)
Connecté à MongoDB Atlas
```

3 - Entraînement et sérialisation du modèle

```
train_data = pd.read_csv("/content/drive/MyDrive/DATA/digit-recognizer/train.csv")

X_train = train_data.drop("label", axis=1).values
y_train = train_data["label"].values

X_test = pd.read_csv("/content/drive/MyDrive/DATA/digit-recognizer/test.csv").values
```

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Définir le modèle

```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Compiler le modèle

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

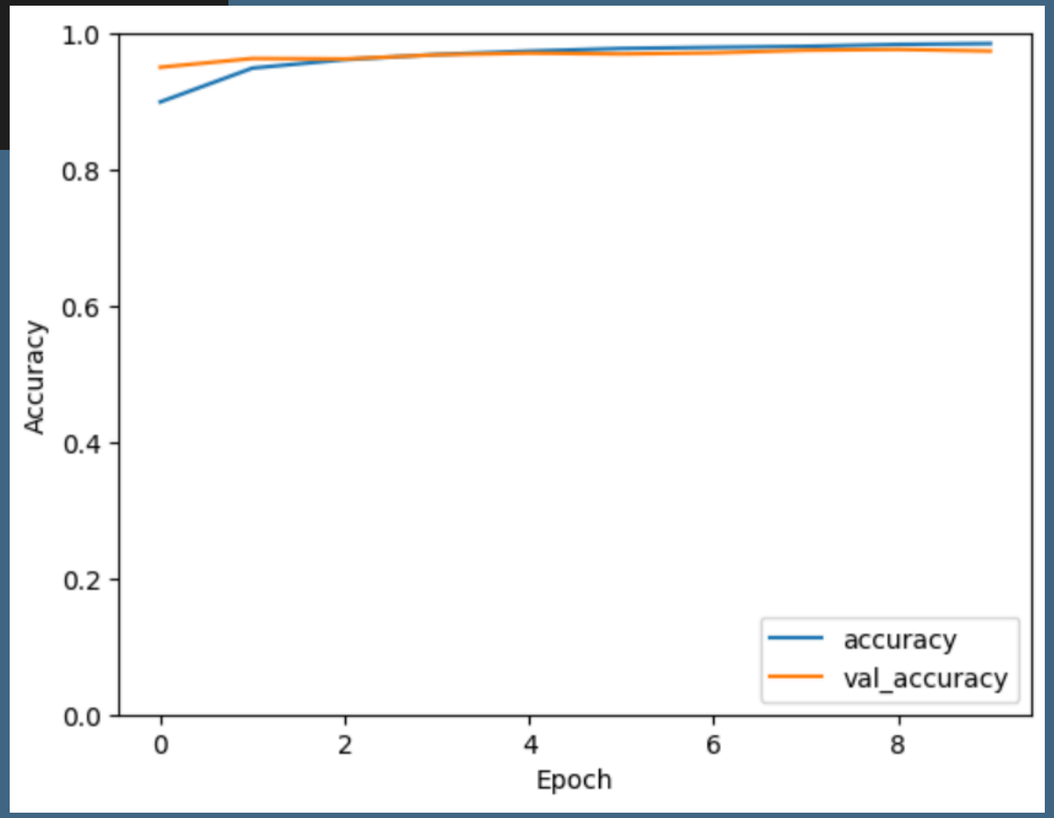
Entraîner le modèle

```
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)
```

```
Epoch 1/10
1182/1182 [==============================] - 8s 6ms/step - loss: 0.3525 - accuracy: 0.8977 - val_loss: 0.1881 - val_accuracy: 0.9436
Epoch 2/10
1182/1182 [==============================] - 7s 6ms/step - loss: 0.1763 - accuracy: 0.9475 - val_loss: 0.1346 - val_accuracy: 0.9593
Epoch 3/10
1182/1182 [==============================] - 5s 4ms/step - loss: 0.1281 - accuracy: 0.9610 - val_loss: 0.1163 - val_accuracy: 0.9631
Epoch 4/10
1182/1182 [==============================] - 7s 6ms/step - loss: 0.1021 - accuracy: 0.9687 - val_loss: 0.1055 - val_accuracy: 0.9676
Epoch 5/10
1182/1182 [==============================] - 5s 5ms/step - loss: 0.0866 - accuracy: 0.9724 - val_loss: 0.0935 - val_accuracy: 0.9712
Epoch 6/10
1182/1182 [==============================] - 6s 5ms/step - loss: 0.0730 - accuracy: 0.9772 - val_loss: 0.0911 - val_accuracy: 0.9726
Epoch 7/10
1182/1182 [==============================] - 8s 7ms/step - loss: 0.0644 - accuracy: 0.9794 - val_loss: 0.0920 - val_accuracy: 0.9736
Epoch 8/10
1182/1182 [==============================] - 5s 5ms/step - loss: 0.0556 - accuracy: 0.9823 - val_loss: 0.0853 - val_accuracy: 0.9736
Epoch 9/10
1182/1182 [==============================] - 7s 6ms/step - loss: 0.0503 - accuracy: 0.9840 - val_loss: 0.0891 - val_accuracy: 0.9736
Epoch 10/10
1182/1182 [==============================] - 7s 6ms/step - loss: 0.0458 - accuracy: 0.9852 - val_loss: 0.0942 - val_accuracy: 0.9736
```

Afficher la courbe d'apprentissage

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



4 - API

Enregistrer le modèle sous format h5

```
model.save("tensorflow_model.h5")
```

Création du server.py

```python
import tensorflow as tf
import numpy as np

from flask import Flask, request, jsonify
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# Chargement du modèle
new_model = tf.keras.models.load_model('tensorflow_model.h5')

new_model.summary()


# Route
@app.route('/')
def index():
    return 'Serveur Flask en cours d\'exécution'

@app.route('/predict', methods=['POST'])
def predict():
    image_data = request.json['image']

    # Prédiction
    prediction = new_model.predict(image_data)

    # Convertion en JSON
    predicted_class = np.argmax(prediction, axis=1)[0]
    result = {'predicted_class': int(predicted_class)}

    # Renvoyer
    return jsonify(result)

if __name__ == '__main__':
    app.run(debug=True)
```

5 - Front-end

PROJET IPSSI DIGIT

Cliquez pour la prediction

Merci de votre écoute