École Nationale Supérieure
d'Informatique et d'Analyse des Systèmes

Université Mohammed V RABAT

Projet Intelligence Artificielle

Reconnaissance des chiffres manuscrits

17 fev 2021

Réalisé par :
Moncef BENDER, moncefbender@gmail.com
e-MBI 12

Anass BENHIMA, anassbenhima@gmail.com
e-MBI 13

Encadré par :
MME Houda BENBRAHIM

Année académique 2020-2021

This work is dedicated to ensias' teachers especially our dear Benhiba, a big thanks for your efforts and being available all the time.

Résumé

Ce projet consiste à réaliser un système de reconnaissance des chiffres manuscrits. Le système est développé en utilisant un réseau de neurones (multilayer perceptron backpropagation) entraîné avec la base de données MNIST.
Un modèle d'extraction d'attributs LLF (Local Line Fitting) est utilisé. cette méthode, basée sur des opérations géométriques simple, est très efficient, permet une représentation invariante de distorsion de relativement faible dimension et une grande vitesse de reconnaissance.

Abstract

This project's aim is to realize a system to recognize handwritten digits. The system is developed using a multilayer perceptron backpropagation neural network trained by the MNIST dataset.
An effective feature extraction called LLF (Local Line Fitting) is used. This method, based on simple geometric operations, is very efficient, yields a relatively low-dimensional, distortion invariant representation and high recognition speed.

Table des matières

1 Introduction

À l'ère actuelle de la digitalisation, la reconnaissance de l'écriture manuscrite joue un rôle important dans le traitement de l'information. D'ailleurs, de nombreuses informations / données sont disponibles sur papier, alors que le traitement des fichiers numériques est moins cher que le traitement des dossiers papier traditionnels.
Le but d'un système de reconnaissance de l'écriture manuscrite est de convertir les caractères manuscrits dans des formats lisibles par machine. Les principales applications sont, par exemple, la reconnaissance des plaques d'immatriculation des véhicules, le tri des lettres par code postal, etc. Tous ces domaines traitent de grandes bases de données et exigent donc une précision de reconnaissance élevée, et une complexité de calcul bien réduite.

Ce projet consiste à réaliser un système de reconnaissance des chiffres manuscrits avec une précision de reconnaissance élevée et une bonne rapidité.

Le présent rapport retrace l'objectif atteint à travers les différentes étapes que nous avons suivies. Ces étapes peuvent être résumées en trois phases :

— **Phase feature extraction :** Cette phase se base notamment sur la méthode LLF qui se compose de deux étapes : la fragmentation de la figure d'abord et puis l'extraction des attributs recherchés.

— **Phase d'apprentissage :** L'apprentissage est fait par un réseau de neurones (multilayer perceptron backpropagation) entraîné avec la base de données MNIST.

— **Phase du test :** Le test est fait sur deux étapes d'abord en utilisant la partie test du MNIST, et puis une interface développée lors du projet pour l'utilisation du modèle.

2 Phase feature extraction

L'objectif du feature extraction est de trouver les paramètres (features) qui définissent la figure présentée avec une bonne *précision, unicité et continuité.* Autres facteurs importants sont *la rapidité* de la reconnaissance de la figure et *l'invariance* de la représentation aux distorsions qui ne changent pas la nature de l'objet représenté. Malheureusement, cette propriété hautement souhaitable est difficile à obtenir par tout type de moyen analytique ou théorique.

D'où le recours à la méthode LLF (Local Line Fitting) qui se base sur des opérations géométriques simples qui permettent de répondre aux exigences citées. Comme cette technique ne demande pas un pré-traitement de la figure. En fait, la figure est utilisée directement et par conséquent une vitesse de reconnaissance élevée peut être obtenue. La complexité et la rapidité du modèle sont détaillées dans la section 4 ; phase du test.

2.1 Fragmentation de la figure

D'abord la figure doit être divisée en k cellules, les paramètres (features) sont définis sur chaque cellules.
D'après les expériences de la validité de la technique LLF, une division en $4 * 4$ permet d'obtenir des bons résultats, d'où $K = 16$ [1].

La fragmentation de la figure est fait en Numpy.

```
# Split figure into 16 cells
cells = [] #16

x = np.vsplit(figure, 4)
for i in range(len(x)):
cells.extend(np.hsplit(x[i],4))
```

2.2 Extraction des attributs

Après la sélection des 16 cellules, nous définissons les attributs à extraire en chaque cellule. La méthode LLF propose l'extraction de trois attributs suffisants pour répondre aux exigences définies auparavant. [1]

Le premier attribut est la densité des pixels noirs dans la cellule.

$$f_{i1} = \frac{n_i}{N} \tag{1}$$

Le deuxième ainsi le troisième attributs sont définis en fonction de l'application de la régression linéaire. En effet, les pixels de la cellule sont ajustés à une ligne droite $y = a_i + b_i x$.

$$f_{i2} = \frac{2b_i}{1 + b^2} \tag{2}$$

$$f_{i3} = \frac{1 - b^2}{1 + b^2} \tag{3}$$

La regression linéaire est faite en sklearn.linear_model.

```
# Extract 3 features from each cell
feature_ex = [] #3*16

for i in range(16):
f1 = np.count_nonzero(cells_np[i] > 30)/49 #feature 1
indices_nonzero = np.nonzero(cells_np[i] > 30)
(y,x) = indices_nonzero
if x.shape==(0,):
  f2=0.
  f3=1.
else:
  x = x.reshape(-1, 1)
  y = y.reshape(-1, 1)
  reg = LinearRegression().fit(x, y)
  f2 = (2*float(reg.coef_))/(1+float(reg.coef_)**2) #feature 2
  f3 = (1-float(reg.coef_)**2)/(1+float(reg.coef_)**2) #feature 3

feature_ex.append(f1)
feature_ex.append(f2)
feature_ex.append(f3)
```

La fragmentation et l'extraction des attributs sont regroupés dans une fonction appelée feature_extraction(figure).
La représentation de chaque figure est alors un **vecteur de 16*3 = 48 elements**.

3 Phase d'apprentissage

La classification peut être faite par plusieurs méthodes, Nous avons choisi un réseau de neurones (Multilayer Perceptron backpropagation) pour faire la classification.

Les réseaux de neurones sont des structures construites à l'aide d'éléments de processus très simples. La topologie d'un Perceptron multicouche est basée sur un certain nombre de couches d'unités connectées les unes après les autres. Les échantillons de taille fixe sont présentés à la première couche (couche d'entrée). La dernière couche (couche de sortie) fournit un autre vecteur de taille fixe qui dépend de l'entrée et des poids des connexions entre les unités. Ces poids sont adaptés dans la phase d'apprentissage par des algorithmes qui comparent le résultat net avec la sortie souhaitée et modifient chaque poids de connexion pour minimiser l'écart. Dans notre cas, la loi de descente de gradient a été utilisée sur un perceptron à trois couches.

3.1 Base de données MNIST

MNIST est une base de données de chiffres écrits à la main de 60.000 figures représentées en 28*28 pixels.
Nous appliquerons notre fonction feature_extraction(figure) sur l'ensemble de données MNIST : chaque image sera définie alors par un vecteur de 16*3.

```
# Download train_data
train_data = datasets.MNIST('data', train=True, download=True)
```

```
3
4 # Create new train data
5 new_train_data = []
6
7 for i in range (len(train_data):
8   feature_np = np.array(train_data[i][0])
9   figure = feature_np.reshape(28,28)
10  feature_ex = feature_extraction(figure)
11  new_element = (feature_ex, train_data[i][1])
12  new_train_data.append(new_element)
```

On divise par la suite la nouvelle dataset d'apprentissage (new_train_data) en deux division aléatoire, la première est dédiée pour l'apprentissage et la deuxième pour la validation.

```
1 # Define Train validation loaders
2 train, val = random_split(new_train_data, [55000, 5000])
3 train_loader = DataLoader(train, batch_size=24)
4 val_loader = DataLoader(val, batch_size=24)
```

3.2 Structure du réseau de neurones

— La validation de la technique $4 * 4$ LLF propose la structure suivante : **48*50*50*10** où $48 = 16*3$ sont les attributs qui définissent la figure. [1]

— La fonction Relu est utilisée comme fonction d'activation avec un "dropout" pour éliminer le surapprentissage.

— Le calcul d'erreur et l'optimisation des poids sont faits selon la fonction ADAM et la CROSSENTROPY.

```
1 # Define the model
2 class ResNet(nn.Module):
3   def __init__(self):
4     super().__init__()
5     self.l1 = nn.Linear(16*3, 50)
6     self.l2 = nn.Linear(50, 50)
7     self.do = nn.Dropout(0.1)
8     self.l3 = nn.Linear(50, 10)
9   def forward(self, x):
10    h1 = nn.functional.relu(self.l1(x))
11    h2 = nn.functional.relu(self.l2(h1))
12    do = self.do(h2 + h1)
13    output = self.l3(do)
14    return output
15
16 model = ResNet()
17
18 # Define the optimzer
19 optim = optim.Adam(model.parameters(), lr=0.001)
20
21 # Define the loss
22 loss = nn.CrossEntropyLoss()
```

3.3 Apprentissage & Validation

La boucle d'apprentissage et validation est définie sur 12 epochs.
La boucle prend 40 secondes pour s'exécuter, avec un taux de validation allant

jusqu'au 91% dans la dernière epoch.

4 Phase du test

4.1 test sur MNIST

Pour confirmer les résultats obtenus lors de la validation, nous passons d'abord un test avec MNIST test dataset. Nous appliquons alors la fonction feature_extraction(figure) sur un ensemble issu du MNIST pour le test.

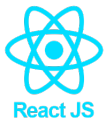
```
# Download test_data
test_data = datasets.MNIST('data', train=False, download=True)

# Create new test_data
new_test_data = []
for i in range (10000):
  feature_ex = feature_extraction(test_data[i][0])
  new_element = (feature_ex, test_data[i][1])
  new_test_data.append(new_element)

# Define test loader
test_loader = DataLoader(new_test_data, batch_size=32, shuffle=True
    )
```

Les résultats restent aux alentours des 90% comme taux de prédiction.

4.2 test sur application

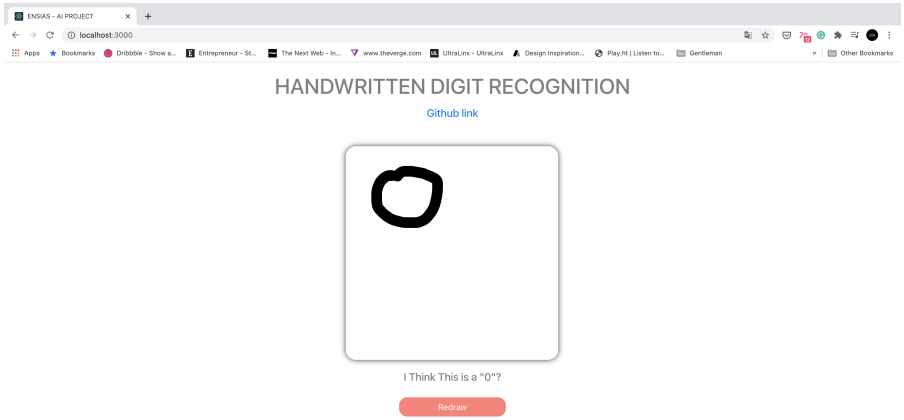
4.2.1 Dévelopement de l'application

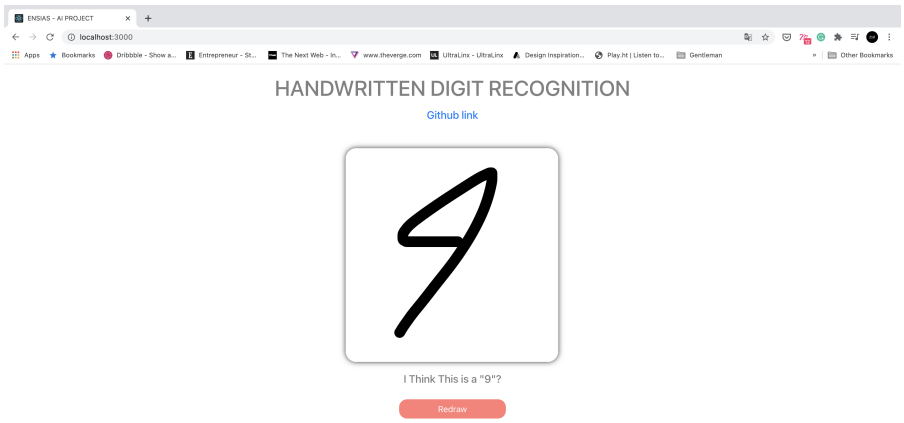
Nous testons notre modèle sur une interface pour un deuxieme test. Il faut d'abord préparer l'application. Nous avons choisi Django pour le backend et le React JS pour le frontend.



4.2.2 Résultat

L'interface est de taille 500*500 pixels, les résultats restent toujours fiable comme la prédiction se fait **instantanément**.


ENSIAS - AI PROJECT
localhost:3000
HANDWRITTEN DIGIT RECOGNITION
Github link
I Think This is a "0"?
Redraw
Made In ENSIAS. By Bender & Benhima. Supervised by Mme Benbrahim.


ENSIAS - AI PROJECT
localhost:3000
HANDWRITTEN DIGIT RECOGNITION
Github link
I Think This is a "9"?
Redraw
Made In ENSIAS. By Bender & Benhima. Supervised by Mme Benbrahim.

5 Conclusion

La nouvelle méthode d'extraction de caractéristiques pour la reconnaissance de caractères manuscrits a été testée par un réseau de neurones. La méthode s'appelle Local Line Fitting (LLF) et produit des vecteurs de taille fixe qui peuvent être utilisés par des systèmes de classification basés sur la géométrie tels que les réseaux de neurones.

Il faut souligner que LLF est une méthode très rapide qui ne nécessite aucun prétraitement (amincissement, binarisation, suppression du bruit, etc.) et qui peut être directement appliquée à des images binaires ou de niveaux de gris de toutes tailles et résolutions.

Le travail est archivé dans un Github repository : HDR link.

Références

[1] Juan-Carlos Perez, Enrique Vidal, and Lourdes Sanchez. Simple and effective feature extraction for optical character recognition. In *Selected Papers From the 5th Spanish Symposium on Pattern Recognition and Images Analysis : Advances in Pattern Recognition and Applications, Valencia, Spain*, 1994.