ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Projet en Intelligence Artificielle

Réalisation d'un interpréteur de chiffres manuscrits avec Python

Filière : Sécurité des Systèmes d'Information

Année universitaire : 2020/2021

Réalisé par :

BELLEFKIH Hajar

CHAKI Aymane

Examiné par :

Mme. Houda Benbrahim

Mme. Lamia Benhiba

Résumé

Durant ce travail nous allons construire une application bureau qui reconnaît les chiffres écrits par un utilisateur sur le canevas à l'aide du curseur de sa souris. Cette reconnaissance sera basée sur un modèle de réseau de neurones que nous allons construire et entraîner à l'aide du dataset MNIST.

Abstract

In this project we will build, from scratch, a desktop application that recognizes the numbers written by a user on the canvas using his mouse cursor. This recognition will be based on a neural network model built and trained using the MNIST dataset.

Liste des abréviations

Abréviation	**Désignation**
RN	Réseau de Neurones
MNIST	Modified National Institute of Standards and Technology
GUI	Graphical User Interface
HOG	Histogram of Oriented Gradients
LLF	Local Line Fitting
GPU	Graphics Processing Unit
CPU	Central Processing Unit

Liste des figures

Liste des codes

Table de matière

Introduction générale

La croissance rapide des nouveaux documents et des nouvelles multimédias a créé de nouveaux défis en matière de reconnaissance de formes et d'apprentissage automatique. La reconnaissance des caractères de l'écriture manuscrite est devenue un domaine de recherche standard en raison des progrès des technologies telles que les dispositifs de capture de l'écriture manuscrite et les ordinateurs portables puissants. Pourtant, étant donné que l'écriture manuscrite dépend beaucoup de la personne qui écrit, la construction d'un système de reconnaissance à haute fiabilité est un défi.

Ce projet vise la reconnaissance des chiffres manuscrits, c'est à-dire les caractères entre 0 et 9, à l'aide d'un modèle simple de réseaux de neurones. La reconnaissance des chiffres manuscrits est une fonction essentielle dans une variété d'applications pratiques, par exemple dans l'administration et la finance. Ces industries nécessitent un excellent taux de reconnaissance avec la plus grande fiabilité. La reconnaissance **sans contrainte** a été appliquée avec d'excellents résultats, des montants écrits sur les chèques, aux formulaires remplis à la main. D'autre part, la reconnaissance **avec contraintes**, fait référence à la mesure dans laquelle les individus croient que des facteurs qui sortent de leur volonté limitent leur comportement.

Le système que nous allons construire est un système de reconnaissance sans contraintes, ce genre de systèmes est décomposé en plusieurs parties, à savoir : le pré-traitement, l'extraction d'attributs, la classification, l'évaluation et la vérification. Ce sont les parties que nous allons principalement couvrir dans ce rapport de projet.

CHAPITRE 1

CRÉATION DES DATASETS MNIST PERSONNALISÉS

Chapitre 1 : Création des datasets MNIST personnalisés

1.1 Introduction

Cette première phase de travail a comme but principal la création des Datasets qui seront utilisés lors de l'apprentissage et l'évaluation de notre réseau de neurones. On commencera d'abord par télécharger les datasets MNIST d'apprentissage et d'évaluation.

```
from torchvision import datasets
mnist_train = datasets.MNIST(root='.', train=True, download=True)
mnist_test = datasets.MNIST(root='.', train=False, download=True)
```

Listing 1: Télécharger les datasets MNIST

1.2 Extraction des attributs

Avant de commencer l'extraction des attributs, on fait d'abord un simple traitement qui consiste à rendre l'image en noir et blanc (écriture en blanc et arrière plan en noir) au lieu de l'échelle du gris. Un simple appel à la fonction `threshold()` du module cv2 réalise ces deux opérations :

```
import cv2
image_binary = cv2.threshold(image_tensor.numpy(), 0, 255, cv2.THRESH_BINARY)[1]
```

Listing 2: Pré-traitement de MNIST

Il existe plusieurs méthodes d'extraction d'attributs tels que la LLF (Local Line Fitting) et la méthode HOG (Histogram of Oriented Gradients). Pourtant, pour ce projet nous n'allons pas utiliser une méthode d'extraction prédéfinie mais nous allons réaliser notre propre extraction manuelle d'attributs. Pour cela, on divise chaque image en un nombre fixe de cellules, puis on extrait de chaque cellule 3 attributs (en se basant sur le papier *"Simple and Effective Feature Extraction for Optical Character Recognition"*, Juan-Carlos Perez, Enrique VidalL Lourdes Sanchez).

1.2.1 Premier attribut :

Le premier attribut est calculé à l'aide de la formule suivante :

$$feature_1 = \frac{n}{N}$$

avec n : *nombre de pixels blancs dans la cellule*
N : *nombre de pixels blancs dans l'image entière*

Pour le calcul des deux attributs suivants, on concidère les cellules comme des plans, les pixels blancs étant des points dans ces plans. Puis on calcule la droite de régression des points de chaque cellule, l'équation de la droite est la suivante : $Y = a + b.X$, avec X l'axe des abscisses

et Y l'axe des ordonnées. En utilisant la méthode des moindres carrés on peut obtenir la valeur du coéfficient b :

$$b = \frac{cov(X,Y)}{V(X)}$$

1.2.2 Deuxième et troisième attributs :

Après calculer la valeur du coefficient de la régression linéaire b, on calcule les deux attributs à l'aide des formules :

$$feature_2 = \frac{2b}{1+b^2} \quad ; \quad feature_3 = \frac{1-b^2}{1+b^2}$$

1.3 Paramètres et résultats de l'extraction

Pour préparer nos propres datasets personnalisés, on crée une fonction python qui réalise le pré-traitement des images MNIST puis en extrait les attributs souhaités dans un torseur. On applique cette fonction aux données de l'apprentissage ainsi qu'aux données de l'évaluation, ce qui donne un total de 70000 images traitées. On choisit de diviser les images en 16 cellules (4x4), donc à chaque image il serait associé un torseur de taille (16,3).

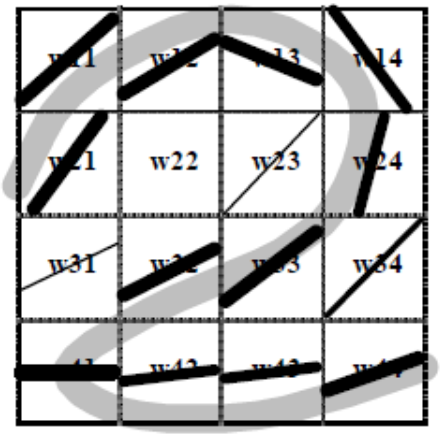

Figure 1: Division de l'image en 16 cellules

L'exécution du script de l'extraction des attributs nous affiche les informations suivantes :

```
Train data size :  torch.Size([60000, 16, 3])
Test data size :  torch.Size([10000, 16, 3])
Feature extraction took 2282.9031821 seconds to finish
```

Figure 2: Résultat de l'extraction des attributs

1.4 Création des datasets

La préparation de notre modèle de reconnaissance va passer par 3 étapes principales : l'entraînement, la validation et l'évaluation. Pour cela, on aura besoin de 3 datasets différents pour chacune de ces étapes. Une classe est donc créée pour regrouper les caractéristiques de nos datasets. Pour créer le dataset de validation, on prend une partie (1/6) du dataset de l'entrainement.

```
from sklearn.model_selection import train_test_split
training_data, validation_data, training_targets, validation_targets = train_test_split(
new_train_data,mnist_train.targets, test_size=10000)
```

Listing 3: Appliquer le split au dataset d'entraînement

Après la préparation de nos données, la création des datasets n'est maintenant qu'une instanciation de la classe de datasets créée.

```
training_dataset = MyDataset(training_data, training_targets)
test_dataset = MyDataset(new_test_data, mnist_test.targets)
validation_dataset = MyDataset(validation_data, validation_targets)
```

Listing 4: Création des datasets

1.5 Conclusion

Vu la grande taille des datasets MNIST, la phase de l'extraction des attributs a une grande complexité temporelle. Cette opération d'extraction d'attributs ne doit pas s'exécuter chaque fois que l'application se lance. Une solution à ce problème est de préparer les datasets avant le lancement de l'application et les sauvegarder dans des fichiers qui seront lus chaque fois qu'on en aura besoin. La fonction `save()` du module **torch** fait l'affaire.

```
from torch import save
save(training_dataset,'datasets\\training.pt')
save(test_dataset,'datasets\\test.pt')
save(validation_dataset,'datasets\\validation.pt')
```

Listing 5: Sauvegarde des datasets

CHAPITRE 2

IMPLÉMENTATION DU RÉSEAU DE NEURONES

Chapitre 2: Implémentation du réseau de neurones

2.1 Introduction

Pour réaliser notre modèle de reconnaissance on se servira d'un simple réseau de neurones de classification. Notre travail dans ce chapitre consiste à conceptualiser puis implémenter le réseau de neurones en essayant de choisir les bons paramètres et hyper-paramètres pour nous rapprocher le plus possible du modèle **Best-Fit**.

2.2 Paramètres du réseau de neurones

Notre réseau de neurones contiendra principalement une couche d'entrée et une couche de sortie. La couche d'entrée doit contenir assez de neurones que d'attributs de données qu'on est censé lui passer en entrée. Le nombre total d'attributs contenus dans une seule image dépendra donc de la division de cellules réalisée dans la phase de l'extraction des attributs. En premier temps, le nombre d'attributs extraits d'une image est **48 attributs**, c'est donc le nombre de neurones de la couche d'entrée du réseau de neurones. Le nombre de neurones en couche de sortie est **10 neurones** vu que le nombre d'étiquettes des images est 10 (de 0 à 9). Pour choisir le nombre de couches cachées et la nombre de neurones dans ces couches on se base sur des bonnes pratiques fournies par la communauté de ce domaine.

```
class MyNetwork(nn.Module):
    def __init__(self, n_features, hid1, hid2, out):
        #the parameter n_features implies the total number of features in a single image
        super(MyNetwork, self).__init__()
        self.hidden1 = nn.Linear(n_features, hid1)
        self.hidden2 = nn.Linear(hid1, hid2)
        self.output = nn.Linear(hid2, out)

    def forward(self, x):
        x = F.relu(self.hidden1(x))
        x = F.relu(self.hidden2(x))
        return self.output(x)
```

Listing 6: Implémentation du réseau de neurones

Pour instancier notre modèle, on précise le nombre de neurones contenues dans chaque couche. On choisie dans un premier lieu les valeurs 64 et 42 pour le nombres de neurones de la première et la deuxième couches cachées (respectivement).

```
from my_classes import MyNetwork
model = MyNetwork(n_features = 48, hid1=64, hid2=42, out=10)
```

Listing 7: Instanciation du modèle

Remarque :
Tout paramètre modifiable pouvant affecter l'apprentissage de notre réseau de neurones s'appelle un **hyper-paramètre**. Ces paramètres là doivent être facilement modifiables dans nos scripts pour pouvoir facilement tester les différentes valeurs et finalement aboutir à la meilleur combinaison possible de ces paramètres. Les principaux hyper-paramètres de notre modèle sont :

- La division des cellules
- Le nombre de couches cachées
- Le nombre de neurones dans chaque couche cachée

2.3 Entraînement, validation et vérification

Après avoir chargé les datasets préparés dans la phase précédente à partir de leurs fichiers sauvegardés, on se sert des Dataloader pour passer les données au réseau de neurones sous forme de mini-lots (mini-batches). On peut aussi préciser le nombre d'itérations sur lesquels le modèle vas boucler pendant son apprentissage, ce nombre est appelé **epoch**.
On précise ensuite chacune des fonctions de perte et d'optimisation, ainsi que le taux d'apprentissage. Après chaque itération, les moyennes de pertes et la précision des prédictions du modèle sont calculées et affichées pour qu'on puisse suivre l'avancement de l'apprentissage de notre modèle lors de l'exécution du script. Pour notre cas, les choix suivants ont été faits :

- **taille de lot :** 32
- **nombre d'epochs :** 10
- **taux d'apprentissage :** 0.0001
- **fonction de perte :** *Cross entropy loss*
- **fonction d'optimisation :** Adam

Ces choix, combinés avec les choix des paramètres du réseau de neurones et d'extraction des attributs nous ont permis d'atteindre une précision de **79%**.

2.4 Conclusion

Similairement à la phase d'extraction d'attributs, la phase d'apprentissage du modèle de reconnaissance prend aussi beaucoup de temps, et elle prendra encore plus de temps lorsqu'on essayera d'augmenter le nombre d'epochs. Ainsi, cette phase ne doit pas s'exécuter avec le démarrage de l'application. Donc, en se servant encore une fois de la fonction `save()` de **torch** on peut sauvegarder le modèle entraîné dans un fichier pour l'utiliser ultérieurement sans avoir besoin de refaire l'apprentissage.

```
1 from torch import save
2 save(model, 'models\\digit_model.pt')
```

Listing 8: Sauvegarde du modèle

CHAPITRE 3

RÉALISATION DE L'APPLICATION

Chapitre 3 : Réalisation de l'application

3.1 Introduction

Maintenant que nous avons préparé et sauvegardé notre modèle de reconnaissance, on construit notre application de reconnaissance des chiffres manuscrits. Cette application sera composée d'une interface graphique où l'utilisateur écrit le chiffre qui doit être reconnu à l'aide du modèle de reconnaissance, et d'une partie de traitement qui a comme but principale préparer ce que l'utilisateur a saisi et le passer au modèle pour que ce dernier retourne sa prédiction qui sera après affichée à l'utilisateur.

3.2 Description de l'interface graphique

Lorsque le programme est exécuté, l'interface ci-dessous s'affiche. Dans cette interface graphique, l'utilisateur intéragit avec l'application en écrivant un chiffre à l'aide du curseur dans la grande zone carrée (canevas). Le bouton «Show result» affiche la valeur prédite par notre modèle de reconnaissance.

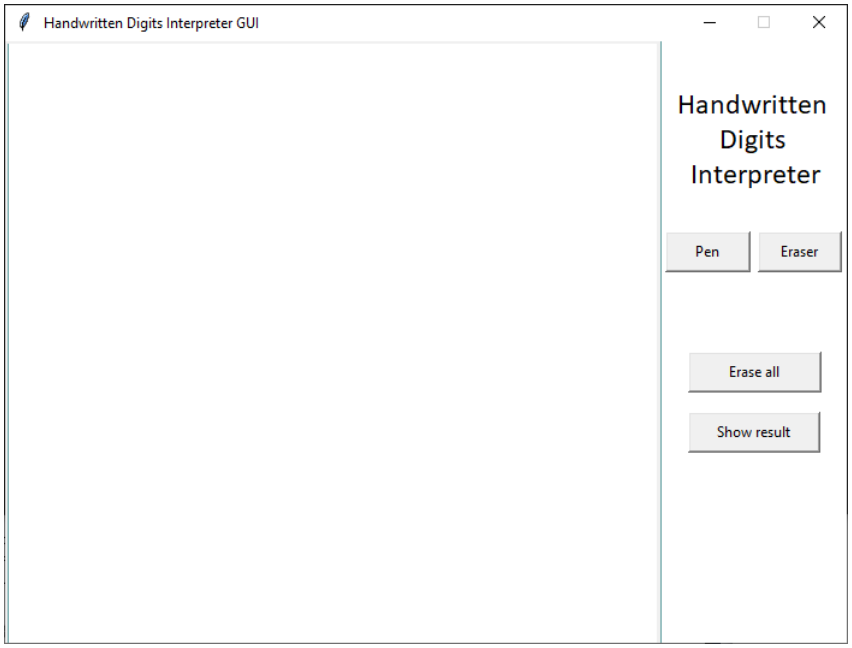

Figure 3: Interface graphique de l'application

Le bouton « Erase all » supprime tout le contenu de la zone de dessin, et les boutons «Pen» et «Eraser» permettent de basculer entre le *stylo* et la *gomme*.

3.3 Fonctionnement de l'application

Afin de pouvoir reconnaître le chiffre écrit par un utilisateur, plusieurs tâches d'arrière-plan sont déclenchées lorsque le bouton «Show result» est appuyé.

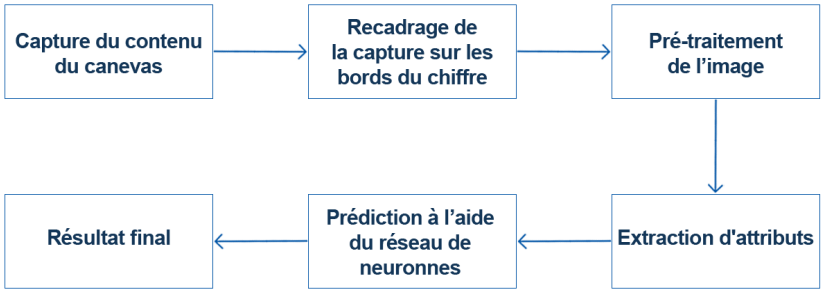

Figure 4: Processus d'arrière-plan de l'application

3.3.1 Capture du contenu du canevas

Dans cette étape l'application ne fait que capturer exactement ce que l'utilisateur a écrit à l'aide de la fonction `postscript()`.

```
ps = myCanvas.postscript(colormode='mono') # takes a snapshot of the whole canva
img = Image.open(io.BytesIO(ps.encode('utf-8')))
```

Listing 9: Capture du canvas

3.3.2 Re-cadrage de la capture

Pour avoir une image qui délimite le chiffre écrit par l'utilisateur par ses bords, on utilise la fonction `findContours()` du module **cv2**.

```
gray= cv2.cvtColor(open_cv_image,cv2.COLOR_BGR2GRAY)
edges= cv2.Canny(gray, 50,200)
contours= cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
sorted_contours= sorted(contours, key=cv2.contourArea, reverse= True)
    for (i,c) in enumerate(sorted_contours):
        x,y,w,h= cv2.boundingRect(c)
        cropped_contour= original_image[y-10:y+h+10, x-10:x+w+10]
        image_name= "processed_images\\croppedImage.png"
        cv2.imwrite(image_name, cropped_contour)
        break   #on arrete apres une iteration pour ne prendre que l'image avec la plus grande
     surface, c'est l'image qu'on veut
```

Listing 10: Re-cadrage

La nouvelle image re-cadrée est sauvegardée dans le dossier "processed_images" de notre répertoire courant.

3.3.3 Pré-traitement de l'image

On récupère l'image re-cadrée du dossier "processed_images", on convertit sa taille à 28x28 avec la fonction `resize()`, puis on la transforme en torseur. Ensuite, on applique la fonction threshold pour rendre l'image binaire et inverser ses pixels noirs et blancs (l'écriture devient en blanc et l'arrière-plan en noir), et on la transforme en torseur encore. Maintenant notre image correspond aux images de la librairie MNIST. On peut donc passer à l'extraction des attributs.

```
resized_img = img.resize((28, 28))
resized_img.save("processed_images\\resized_image.png")
image_tensor = img_to_Tensor('processed_images\\resized_image.png') # transform the resized
    and cropped image to tensor
image_binary = cv2.threshold(image_tensor.numpy(), 0, 255, cv2.THRESH_BINARY_INV)[1]
image_binary_tensor = from_numpy(image_binary)
```

Listing 11: Pré-traitement

3.3.4 Extraction d'attributs

On appelle notre fonction d'extraction d'attributs et on sauvegarde les attributs sauvegardés dans un torseur qui sera passé comme paramètre du modèle.

```
features = extract_features(image_binary_tensor)
feature_list = []
feature_list.append(features)
data = from_numpy(np.array(feature_list))
```

Listing 12: Extraction d'attributs

3.3.5 Prédiction de l'étiquette du chiffre écrit

On charge notre modèle sauvegardé dans un fichier, et on l'utilise pour prédir l'étiquette de notre image. L'étiquette prédite est la sortie du modèle de reconnaissance la plus excitée.

```
digit_recognition_model = load('models\\digit_model_7x7.pt')
test_output = digit_recognition_model(flatten(data, start_dim=1).float())
prediction = max(test_output, 1)[1].data.numpy().squeeze()
```

Listing 13: Prédiction

3.3.6 Affichage du résultat

Le résultat prédit est affiché dans un label.

```
result_label = Label(rightCanvas,text="The written number is : \n "+ str(prediction))
```

Listing 14: Résultat

3.4 Essai de l'application

Maintenant, mettons à l'épreuve notre application. On lance l'application en exécutant le code python "GUI.py". Testons quelques chiffres.

- **Le chiffre 0 :**

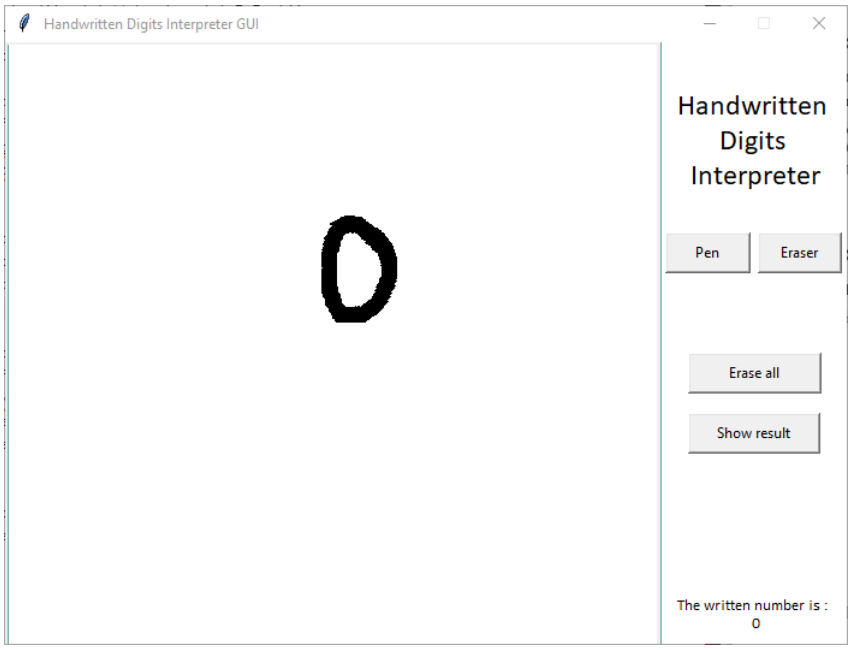

Figure 5: Reconnaissance du chiffre 0

Ce chiffre est reconnu correctement, voici le résultat du pré-traitement de cette saisie :

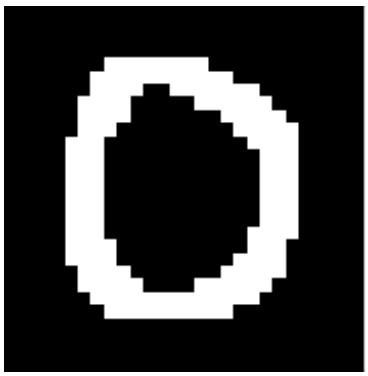

Figure 6: Chiffre 0 manuscrit après traitement

- **Le chiffre 2 :**

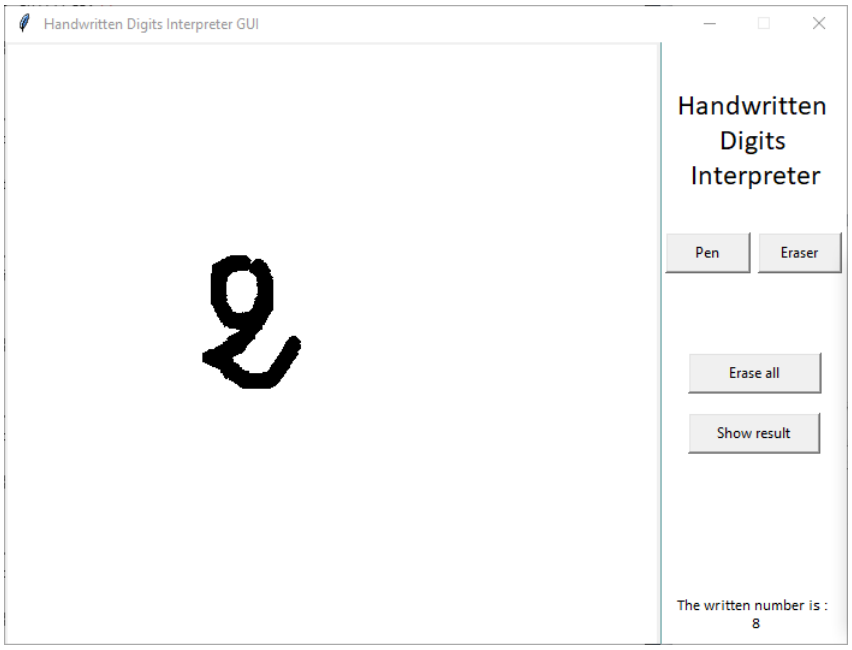

Figure 7: Reconnaissance du chiffre 2 : Échec

La reconnaissance de ce chiffre a échoué, voici le résultat du pré-traitement de cette saisie.

Figure 8: Chiffre 2 manuscrit après traitement

L'image traitée est bonne, pourtant la prédiction est mauvaise.

3.5 Conclusion

On a vérifié que notre application marche bien et reconnaît les chiffres manuscrits avec une précision intéressante. Cependant, pour quelques chiffres (le chiffre 2 par exemple), la reconnaissance n'est pas toujours correcte. En analysant les images traitées on remarque qu'elles sont relativement bonnes, mais malgré ça les prédictions échouent parfois. Donc la cause de cette faible précision ne serait pas le pré-traitement mais le modèle lui même. Essayons donc d'augmenter la précision du modèle.

CHAPITRE 4

QUELQUES AMÉLIORATIONS

Chapitre 4 : Quelques améliorations

4.1 Introduction

Les tests du chapitre précédent ont montré que notre modèle n'a pas une précision acceptable, dans ce chapitre nous allons donc aborder quelques moyens pour améliorer les performances de notre modèle.

4.2 Modification des hyper-paramètres

Les hyper-paramètres de notre modèle actuel de précision 79% sont :

- **Division des cellules :** 4x4
- **Nombre de neurones de la première couche cachée :** 64
- **Nombre de neurones de la deuxième couche cachée :** 42
- **Taille de lot :** 32
- **Nombre d'epochs :** 10
- **Taux d'apprentissage :** 0.0001

Après plusieurs modifications et tentatives, nous avons pu atteindre une précision de **91%** seulement en jouant sur les hyper-paramètres, et voici notre nouvelle et meilleure configuration.

- **Division des cellules :** 7x7 (cela affecte aussi le nombre de couches d'entrée du RN qui est devenu 147)
- **Nombre de neurones de la première couche cachée :** 120
- **Nombre de neurones de la deuxième couche cachée :** 96
- **Taille de lot :** 32
- **Nombre d'epochs :** 40
- **Taux d'apprentissage :** 0.00001

Maintenant, avec une précision de 91%, les chiffres qui posaient des problèmes auparavant sont plus faciles à détecter.

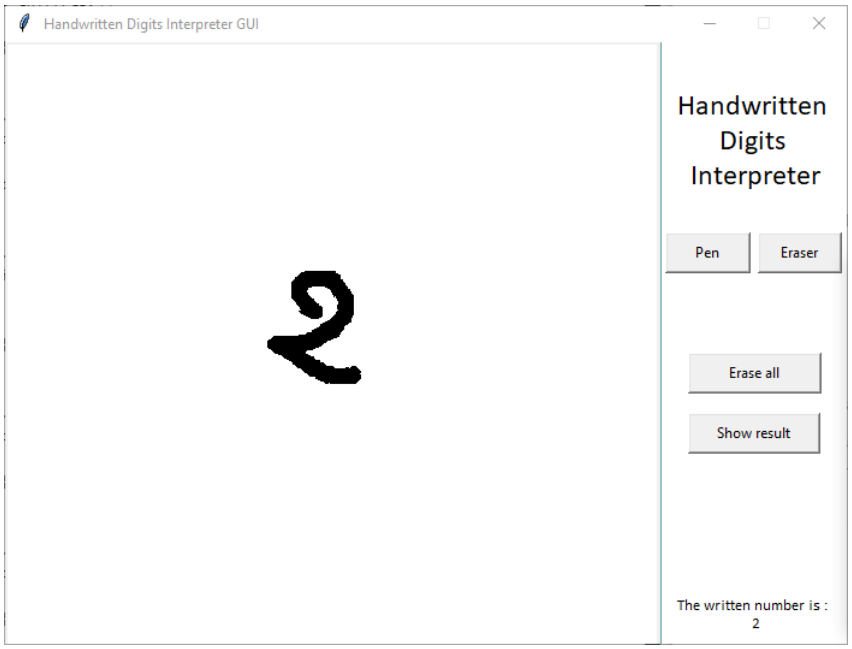

Figure 9: Reconnaissance du chiffre 2 : Succès

Le chiffre est reconnu avec succès!

4.3 Utilisation de la GPU

En augmentant le nombre de neurones de notre réseau, et en diminuant le taux d'apprentissage et la taille des lots de données, le temps nécessaire pour préparer notre modèle augmente, sans oublier le temps nécessaire pour extraire les attributs de nos 70000 images maintenant qu'ils sont divisés en 49 cellules au lieu de 16.
Ces augmentations dans les durées d'exécution rendent la nouvelle configuration une nuisance alors qu'elle est censée être un avantage et une amélioration. La solution à ce problème est d'utiliser la carte graphique (GPU) de notre machine au lieu de la CPU. La carte graphique, étant plus performante que la CPU, nous permettra de réaliser l'extraction des attributs, la préparations des datasets et l'apprentissage du réseau de neurone dans un temps très inférieur au temps que prendrait le CPU pour réaliser ces tâches.
Heureusement, la framework Pytorch supporte l'utilisation des GPU, et c'est aussi facile que jamais. Il suffit d'avoir une carte graphique qui supporte CUDA, et installer CUDA sur notre machine. On utilise la GPU de la manière suivante :

```
######################### computing on the GPU for better performance ##########################
gpu = device('cuda:0' if cuda.is_available() else 'cpu')
#Pour traiter une ressource de pytorch en utilisant la GPU on charge la ressource dans la GPU#
ressource.to(gpu)
```

Listing 15: Utilisation de la GPU

4.4 Conclusion

L'optimalité doit toujours être poursuivie dans n'importe quel travail. Cependant, il est très difficile parfois de trouver une manière exacte pour atteindre ce but. Les méthodes utilisées pour atteindre l'optimalité sont expérimentales en premier lieu.

Conclusion et perspectives

Ce projet se concentre dans le domaine de l'apprentissage et de l'intelligence artificielle, notamment la classification et la reconnaissance. Dans ce travail, nous avons établis un système d'apprentissage sans contraintes qui permet la reconnaissance des chiffres manuscrits. Nous avons pris le temps d'expliquer et décrir chaque phase du processus de sa création, nous avons aussi introduit quelques méthodes d'optimisation de notre modèle et de notre utilisation des ressources.

Pour conclure, le projet nous a permis d'explorer et manipuler les caractéristiques des images et du réseau de neurones également, et se familiariser avec la création des modèles de reconnaissance et leur utilisation.

Références

[1] Juan-Carlos Perez ; Enrique Vidal ; Lourdes Sanchez. «Simple and effective feature extraction for optical character recognition» [PDF].

[2] Victor Basu, (Octobre 2018). «Handwritten Digits Recognition», disponible sur medium.com

[3] Jeff Heaton, (2008). «Introduction to Neural Networks for Java», $2^{ème}$ édition, aperçu disponible gratuitement sur Google Books.

[4] «Build Your First Neural Network with PyTorch», (Février 2020), disponible sur curiousily.com

[5] Aayush Agrawal, (Juin 2018). «Building Neural Network from scratch», disponible sur towardsdatascience.com

[6] Jason Brownlee, (Juillet 2018). «Difference Between a Batch and an Epoch in a Neural Network», disponible sur machinelearningmastery.com

[7] Mesay Hailemariam Moreda, (Juillet 2003). «Line fitting to Amharic OCR : The case of postal address» [PDF], disponible sur etd.aau.edu.et

[8] Hitesh Kumar Kushwaha ; Akanksha Rai (Octobre 2019). «Running Python script on GPU», disponible sur www.geeksforgeeks.org

[9] Kanghui, (Mars 2020). «Pytorch 3 : Tensor and Images», disponible sur www.bigrabbitdata.com

[10] «Running on the GPU - Deep Learning and Neural Networks with Python and Pytorch p.7» (Octobre 2019), disponible sur pythonprogramming.net

[11] George Liu (Janvier 2019). «Optimizing Neural Networks — Where to Start ?», disponible sur towardsdatascience.com

[12] «How to run a basic RNN model using Pytorch ?», disponible sur www.dezyre.com

[13] Bastien L. (Février 2021). «Python : tout savoir sur le principal langage Big Data et Machine Learning», disponible sur www.lebigdata.fr

[14] «Code Visual Studio», disponible sur wikipedia.org

[15] «Tkinter», disponible sur wikipedia.org