Université Hassan 1er

Ecole Nationale des Sciences Appliquées de Berrechid

Département de mathématique et informatique

Filière : Ingénierie des Systèmes d'Information et BIG DATA

Module : Introduction sur Python

Semestre : S7

Rapport de Projet

Réalisation des applications en Deep Learning

- **Handwritten Recognition Digits**
- **Conversational agent (Chatbot)**

Equipe de travail :

Salaheddine EL BAIDOURY

Mohamed SAIDI

Encadré Par :

Lahcen MOUMOUN

Année Universitaire : 2020/2021

Table des matières

Liste des figures

Introduction

Aujourd'hui, d'une simple phrase, on peut connaître la météo, programmer une alarme, être averti de ses rendez-vous, avec les trajets pour s'y rendre et des résultats de plus en plus précis au fil du temps… C'est la face émergée de l'IA dans notre vie quotidienne. Mais au-delà de ces usages, pratiques mais un peu anecdotiques, l'intelligence artificielle recouvre un vaste domaine et des champs d'application existants ou potentiels quasi-infinis.

Traduction automatique, reconnaissance d'image, reconnaissance faciale, assistants vocaux, chatbots, véhicules autonomes, agriculture, secteur bancaire, optimisation de la prospection commerciale, maintenance prédictive, aide médicale au diagnostic, Facility Management et smart building, amélioration des processus de recrutement, réduction des consommations d'énergie… L'intelligence artificielle est une science qui bouleverse tous les secteurs, tous les domaines.

Chapitre 1

I. L'intelligence artificiel :

L'intelligence artificielle peut se définir comme « l'ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence », selon le Larousse. Soit des ordinateurs ou des programmes capables de performances habituellement associées à l'intelligence humaine, et amplifiées par la technologie :

- Capacité de raisonner
- Capacité de traiter de grandes quantités de données
- Faculté de discerner des patterns et des modèles indétectables par un humain
- Aptitude à comprendre et analyser ces modèles
- Capacités à interagir avec l'homme
- Faculté d'apprendre progressivement
- Et d'améliorer continuellement ses performances

II. Machine Learning :

Le machine Learning est le fondement de la plupart des solutions d'IA et permet la création de modèles qui prédisent des valeurs inconnues et déduisent des informations à partir des données observées.

- Comment les machines apprennent-elles ?

La réponse est : à partir de données. Dans le monde d'aujourd'hui, nous créons d'énormes volumes de données au fur et à mesure de nos activités quotidiennes. Des messages texte, des e-mails et des publications sur les réseaux sociaux que nous envoyons aux photos et vidéos que nous prenons sur nos téléphones, nous générons d'énormes quantités d'informations. De plus en plus de données sont constamment créées par des millions de capteurs dans nos maisons, nos voitures, nos villes, nos infrastructures de transport public et nos usines.

Les scientifiques des données peuvent utiliser toutes ces données pour former des modèles Machine Learning capables de faire des prédictions et des inférences en fonction des relations qu'ils découvrent dans les données.

1. L'apprentissage supervisé :

Ici, la machine s'appuie sur des classes prédéterminées et sur un certain nombre de paradigmes connus pour mettre en place un système de classement à partir de modèles déjà catalogués. Dans ce cas, deux étapes sont nécessaires pour compléter le processus, à commencer par le stade d'apprentissage qui consiste à la modélisation des données cataloguées. Ensuite, il s'agira au second stade de se baser sur les données ainsi définies pour attribuer des classes aux nouveaux modèles introduits dans le système, afin de les cataloguer eux aussi.

1. 1 Régression

La régression est une forme d'apprentissage automatique qui est utilisée pour prédire une étiquette numérique en fonction des fonctionnalités d'un élément. La régression est un exemple de technique d'apprentissage automatique *supervisé* dans laquelle vous formez un modèle à l'aide de données qui incluent à la fois les fonctionnalités et les valeurs connues de l'étiquette, afin que le modèle apprenne à *ajuster* les combinaisons de fonctionnalités à l'étiquette. Ensuite, une fois la formation terminée, vous pouvez utiliser le modèle formé pour prédire les étiquettes des nouveaux éléments dont l'étiquette est inconnue.

1. 2 Classification

La classification est une forme de machine Learning utilisée pour prédire à quelle catégorie, ou classe, un élément appartient. Par exemple, une clinique peut utiliser les caractéristiques d'un patient (telles que l'âge, le poids, la tension artérielle, etc.) pour prédire si le patient présente un risque de diabète. Dans ce cas, les caractéristiques du patient sont les traits, et l'étiquette est une classification de 0 ou 1, représentant non diabétique ou diabétique.

Comme la régression, la classification est un exemple de technique de Machine Learning supervisé dans laquelle vous formez un modèle à l'aide de données qui incluent à la fois les fonctionnalités et les valeurs connues de l'étiquette, afin que le modèle apprenne à ranger les combinaisons de fonctionnalités dans l'étiquette. Ensuite, une fois la formation terminée, vous pouvez utiliser le modèle formé pour prédire les étiquettes des nouveaux éléments dont l'étiquette est inconnue.

2. L'apprentissage non supervisé :

Dans ce mode de fonctionnement du Machine Learning, il n'est pas question de s'appuyer sur des éléments prédéfinis, et la tâche revient à la machine de procéder toute seule à la catégorisation des données. Pour ce faire, le système va croiser les informations qui lui sont soumises, de manière à pouvoir rassembler dans une même classe les éléments présentant certaines similitudes. Ainsi, en fonction du but recherché, il reviendra à l'opérateur ou au chercheur de les analyser afin d'en déduire les différentes hypothèses.

1. 3 Clustering

Le clustering est une forme de Machine Learning utilisée pour regrouper des éléments en grappes ou clusters en fonction des similitudes de leurs fonctionnalités. Par exemple, un botaniste peut mesurer des plantes et les regrouper en fonction de similitudes dans leurs proportions.

Le clustering est un exemple de Machine Learning non supervisé, dans lequel vous formez un modèle à séparer les éléments en clusters uniquement en fonction de leurs caractéristiques, ou traits. Il n'y a pas de valeur de cluster connue auparavant (ou « étiquette ») à partir de laquelle former le modèle.

III. Deep Learning :

Le terme Deep Learning (en français : apprentissage profond) est très en vogue ces derniers temps. C'est bien simple, lorsque l'on parle d'intelligence artificielle, on parle presque systématiquement de Deep Learning. A tel point que dans l'esprit de beaucoup, ces deux termes sont synonymes. C'est pourtant inexact. Par définition, le Deep Learning désigne l'apprentissage profond, une branche de l'IA.

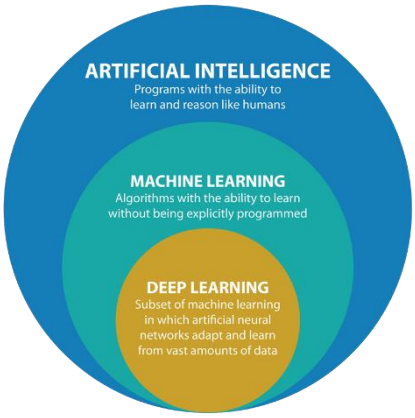

Figure 1 : AI, ML, et Deep Learning

1. Réseau de neurones :

L'intelligence artificielle a connu une longue histoire puisqu'elle a été conceptualisée dès l'antiquité. Bien sûr ce n'est que suite à la création des premiers ordinateurs qu'elle a pu être mise en œuvre de façon concrète. Différents courants se sont alors développés. L'un de ces courants consistait à s'inspirer du fonctionnement du cerveau humain afin de tenter de créer des neurones artificiels. Un neurone artificiel n'est rien d'autre qu'une opération mathématique relativement simple. La complexité repose avant tout dans l'interconnexion de plusieurs neurones.

Le premier réseau de neurones artificiels a été mis au point en 1951 par Marvin Minsky et Dean Edmonds de l'Université de Harvard. Peu de temps après, en 1956, Frank Rosenblatt a mis au point le Perceptron qui a suscité un grand engouement. Les scientifiques misaient alors énormément sur les réseaux de neurones mais les résultats ont fini par décevoir.

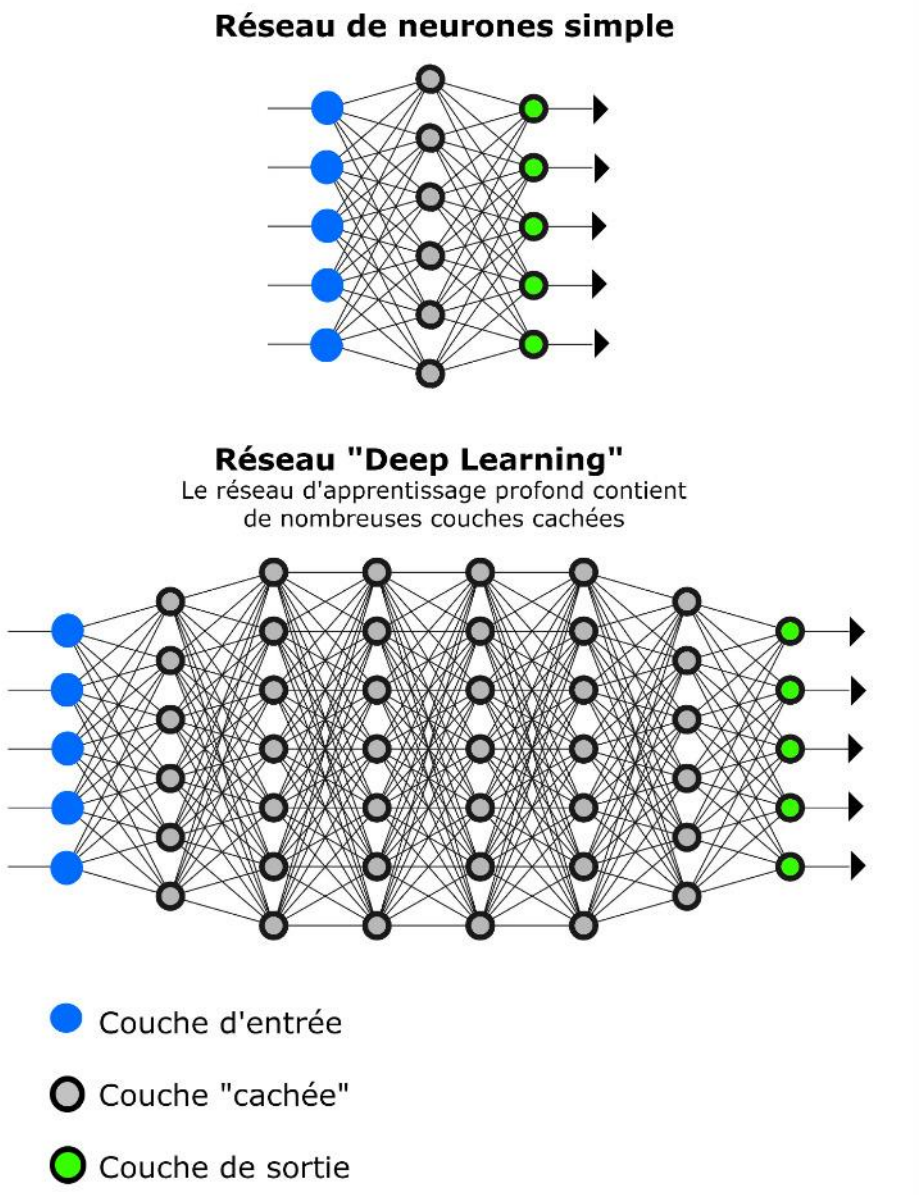

Figure 2 : Réseau de neurones simple et réseau de Deep Learning

Vous le savez désormais, le terme de Deep Learning, Apprentissage profond en français, désigne en fait un réseau de neurones constitué de nombreuses couches. Il ne s'agit que d'une évolution des réseaux de neurones.

2. Réseau de neurones convolutifs

Les réseaux de neurones convolutifs désignent une sous-catégorie de réseaux de neurones : ils présentent donc toutes les caractéristiques listées ci-dessus. Cependant, les CNN sont spécialement conçus pour traiter des images en entrée. Leur architecture est alors plus spécifique : elle est composée de deux blocs principaux.

Le premier bloc fait la particularité de ce type de réseaux de neurones, puisqu'il fonctionne comme un extracteur de features. Pour cela, il effectue du template matching en appliquant des opérations de filtrage par convolution. La première couche filtre l'image avec plusieurs noyaux de convolution, et renvoie des "feature maps", qui sont ensuite normalisées (avec une fonction d'activation) et/ou redimensionnées.

Ce procédé peut être réitéré plusieurs fois : on filtre les features maps obtenues avec de nouveaux noyaux, ce qui nous donne de nouvelles features maps à normaliser et redimensionner, et qu'on peut filtrer à nouveau, et ainsi de suite. Finalement, les valeurs des dernières feature maps sont concaténées dans un vecteur. Ce vecteur définit la sortie du premier bloc, et l'entrée du second.

Le second bloc n'est pas caractéristique d'un CNN : il se retrouve en fait à la fin de tous les réseaux de neurones utilisés pour la classification. Les valeurs du vecteur en entrée sont transformées (avec plusieurs combinaisons linéaires et fonctions d'activation) pour renvoyer un nouveau vecteur en sortie. Ce dernier vecteur contient autant d'éléments qu'il y a de classes : l'élément ii représente la probabilité que l'image appartienne à la classe ii. Chaque élément est donc compris entre 0 et 1, et la somme de tous vaut 1. Ces probabilités sont calculées par la dernière couche de ce bloc (et donc du réseau), qui utilise une fonction logistique (classification binaire) ou une fonction softmax (classification multi-classe) comme fonction d'activation.

Comme pour les réseaux de neurones ordinaires, les paramètres des couches sont déterminés par rétropropagation du gradient : l'entropie croisée est minimisée lors de la phase d'entraînement. Mais dans le cas des CNN, ces paramètres désignent en particulier les features des images.

3. Les Couches de CNN

Il existe quatre types de couches pour un réseau de neurones convolutif : la couche de convolution, la couche de pooling, la couche de correction ReLU et la couche fully-connected. Dans ce chapitre, je vais vous expliquer le fonctionnement de ces différentes couches.

3. 1 La couche de convolution

La couche de convolution est la composante clé des réseaux de neurones convolutifs, et constitue toujours au moins leur première couche.

Son but est de repérer la présence d'un ensemble de features dans les images reçues en entrée. Pour cela, on réalise un filtrage par convolution : le principe est de faire "glisser" une fenêtre représentant la feature sur l'image, et de calculer le produit de convolution entre la feature et chaque portion de l'image balayée. Une feature est alors vue comme un filtre : les deux termes sont équivalents dans ce contexte.

Cette technique est très proche de celle étudiée dans la partie précédente pour faire du template matching : ici, c'est le produit convolution qui est calculé, et non la corrélation croisée.

La couche de convolution reçoit donc en entrée plusieurs images, et calcule la convolution de chacune d'entre elles avec chaque filtre. Les filtres correspondent exactement aux features que l'on souhaite retrouver dans les images.

On obtient pour chaque paire (image, filtre) une carte d'activation, ou feature map, qui nous indique où se situent les features dans l'image : plus la valeur est élevée, plus l'endroit correspondant dans l'image ressemble à la feature.

Contrairement aux méthodes traditionnelles, les features ne sont pas pré-définies selon un formalisme particulier (par exemple SIFT), mais apprises par le réseau lors la phase d'entraînement ! Les noyaux des filtres désignent les poids de la couche de convolution. Ils sont initialisés puis mis à jour par rétropropagation du gradient.

C'est là toute la force des réseaux de neurones convolutifs : ceux-ci sont capables de déterminer tout seul les éléments discriminants d'une image, en s'adaptant au problème posé. Par exemple, si la question est de distinguer les chats des chiens, les features automatiquement définies peuvent décrire la forme des oreilles ou des pattes.

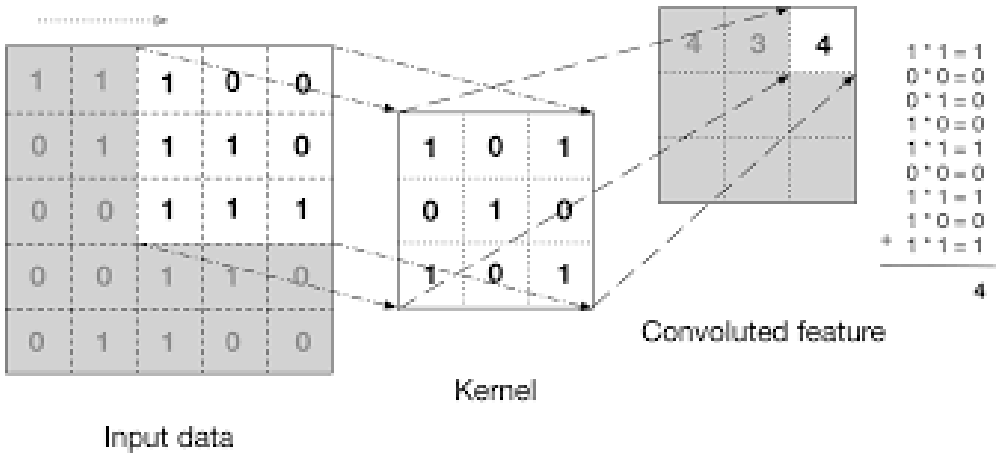

Figure 3 : Couche de Pooling

3. 2 La couche de pooling

Ce type de couche est souvent placé entre deux couches de convolution : elle reçoit en entrée plusieurs feature maps, et applique à chacune d'entre elles l'opération de pooling. L'opération de pooling consiste à réduire la taille des images, tout en préservant leurs caractéristiques importantes. Pour cela, on découpe l'image en cellules régulière, puis on garde au sein de chaque cellule la valeur maximale. En pratique, on utilise souvent des cellules carrées de petite taille pour ne pas perdre trop d'informations. Les choix les plus communs sont des cellules adjacentes de taille 2 ×× 2 pixels qui ne se chevauchent pas, ou des cellules de taille 3 ×× 3 pixels, distantes les unes des autres d'un pas de 2 pixels (qui se chevauchent donc).

On obtient en sortie le même nombre de feature maps qu'en entrée, mais celles-ci sont bien plus petites. La couche de pooling permet de réduire le nombre de paramètres et de calculs dans le réseau. On améliore ainsi l'efficacité du réseau et on évite le sur-apprentissage. Les valeurs maximales sont repérées de manière moins exacte dans les feature maps obtenues après pooling que dans celles reçues en entrée – c'est en fait un grand avantage ! En effet, lorsqu'on veut reconnaître un chien par exemple, ses oreilles n'ont pas besoin d'être localisées le plus précisément possible : savoir qu'elles se situent à peu près à côté de la tête suffit ! Ainsi, la couche de pooling rend le réseau moins sensible à la position des features : le fait qu'une feature se situe un peu plus en haut ou en bas, ou même qu'elle ait une orientation légèrement différente ne devrait pas provoquer un changement radical dans la classification de l'image.

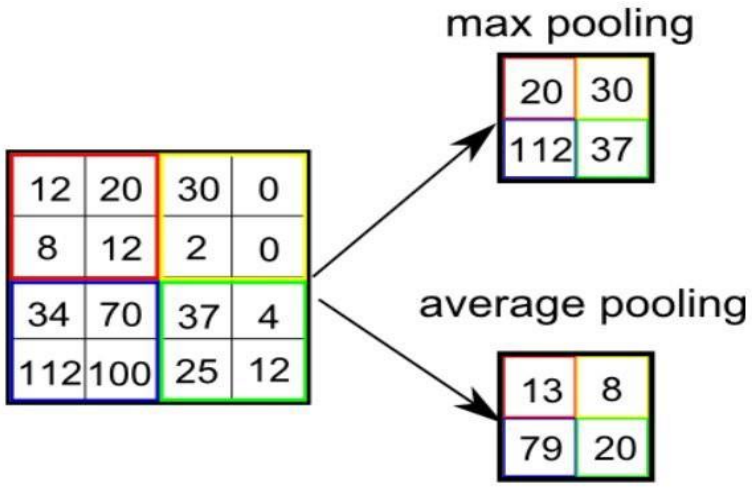

Figure 4 : Couche de Pooling

3. 3 La couche de correction ReLU

ReLU[1] désigne la fonction réelle non-linéaire définie par Relu(x) = max (0, x)

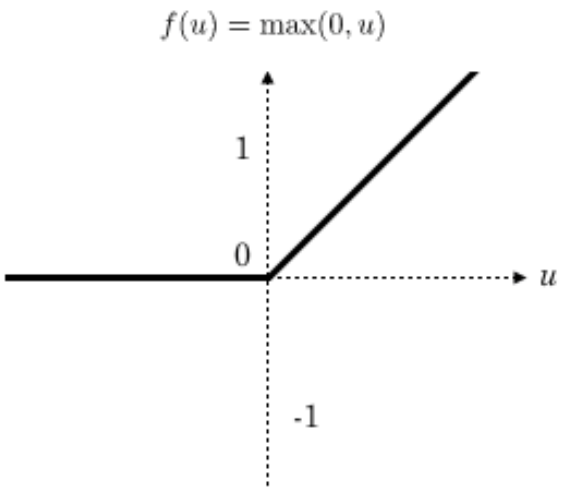

Figure 5 : Allure de la fonction ReLU

La couche de correction ReLU remplace donc toutes les valeurs négatives reçues en entrées par des zéros. Elle joue le rôle de fonction d'activation.

3. 4 La couche fully-connection

La couche fully-connection constitue toujours la dernière couche d'un réseau de neurones, convolutif ou non – elle n'est donc pas caractéristique d'un CNN. Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée. La dernière couche fully-connection permet de classifier l'image en entrée du réseau : elle renvoie un vecteur de taille NN, où NN est le nombre de classes dans notre problème de classification d'images. Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe.

Chaque valeur du tableau en entrée "vote" en faveur d'une classe. Les votes n'ont pas tous la même importance : la couche leur accorde des poids qui dépendent de l'élément du tableau et de la classe. Pour calculer les probabilités, la couche fully-connection multiplie donc chaque élément en entrée par un poids, fait la somme, puis applique une fonction d'activation (logistique si N=2N=2, softmax si N>2N>2) :

Ce traitement revient à multiplier le vecteur en entrée par la matrice contenant les poids. Le fait que chaque valeur en entrée soit connectée avec toutes les valeurs en sortie explique le terme fully-connection.

[1] Rectified Linear Units

Le réseau de neurones convolutif apprend les valeurs des poids de la même manière qu'il apprend les filtres de la couche de convolution : lors de phase d'entraînement, par rétropropagation du gradient. La couche fully-connected détermine le lien entre la position des features dans l'image et une classe. En effet, le tableau en entrée étant le résultat de la couche précédente, il correspond à une carte d'activation pour une feature donnée : les valeurs élevées indiquent la localisation (plus ou moins précise selon le pooling) de cette feature dans l'image. Si la localisation d'une feature à un certain endroit de l'image est caractéristique d'une certaine classe, alors on accorde un poids important à la valeur correspondante dans le tableau.

Enfin, On résume l'architecture de CNN par cette figure ci-dessus :

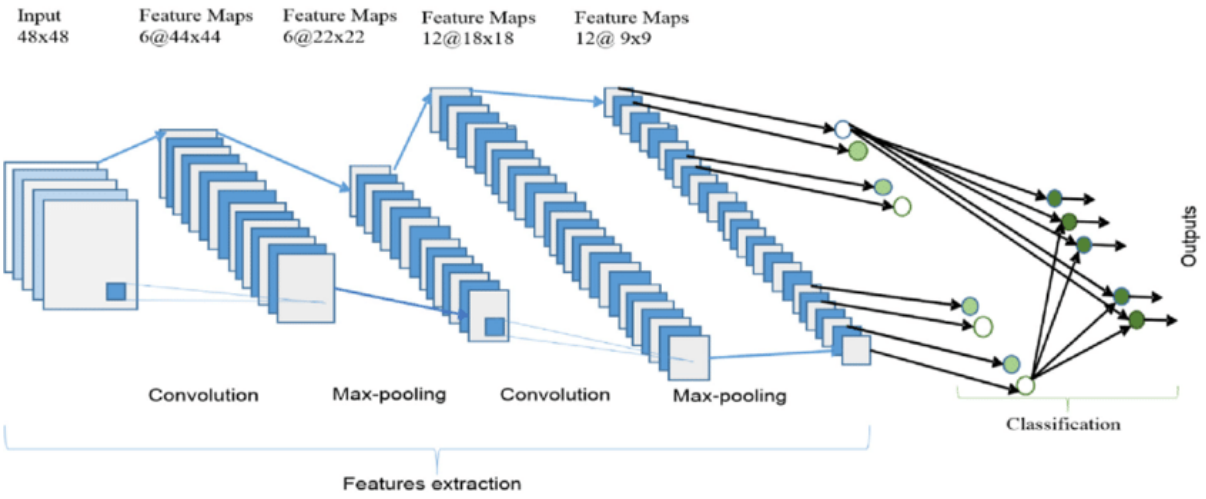

Figure 6 : Architecture de CNN

Projet 1 :
Handwritten Recognition Digits

IV. Introduction :

Une démonstration populaire de la capacité des techniques d'apprentissage en profondeur est la reconnaissance d'objets dans données d'image. Le" Hello world " de la reconnaissance d'objets pour l'apprentissage automatique et l'apprentissage en profondeur est le MNIST dataset pour le chiffre Manuscrit recognition., vous découvrirez comment développer un modèle d'apprentissage profond pour obtenir des performances de pointe sur la tâche de reconnaissance des chiffres manuscrits MNIST en Python à l'aide de la bibliothèque D'apprentissage Keras deep.

V. Objectifs :

Le but de ce projet est de concevoir un réseau de neurone de convolution afin de classifier une image. Les premiers tests se feront sur la base de données MNIST qui comporte des images manuscrites de chiffres. Une fois que le réseau a appris, on lui soumet une image d'un chiffre qu'il n'a pas vu lors de son apprentissage, et il doit produire en sortie le chiffre reconnu.

VI. La base de données MNIST :

MNIST est un ensemble de données développé par *Yann LeCun, Corinna Cortes* et *Christopher Burges* pour évaluer les modèles d'apprentissage automatique sur le problème de classification des chiffres manuscrits.

L'ensemble de données a été construit à partir d'un certain nombre de documents numérisés disponibles auprès de *l'Institut national des normes et de la technologie (NIST)*. C'est d'où vient le nom de l'ensemble de données, en tant qu'ensemble de données NIST ou MNIST modifié.

Les Images de chiffres ont été prises à partir d'une variété de documents numérisés, normalisés en taille et centré. Cela en fait un excellent ensemble de données pour évaluer les modèles,

permettant au développeur de se concentrer sur l'apprentissage automatique avec très peu de nettoyage ou de préparation des données nécessaires.

Chaque image est un carré de 28 x 28 pixels (784 pixels au total). Une broche standard de l'ensemble de données est utilisée pour évaluer et comparer des modèles, où 60 000 images sont utilisées pour entraîner un modèle et un ensemble séparé de 10 000 images sont utilisées pour tester.

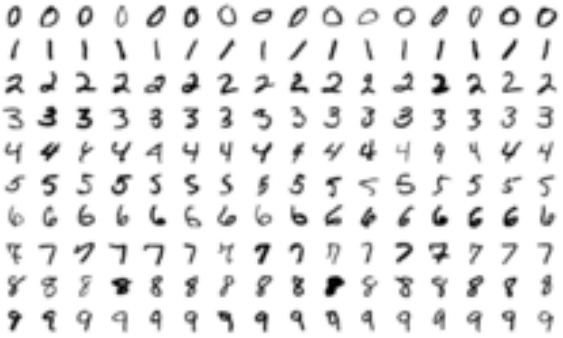

Figure 7 : Base de données MNIST

VII. Modèle Utilisé :

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

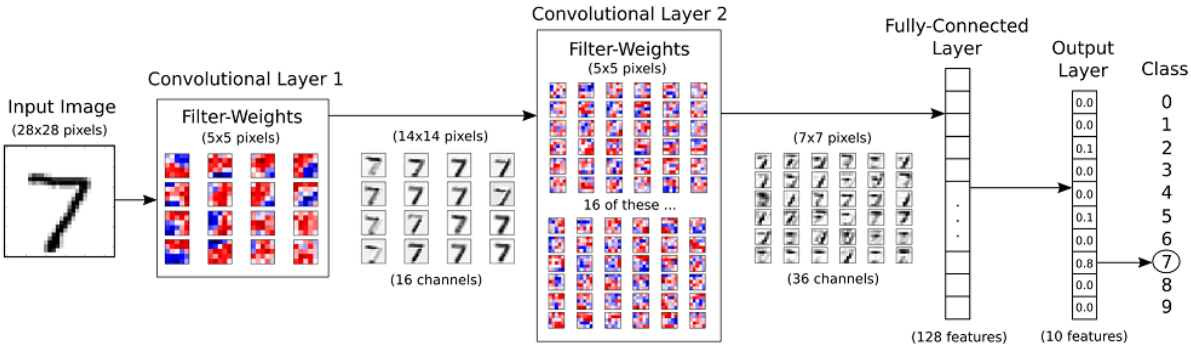

Figure 8 : Représentation de modèle CNN

VIII. Etude technique :

4. Environnement de travail :

Anaconda est une distribution libre et open source des langages de programmation Python et R appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique, qui vise à simplifier la gestion des paquets et de déploiement

Le notebook Jupyter est un environnement HTML pour Python, Il fournit un environnement organisé en cellules interactives qui peuvent être exécutées, permettant l'organisation et la documentation de calculs de façon structurée.

5. Framework et librairies utilisé :

TensorFlow est un outil open source d'apprentissage automatique développé par Google. Le code source a été ouvert le 9 novembre 2015 par Google et publié sous licence Apache. Il est fondé sur l'infrastructure DistBelief, initiée par Google en 2011, et est doté d'une interface pour Python, Julia et R

La bibliothèque Keras permet d'interagir avec les algorithmes de réseau de neurones profonds et d'apprentissage automatique, Conçue pour permettre une expérimentation rapide avec les réseaux de neurones profonds, elle se concentre sur son ergonomie, sa modularité et ses capacités d'extension. Elle a été développée dans le cadre du projet ONEIROS[2]

[2] Open ended neuro electr

OpenCV est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage et la société ItSeez se sont succédé au support de cette bibliothèque.

Tkinter est la bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. Elle vient d'une adaptation de la bibliothèque graphique Tk écrite pour Tcl

IX. Application : Handwritten Recognition GUI App :

Afin de construire une application de reconnaissance de chiffres en utilisant le deep learning par concevoir un réseau de neurone afin de classifier les images des chiffres.

Dans cette partie, on va suivre ses étapes afin d'implémenter une application qui consiste à reconnaitre le chiffre (ou les chiffres) soumet par un utilisateur et de les classifier par suite.

Pour ce projet, on' a utilisé le langage python, *OpenCV*, Deep learning avec *Keras*, et la librairie *Tkinter* pour la partie GUI de l'application

1. Importation des librairies

Premièrement, on doit importer toutes les bibliothèques nécessaires pour construire le model et entrainer le modèle.

```
In [1]: #import the librairies

        import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Dropout
        from keras.layers import Flatten
        from keras.layers.convolutional import Convolution2D
        from keras.layers.convolutional import MaxPooling2D
        from keras.utils import to_categorical
        from keras import backend as k
        import numpy as np
        import pandas as pd
```

Figure 9 : code d'importation des librairies

2. Chargement de la base de données MNIST

Pour charger le *MNIST Dataset*, Keras fournit une méthode pratique pour charger L'ensemble de données MNIST.

Nous pouvons donc facilement La base de données MNIST, et pour ce faire en appelant tout simplement la fonction *mnist.load_data()*

```
In [2]: # load mnist data from keras directly

        (X_train, Y_train) , (X_test, Y_test) = mnist.load_data()
```

Figure 10 : code de chargement de MNIST Dataset

Après le chargement de *dataset*, et avant de l'utiliser, on peut visualiser un échantillon en utilisant le code suivant :

```
In [3]: # visualise some mnist digits data

        import matplotlib.pyplot as plt

        for i in range(6):
            plt.subplot(int('23'+str(i+1)))
            plt.imshow(X_train[i],cmap=plt.get_cmap('hot'))
```

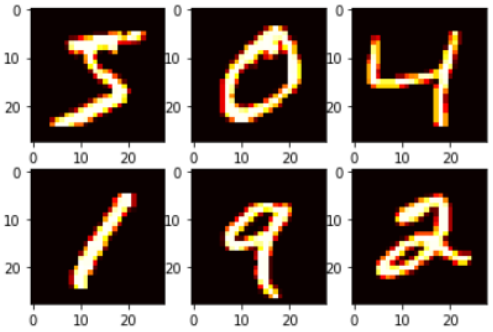

Figure 11 : code de visualisation d'un échantillon de MNIST Dataset

3. Preprocess et normalisation des données

Nous savons que toutes les images de *MNIST Dataset* sont représentées sous la forme d'une matrice ***28×28*** contenants des valeurs de pixels en niveaux de gris. Selon cela, la dimension des données d'entraînement est ***(60000,28,28)*** mais le modèle *CNN* nécessitera une dimension supplémentaire, nous devons donc traiter les données en remodelant la matrice en forme ***(60000,28,28,1),*** Et pour ce faire nous utilisons le bloc de code suivant :

```
In [4]: #reshape format [Samples][width][height][channels]

        X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float')
        X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float')

        # Converts a class vectors (integers) to binary class matrix

        Y_train = to_categorical(Y_train)
        Y_test = to_categorical(Y_test)
```

Figure 12 : code de Preprocessing Mnist Dataset

Puisque les valeurs de pixels sont une échelle de gris comprise entre ***0*** et ***255***, et qu'on est lors de l'utilisation de ***CNN***, il sera bien d'effectuer une mise à l'échelle des valeurs d'entrée pour normaliser les valeurs de pixels à la plage ***0*** et ***1*** en divisant chaque valeur par la valeur maximale qui est ***255***. Et le code suivant montre l'étape de normalisation :

```
In [5]: # normalize inputs

        X_train = X_train / 255
        X_test = X_test / 255
```

Figure 13 : Normalisation des données de MNIST

4. Création de modèle :

Nous allons maintenant créer un modèle ***CNN*** avec une couche convolutionnelle double de la même taille ***3×3***, des couches de mise en commun max et des couches entièrement connectées. La couche de décrochage est utilisée pour désactiver certains neurones afin de réduire le surajustement.

Enfin, la couche de sortie a 10 neurones pour les 10 classes. Nous allons ensuite compiler le modèle avec L'optimiseur ADAM[3].

Et voici ci-dessous la figure qui montre la fonction qui crée le modèle :

```
In [6]: # defining a cnn model

def model():
    num_classes = 10
    model = Sequential()
    model.add(Convolution2D(32,kernel_size=(3,3),activation='relu'))
    model.add(Convolution2D(64,(3,3),activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(256,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes,activation='softmax'))
    model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
    return model

# build the model

model = model()
```

Figure 14 : Code de création de modèle

5. Entrainement de modèle

Après la création de modèle, il suffit maintenant de lui sommet les données et lui laisser un temps pour apprendre et entrainer.

La fonction ***model.fit()*** de Keras prend les données d'entraînement, les données de validation, les époques et la taille du lot pour entraîner le modèle. Quand le modèle termine le processus d'entrainement, nous allons enregistrer le modèle dans un fichier nommé ***model.h5*** qu'on va l'utiliser dans la partie GUI de l'application, et ci-dessous le code utiliser dans cette partie :

[3] Un algorithme d'optimisation de remplacement pour la descente de gradient stochastique pour entrainement des modèles de Deep Learning.

```
In [7]: # fit the model

        model.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=10,batch_size=200, verbose=2)
        print("================================================\n")
        print("the model has successfully trained")

        model.save('model.h5')

        print("================================================\n")
        print("the model has successfully saved")

        Epoch 1/10
        300/300 - 107s - loss: 0.2249 - accuracy: 0.9306 - val_loss: 0.0520 - val_accuracy: 0.9835
        Epoch 2/10
        300/300 - 110s - loss: 0.0714 - accuracy: 0.9782 - val_loss: 0.0386 - val_accuracy: 0.9875
        Epoch 3/10
        300/300 - 108s - loss: 0.0507 - accuracy: 0.9848 - val_loss: 0.0329 - val_accuracy: 0.9886
        Epoch 4/10
        300/300 - 106s - loss: 0.0391 - accuracy: 0.9874 - val_loss: 0.0298 - val_accuracy: 0.9899
        Epoch 5/10
        300/300 - 109s - loss: 0.0332 - accuracy: 0.9893 - val_loss: 0.0252 - val_accuracy: 0.9905
        Epoch 6/10
        300/300 - 106s - loss: 0.0290 - accuracy: 0.9905 - val_loss: 0.0275 - val_accuracy: 0.9908
        Epoch 7/10
        300/300 - 106s - loss: 0.0253 - accuracy: 0.9919 - val_loss: 0.0264 - val_accuracy: 0.9919
        Epoch 8/10
        300/300 - 105s - loss: 0.0209 - accuracy: 0.9930 - val_loss: 0.0232 - val_accuracy: 0.9926
        Epoch 9/10
        300/300 - 107s - loss: 0.0194 - accuracy: 0.9931 - val_loss: 0.0239 - val_accuracy: 0.9922
        Epoch 10/10
        300/300 - 107s - loss: 0.0169 - accuracy: 0.9945 - val_loss: 0.0319 - val_accuracy: 0.9910
        the model has successfully trained
        the model has successfully saved
```

Figure 15 : Code d'entrainement de modèle

6. Evaluation de modèle :

Pour évaluer les performances de notre modèle, nous disposons de 10 000 images de test de chiffres manuscrits. L'ensemble de données MNIST est bien équilibré afin que nous puissions obtenir une précision de 99% (c'est-à-dire une erreur CNN < 1%).

```
In [1]: #Evaluate the model

        scores = model.evaluate(X_test,Y_test, verbose = 0)
        print('================================================\n')
        print("CNN ERROR : %.2f%% " % (100-scores[1]*100))

        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        <ipython-input-1-5691af9533c9> in <module>
              1 #Evaluate the model
              2
        ----> 3 scores = model.evaluate(X_test,Y_test, verbose = 0)
```

Figure 16 : Evaluation de modèle

7. Interface Graphique de l'application

Enfin, il est temps de créer une application GUI en utilisant Tkinter. Et pour se faire nous allons créer un nouveau fichier python pour construire cette interface graphique.

Il existe deux méthodes principales :

- Tk()
- mainloop()

Tk() est la méthode que nous utilisons pour créer la fenêtre principale de l'application et mainloop() est une méthode de la fenêtre principale qui nous exécuter lorsque l'on veut exécuter notre application.

7. 1 Importation des librairies :

Première chose à faire c'est d'importer tous les packages et les bibliothèques qui vont nous aider à construire l'interface graphique de l'application, et ci-dessous le code qui concerne la partie d'importation

```
In [1]: #import libraries

        import os
        import PIL
        import cv2
        import glob
        import numpy as np
        from tkinter import *
        from PIL import Image, ImageDraw , ImageGrab
```

Figure 17 : code d'importation des librairies

7. 2 Chargement de modèle :

Ici, nous chargeons le modèle déjà construit et enregistré précédemment en fournissant le chemin où le modèle a été enregistré.

```
In [1]: # load model

        from keras.models import load_model
        model = load_model(r'C:\Users\elbai\Documents\1-Projects\Untitled Folder\mnist.h5')
        print("=====================================\n")
        print('The model is loaded successfully')

        =====================================

        The model is loaded successfully
```

Figure 18 : Code de chargement de modèle

7. 3 Création de l'interface :

Après l'importation des librairies et le chargement de modèle, on est maintenant sensé que de créer l'interface graphique de l'application.

En premier lieu, nous créons la fenêtre racine, qui est la fenêtre principale de notre application, qui ça se fait par ce code suivant :

```
In [ ]: # Create a main window first (named as root)
root = Tk()
root.resizable(0,0)
root.title("Handwritten Digit Recognition GUI APP")

# Initialize few Variables

lastx , lasty = None, None
image_number = 0

# Create a canvas for drawing

cv = Canvas(root , width=640, height = 480, bg='white')
cv.grid(row = 0, column = 0, pady=2, sticky = W, columnspan=2)



cv.bind('<Button-1>' , activate_event)

# Add Buttons and Labels


btn_save = Button(text="Recognize digit", command = Recognize_Digit)
btn_save.grid(row=2, column=0 , pady=1, padx=1)
btn_clear = Button(text="Clear widget", command = clear_widget)
btn_clear.grid(row=2, column=1 , pady=1, padx=1)

# Mainloop() is used when your application is ready to run

root.mainloop()
```

Figure 19 : code de création de l'interface graphique 'GUI'

7. 4 Les fonctions utilisées dans la partie interface :

Dans cette partie nous avons utilisé 3 fonctions principales qui vont nous aider à dessiner le chiffre à travers l'interface graphique.

- ***clear_widget ()*** est utilisée pour effacer le canevas, nous utiliserons la méthode delete() pour effacer les chiffres dessinés précédemment sur le canevas afin que nous puissions dessiner une autre fois.
- ***activate_event()*** Les événements peuvent provenir de diverses sources, y compris les pressions sur les touches et les opérations de la souris par l'utilisateur. Pour chaque widget, vous pouvez lier des fonctions et des méthodes Python à des événements
- ***draw_lines()*** est utilisée pour tracer une ligne sur le canevas.

Et ci-dessous le bloc de code de chaque fonction :

```
In [ ]: def clear_widget():
            global cv

            # To clear a canvas
            cv.delete("all")

        def activate_event(event):
            global lastx, lasty

            # <B1-Motion>

            cv.bind('<B1-Motion>', draw_lines)
            lastx, lasty = event.x, event.y

        def draw_lines(event):
            global lastx, lasty
            x, y = event.x, event.y

            #do the canva drawings

            cv.create_line((lastx, lasty, x, y), width=8 , fill='black',
                           capstyle=ROUND,smooth=TRUE, splinesteps=12)

            lastx, lasty = x, y
```

Figure 20 : code des fonctions utilisé dans l'interface

7. 5 La fonction de reconnaissance de chiffre :

Dans cette section, Nous allons utiliser le module ImageGrab pour copier le contenu de l'écran ou du presse-papiers dans une mémoire D'image Pil[4] , il prend un instantané de l'écran.

Après avoir pris l'instantané d'écran, nous utiliserons une méthode de recadrage qui prend quatre coordonnées en entrée et renvoie une région rectangulaire de l'image (qui est un instantané dans

le cas), puis nous enregistrerons l'image sous le nom de fichier donné au format png. Et ci-dessous le bloc de code utilisé :

```
In [ ]: def Recognize_Digit():

            global image_number
            prediction = []
            percentage = []
            #image_number = 0

            filename = f'image_{image_number}.png'
            widget = cv

            # get the widget coordinates

            x = root.winfo_rootx()+widget.winfo_x()
            y = root.winfo_rooty()+widget.winfo_y()

            x1 = x + widget.winfo_width()
            y1 = y + widget.winfo_height()

            # grab the image, crop it acoording to my requirement and saved it in png format
            ImageGrab.grab().crop((x,y,x1,y1)).save(filename)
```

Figure 21 : Partie 1 de la fonction Recognize_digit()

[4] Python Imaging Library

Nous allons utiliser OpenCV pour trouver les contours d'une image que nous avons enregistrée précédemment. En utilisant le bloc de code suivant :

```
# read the image in color format
image = cv2.imread(filename, cv2.IMREAD_COLOR)
# Convert  the image in grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# applying Otsu thresholding
ret, th = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# findContours () function helps in extracting the countours from the image
countours = cv2.findContours(th , cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
```

Figure 22 : Partie 2 de la fonction Recognize_digit()

Maintenant, nous devons redimensionner, remodeler et normaliser l'image pour prendre en charge notre entrée de modèle. Et par suite exécuter la méthode ***modèle.predict()*** pour reconnaître le chiffre manuscrit et dessiner la zone de délimitation entourant chaque chiffre présent dans l'image avec la valeur et le pourcentage prédits. Et ci-dessous le code utilisé :

```
countours = cv2.findContours(th , cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]

for cnt in countours:
    # GET Bounding box and extract ROI
    x,y,w,h = cv2.boundingRect(cnt)

    # Create rectangle
    cv2.rectangle(image, (x,y) , (x+w, y+h), (255,0,0), 1)
    top = int(0.05 * th.shape[0])
    bottom = top
    left = int(0.05 * th.shape[1])
    right = left
    th_up = cv2.copyMakeBorder(th, top, bottom, left, right, cv2.BORDER_REPLICATE)

    # Extract the image ROI
    roi = th[y-top:y+h+bottom, x-left:x+w+right]

    # resize roi image to 28*28 pixels
    img = cv2.resize(roi, (28,28), interpolation = cv2.INTER_AREA)

    # reshaping the image to support our model
    img = img.reshape(1,28,28,1)

    # Normalizing the image to support our model
    img = img / 255.0

    # it's time to predict the result

    pred = model.predict([img])[0]

    # numpy.argmax(input array) returns the indices of the maximum values
    final_pred = np.argmax(pred)
    data = str(final_pred) + ' ' + str(int(max(pred)*100)) + '%'

    # cv2.putText() method is used to draw  text string on image
    font = cv2.FONT_HERSHEY_SIMPLEX
    fontScale = 0.5
    color = (255, 0, 0)
    thickness = 1
    cv2.putText(image, data, (x,y-5), font, fontScale, color, thickness)

# showing the predicted results on new window
cv2.imshow('image', image)
cv2.waitKey(0)
```

Figure 23 : Code de fonction de reconnaissance de chiffre

X. Résultats :

1. Chiffres dessinés à travers l'interface :

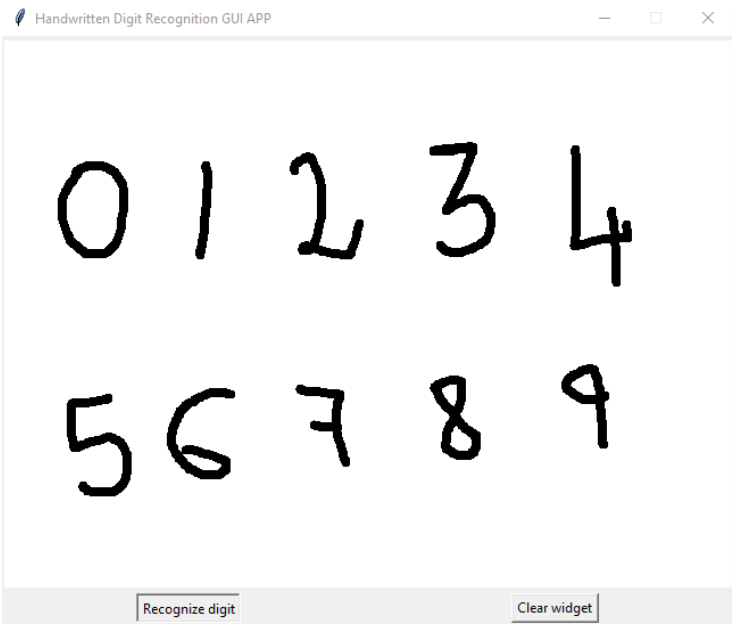

Figure 24 : Test par des chifres dessinés par utilisateur

2. Résultats de la prédiction :

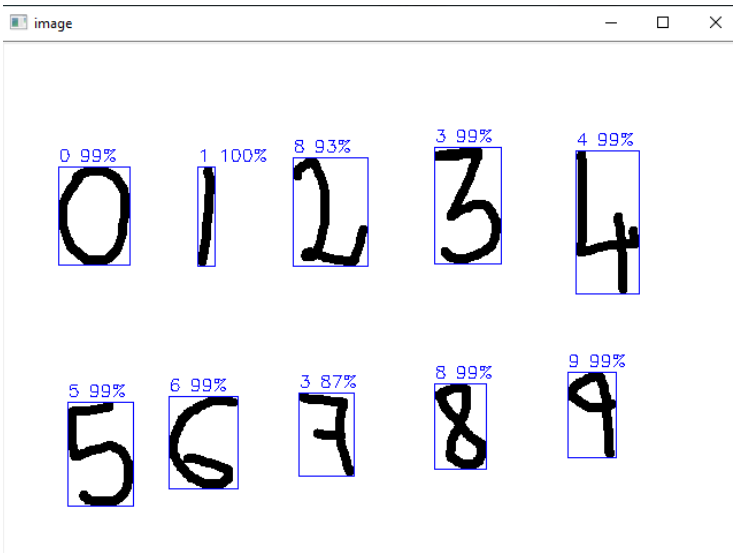

Figure 25 : Résultats de la prédiction

Projet 2 :

Reconnaissance de texte : agent conversationnel " Chatbot "

I. Introduction :

Dans le monde connecté d'aujourd'hui, les gens utilisent plusieurs technologies pour communiquer. Par exemple :

- Les appels vocaux
- Les services de messagerie
- Les applications de conversation en ligne
- Les e-mails
- Les réseaux sociaux
- Les outils collaboratifs en milieu professionnel

Nous sommes à présent tellement habitués à cette connectivité omniprésente que nous nous attendons à ce que les organisations avec lesquelles nous travaillons soient facilement joignables et immédiatement réactives via les canaux que nous utilisons déjà. De plus, nous nous attendons à ce que ces organisations s'engagent individuellement avec nous et soient en mesure de répondre à des questions complexes à un niveau personnel.

De nombreuses organisations publient des informations de support et des réponses aux questions fréquemment posées (FAQ) accessibles via un navigateur web ou une application dédiée. La complexité des systèmes et des services qu'ils offrent signifie que les réponses à des questions spécifiques sont difficiles à trouver. Souvent, le personnel de support de ces organisations est submergé par des demandes de support qui leur arrivent sous forme d'appels téléphoniques, d'e-mails, de SMS, de messages sur les réseaux sociaux et d'autres canaux.

De plus en plus, les organisations se tournent vers des solutions d'intelligence artificielle (IA) qui utilisent des agents d'IA, communément appelés bots, pour fournir une première ligne de support automatisé via la gamme complète de canaux que nous utilisons pour communiquer. Les bots sont conçus pour interagir avec les utilisateurs de manière conversationnelle, comme illustré dans cet exemple d'interface de conversation instantanée :

Un Chatbot est un programme informatique qui simule une conversation avec une personne alors qu'il s'agit d'intelligence artificielle.

Un des premiers chatbots sur internet est celui d'Ikea :

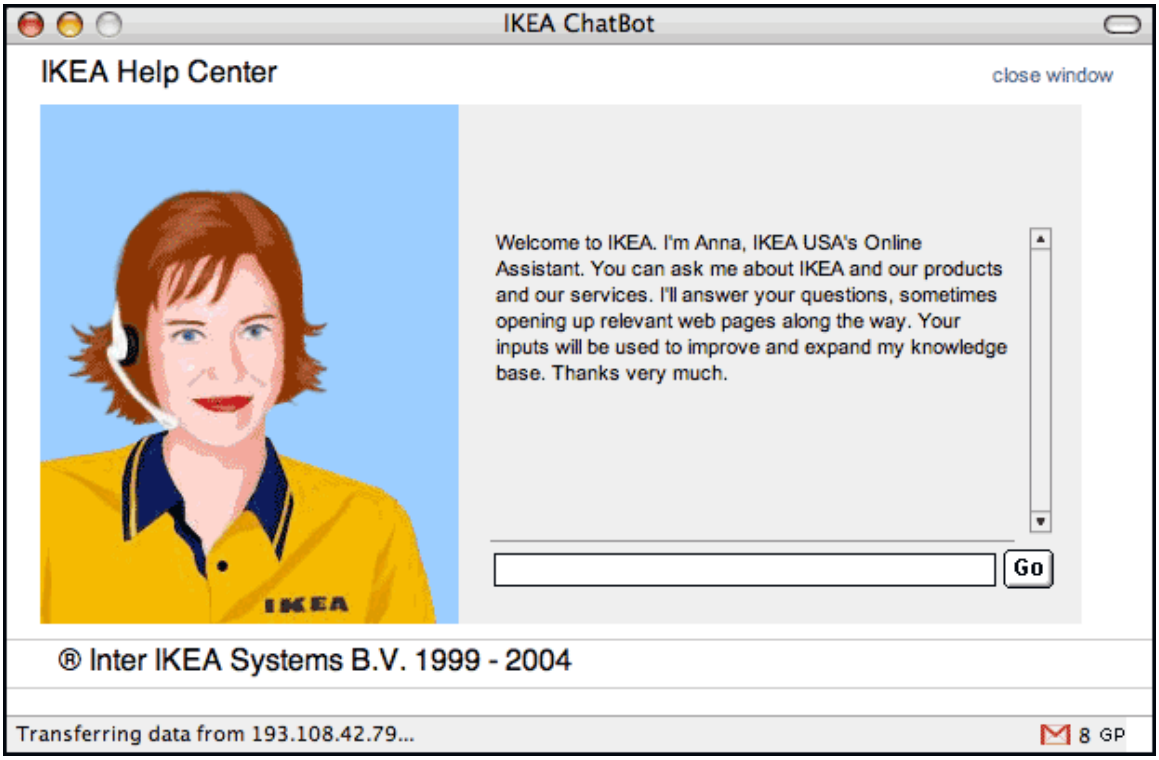

Figure 26 : Ikea Chatbot, Premier chatbot introduit dans le monde

II. Objectifs :

Le but de ce projet est de concevoir un chatbot qui donne des informations sur ENSA BERRECHID.

III. Bibliothèques utilisées :

NLTK : Natural Language Toolkit est une bibliothèque logiciel en Python permettant un traitement automatique des langues, développée par *Steven Bird* et *Edward Loper* du département d'informatique de l'université de Pennsylvanie. En plus de la bibliothèque, NLTK fournit des démonstrations graphiques, des données-échantillon, des tutoriels, ainsi que la documentation de l'interface de programmation API.

Dans ce projet on va utiliser trois fonctions majeures de la bibliothèque NLTK :

- Tokenize : il sert diviser la phrase en tableau de mots / jetons un jeton peut être un mot, un caractère de ponctuation ou nombre.

Figure 27 : tokenisation d’une phrase

- Stem : cette fonction sert à trouver la racine du mot.

	words	stemmed words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

Figure 28 : trouver la racine des mots

- Bag_of_words : retourner un tableau de bag of words :

sentence = ["hello", "how", "are", "you"]

words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]

bag = [0, 1, 0, 1, 0, 0, 0]

Pytorch : est une bibliothèque logicielle Python open source d’apprentissage machine qui s'appuie sur torch développé par Facebook.

PyTorch permet d'effectuer les calculs tensoriels nécessaires notamment pour l'apprentissage profond (deep learning). Ces calculs sont optimisés et effectués soit par le processeur (CPU) soit, lorsque c'est possible, par un processeur graphique (GPU) supportant CUDA. Il est issu des équipes de recherche de Facebook, et avant cela de Ronan Collobert dans l'équipe de Samy Bengio à l’IDIAP, Pytorch permet de :

Manipuler des tenseurs (tableaux multidimensionnels), de les échanger facilement avec Numpy et d'effectuer des calculs efficaces sur CPU ou GPU (par exemple, des produites de matrices ou des convolutions).

Calculer des gradients pour appliquer facilement des algorithmes d'optimisation par descente de gradient. PyTorch utilise la bibliothèque autograd.

Tkinter est la bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. Elle vient d'une adaptation de la bibliothèque graphique Tk écrite pour Tcl

IV. Modèle utilisé :

Le modèle utilisé est un réseau de neurones (Neural Network : NN) pouvant être représenté comme ceci :

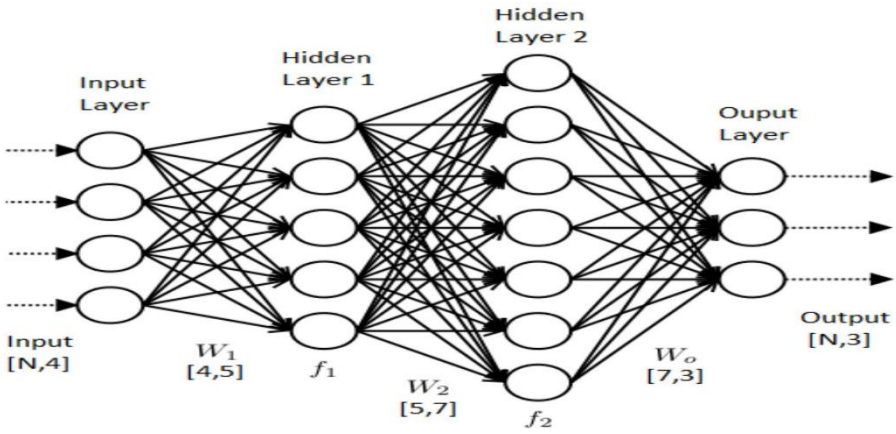

Figure 29 : Modèle utilisé pour la création du Chatbot

V. Application : Conversational agent (chatbot gui app) :

Afin de construire une application de reconnaissance du texte en utilisant le Deep Learning par concevoir un réseau de neurone afin de donner des réponses correspondant à ce texte.

Dans cette partie, on va suivre ses étapes afin d'implémenter une application qui consiste à reconnaitre le texte soumit par un utilisateur et de le répondre par la réponse qui convient.

Pour ce projet, on' a utilisé le langage python, nltk, Deep learning avec pytorch, et la librairie Tkinter pour la partie GUI de l'application.

1. Importation des librairies :

Premièrement, on doit importer toutes les bibliothèques nécessaires pour construire le modèle et l'entrainer.

```python
import numpy as np
import random
import json
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from nltk_utils import bag_of_words, tokenize, stem
from model import NeuralNet
```

Figure 30 : importation des librairies nécessaire pour la création et l'entrainement du modèle

2. Chargement des données à travers le fichier intents.json :

Pour charger le fichier intents.json la bibliothèque json offrit la fonction open('intents.json')

```python
with open('intents.json', 'r') as f:
    intents = json.load(f)
```

Figure 31 : ouvrir le fichier json qui contient les donnés

Puis on fait le parcours de chaque 'sentence' dans notre intents 'patterns'.

```python
all_words = []
tags = []
xy = []
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
    tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
        all_words.extend(w)
        # add to xy pair
        xy.append((w, tag))
```

Figure 32 : importation du fichier intents.json

Maintenant on fait le 'stem' de chaque mot et on rend tous les mots en minuscules

```
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
```

Figure 33 : élimination des mots duplices

On supprime les mots dupliqués et on les trie.

```
all_words = sorted(set(all_words))
tags = sorted(set(tags))
```

Figure 34 : trier les mots par ordre alphabétique

On crée les données d'entraînement

```
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    # y: PyTorch CrossEntropyLoss needs only class labels, not one-hot
    label = tags.index(tag)
    y_train.append(label)

X_train = np.array(X_train)
y_train = np.array(y_train)
```

Figure 35 : création des donnes d'entrainement

3. Création d'une Dataset :

On définit d'abord des hyper paramètres :

```
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)
```

Figure 36 : définition des hyperparamètres

- Num_epochs : passage en avant et en arriére de tous les échatillions d'apprentissage.
- Batch_size : nombre d'échantillions d'apprentissage en passe avant et arriére .

On crée maintenant chatdataset :

```
class ChatDataset(Dataset):
    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train
    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]
    # we can call len(dataset) to return the size
    def __len__(self):
        return self.n_samples
dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)
```

Figure 37 : création d'une dataset

1. Création du modèle :

Nous allons maintenant créer un modèle NN avec 3 couche linéaire.

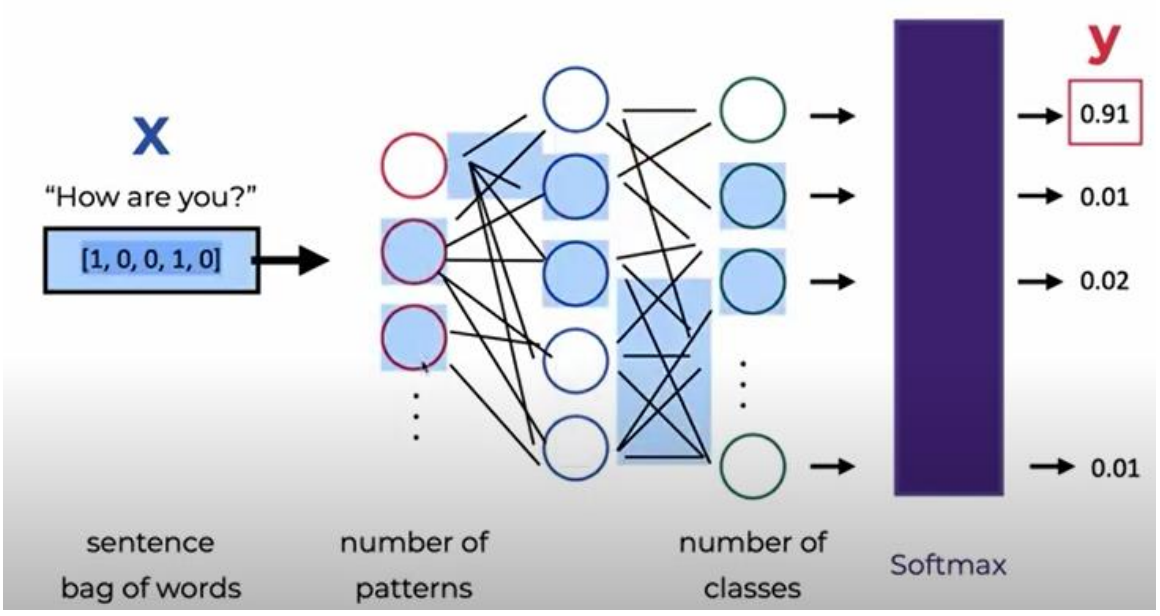

Figure 38 : réseau de neurone

On a 3 trois couche linéaire la premiére couche a comme input le nombres des différents patterns et comme output la taille de output_size qu'on a choisit,la deuxieme couche est une couche cachée qui comme input et ouput aussi la taille de output_size et puis la troisieme couche qui est une couche entierement connécté qui a comme input hidden_size et comme output la taille des différents tags.

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

Figure 39 : création du modèle

Nous allons ensuite compiler le modèle avec L'optimiseur ADAM.

2. Entrainement de modèle :

Après la création de modèle, il suffit maintenant de lui sommet les données et lui laisser un temps pour apprendre et entrainer.

La fonction dataloader() de torch.utils.data qui prend le dataset qu'on a créé dans la partie (3) et le nombres d'échantillons d'apprentissage (batch_size) pour entrainer le modèle.

Quand le modèle termine le processus d'entrainement, nous allons enregistrer le modèle dans un fichier nommé data.pth qu'on va l'utiliser dans la partie GUI de l'application, et ci-dessous le code utiliser dans cette partie.

```
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        # Forward pass
        outputs = model(words)
        # if y would be one-hot, we must apply
        # labels = torch.max(labels, 1)[1]
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 100 == 0:
        print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')


print(f'final loss: {loss.item():.4f}')
```

```
Epoch [100/1000], Loss: 1.0481
Epoch [200/1000], Loss: 0.1215
Epoch [300/1000], Loss: 0.0027
Epoch [400/1000], Loss: 0.0032
Epoch [500/1000], Loss: 0.0015
Epoch [600/1000], Loss: 0.0003
Epoch [700/1000], Loss: 0.0003
Epoch [800/1000], Loss: 0.0001
Epoch [900/1000], Loss: 0.0001
Epoch [1000/1000], Loss: 0.0001
final loss: 0.0001
training complete. file saved to data.pth
```

Figure 32 : entrainement du modèle

3. Interface graphique de l'application :

Enfin, il est temps de créer une application GUI en utilisant Tkinter. Et pour se faire nous allonscréer un nouveau fichier python pour construire cette interface graphique.

Il existe deux méthodes principales :

- Tk()
- mainloop()

Tk() est la méthode que nous utilisons pour créer la fenêtre principale de l'application et mainloop() est une méthode de la fenêtre principale qui nous exécuter lorsque l'on veut exécuter notre application.

Nous allons créer aussi deux frames principales le premier frame de welcoming et la deuxième Frame sera le frame ou le chat et l'interaction avec l'utilisateur se déroulera.

1. Importation des librairies :

Première chose à faire c'est d'importer tous les packages et les bibliothèques qui vont nous aider à construire l'interface graphique de l'application, et ci-dessous le code qui concerne la partie d'importation.

```
from tkinter import *
from tkinter import ttk
import time
import random
import json
import torch
from model import NeuralNet
from nltk_utils import bag_of_words, tokenize
```

Figure 33 : importation des librairies nécessaire pour la réalisation interface graphique

Chargement du modèle :

Ici, nous chargeons le modèle déjà construit et enregistré précédemment en fournissant lechemin où le modèle a été enregistré.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
with open('intents.json', 'r') as json_data:
    intents = json.load(json_data)
FILE = "data.pth"
data = torch.load(FILE)
input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
all_words = data['all_words']
tags = data['tags']
model_state = data["model_state"]
model = NeuralNet(input_size, hidden_size, output_size).to(device)
model.load_state_dict(model_state)
```

Figure 34 : importation des donnes de l'entrainement à partir du fichier data.pth

Création du l'interface :

Après l'importation des librairies et le chargement de modèle, on est maintenant sensé que de créer l'interface graphique de l'application.

En premier lieu, nous créons la fenêtre racine, qui est la fenêtre principale de notre application, et deux frames principales le premier frame sert à la salutation de l'utilisateur et l'autre frame sera responsable de la partie chatbot et interaction avec l'utilisateur.

```
root = Tk()
"""  images used in window  """
back = PhotoImage(file='arrow_behind.png')
front = PhotoImage(file='arrow_ahead.png')
exitt = PhotoImage(file='exit.png')
screen_1 = PhotoImage(file='image_5.png')
submit_img = PhotoImage(file='image_8.png')
"""     WELCOME FRAME    """
"""    first frame containing time date and welcome messages """
frame_welcome = Frame(root, bg=c1, height='670', width='550')
frame_welcome.pack_propagate(0)
frame_welcome.pack()
welcome = Label(frame_welcome, text='Welcome', font="Vardana 40 bold", bg=c1, fg="white")
welcome.place(x=160, y=200)
welcome_chatbot = Label(frame_welcome, text='I am Ensa Berrechid Chatbot ! ', font="Helvetica 15 bold italic", bg=c1, fg=c6)
welcome_chatbot.place(x=140, y=270)
pic_1 = Label(frame_welcome, image=screen_1)
pic_1.place(x=-2, y=357)
button_front = Button(frame_welcome, image=front, relief="flat", bg=c1, bd="3px solid black",
                      command=welcome_to_chat).place(x=470, y=10)
```

```
"""         CHAT FRAME    """
"""""       main chat screen    """
frame_chat = Frame(root, bg=c1, height='670', width='550')
frame_chat.pack_propagate(0)
frame_top = Frame(frame_chat, bg=c3, height='100', width='550')
frame_top.pack()
label_topic = Label(frame_top, bg=c3, fg='white', font='Verdana 20 bold ')
label_topic.pack(pady='40')
frame_spacer = Frame(frame_top, bg=c2, height="10", width="550")
frame_spacer.pack()
bottom_frame = Frame(frame_chat, bg=c2, height='100', width='550')
bottom_frame.pack_propagate(0)
bottom_frame.pack(side=BOTTOM)
button = Button(bottom_frame, image=submit_img, relief="flat", font='Vardana 10 bold', bg=c3,command=submit )
button.place(x=410, y=27)
entry = Text(bottom_frame, bg='white', fg=c6, height='5', width='45', font='Verdana 10')
entry.bind('<Return>')
entry.place(x=30, y=10)
frame_chats = Frame(frame_chat, bg=c1, height='450', width='500')
frame_chats.pack_propagate(0)
frame_chats.pack()
label_space = Label(frame_chats, bg=c1).pack()
button_refresh = Button(frame_chat, bg=c3, fg=c2, text='refresh', font='Vardana 10 bold',command=refresh_screen)
button_refresh.place(x=440, y=80)
button_back = Button(frame_chat, image=back, relief="flat", bg=c3, command=chat_to_welcome).place(x=10, y=10)
button_front = Button(frame_chat, image=exitt, relief="flat", bg=c3, command=root.destroy).place(x=440, y=10)
# ------------------------------------------------------------------------------------------------------------
root.mainloop()
```

Figure 35 : création de l'interface graphique

La fenêtre de salutation de l'utilisateur est comme ceci :

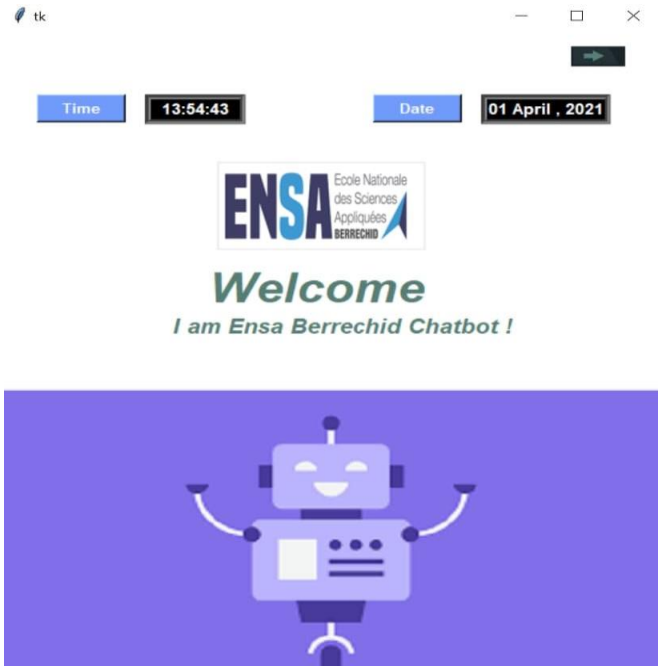

Figure 36 : la fenêtre principale de l'interface

La fenêtre de chat principale est représentée comme ci-dessous :

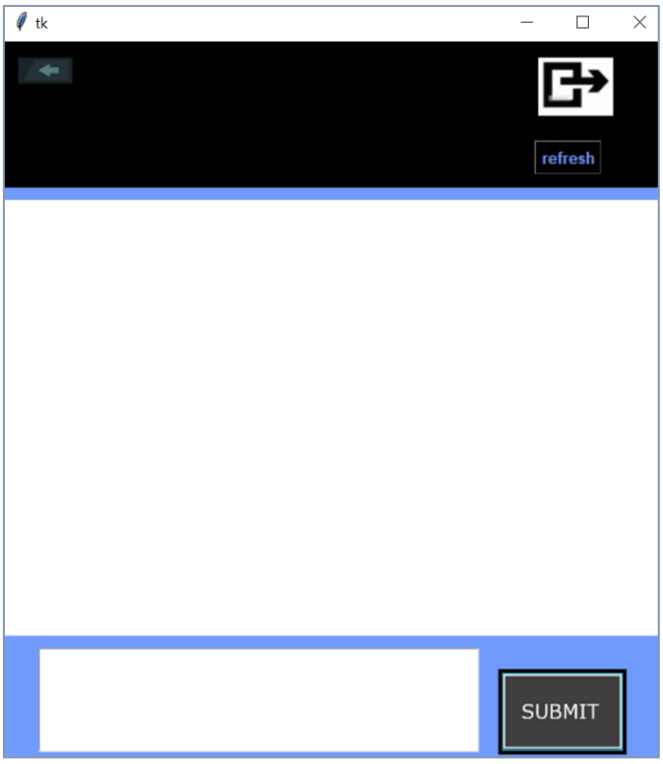

Figure 37 : la figure de la partie de chat et interaction avec l'utilisateur de l'interface

a. Les fonctions utilisées dans la partie interface :

Dans cette partie nous avons utilisé 4 fonctions principales qui vont nous aider à dessiner le chiffre à travers l'interface graphique.

- *Forget()* : cette fonction donne la possibilité de cacher une frame de la fenêtre principale et par suite on aura capable d'ajouter une autre frame au lieu de celle qu'on cacher .
- *Pack ()* : permet d'ajouter un frame ou bien un Button un label à la fenêtre
- *Place ()* : a le même rôle que la fonction *pack()* et il a comme plus qu'on peut contrôler l'ajout en manipulant les coordonnées x et y.
- *Widget.Destroy()* : est utilisée pour effacer la partie du chat.

b. La fonction qui prépare la réponse :

Dans cette section nous allons voir comment on a implémenté le programme qui prend la phrase qui a été saisit par l'utilisateur et puis il prédit le tag qui est correspondant à cette phrase en utilisant le module torch.max et puis on applique la function softmax[5] () pour tester si la probabilité de la prédiction est assez haut pour le choisir (plus que 0.75) le code d'implémentation de cette partie est comme ceci :

```
def submit():
    global chat_raw
    chat_raw = entry.get('1.0', 'end-1c')
    entry.delete('1.0', END)
    chat=tokenize(chat_raw)
    X = bag_of_words(chat, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)
    output = model(X)
    _, predicted = torch.max(output, dim=1)
    tag = tags[predicted.item()]
    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    global label_request
    label_request = Label(frame_chats, text=chat_raw, bg=c4, fg='white', justify=LEFT, wraplength=300,
                          font='Verdana 10 bold')
    label_request.pack(anchor='w')
    global answer
    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag == intent["tag"]:
                answer=random.choice(intent['responses'])
    else:

        answer="I don't understand........."
    get_response()
```

Figure 38 : la fonction qui prépare la réponse qui sera destinée à l'utilisateur

[5] Softmax est une fonction mathématique qui convertit un vecteur de nombres en un vecteur de probabilités, où les probabilités de chaque valeur sont proportionnelles à l'échelle relative de chaque valeur du vecteur.

VI. Résultat :

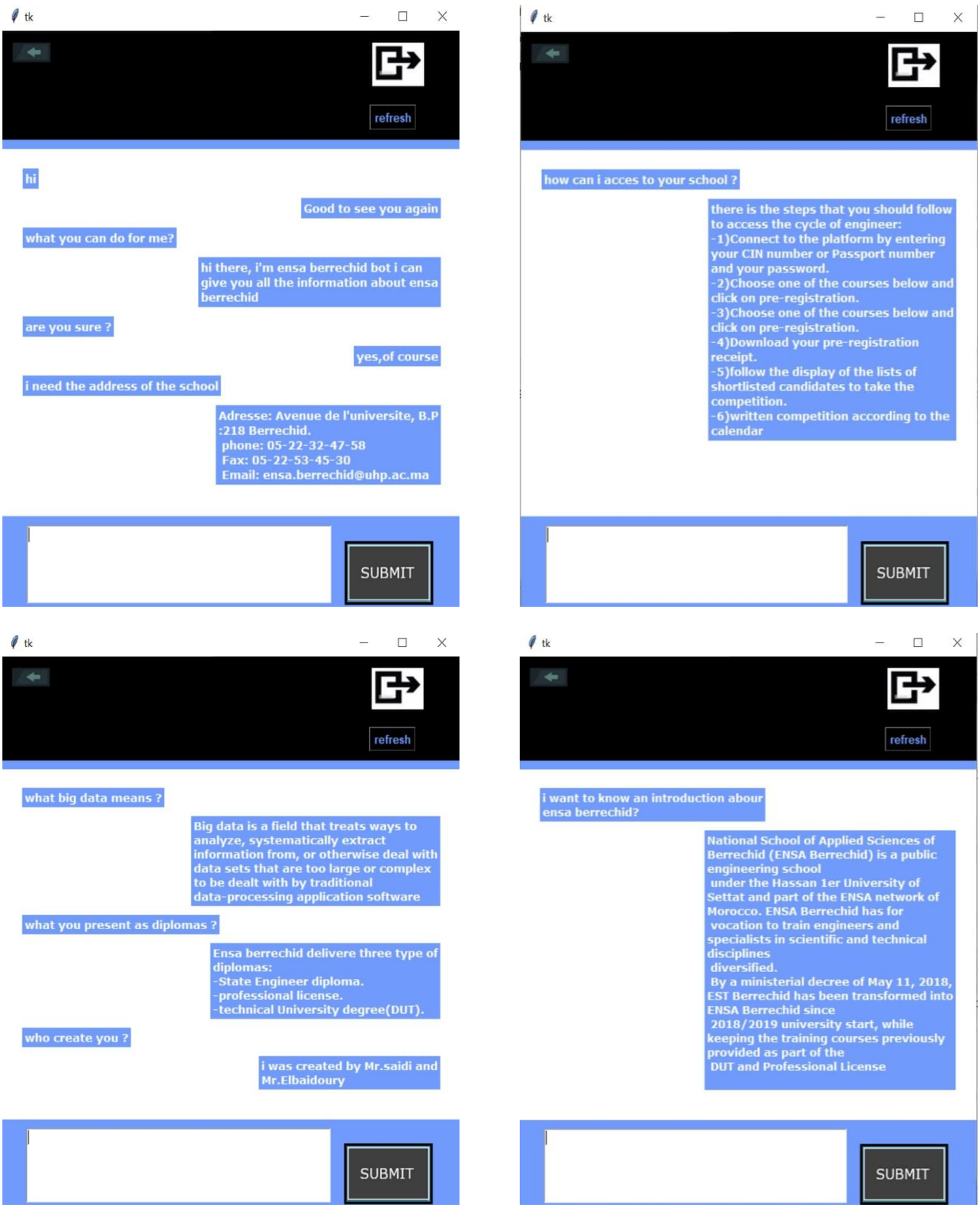

Figure 39 : les résultats du chat

Conclusion

Enfin après la réalisation de la premeier application du deep learning ,il est claire que l'application du handwritten recognition digits a un taux élevé de réussite , elle est rapide et efficace par rapport à l'ancienne méthodolie des pixels d'image il peux aussi etre utile dans plusieurs application (trier les courries éléctroniques,traitement des cheques bancaires,la saisie des donnée formulaire).

Pour le deuxieme qui application correspond à un chatbot , a été une occasion pour comprendre l'architecture et le fonctionnement des chatbot avec nlp , ces chatbots ouvrent d'immenses opportunités pour enrichir voire réinventer l'expérience client et facilite la tache dans plusieurs aspects (amélioration de la réactivité 24/7, automatisation des tâches répétitives et simples, économisez l'argent, amélioration de l'expérience de l'utilisateur).

Références

Bookshelf Microsoft Azure AI - Notions fondamentales AI-900T00FR-A

https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles

https://experiences.microsoft.fr/business/intelligence-artificielle-ia-business/comprendre-utiliser-intelligence-artificielle/

https://experiences.microsoft.fr/business/intelligence-artificielle-ia-business/chatbot-arte/

https://en.wikipedia.org/wiki/Neural_network

https://medium.com/godatascience/guide-to-spam-classification-using-nltk-library-stemming-and-bag-of-words-2f9450bc1485

https://www.lebigdata.fr/traitement-naturel-du-langage-nlp-definition