UNIVERSITÉ
Clermont Auvergne

IA : Reconnaissance manuscrite

Souchon Théo & Cyril Toffin

IUT Informatique
Année universitaire 2020 – 2021

Table des matières

INTRODUCTION

Identification du problème

Le sujet du projet est de faire une intelligence artificielle qui soit capable de reconnaître des caractères manuscrits (écrits à la main). Mais beaucoup de zones d'ombres restent autour de ce simple intitulé. On cherche donc à mieux cerner le problème avec une analyse plus rigoureuse du sujet.

Les entrées de notre IA sont des images (28x28 pixels) qui représentent des lettres écrites à la main ou à la souris pour plus de facilité d'utilisation. Ces lettres peuvent être en majuscule ou en minuscule. La sortie de l'IA est une prédiction du résultat selon notre entrée, cette prédiction est une lettre qui est supposée correspondre à celle manuscrite selon les capacités de notre intelligence artificielle.

Notre IA se base sur un arbre de classification. En effet, le résultat à la sortie est une lettre fixe comprise entre A et Z. Cette valeur est discrète, comparé à un arbre de régression qui sera pour des variables continues (inexactes et sensibles à des changements).

1. Comprendre les données

A - Quantité de donnés

- Les données sont contenues dans un fichier de type .csv qui possède 370 000 lignes incluant pour chaque ligne une image (matrices de pixels) ainsi qu'un nombre représentant la position de la lettre dans l'alphabet associée à la matrice.

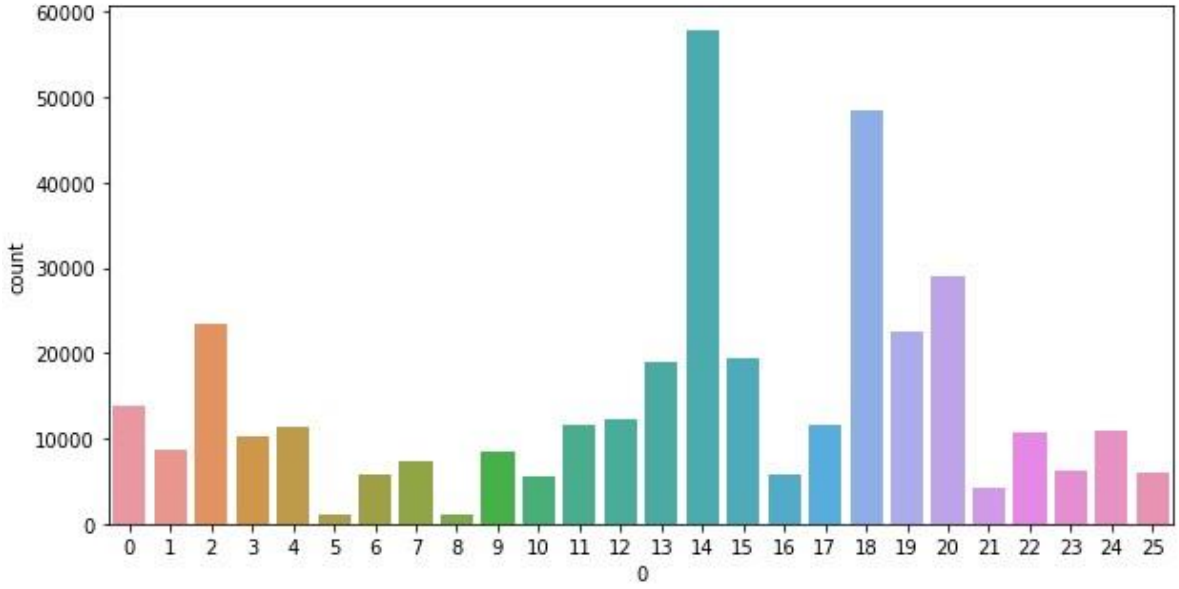
- Le fichier .csv possède environ 370 000 lignes. Après génération de la base sous le format d'image .png / .jpeg, on obtient 372 451 images exactement, réparties de manière hétérogène parmi les différentes lettres

B - Qualité des données

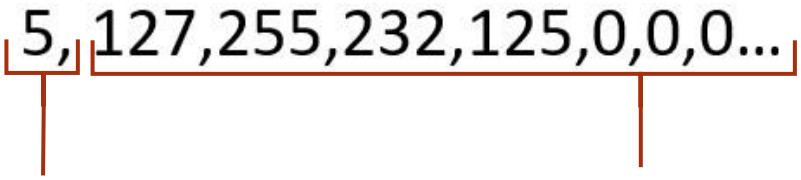
- La pertinence des données est correcte, chaque ligne du fichier .csv représente un identifiant correspondant à la lettre appartenant à la ligne. A la suite une matrice de pixel représentant l'image.
- Le type de donnée utilisé est numérique, l'identifiant est un nombre compris entre 0 (A) et 25 (Z). Tandis que l'image une matrice de nombre compris entre 0 et 255 représentants chaque pixel en nuance de gris.
- Un grand nombre de données est présenté, mais des lettres possèdent plus de données que d'autres, ce qui donne une précision différente entre les lettres. Par exemple, " a " possède 13 870 éléments tandis que " i " n'en possède que 1120 (soit plus de 10x moins).

C - Représentation des données

La répartition du nombre d'images pour chaque lettre de l'alphabet :



Structure d'une ligne du dataset :



Position de la lettre dans l'alphabet

Matrice de 28*28 représentants les pixels de l'image

2. Exercice d'entrainement

Afin de mieux comprendre notre base de données et pouvoir les visualiser nous les avons extraites dans un dossier hiérarchique pour chaque lettre. Dans la création du modèle nous n'utilisons pas cette base d'image car nous utiliserons le .csv qui colle mieux à la lecture de nos données. Mais il est tout de même bien de se donner à l'exercice pour faciliter notre compréhension du sujet.

1 - Initialisation des variables :

```
#Initialisation variable
csv_File_Path = "A_Z_Handwritten_Data.csv"
count = 1
last_digit_Name = None
image_Folder_Path = "..\..\..\LETTRES"
Alphabet_Mapping_List = list(string.ascii_uppercase)
```

2 - Création des répertoires pour chaque lettre dans l'alphabet :

```
#Pour chaque lettre on créer un dossier avec le nom de la lettre
for alphabet in Alphabet_Mapping_List:
    path = image_Folder_Path + '\\' + alphabet
    if not os.path.exists(path): #Si le dossier n'existe pas on le créer
        os.makedirs(path)
```

3 - Lecture des données d'une ligne :

```
with open(csv_File_Path, newline='') as csvfile: #Se deplacer dans le fichier de la lettre actuelle
    reader = csv.reader(csvfile, delimiter=',', quotechar='|') # On lit une ligne définit par les delimiter
    count = 0
    for row in reader:
        digit_Name = row.pop(0) #Obtenir l'identifiant de l'image, la lettre à laquelle il correspond
        image_array = np.asarray(row) #la matrice de l'image
        image_array = image_array.reshape(28, 28) #On redéfinit sa taille en 28x28
        new_image = Image.fromarray(image_array.astype('uint8')) #On récupère la matrice en tant qu'image
```

4 - Changer de lettre selon l'identifiant de la ligne :

```
        if last_digit_Name != str(Alphabet_Mapping_List[(int)(digit_Name)]): #Si on change de lettre
            last_digit_Name = str(Alphabet_Mapping_List[(int)(digit_Name)]) #On actualise la nouvelle lettre
            count = 0
            print("")
            print("Prcessing Alphabet - " + str(last_digit_Name))
```

5 - Sauvegarde de l'image dans le répertoire :

```
        image_Path = image_Folder_Path + '\\' + last_digit_Name + '\\' + str(last_digit_Name) + '-' + str(
            count) + '.png' #chemin de sauvegarde de l'image
        new_image.save(image_Path) #On sauvegarde l'image au chemin précédent
        count = count + 1

        if count % 1000 == 0: #Comptez toutes les 1000 images faites
            print("Images processed: " + str(count))
```

Cet exercice nous a permis de comprendre le fonctionnement du dataset et nous a aidé pour fabriquer le modèle de notre IA car nous avons réutilisé les mêmes procédés.

3. Le modèle

A - Construction du modèle

Pour notre IA il faut construire un modèle afin de contenir l'apprentissage que fait l'IA, pour cela on va créer un modèle et l'entrainer avec notre dataset.

```
dataset = pd.read_csv("A_Z Handwritten Data.csv").astype('float32')#Lecture du data set contenant les données
dataset.rename(columns={'0': 'label'}, inplace=True)#Renomme l'identifiant de la ligne du fichier en tant que "label"
```

Le programme va donc travailler avec le dataset que l'on a téléchargé et déjà utilisé dans l'exercice d'avant.

```
alphabets_mapper = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L',
                    12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W',
                    23: 'X', 24: 'Y', 25: 'Z'}
```

On associe chaque lettre à un numéro représentant sa place dans l'alphabet.

```
X_train, X_test, y_train, y_test = train_test_split(X, y) #Séparation des données en sous ensemble pour pouvoir tester le modèle entrainé
```

On peut séparer les données pour pouvoir réaliser les tests.

```
# Construction du modèle#
cls = Sequential() #Creation du modele
cls.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
cls.add(MaxPooling2D(pool_size=(2, 2)))
cls.add(Dropout(0.3))
cls.add(Flatten())
cls.add(Dense(128, activation='relu'))
cls.add(Dense(len(y.unique()), activation='softmax'))
```

On construit le modèle avec les données du dataset, on le fait pour toutes les données disponibles afin de maximiser la précision de note modèle.

```
cls.save('modele_ecriture.model')
```

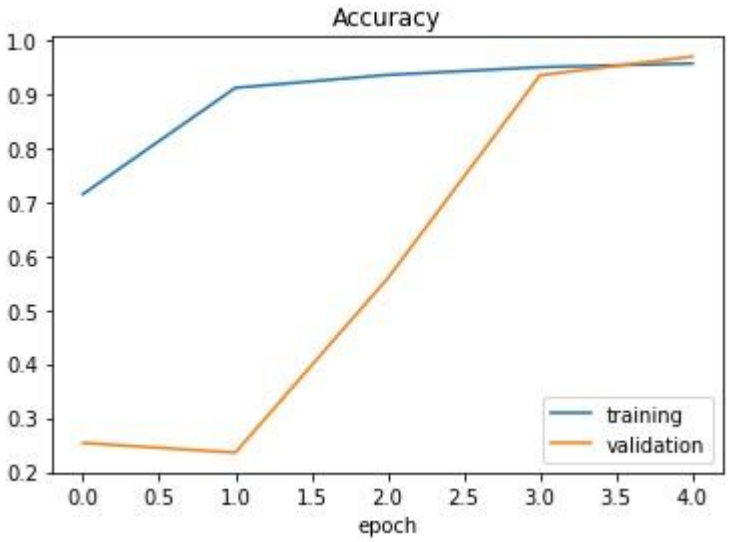
Ensuite on enregistre le modèle pour l'utiliser lors de nos phases de test.

B - Évaluation du modèle

Afin de savoir si notre modèle est précis, il faut connaître la précision de celui-ci, pour cela, on va le tester sur un échantillon d'images dont on connaît déjà la lettre associée à la matrice puis on va vérifier que la lettre prédite correspond bien à la lettre. On fait cette opération avec un certain nombre d'images ce qui va nous donner une précision correspondant au modèle.

```
1397/1397 - 60s - loss: 0.2435 - accuracy: 0.9324 - val_loss: 0.0903 - val_accuracy: 0.9750
Précision du modèle : 0.9749873876571655
```

Grâce à cette évaluation du modèle cela permet une approximation de la précision du modèle dans les futures prédictions lors des tests que nous réaliserons en utilisant ce modèle.

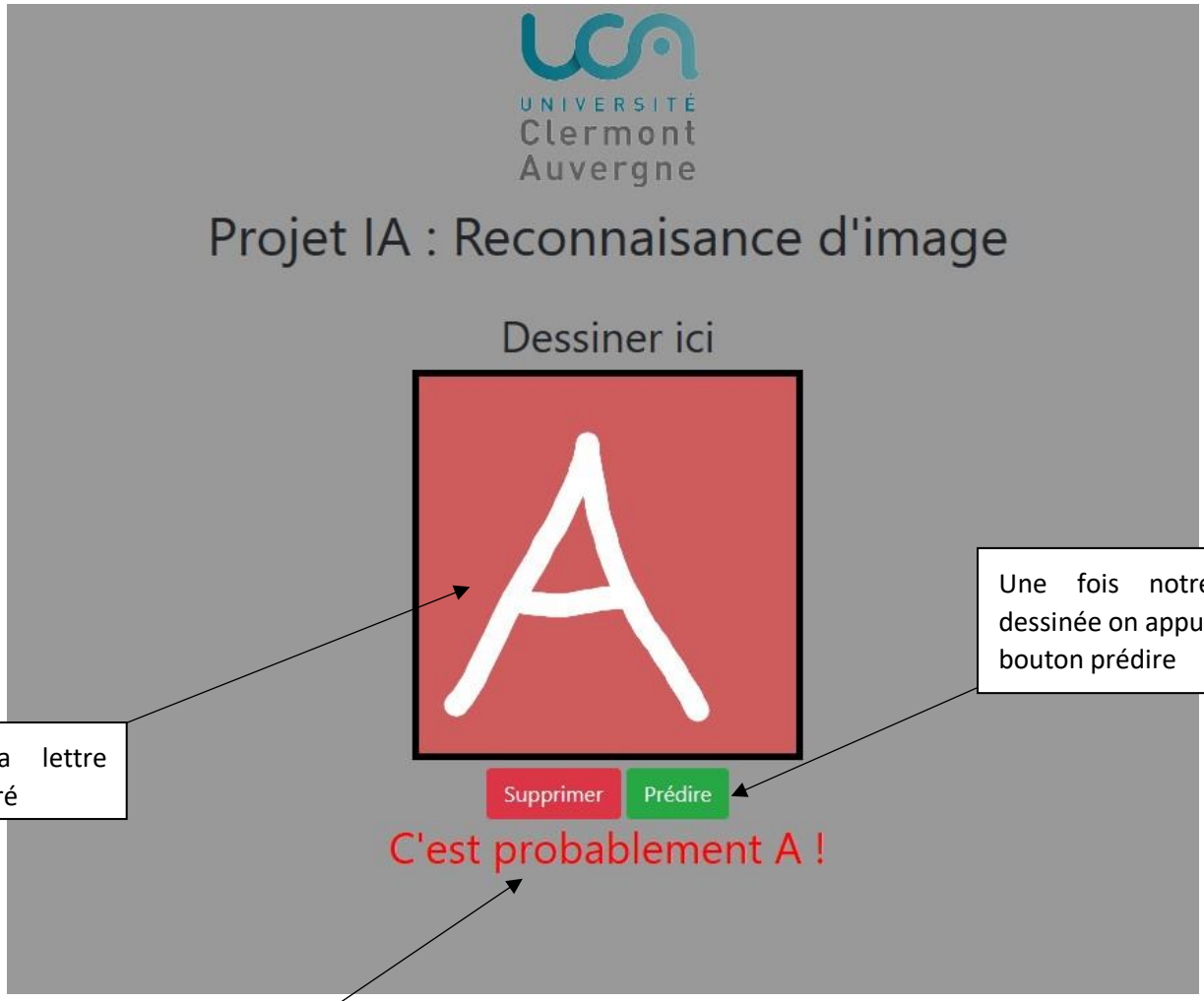


Ce graphique montre la précision du modèle, on constate que plus l'entrainement est long plus la précision du modèle sera haute. Ce que l'on a fait afin d'optimiser notre modèle.

4. Les tests

A - Programme de test

Pour pouvoir tester notre modèle nous avons cherché un moyen simple de l'utiliser, nous avons donc trouvé sur internet un code qui permettait de dessiner les caractères à la main dans une fenêtre qui se lançait dans le navigateur, nous avons donc repris le code et nous avons apporté quelques modifications pour pouvoir l'adapter à notre code et implémenter notre modèle à cette interface. Une fois le code modifié voici le rendu de note programme de test :



Dessine la lettre dans le carré

Une fois notre lettre dessinée on appuie sur le bouton prédire

Annonce de la prédiction du modèle

B - Fonctionnement du programme

Ce programme fonctionne en utilisant le modèle que nous avons créé juste avant il est relativement simple à comprendre et n'est pas très long.

```
image = np.asarray(bytearray(draw_decoded)) #Transformationde de draw_decode en un tableau d'octet
image = cv2.imdecode(image, cv2.IMREAD_GRAYSCALE) #Retouche de l'image en nuance de gris
resized = cv2.resize(image, (28,28), interpolation=cv2.INTER_AREA)#Redimensionnement de l'image en 28*28 px
```

En premier on récupère l'image qui est dessiné dans le cadre, puis on l'a convertie en une matrice de pixels que l'on passe en nuance de gris. Car le modèle s'est entrainé avec des images en noir et blanc, puis l'image est recadrée au format 28*28.

```
pred = model.predict(vect) #Test de prediction
index_pred = np.argmax(pred) #Choix de l'index en se basant sur argument max
```

Ensuite le programme prédit la lettre en se basant sur le modèle avec lequel il s'est entrainé. Il associe une probabilité à chaque lettre de l'alphabet en mettant à chaque fois une probabilité que ce soit la lettre dessinée. Il suffit de choisir la lettre avec la plus haute probabilité pour pouvoir prédire quelle est la lettre à deviner. Il nous reste plus qu'à l'afficher en espérant que ce soit la bonne lettre .

CONCLUSION

En conclusion, nous avons vraiment apprécié faire ce projet, cela a été une réelle découverte de l'intelligence artificielle pour nous, du champ des possibilités qu'elle permettait, nous sommes assez fiers du rendu final que nous avons fait.

Cependant, nous avons remarqué certaines limites lors du projet notamment la précision de nos tests qui ne correspondent pas totalement à la précision annoncée lors de la création de notre modèle. La résolution des images que nous utilisons est assez basse (28*28 pixels) ce qui pourrait expliquer les différentes erreurs que le modèle fait lors des tests.

Comme observé dans notre dataset, certaines lettres sont plus présentes ce qui fait que le modèle s'est plus ou moins entrainé selon les lettres, la conséquence est que le modèle reconnait mieux certains lettes que d'autres ce qui pose un problème.