ENSAE ParisTech

Projet de programmation

Reconnaissance de Captchas

Paul-Noël Digard, Louis François (Février - Mai 2018)

Préambule

Intéressés par le Deep Learning et fascinés par ses différentes utilisations (détection d'objets sur des photographies, voitures autonomes, colorisation d'images en noir et blanc, etc.), nous souhaitions comprendre son fonctionnement. Or, ce dernier repose essentiellement sur la notion de réseau de neurones. Nous nous sommes donc intéressés à une utilisation « simple » de ces derniers : la reconnaissance de caractères (lettres et chiffres). Toutefois, le problème de reconnaissance de caractères étant un « classique » de l'introduction au Deep Learning, nous avons décidé de ne pas limiter notre projet à la programmation du problème MNIST. Nous avons d'abord pensé à programmer une méthode de reconnaissance et de résolution d'équations mathématiques manuscrites. Toutefois, nous anticipions certaines difficultés que nous ne pensions pas pouvoir surmonter (constitution d'une base de données assez grande pour pouvoir entraîner notre réseau de neurones, reconnaissance de racines ou d'exposants,...). Nous avons alors décidé de nous pencher sur le problème de la reconnaissance de « captchas », ces images de séquences de caractères utilisées lors de tests d'accès à certains services sur Internet. Ce qui nous intéressait tout particulièrement était de voir comment, dans le cas de captchas « simples », il était possible pour un robot de se faire passer pour un humain en réussissant un tel test.

Sommaire

I Définition du problème

I.1 Vue d'ensemble

La reconnaissance de caractères est un problème connu qui a nombre d'applications. La reconnaissance d'adresses postales par les services postaux, la lecture automatique de chèques ou encore la lecture automatique pour les personnes malvoyantes ne sont que des exemples de ces applications. Ceci a encouragé la recherche dans ce domaine afin d'améliorer les techniques de reconnaissance de caractères de toute sorte. La reconnaissance de chiffres est la version du problème la plus étudiée, si bien que depuis les années 1990, la base de données MNIST permet d'obtenir un taux de réussite (proportion de chiffres correctement reconnus) de 99,9%. Aujourd'hui, des techniques telles que les réseaux de neurones « profonds » (*deep neural networks*) sont utilisées pour reconnaître des lettres de l'alphabet latin ou des caractères chinois. On se propose ici d'utiliser les réseaux de neurones pour reconnaître des captchas. On se limitera à la reconnaissance de captchas de quatre caractères. Toutefois, les programmes que nous implémenterons fonctionneront également sur des captchas plus longs. Le set de données que nous utiliserons contient environ 10 000 captchas de 4 caractères.

FIGURE 1 – Un exemple de captcha de la base de données

Cette base donne donc accès à un set de données d'environ 40 000 caractères (en extrayant les 4 caractères de chaque captcha) réparties en 35 classes (9 chiffres, 1 à 9, et 26 lettres). Notre set de données offre donc environ 1000 exemples par classe. Il est important de noter que ce set de données est relativement petit, notamment en comparaison au set de données MNIST offrant 6000 exemples par classe (9 classes pour 9 chiffres allant de 1 à 9). De plus, parmi ces 1000 exemples par classe, seulement une partie sera utilisée pour « entraîner » notre modèle, l'autre étant utilisée pour le tester. Ainsi, le taux de réussite de la reconnaissance de captchas par notre modèle, que nous tenterons d'optimiser, sera nécessairement moins élevé que celui de la reconnaissance de chiffres utilisant la base de données MNIST.

I.2 Énoncé du problème

Le problème que nous nous proposons de résoudre est le suivant : peut-on déterminer une fonction prenant en argument une capture d'écran de captcha et renvoyant les caractères de celui-ci. Pour déterminer un tel algorithme

et résoudre ce problème, nous allons utiliser un set de données de 10 000 captchas au format .png ainsi que des techniques de Deep Learning telles que les réseaux de neurones. Nous suivrons les étapes suivantes :

- Séparation du set de captchas en un set d'entraînement et un set de validation (90% - 10%).
- Création d'un set de données chiffres/lettres à partir du set d'entraînement (découpage des captchas : 1 captcha = 4 caractères).
- Entraînement d'un modèle de prédiction de caractère unitaire (réseau de neurones) avec une partie de ce nouveau set (90%).
- Test de ce modèle avec l'autre partie (10%) du set d'entraînement.
- Implémentation d'une fonction prenant en entrée un captcha au format .png et renvoyant son texte grâce à notre modèle de prédiction.
- Validation de notre fonction en déterminant son taux de réussite sur notre set de validation.

Si notre méthode de découpage pour isoler chaque caractère du captcha est efficace nous devrions logiquement avoir un taux de réussite en R^4 (avec R le taux de réussite de notre modèle de prédiction de caractère unitaire) car chaque captcha est une image de quatre caractères.

I.3 Métrique

Pour pouvoir conclure quant à l'efficacité de notre fonction de reconnaissance de captcha, il nous faut définir ce que l'on entend par "taux de réussite". Dans ce projet, nous mesurerons notre performance par le rapport entre le nombre de captchas correctement prédits et le nombre total de captchas prédits. Nous nommons ce taux de réussite "accuracy" :

$$accuracy = \frac{nombre\ de\ prédictions\ correctes}{nombre\ total\ de\ prédictions} \tag{1}$$

II Méthodologie

Pour résoudre notre problème nous avons suivi l'adage "Diviser pour mieux régner" et avons procédé en deux temps :

— Création d'un set de données de caractères unitaires prêts à l'emploi pour entraîner notre modèle de prédiction.
— Entraînement d'un modèle de prédiction de caractères.

Nous allons dans un premier temps détailler la résolution théorique de chacune de ces parties puis nous nous attarderons sur leur implémentation en Python.

II.1 Théorie

a Création d'un set de données de caractères unitaires

Tout d'abord nous séparons notre set de données de travail (contenant 10 000 captchas) en deux sets :

— `training_set` : 90% du set initial (environ 9000 captchas).
— `validation_set` :10% du set initial (environ 1000 captchas).

La base `validation_set` nous permettra de déterminer si notre fonction finale (prenant en entrée un captcha au format .png et renvoyant son texte) est efficace ou non.

La base `training_set` va quant à elle nous permettre d'élaborer une nouvelle base de données contenant les caractères des captchas isolés. Pour cela on va "découper" chaque captcha présent dans `training_set`. Le point clé de la création de notre base de données réside donc dans un découpage efficace des captchas. Notre méthode de découpage consiste à déterminer les "frontières" à droite et à gauche de chaque caractère dans le captcha. On procède comme suit :

— (1) Initialement, on se place à l'extrémité gauche du captcha et on parcourt les pixels de haut en bas et de gauche à droite.

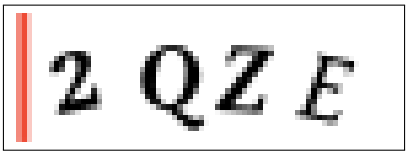

FIGURE 2 – Initialement, nous sommes dans une zone "blanche"

— (2) Puisque nous sommes initialement dans une zone où tous les pixels sont blancs, on arrête le parcours dès que l'on rencontre un pixel noir et on mémorise cette première "frontière".

FIGURE 3 – Frontière gauche fixée lors de la "rencontre" du 1er caractère

— (3) Nous sommes désormais dans une zone avec un mélange de pixels noirs et blancs, la frontière droite est donc caractérisée par la première colonne rencontrée constituée exclusivement de pixels blancs.

FIGURE 4 – Nous sommes dans une zone "noire et blanche"

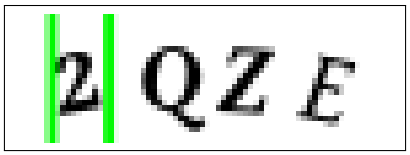

FIGURE 5 – Frontière droite (1ère colonne blanche)

— (4) On continue le balayement jusqu'à l'extrémité droite de l'image.

Néanmoins, cette méthode atteint ses limites lorsque deux caractères sont collés comme c'est le cas dans le captcha suivant :

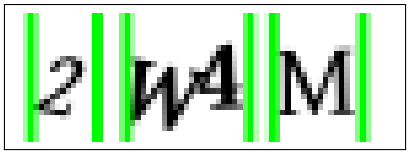

FIGURE 6 – Captcha avec deux caractères "collés"

Dans cette situation, on décide de couper l'image extraite des deux caractères collés en son centre. L'expérience nous a montré que cette précision était suffisante.

Ce processus nous permet d'avoir un set de $9000 \times 4 = 36000$ caractères environ. Ce set va nous permettre d'entraîner un modèle de reconnaissance de caractères.

b Création d'un modèle de prédiction

Pour reconnaître des caractères unitaires, nous avons choisi d'utiliser un modèle simple de réseau de neurones : une *Softmax Regression*. L'objectif

est de créer une fonction prenant en entrée l'image du caractère sous la forme d'un vecteur dont le *j-ème* élément code le *j-ème* pixel de l'image, et renvoyant un vecteur de $\mathbb{R}^{35}$ dont le *j-ème* élément correspond à la probabilité que le caractère appartienne à la classe j (rappel : il y a 35 classes : 9 chiffres et 26 lettres). Dans le modèle *Softmax Regression* on cherche la fonction sous la forme :

$$f(X) = softmax(WX + b) \tag{2}$$

où :

- X est un vecteur de taille n avec n le nombre de pixels de l'image.
- W est une matrice de dimension (n, 35).
- b est un vecteur de taille 35.
- *softmax* est la fonction définie par : $softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ (avec $z = (z_1, ..., z_K)$ un vecteur réel).

L'enjeu est alors d'utiliser le set de caractères que l'on vient de créer pour déterminer les coefficients de W et de b qui minimisent les écarts de prédiction. Le problème devient ainsi un problème d'optimisation. L'outil informatique est alors indispensable face à ces grandes dimensions.

II.2 Implémentation en Python

a Création de la base de données de caractères unitaires

Pour créer un tel set de données à partir de `training_set` nous avons procédé en trois étapes :

- Première étape : *Preprossessing* des images.
- Deuxième étape : Découpage des images.
- Troisième étape : Transformation des images sous forme de listes.

En effet le but est d'avoir un set constitué de couple (Image sous forme de vecteur, Label) prêt à l'emploi pour entraîner notre modèle.

1ère étape : *Preprossessing* des images Le set `training_set` est constitué d'enviroon 9000 captchas au format .png. Nous les ouvrons dans Python et les transformons en tableau numpy RGB puis "teinte de gris" à l'aide notamment des fonctions de la librairie PIL (Python Image Library). Pour plus de simplicité et pour alléger les calculs on recode les tableaux obtenus en des tableaux contenant uniquement des 0 (pour le noir) et des 1 (pour le blanc). Pour attribuer un label à chaque tableau nous avons utilisé des fonctions de la librairie OS et avons récupéré le nom de base de chaque image (qui n'est autre que le texte du captcha).

À la fin de cette étape nous avons une liste de couples (tableaux numpy, label). Nous la sauvegardons sur le disque au format pickle pour pouvoir la réutiliser plus tard.

2ème étape : Découpage des images Ce point est certainement le plus important de notre projet. En effet, tout le problème réside dans un découpage efficace de captchas en quatre caractères. Nous avons pour cela implémenté la procédure suivante qui met en application la théorie que nous avons expliquée dans la partie précédente. (`l` et `h` sont la largeur et la hauteur de chaque captcha sous forme de tableau ; `captcha_noir` est la liste obtenue à la fin de la 1ère étape)

```
#decoupage des captchas (ie creation d'un set de donnees (lettre/label))
letters = []
for (i, (captcha_noir, lbl)) in enumerate(captchas_noirs):
    print("[decoupage] processing captcha {}/{}".format(i + 1, len(captchas_noirs)))
    frontieres = []  #liste contenant les indices des colonnes constituant une frontieres entre deux caracteres
    blanc = True
    noir = False
    for j in range(l):
        if blanc:
            i=0
            while blanc and i<h:
                if captcha_noir[i][j] == 0:
                    frontieres.append(j)
                    blanc = False
                    noir = True
                i=i+1
        else: #cad si noir, on cherche la frontiere de droite
            i = 0
            ok = True   #si ok alors on a que des pixels blancs sur la colonne
            while ok and i<h:
                if captcha_noir[i][j] == 0:
                    ok = False
                i+=1
            if ok:
                frontieres.append(j)
                blanc = True
                noir = False
    #on connait maintenant les indices des contours des caracteres du captcha, on peut proceder au decoupage
    split_captcha = []
    for i in range(0, len(frontieres)-1, 2):
```

```
31          fg, fd = frontieres[i], frontieres[i+1] #il s'
               agit des indices des colonnes encerclant le
               premier caractere (ou groupe de caracteres)
32          if (fd - fg) < 0.9*h:    #dans ce cas cela
               signifie qu'il n'y a qu'un seul caractere (et
                pas deux colles)
33              split_captcha.append(np.array([captcha_noir[
                   i][fg-1:fd+1] for i in range(h)]))
34          else: #ici on traite le cas ou deux caracteres
               sont colles et n'ont pu etre separes
35              m = int((fd + fg)/2)
36              split_captcha.append(np.array([captcha_noir[
                   i][fg-1:m+1] for i in range(h)]))
37              split_captcha.append(np.array([captcha_noir[
                   i][m:fd+1] for i in range(h)]))
38  #on associe maintenant chaque lettre à son label
39      if len(split_captcha) == 4:  #sinon cela signifie
           que le decoupage n'a pas bien marche, on
           utilisera donc pas ce captcha pour entrainer
           notre modele car il est de mauvaise qualite
40          for k in range(4):
41              letters.append((split_captcha[k], lbl[k]))
42
43  import pickle
44  with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
       Projet_python/letters_pckl', 'wb') as fp:
45      pickle.dump(letters, fp)
```

Cette procédure permet de découper chaque captcha selon notre méthode expliquée dans la partie théorique. Pour chaque couple (captcha,label) de la liste `captcha_noir` (qui est la liste obtenue à la fin de la 1ère étape), la procédure python parcourt le tableau et retient les indices des colonnes correspondant aux frontières gauches et droites de chaque caractère. On procède ensuite à la séparation du tableau en quatre et on attribue le caractère associé (l'un des quatre caractères du label du captcha) à chaque partie du tableau. Le résultat de cette étape (`letters`) est une liste de couples (caractère, label) que l'on sauvegarde au format pickle sur le disque.

Complexité : En terme de temps de calcul cette étape est assez longue (du moins pas instantanée). En effet, la procédure effectue un parcours de la liste de couples (captcha sous forme de tableau, label) `captcha_noir` de longueur 9000, et pour chaque couple de cette liste, elle parcourt tous les éléments du tableau représentant le captcha (tableau de dimension 72 par 24). Cela donne donc un total de l'ordre de $9000 \times 72 \times 24 = 15\,552\,000$ de calculs. Sur machine cela s'effectue en un temps de l'ordre de la minute. Nous avons donc décidé d'utiliser la fonction `enumerate` et de placer un `print` au début de la boucle de parcours de la liste de (captcha,label) pour connaître l'avancement

de la procédure et savoir combien de captchas il restait à traiter.

3ème étape : Transformation des images sous forme de listes La liste `letters`, obtenue à l'étape précédente, n'est pas immédiatement utilisable pour entraîner notre modèle de prédiction. En effet, du fait de notre méthode de découpage, les tableaux représentant les caractères n'ont pas tous la même dimension. Nous faisons donc d'abord appel à la fonction `img.resize` du module PIL. Puis, nous transformons chaque tableau en liste car la fonction de prédiction de notre modèle prend en entrée une liste des pixels de l'image à prédire. Il reste à transformer chaque label en vecteur codant la classe. Pour cela on utilise une liste `outputs` regroupant l'ensemble des classes et chaque label (au format string) est alors transformé en une liste de 0 avec un 1 au niveau de la classe correspondante.

```
1 outputs = ['1', '2', '3', '4', '5', '6', '7', '8', '9',
     'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
2              'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
                   'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
```

Par exemple le label 'Z' sera codé par le vecteur [0, ..., 0, 1].

Nous avons désormais une liste de couples (caractère sous forme de liste, label sous forme de liste) prête à l'emploi pour entraîner notre modèle. Nous la nommons `features_labels` et la sauvegardons au format pickle.

b Entraînement du modèle de prédiction

Dans cette partie de notre projet, nous avons décidé d'utiliser le framework TensorFlow. Il permet en effet d'entraîner notre modèle de manière concise.

Nous séparons dans un premier temps notre liste `features_labels` en deux :

- `features_labels_train` : 90% de `features_labels` pour entraîner le modèle.
- `features_labels_test` : 10% de `features_labels` pour tester le modèle.

Pour minimiser l'écart entre prédictions et labels de `features_labels_train`, nous minimisons la fonction de perte grâce à la fonction `GradientDescentOptimizer` de TensorFlow. Une fois cette étape effectuée, nous testons notre modèle sur `features_labels_test` et obtenons une *accuracy* de 93%, ce qui nous semble un bon résultat compte tenu du résultat similaire obtenu par une Softmax Regression sur le problème MNIST. Nous sauvegardons alors les matrices W et b sur le disque au format pickle pour les réutiliser dans notre fonction finale de prédiction qui prendra en entrée un captcha au format .png.

II.3 Difficultés rencontrées

La majorité des difficultés rencontrées concernent le traitement des images. La première difficulté fut de trouver un moyen efficace et précis de séparer les uns des autres les caractères d'un même captcha. En effet, si l'idée naturelle pour faire cela est de couper l'image captcha verticalement en quatre images de mêmes dimensions, ce n'est pas la meilleure solution. En effet, dans certains captchas, les quatre caractères ne sont pas répartis de manière homogène sur l'image. C'est notamment le cas dans le captcha ci-dessous, dans lequel tous les caractères sont concentrés dans la partie gauche de l'image, si bien que le dernier quart de l'image (celui de droite) ne contient aucun caractère.

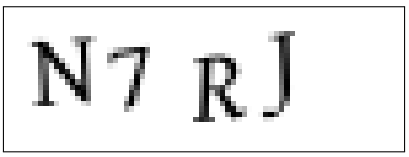

FIGURE 7 – Captcha dont les caractères sont concentrés à gauche

Nous avons finalement opté pour la méthode décrite plus tôt, qui détermine les frontières de chaque caractère du captcha en trouvant les indices de début et de fin des caractères dans le « tableau » représentant l'image.

La seconde difficulté rencontrée découle de la première et de la méthode utilisée pour la surmonter. En effet, certains captchas, dont un exemple est donné ci-dessous, contiennent des caractères qui se superposent.

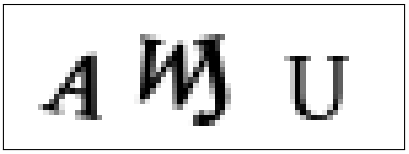

FIGURE 8 – Ici le W et le J se superposent

Dans ce cas, la méthode utilisée pour la séparation des différents caractères au sein d'un captcha comprendra la continuité entre les traits formant la lettre « W » et ceux formant la lettre « J » comme la présence d'une seule lettre et non pas deux. Pour prévenir ce genre de problème, nous avons ajouté un « double-contrôle » à notre méthode de séparation de caractères. Un premier contrôle consiste à n'effectuer le découpage d'un supposé caractère que si le rapport largeur/hauteur de l'image à découper est strictement inférieur à 0.9. Le choix du rapport largeur/hauteur limite de 0.9 est empirique et découle de nombreuses observations sur les captchas du set de données utilisé. Le second contrôle consiste à n'ajouter les images de caractère obtenues après séparation au set d'entraînement de notre modèle que si 4 images sont obtenues pour un captcha donné. Le cas contraire, le captcha ne sera pas

utilisée. Les captchas aux caractères superposés étant rares dans notre set de 10000 captchas, il nous semblait plus judicieux de se priver d'une poignée d'entre elles plutôt que d'entraîner notre modèle avec des données erronées.

III Résultats

III.1 Fonction finale

Le but de cette fonction est de prendre en entrée un captcha au format .png et de renvoyer le texte de ce captcha. Pour cela nous avons codé la fonction suivante suivante qui fait appel à des fonctions de *preprossessing* et de découpage qui reprennent principalement les procédures utilisées pour la création de la base de données.

```
def predictor(captcha):     #en entree on attend un
    chemin vers un captcha dont on veut predire le texte
    captcha_noir_blanc = formatage_captcha(captcha)
    letters = decoupage(captcha_noir_blanc)
    lettres_a_predire = []
    for letter in letters:
        lettres_a_predire.append(ready_to_test(letter))
        prediction = ""
        for x in lettres_a_predire:
            y = softmax(np.dot(x, W) + b)
            prediction+= outputs[y.tolist().index(max(y))]
    return(prediction)
```

où :

- `formatage_captcha` ouvre le captcha et le transforme en un tableau de 0 (noir) et de 255 (blanc).
- `decoupage` isole les caractères du captcha et redimensionne les tableaux de chaque caractère.
- `ready_to_test` transforme un tableau en liste de 0 (noir) et de 1 (blanc).
- `outputs` est la liste des 35 classes vue précédemment.
- `W` et `b` sont les matrices récupérées à la fin de l'étape d'entraînement de reconnaissance de caractères unitaires.

III.2 Évaluation et validation

Pour évaluer notre fonction finale et pouvoir valider ou non notre projet nous utilisons le set `validation_set` et calculons l'*accuracy* de notre fonction `predictor` sur ce set. Nous obtenons un score de 74%. Or, nos captchas sont constitués de quatre caractères et nous avions un score de 93% pour la reconnaissance de caractères unitaires et $0.93^4 \approx 0.74$. Ce résultat est donc cohérent et montre que notre méthode de découpage est efficace puisqu'elle n'induit pas d'erreur supplémentaire sur la prédiction.

Conclusion

Nous avons donc réussi à entraîner un modèle de façon à ce qu'il soit capable de reconnaître des captchas de 4 caractères et de même « difficulté » que ceux de notre set de données. Le taux de réussite obtenu, 74%, semble faible en comparaison à celui de 99% que certains réseaux de neurones permettent d'obtenir aujourd'hui pour la reconnaissance de caractères. Ceci vient principalement du fait que notre modèle reconnaît chaque caractère un par un et que le taux de réussite de la reconnaissance du captcha est donc égal au taux de réussite de la reconnaissance d'un caractère à la puissance le nombre de caractères dans le captcha. Néanmoins, le taux de réussite de notre modèle pour la reconnaissance d'un caractère est lui beaucoup plus satisfaisant : 93%. Ce résultat est d'autant plus satisfaisant que notre set de données n'était pas très grand (1140 exemples par classe contre 6000 pour la base de données MNIST par exemple). Toutefois, il semble également juste de noter que certains réseaux de neurones entraînés avec des sets de données beaucoup plus petits sont pourtant plus performants que le nôtre. C'est notamment le cas de certains modèles permettant, à partir de sets de données de 200 exemples par classe, de reconnaître des caractères Devanagari avec un taux de réussite supérieur à 95%. Ainsi, malgré la petite taille de notre set de données, nous aurions tout de même pu améliorer le taux de réussite de notre modèle. Nous aurions, par exemple, pu utiliser un réseau de neurones plus complexe que le modèle Softmax Regression, un CNN (Convolutional Neural Network ou Réseau neuronal convolutif) à plusieurs couches par exemple. Cependant, comme expliqué dans le préambule, nous n'avions aucune connaissance en Deep Learning avant de commencer ce projet et l'objectif était d'en découvrir les bases. Or, la compréhension et l'utilisation de CNN nécessitent d'avoir de bases solides en Deep Learning.

Outre l'amélioration du taux de réussite de notre modèle, notre projet pourrait être prolongé de différentes manières. Premièrement, on pourrait essayer de reconnaître des mots manuscrits, écrits de manière continue, puis des phrases tout entières. La difficulté majeure résiderait là encore dans le découpage de la photo d'un mot en images contenant chacune une des lettres composant le mot, difficulté que la méthode que nous avons utilisée dans notre projet ne permettrait de dépasser. Pour la dépasser, on pourrait utiliser un filtre passe-bas sur la photo du mot afin d'effacer des traits « isolés », notamment ceux joignant les différentes lettres composant le mot. On pourrait également réaliser un seuillage en traçant la courbe représentant le nombre de pixels noirs en fonction de l'abscisse sur l'image du mot. On couperait alors à l'endroit où ce nombre est le plus faible. Ceci nous ramènerait à un problème plus proche de celui que nous avons résolu dans ce projet. Toutefois, aucune de ces méthodes n'est parfaite et il faudrait faire de nombreux tests afin de les parfaire. La méthode du passe-bas pourrait notamment conduire à l'effacement d'autres traits que ceux que l'on souhaite supprimer et la méthode

du seuillage pourrait être mise à défaut lors de la lecture d'images de mots contenant les lettres « u » ou « m ».

Ensuite, on pourrait également poursuivre dans la reconnaissance de captchas, plus compliqués que ceux sur lesquels nous avons travaillés, et plus proches de ceux réellement utilisés aujourd'hui (lettres barrées ou partiellement effacées par exemple). Cela permettrait notamment aux sites Internet voulant sécuriser l'accès à leurs services de tester la « difficulté » de leurs captchas. Une multitude d'autres problèmes concrets nécessitant la reconnaissance de caractères « collés » pourrait servir de prolongement à ce projet.

Ce projet nous a permis de nous rendre compte de l'étendue des bibliothèques Python et des choses qu'elles permettent de réaliser dans le domaine du Deep Learning. Il a également cultivé notre intérêt pour ce domaine, que nous continuerons d'explorer.

Annexe

Code Python : Création du set d'entraînement

```
"""
1ere étape : Associer à chaque captcha de la base son texte (ie son label), puis separer notre nouvelle base (features/labels)
en un train set et un test set.
"""
import os
import os.path
import glob
import random
import pickle

#tout d'abord on recupere les captchas dans le dossier "captchas" (il y en a 9 956)
captchas_list = glob.glob("/Users/Paul-Noel/Desktop/ENSAE/Info/Projet_python/captchas/*.png") #liste de chemins vers chaque captchas

#ensuite on attribue le label du captcha à son chemin
captchas_and_labels = []
for captcha in captchas_list:
    filename = os.path.basename(captcha)
    label = os.path.splitext(filename)[0]
    captchas_and_labels.append((captcha, label))

#captchas_and_labels est maintenant une liste de couple chemin vers le captcha/texte du captcha (ie label du captcha)

#On va maintenant decouper notre base en un train set et un test set
random.shuffle(captchas_and_labels)
captchas_train = captchas_and_labels[:9000]
captchas_test = captchas_and_labels[9000:]

with open('/Users/Paul-Noel/Desktop/ENSAE/Info/Projet_python/captchas_test', 'wb') as fp:
    pickle.dump(captchas_test, fp)

#On travail desormais uniquement sur captchas_train
"""
2eme étape : On va decouper tout les captchas de èmanire à isoler chaque (caractere, label). Ces (caracteres, labels) vont
```

```
34 etre utilise pour entrainer notre fonction de
      reconnaissance de caractere.
35 """
36 import numpy as np
37 from PIL import Image
38
39 #ouverture et transformation des images png de captcha
      en tableaux
40 h, l, r =0, 0, 0
41 captchas_sous_forme_de_tableaux = []
42 for (i, (captcha_path,lbl)) in enumerate(captchas_train)
      :
43     print("[mise sous forme de tableau] processing
          captcha {}/{}".format(i + 1, len(captchas_train))
          )
44     captcha = Image.open(captcha_path)  #on ouvre le
          captcha correspondant au chemin captcha_path
45     tab_captcha = np.array(captcha)     #on le convertie
           en tableau de type RVB 0-255
46     h,l,r=tab_captcha.shape             #on recupere les
           dimensions des captchas (ils sont tous de meme
          taille)
47     captchas_sous_forme_de_tableaux.append((tab_captcha,
           lbl))
48
49 #transformation des tableaux RVB au format teintes de
      gris
50 captchas_gris = []
51 for (i, (tab_captcha, lbl)) in enumerate(
      captchas_sous_forme_de_tableaux):
52     print("[niveaux de gris] processing captcha {}/{}"
          .format(i + 1, len(
          captchas_sous_forme_de_tableaux)))
53     tab_captcha_gris = np.zeros((h, l), dtype="uint8")
54     for i in range(h):
55         for j in range(l):
56             tab_captcha_gris[i][j] = int(0.299*
                tab_captcha[i][j][0] + 0.587*tab_captcha[
                i][j][1] + 0.114*tab_captcha[i][j][2])
57     captchas_gris.append((tab_captcha_gris, lbl))
58
59 #transformation du format teintes de gris au format noir
       blanc
60 captchas_noirs = []
61 for (i, (tab_captcha_gris, lbl)) in enumerate(
      captchas_gris):
62     print("[noir et blanc] processing captcha {}/{}"
          .format(i + 1, len(captchas_gris)))
```

```
63      captcha_noir = np.zeros((h, l), dtype="uint8")
64      for i in range(h):
65          for j in range(l):
66              if tab_captcha_gris[i][j]>200:
67                  captcha_noir[i][j] = 255
68              else:
69                  captcha_noir[i][j] = 0
70      captchas_noirs.append((captcha_noir, lbl))
71
72  #decoupage des captchas (ie creation d'une bdd (lettre/
        label))
73  letters = []
74  for (i, (captcha_noir, lbl)) in enumerate(captchas_noirs
        ):
75      print("[édcoupage] processing captcha {}/{}".format(
            i + 1, len(captchas_noirs)))
76      frontieres = []  #liste contenant les indices des
            colonnes constituant une frontieres entre deux
            caracteres
77      blanc = True
78      noir = False
79      for j in range(l):
80          if blanc:
81              i=0
82              while blanc and i<h:
83                  if captcha_noir[i][j] == 0:
84                      frontieres.append(j)
85                      blanc = False
86                      noir = True
87                  i=i+1
88          else: #cad si noir, on cherche la frontiere de
                droite
89              i = 0
90              ok = True   #si ok alors on a que des pixels
                    blancs sur la colonne
91              while ok and i<h:
92                  if captcha_noir[i][j] == 0:
93                      ok = False
94                  i+=1
95              if ok:
96                  frontieres.append(j)
97                  blanc = True
98                  noir = False
99      #on connait maintenant les indices des contours des
            caracteres du captcha, on peut proceder au
            decoupage
100     split_captcha = []
101     for i in range(0, len(frontieres)-1, 2):
```

```
102         fg, fd = frontieres[i], frontieres[i+1] #il s'
               agit des indices des colonnes encerclant le
               premier caractere (ou groupe de caracteres)
103         if (fd - fg) < 0.9*h:    #dans ce cas cela
               signifie qu'il n'y a qu'un seul caractere (et
                pas deux colles)
104             split_captcha.append(np.array([captcha_noir[
                   i][fg-1:fd+1] for i in range(h)]))
105         else: #ici on traite le cas ou deux caracteres
               sont colles et n'ont pu etre separes
106             m = int((fd + fg)/2)
107             split_captcha.append(np.array([captcha_noir[
                   i][fg-1:m+1] for i in range(h)]))
108             split_captcha.append(np.array([captcha_noir[
                   i][m:fd+1] for i in range(h)]))
109 #on associe maintenant chaque lettre à son label
110     if len(split_captcha) == 4:  #sinon cela signifie
           que le decoupage n'a pas bien marche, on
           utilisera donc pas ce captcha pour entrainer
           notre modele car il est de mauvaise qualite
111         for k in range(4):
112             letters.append((split_captcha[k], lbl[k]))
113
114 import pickle
115 with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
       Projet_python/letters_pckl', 'wb') as fp:
116     pickle.dump(letters, fp)
117
118 #on enregistre notre bdd d'entrainement pour notre
       modele de reconnaissance de caractere
119 for (i, (letter, lbl)) in enumerate(letters):
120     print("[écration bdd letters] processing letter {}/
           {}".format(i + 1, len(letters)))
121     im = Image.fromarray(letter)
122     path = "/Users/Paul-Noel/Desktop/ENSAE/Info/
           Projet_python/letters/" + lbl + '(' + str(i) + ')
           ' + ".png"
123     im.save(path)
124
125 """
126 3eme étape : redimensionner nos image pour avoir une bdd
       utilisable
127 """
128 #il reste à redimensionner ttes les lettres de la bdd
       letters pour qu'elles aient ttes la meme dimensions
129 #tout d'abord on determine la taille moyenne des lettres
       que l'on vient d'extraire
130 ls = []  #tableau regroupant les largeurs des lettres
```

```
131 for (letter, lbl) in letters:
132     ls.append(letter.shape[1])
133 mean_l = int(np.mean(ls))
134 mean_h = h
135 
136 #on redimensionne alors toute les lettres en dimension (
    mean_h, mean_l)
137 list_letters = glob.glob("/Users/Paul-Noel/Desktop/ENSAE
    /Info/Projet_python/letters/*.png")
138 for (i, letter) in enumerate(list_letters):
139     print("[redimensionnement] processing letter {}/{}"
        .format(i + 1, len(list_letters)))
140     img = Image.open(letter)
141     img = img.resize((mean_l, mean_h), Image.ANTIALIAS)
142     name = os.path.basename(letter)
143     path = "/Users/Paul-Noel/Desktop/ENSAE/Info/
        Projet_python/letters_resized/" + name
144     img.save(path)
145 
146 #on realise qu'apres redimensionnement, certaines
    lettres ont des contour un peu flou.
147 #on va donc leur appliquer le meme traitement que celui
    pour passer de teintes de gris à noir et blanc
148 list_blurred_letters = glob.glob("/Users/Paul-Noel/
    Desktop/ENSAE/Info/Projet_python/letters_resized/*
    .png")
149 for (i, letter) in enumerate(list_blurred_letters):
150     print("[édfloutage] processing letter {}/{}".format(
        i + 1, len(list_blurred_letters)))
151     img = Image.open(letter)
152     tab_img = np.array(img)
153     clear_img = np.zeros((mean_h, mean_l), dtype="uint8"
        )
154     for i in range(mean_h):
155         for j in range(mean_l):
156             if tab_img[i][j]>150:
157                 clear_img[i][j] = 255
158             else:
159                 clear_img[i][j] = 0
160     name = os.path.basename(letter)
161     path = "/Users/Paul-Noel/Desktop/ENSAE/Info/
        Projet_python/clear_letters/" + name
162     Image.fromarray(clear_img).save(path)
163 
164 """
165 4eme étape : mettre les images des lettres et leur label
    sous forme de liste
166 """
```

```
167
168 h, l = 24, 12 #tailles de nos images dans le dossier
    clear_letters
169
170 outputs = ['1', '2', '3', '4', '5', '6', '7', '8', '9',
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
171             'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
                    'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
172
173 features_labels = []
174 list_features = glob.glob("/Users/Paul-Noel/Desktop/
    ENSAE/Info/Projet_python/clear_letters/*.png")
175 for (ii, feature) in enumerate(list_features):
176     print("[conversion letters and label to lists]
        processing letter {}/{}".format(ii + 1, len(
        list_features)))
177     img = Image.open(feature)
178     tab = np.array(img).tolist()
179     lettre_sous_forme_de_liste = []
180     for i in range(h):
181         for j in range(l):
182             lettre_sous_forme_de_liste.append(tab[i][j]/
                255)
183     lettre_sous_forme_de_liste = np.array(
        lettre_sous_forme_de_liste, dtype='float32')
184     label_dict = { _ : 0 for _ in outputs}
185     label = list(os.path.basename(feature))[0]
186     label_dict[label] = 1
187     label_vect = np.array(list(label_dict.values()))
188     features_labels.append((lettre_sous_forme_de_liste,
        label_vect))
189
190 #on sauvegarde features_labels sur le disque pour
    pouvoir les utiliser dans la partie II : entrainement
     du model
191 import pickle
192 with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
    Projet_python/features_and_labels', 'wb') as fp:
193     pickle.dump(features_labels, fp)
194
195 #On a desormais une bdd (features_labels) propre avec un
     label pour chaque caractere sous forme de liste.
    Pret à l'emploi pour
196 #entrainer notre modele
```

Code Python : Entraînement du modèle

```
1 #on recupere notre liste de couple (feature/label)
```

```
    construite dans la partie I
import pickle
with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
    Projet_python/features_and_labels', 'rb') as fp:
     features_labels = pickle.load(fp)

m = int((90/100)*len(features_labels))
features_labels_train = features_labels[:m]
features_labels_test = features_labels[m:]


#on va utiliser la librairie TensorFlow
import tensorflow as tf
import numpy as np
import random

h, l = 24, 12 #tailles de nos images dans le dossier
    clear_letters
N = h*l

#creation d'un placeholder (ie variable à laquelle on
    donnera les valeurs des images)
x = tf.placeholder(tf.float32, [None, N])

#initialisation du poids w et du biais b
W = tf.Variable(tf.zeros([N, 35]))      #35 car 9
    chiffres (pas le 0) + 26 lettres
b = tf.Variable(tf.zeros([35]))

#écration de notre modele (le but va etre de trouver w
    et b tq y se rapproche le plus du label associe à x)
y = tf.nn.softmax(tf.matmul(x, W) + b)

#il est maintenant temps de determiner w et b grace à la
     bdd mnist
#tout d'abord on definie un distance pour determiner à
    quel point notre prediction est loin du label
#on cree un nouveau placeholder pour le label
y_ = tf.placeholder(tf.float32, [None, 35])

#puis on definie la distance entre y_(le label) et y (la
     prediction)
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ *
    tf.log(y), reduction_indices=[1]))


#on definie maintenant l'etape d'entrainement de notre
    modele
```

```
39 train_step = tf.train.GradientDescentOptimizer(0.5)
       .minimize(cross_entropy)
40
41 #on entraine enfin reellement notre modele
42 #pour cela on ouvre une "session", c'est la procedure
       dans tensorflow
43 sess = tf.InteractiveSession()
44 tf.global_variables_initializer().run()
45
46 #on fait tourner 1000 fois l'algo de descente de
       gradient
47 for i in range(1000):
48     random.shuffle(features_labels_train)
49     batch_xs, batch_ys = np.array(list(zip(\*
           features_labels_train))[0]), np.array(list(zip(\*
           features_labels_train))[1])
50     sess.run(train_step, feed_dict={x: batch_xs, y_:
           batch_ys})
51     print("Loss {}/1000 : {}".format(i + 1, sess.run(
           cross_entropy, feed_dict={x: batch_xs, y_:
           batch_ys})))
52
53 #evaluation du modele
54 #evaluer une seule image :
55 correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(
       y_,1))
56
57 #predire une seule image
58 prediction = tf.argmax(y)
59
60 #determiner l'accuracy :
61 accuracy = tf.reduce_mean(tf.cast(correct_prediction,
       tf.float32))
62
63 #Finalement, on observe l'accuracy sur nos donnees de
       test pour verifier notre modele
64 features_test, labels_test = np.array(list(zip(\*
       features_labels_test))[0]), np.array(list(zip(\*
       features_labels_test))[1])
65
66 print("Accuracy : {}".format(100 * sess.run(accuracy,
       feed_dict={x: features_test, y_: labels_test})))
67 # on a un score de 93\%
68
69 #on sauvegarde les poids (W) et le biais (b) pour les
       reutiliser dans utilisation_du_model.py sans avoir à
       refaire tourner l'entrainement à chaque fois
70 import pickle
```

```
71 with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
      Projet_python/weigths', 'wb') as fp:
72     pickle.dump(sess.run(W), fp)
73 with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
      Projet_python/bias', 'wb') as fp:
74     pickle.dump(sess.run(b), fp)
```

Code Python : Fonction finale

```
1  outputs = ['1', '2', '3', '4', '5', '6', '7', '8', '9',
      'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
2             'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
                  'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
3
4  import numpy as np
5  from PIL import Image
6  import os
7  import os.path
8  import glob
9  import random
10 import pickle
11
12 #on recupere tout d'abord les poids (W) et le biais (b)
      du modele que l'on vient d'entrainer
13 with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
      Projet_python/weigths', 'rb') as fp:
14     W = np.array(pickle.load(fp))
15 with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
      Projet_python/bias', 'rb') as fp:
16     b = np.array(pickle.load(fp))
17
18 #on construit ensuite toute les fonctions dont on va se
      servir
19 def softmax(y):
20     n = len(y)
21     z = np.zeros(n)
22     S = 0
23     for yi in y:
24         S+=np.exp(yi)
25     for (i, yi) in enumerate(y):
26         zi = np.exp(yi)/S
27         z[i] = zi
28     return(z)
29
30 def formatage_captcha(img): #prend en entree un chemin
      vers une image et renvoie un tableau correspondant en
       noir et blanc (format 0 ou 1)
31     tab_img = np.array(Image.open(img))
```

```
32      h, l, r = tab_img.shape
33      tab_img_noir_blanc = np.zeros((h, l), dtype='float32
            ')
34      for i in range(h):
35          for j in range(l):
36              pixel = int(0.299*tab_img[i][j][0] + 0.587*
                    tab_img[i][j][1] + 0.114*tab_img[i][j
                    ][2])
37              if pixel>200:
38                  tab_img_noir_blanc[i][j] = 255
39              else:
40                  tab_img_noir_blanc[i][j] = 0
41      return(tab_img_noir_blanc)
42
43  def decoupage(captcha): #prend en entree un captcha sous
        forme de tableau et renvoie les caracteres du
        captcha sous forme de tableau
44      h, l = captcha.shape
45      frontieres = []  #liste contenant les indices des
            colonnes constituant une frontieres entre deux
            caracteres
46      blanc = True
47      noir = False
48      for j in range(l):
49          if blanc:
50              i=0
51              while blanc and i<h:
52                  if captcha[i][j] == 0:
53                      frontieres.append(j)
54                      blanc = False
55                      noir = True
56                  i=i+1
57          else: #cad si noir, on cherche la frontiere de
                droite
58              i = 0
59              ok = True   #si ok alors on a que des pixels
                    blancs sur la colonne
60              while ok and i<h:
61                  if captcha[i][j] == 0:
62                      ok = False
63                  i+=1
64              if ok:
65                  frontieres.append(j)
66                  blanc = True
67                  noir = False
68      #on connait maintenant les indices des contours des
            caracteres du captcha, on peut proceder au
            decoupage
```

```
69      split_captcha = []
70      for i in range(0, len(frontieres)-1, 2):
71          fg, fd = frontieres[i], frontieres[i+1] #il s'
               agit des indices des colonnes encerclant le
               premier caractere (ou groupe de caracteres)
72          if (fd - fg) < 0.9*h:    #dans ce cas cela
               signifie qu'il n'y a qu'un seul caractere (et
                pas deux colles)
73              split_captcha.append(np.array([captcha[i][
                   fg-1:fd+1] for i in range(h)]))
74          else: #ici on traite le cas ou deux caracteres
               sont colles et n'ont pu etre separes
75              m = int((fd + fg)/2)
76              split_captcha.append(np.array([captcha[i][
                   fg-1:m+1] for i in range(h)]))
77              split_captcha.append(np.array([captcha[i][m:
                   fd+1] for i in range(h)]))
78  #on associe maintenant chaque lettre à son label
79      if len(split_captcha) == 4:  #sinon cela signifie
           que le decoupage n'a pas bien marche
80          letters = []
81          for k in range(4):
82              img = Image.fromarray(split_captcha[k])
83              img = img.resize((12, 24), Image.ANTIALIAS)
84              letters.append(np.array(img))
85          return(letters)
86      else:
87          return('Trop écompliqu àé dcouper')
88
89  def ready_to_test(img):  #passage d'un tableau à une
       liste et normalisation
90      L = []
91      h, l = img.shape
92      for i in range(h):
93          for j in range(l):
94              if img[i][j]>150:
95                  L.append(1)
96              else:
97                  L.append(0)
98      return(np.array(L, dtype='float32'))
99
100
101
102 #on peut enfin coder notre fonction de prediction
103 def predictor(captcha):     #en entree on attend un
       chemin vers un captcha dont on veut predire le texte
104     captcha_noir_blanc = formatage_captcha(captcha)
105     letters = decoupage(captcha_noir_blanc)
```

```
106     if isinstance(letters, str):
107         return('Trop complique à decoder')
108     else:
109         lettres_a_predire = []
110         for letter in letters:
111             lettres_a_predire.append(ready_to_test(
                    letter))
112         prediction = ''
113         for x in lettres_a_predire:
114             y = softmax(np.dot(x, W) + b)
115             prediction+= outputs[y.tolist().index(max(y)
                    )]
116         return(prediction)
```

Code Python : Test du modèle

```
1  import numpy as np
2  from PIL import Image
3  import os
4  import os.path
5  import glob
6  import random
7  import pickle
8  from utilisation_du_model import predictor
9
10 def accuracy(liste_path_to_captchas_labeled):
11     n_correct = 0
12     N = len(liste_path_to_captchas_labeled)
13     for (i, (captcha, lbl)) in enumerate(
           liste_path_to_captchas_labeled):
14         print(predictor(captcha), lbl)
15         if predictor(captcha) == lbl:  #not isinstance(
               predictor(captcha), str) and
16             n_correct+=1
17             print("Captcha {}/{} correctly predicted"
                   .format(i + 1, N))
18         else:
19             print("Captcha {}/{} Fail !".format(i + 1, N
                   ))
20     return(n_correct/N)
21
22 with open('/Users/Paul-Noel/Desktop/ENSAE/Info/
       Projet_python/captchas_test', 'rb') as fp:
23     captchas = pickle.load(fp)
24
25 print(accuracy(captchas))
26 #on a un score de .74 logique car notre algo de
       reconnaissance des lettres a un score de .93 et .74 =
```

```
    .93^4 (4 lettres par captcha)
```

Bibliographie

— Tensorflow : https ://www.tensorflow.org
— The MNIST Database, Yann LeCun, Corinna Cortes, Christopher J.C. Burges
— Udacity course : Intro to Deep Learning