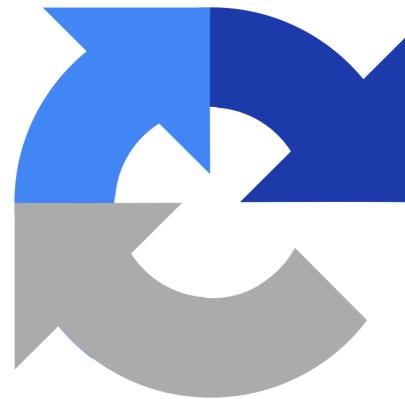


Résolution de Captcha par Deep Learning

Samy BENRAHHALATE
Amine KHARROUBI



reCAPTCHA

SOMMAIRE:

I.Introduction : Définition d'un Captcha.....	3 à 5
II. Création d'un modèle structuré.....	6 à 7
III.Segmentation.....	7 à 9
IV.Training du DataSet.....	9 à 10
V. Solveur du Captcha.....	10 à 12
VI. Conclusion.....	12 à 13

I/ Définition d'un Captcha :

Lorsque nous naviguons sur Internet, nous rencontrons fréquemment une authentification qui protège les sites Webs contre les robots en demandant à l'utilisateur d'effectuer une série de tests afin de vérifier qu'il est bien humain. Cette authentification, c'est le Captcha (**C**ompletely **A**utomated **P**ublic **T**uring **T**est to Tell **C**omputers and **H**umans **A**part). Inventé en 2000, cette version du programme vous demande de taper ce que vous voyez dans l'image CAPTCHA, qu'il s'agisse d'un nombre, d'un mot ou de plusieurs mots.

L'objectif principal du CAPTCHA est d'empêcher l'automatisation de l'accès aux formulaires sur Internet par les robots. En d'autres termes, il correspond à un test utilisé en informatique pour déterminer si l'utilisateur est humain ou non.

Le CAPTCHA est simplement, un texte avec des bruits, des couleurs différentes, des symboles tournés ou modifiés de différentes manières pour rendre sa reconnaissance plus difficile à un ordinateur. Parfois, même pour un humain, il peut s'avérer compliqué de reconnaître ce qui est écrit sur l'image, il est donc assez difficile de créer des robots capables de freiner ces images.

Quelle que soit l'évolution du CAPTCHA, il y aura toujours des individus qui trouveront des méthodes pour le casser et passer au travers de sa sécurité. L'une des méthodes les plus connues consiste à utiliser l'apprentissage automatique, et nous allons nous concentrer sur un type spécifique de réseau neuronal appelé **réseau neuronal convolutif (CNN)**.

Le fonctionnement du CNN est très semblable à celui de notre cerveau, qui est capable de reconnaître les choses et de différencier un objet d'un autre. Pour vous donner une explication claire, lorsque vous regardez l'image ci-dessous, vous pouvez immédiatement dire que ces deux animaux ne sont pas de la même espèce, mais vous allez vous demander, comment ? La réponse serait, c'est évident...



Cela vient du fait que nous avons vu des millions de photos que ce soit de chiens, de chats ou d'autres animaux, et que nous les avons vus dans la vie réelle. Durant notre enfance, on nous a appris qu'ils étaient différents. Puis, notre cerveau a lentement compris les distinctions entre ces deux animaux. Nos souvenirs nous donnent ainsi, la capacité de pouvoir reconnaître correctement lequel est un chien et lequel est un chat en distinguant leur nombreuses différences.

En se basant sur la même idée, nous allons faire quasiment de même pour notre réseau neuronal de détection de CAPTCHA car notre ordinateur ne perçoit pas l'image de la même façon que nous. En effet, il voit des groupes de symboles qui indiquent une intensité de couleur sur ce pixel particulier. Si nous avons une image RVB, l'une des façons de les afficher serait sous forme de tableau RGBA. Les couches dans les CNN sont spéciales car elles sont organisées en 3 dimensions: la largeur, la hauteur et la profondeur, ce qui nous permet ainsi d'intégrer une image dans le réseau. La couche finale, qui est la couche entièrement connectée, nous dit ce qu'elle prédit.

Pour que tout soit plus clair, voici un exemple de photo : ce que nous voyons et ce que l'ordinateur voit dans la même image, dans cette photo nous voyons un chiot :



Si nous voulons voir la photo à la manière de l'ordinateur, nous pouvons utiliser ce script simple sur cette image :

```
from PIL import Image

im = Image.open("puppy.jpg", "r")
pix_val = list(im.getdata())
print(pix_val)
```

Le résultat que nous obtenons sont des milliers de nombres, qui représentent chaque pixel de la photo sous forme de valeur RVB :

Une fois le concept de CNN introduit, nous allons appliquer cette méthode pour décomposer le CAPTCHA et voir avec quelle précision nous pouvons le résoudre.

II/ Création d'un modèle structuré pour casser le CAPTCHA

Focalisons à nouveau le CAPTCHA. Supposons qu'il s'agisse d'une combinaison prise dans les 26 lettres alphabets français et de chiffres de 0 à 9. À la fin, avec notre méthode, nous serons donc en mesure de résoudre les CAPTCHA avec différentes quantités de symboles.

La première étape de notre projet consiste à générer un dataset et d'effectuer une série d'entraînement sur notre modèle. Notre objectif est de détecter les objets de notre modèle d'entraînement qui nous serviront à décoder l'image des captcha.

Au début, nous allons essayer de détecter non pas tous les captcha à la fois, mais nous allons essayer de trouver combien de symboles nous avons dans notre image et ensuite essayer de les reconnaître.

Pour pouvoir résoudre le captcha, nous aurons besoin de nombreux exemples résolus en vue de l'entraînement. Pour cela 2 possibilités s'offrent à nous:

- Trouver un dataset qui en fournit;
 - Utiliser un Captcha dont nous avons le code source pour pouvoir le manipuler en vue de générer les données d'entraînement.

La génération du dataset demandera de développer 3 méthodes :

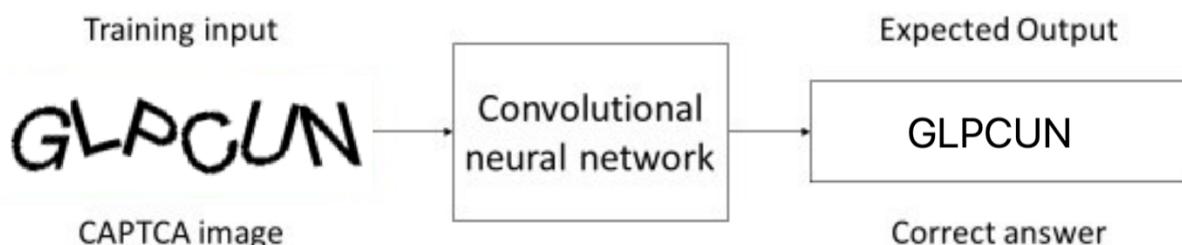
- L'une chargée de déclencher le nombre voulu de fois le captcha dans sa page
- L'une chargée d'injecter la bonne réponse avec l'image générée
- Et la dernière de récupérer les images avec les bonnes réponses

Pour utiliser n'importe quel système d'apprentissage automatique, nous devons collecter des données d'entraînement. Pour casser un système CAPTCHA, nous voulons un modèle entraîné qui fonctionne en prenant une image captcha et nous donne son résultat.

GLPCUN

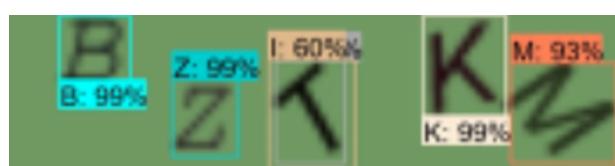
>Résultat attendu : **GLPCUN**

Lorsque nous aurons nos données d'entraînement, nous pourrons les utiliser pour entraîner un réseau de neurones convolutif qui ressemble à ceci:



Avec suffisamment de données de formation collectées, notre approche doit fonctionner, mais nous pouvons rendre le problème encore plus simple à résoudre. Plus le problème est simple, moins il y a de données d'entraînement et moins il nous faudra de puissance et de temps de calcul pour le résoudre. Nous savons que les images CAPTCHA sont toujours constituées d'une certaine quantité de symboles séparés. Si nous pouvions d'une manière ou d'une autre diviser l'image afin que chaque lettre soit une image distincte, alors nous n'avons besoin que d'entraîner le réseau neuronal à reconnaître une seule lettre à la fois

Nous apprenons donc à notre CNN à détecter une seule lettre à partir d'un captcha et non une chaîne complète à la fois, de cette façon, nous aurons besoin de beaucoup moins de données d'entraînement.



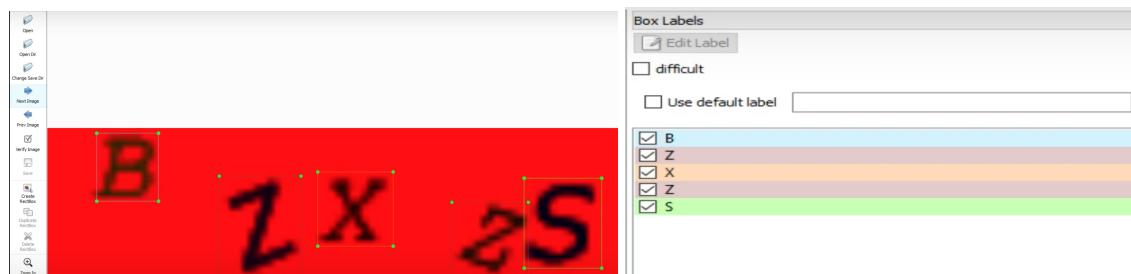
Lorsque nous donnerons notre image CAPTCHA et que la sortie nous donne une autre image CAPTCHA avec des détections. Mais les détections ne sont pas toujours sûres à 100%, ce pourcentage de détection dépend des données d'entraînement que nous utilisons. De plus, notre CNN peut détecter encore plus de symboles qu'il n'y en a sur l'image CAPTCHA. Comme vous pouvez le voir sur l'image ci-dessus, notre modèle a vu la lettre I à 60% au lieu de la lettre T. Nous utiliserons un filtre de détection pour résoudre ce problème.

III/ Segmentations : Extractions et Reconnaissance

Nous allons collecter des données d'entraînement et entraîner notre modèle. Nous utiliserons le modèle de détection d'objet TensorFlow pour déchiffrer les images Captcha

Dans un premier temps, nous essaierons de détecter un CAPTCHA non complet à la fois, mais nous essaierons de savoir combien de symboles nous avons dans notre image, puis nous essaierons de les reconnaître.

La première étape consiste à rassembler quelques centaines d'images CAPTCHA que l'on souhaite déchiffrer. Avec toutes les images rassemblées, il faut étiqueter des symboles dans chaque image CAPTCHA.



Voici un exemple de résultat après avoir classifié et identifier la lettre "W" de notre dataset.



Avec suffisamment de données de formation collectées, nous pouvons construire notre “Neural Network” et l’entraîner.

III/ Training :

Une fois que nous aurons étiqueté nos images, nous allons les séparer en groupes de formation et de test. Pour ce faire, on copie simplement environ 20% de nos images et leurs fichiers d’annotation XML dans un nouveau répertoire appelé test, et les autres dans un nouveau répertoire appelé train.

Avec les images étiquetées, il est possible de générer les TFRecords qui servent de données d’entrée au modèle d’entraînement TensorFlow. En utilisant les scripts [xml_to_csv.py](#) et [generate_tfrecord.py](#). Les données .xml de l’image seront utilisées pour créer des fichiers .csv contenant toutes les données pour les images de formation et de test.

On génère TFRecord en lançant le fichier [generate_tfrecord.bat](#), que l’on a récupéré du code source.

```
python generate_tfrecord.py --csv_input=CAPTCHA_images\train_labels.csv --image_
python generate_tfrecord.py --csv_input=CAPTCHA_images\test_labels.csv --image_d
```

Ces lignes génèrent un fichier train.record et un test.record dans le dossier de formation. Ils seront utilisés pour entraîner la détection d’objets.

On lance l’entraînement grâce à “faster_rcnn_inception_v2_coco” qui est un modèle fourni par ce code source et qui utilise un algorithme de classification RNN. Si tout a été configuré correctement, TensorFlow va initialiser le training. L’initialisation peut prendre jusqu’à 60 secondes avant que l’entraînement réel ne commence. Lorsque le training commence, voilà à quoi ressemble le terminal :

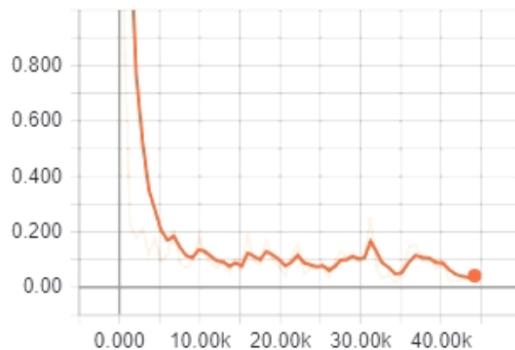
```
I0110 09:06:23.001857 8464 tf_logging.py:115] global step 11: loss = 3.4306 (0.380 sec/step)
INFO:tensorflow:global step 12: loss = 4.1787 (0.297 sec/step)
I0110 09:06:23.301581 8464 tf_logging.py:115] global step 12: loss = 4.1787 (0.297 sec/step)
INFO:tensorflow:global step 13: loss = 4.3128 (0.333 sec/step)
I0110 09:06:23.637682 8464 tf_logging.py:115] global step 13: loss = 4.3128 (0.333 sec/step)
INFO:tensorflow:global step 14: loss = 1.7454 (0.262 sec/step)
I0110 09:06:23.901975 8464 tf_logging.py:115] global step 14: loss = 1.7454 (0.262 sec/step)
INFO:tensorflow:global step 15: loss = 3.7867 (0.247 sec/step)
I0110 09:06:24.152833 8464 tf_logging.py:115] global step 15: loss = 3.7867 (0.247 sec/step)
INFO:tensorflow:global step 16: loss = 3.5090 (0.257 sec/step)
I0110 09:06:24.416132 8464 tf_logging.py:115] global step 16: loss = 3.5090 (0.257 sec/step)
INFO:tensorflow:global step 17: loss = 3.1237 (1.007 sec/step)
I0110 09:06:25.427454 8464 tf_logging.py:115] global step 17: loss = 3.1237 (1.007 sec/step)
INFO:tensorflow:global step 18: loss = 3.6378 (0.316 sec/step)
I0110 09:06:25.752712 8464 tf_logging.py:115] global step 18: loss = 3.6378 (0.316 sec/step)
INFO:tensorflow:global step 19: loss = 3.7932 (0.239 sec/step)
I0110 09:06:25.995068 8464 tf_logging.py:115] global step 19: loss = 3.7932 (0.239 sec/step)
INFO:tensorflow:global step 20: loss = 3.6955 (0.284 sec/step)
I0110 09:06:26.288840 8464 tf_logging.py:115] global step 20: loss = 3.6955 (0.284 sec/step)
INFO:tensorflow:global step 21: loss = 3.8363 (0.318 sec/step)
I0110 09:06:26.609981 8464 tf_logging.py:115] global step 21: loss = 3.8363 (0.318 sec/step)
INFO:tensorflow:global step 22: loss = 3.7909 (0.238 sec/step)
I0110 09:06:26.853330 8464 tf_logging.py:115] global step 22: loss = 3.7909 (0.238 sec/step)
INFO:tensorflow:global step 23: loss = 3.7022 (0.227 sec/step)
```

Grâce à la commande suivante :

```
C:\TensorFlow\research\object_detection>tensorboard --logdir=CAPTCHA_training_dir
```

On obtient le LOSS Graph suivant :

Loss/BoxClassifierLoss/classification_loss
tag: Losses/Loss/BoxClassifierLoss/classification_loss



Ainsi, il est important de noter qu'on peut toujours vérifier le LOSS sur le serveur local pendant l'exécution de tensorboard, et entraîner le modèle jusqu'à ce qu'on voit que le graphique LOSS ne diminue plus. Ainsi, on a entraîné notre modèle de détection CAPTCHA pour 40 000 étapes, et comme vous pouvez le voir sur le graphique, il ne diminue plus aux environs de 40 000 étapes. Donc, dans ce cas, il n'est pas utile d'entraîner ce modèle plus longtemps, il est préférable d'obtenir plus de données d'entraînement, et de l'entraîner avec de nouvelles données, dans ce cas, il sera plus précis.

IV/ Solveur du Captcha :

Le code final permet de saisir tous les symboles du CAPTCHA, vérifie l'ordre des symboles, la précision de la détection et les chevauchements. Nous utilisons tous ces composants pour écrire le solveur CAPTCHA final:

```

1  # Imports
2  import cv2
3  import numpy as np
4  import os
5  import sys
6  # run on CPU
7  os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
8  import tensorflow as tf
9  from distutils.version import StrictVersion
10 from collections import defaultdict
11
12 # title of our window
13 title = "CAPTCHA"
14
15 # Env setup
16 from object_detection.utils import ops as utils_ops
17 from object_detection.utils import label_map_util
18 from object_detection.utils import visualization_utils as vis_util
19
20
21 # Model preparation
22 PATH_TO_FROZEN_GRAPH = 'CAPTCHA_frozen_inference_graph_v2.pb'
23 # List of the strings that is used to add correct label for each box.
24 PATH_TO_LABELS = 'CAPTCHA_labelmap.pbtxt'
25 NUM_CLASSES = 37
26
27
28 # Load a (frozen) Tensorflow model into memory.
29 label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
30 categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
31 category_index = label_map_util.create_category_index(categories)
32
33 detection_graph = tf.Graph()
34 with detection_graph.as_default():
35     od_graph_def = tf.GraphDef()
36     with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
37         serialized_graph = fid.read()
38         od_graph_def.ParseFromString(serialized_graph)
39         tf.import_graph_def(od_graph_def, name='')
40

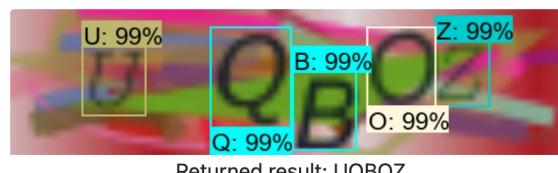
```

```

41 # Detection
42 def Captcha_detection(image):
43     with detection_graph.as_default():
44         with tf.Session(graph=detection_graph) as sess:
45             # Open image
46             image_np = cv2.imread(image)
47             # To get real color we do this:
48             image_np = cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
49             # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
50             image_np_expanded = np.expand_dims(image_np, axis=0)
51             # Actual detection.
52             image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
53             boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
54             scores = detection_graph.get_tensor_by_name('detection_scores:0')
55             classes = detection_graph.get_tensor_by_name('detection_classes:0')
56             num_detections = detection_graph.get_tensor_by_name('num_detections:0')
57             # Visualization of the results of a detection.
58             (boxes, scores, classes, num_detections) = sess.run(
59                 [boxes, scores, classes, num_detections],
60                 feed_dict={image_tensor: image_np_expanded})
61             vis_util.visualize_boxes_and_labels_on_image_array(
62                 image_np,
63                 np.squeeze(boxes),
64                 np.squeeze(classes).astype(np.int32),
65                 np.squeeze(scores),
66                 category_index,
67                 use_normalized_coordinates=True,
68                 line_thickness=2)
69             # Show image with detection
70             cv2.imshow(title, cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB))
71             #cv2.imwrite("Predicted_captcha.jpg", cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB))
72
73
74 Captcha_detection("11.jpg") #11 représente captcha.jpg à déchiffrer
75

```

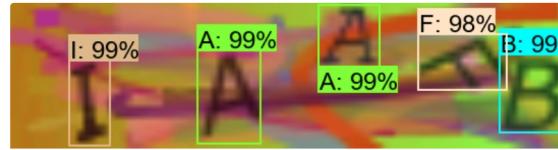
Après compilation, voici les différents résultats attendus en fonction des captcha entrée en paramètre :



Returned result: UQBOZ



Returned result: POQQP



Returned result: IAAFB

V/ Conclusion :

Pour conclure, nous avons trouvé ce projet particulièrement intéressant, nous permettant d'étudier en profondeur le fonctionnement d'un captcha et comment contourner sa sécurité avec l'utilisation du deep learning.

Via l'implémentation de ce code, nous ne pouvons pas résoudre les CAPTCHAS très difficiles, qui sont composés de peu de mots. Cependant grâce à l'utilisation du CNN(réseau de neurone convolutif), nous pouvons résoudre approximativement 95% des CAPTCHA utilisés sur Internet, à l'exception du reCAPTCHA de Google.

Pour résumé, pour "break" un captcha grâce au deep learning, il faut passer par 4 étapes :

- Création du Dataset
- Simplifier le problème grâce à la segmentation
- Construire et Entraîner le Réseau de Neurone
- Utiliser le Training modèle afin de résoudre davantage de CAPTCHA.