Détecteur de nombre

# Qu’est-ce que c’est ?

Le détecteur de nombre permet de lire un nombre dans une image.

# A quoi ça sert ?

Ce projet peut servir à beaucoup de tâches fastidieuses. L’idée originale était de faire un programme qui détecte automatiquement à qui appartient une photo

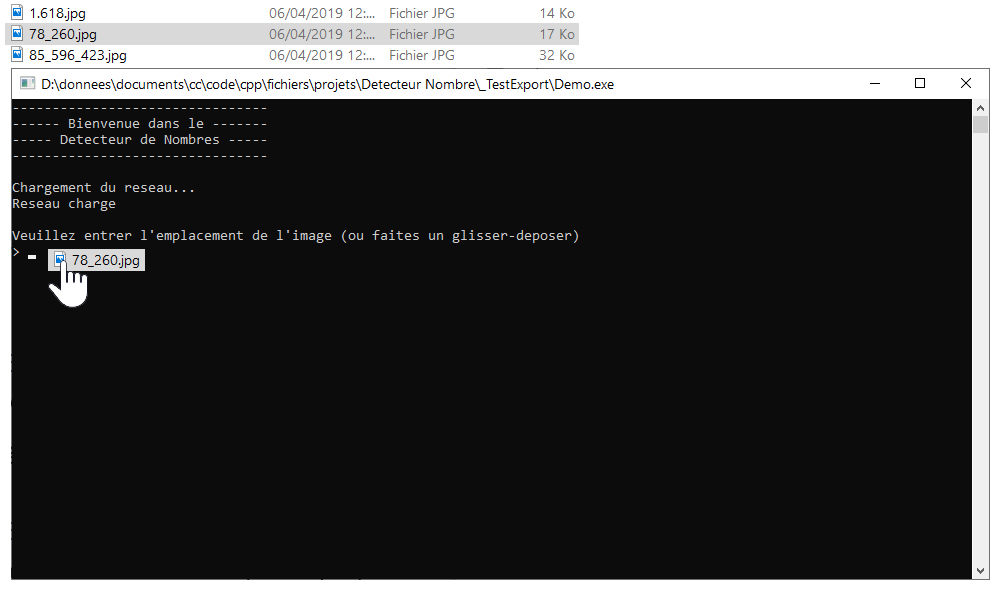
Par exemple quand un photographe prend en photo des coureurs dans une course, il doit envoyer plusieurs milliers de photos à chaque coureur, il doit lire le numéro de dossard, trouver l’adresse email correspondante puis l’envoyer ce qui prend généralement plusieurs jours.

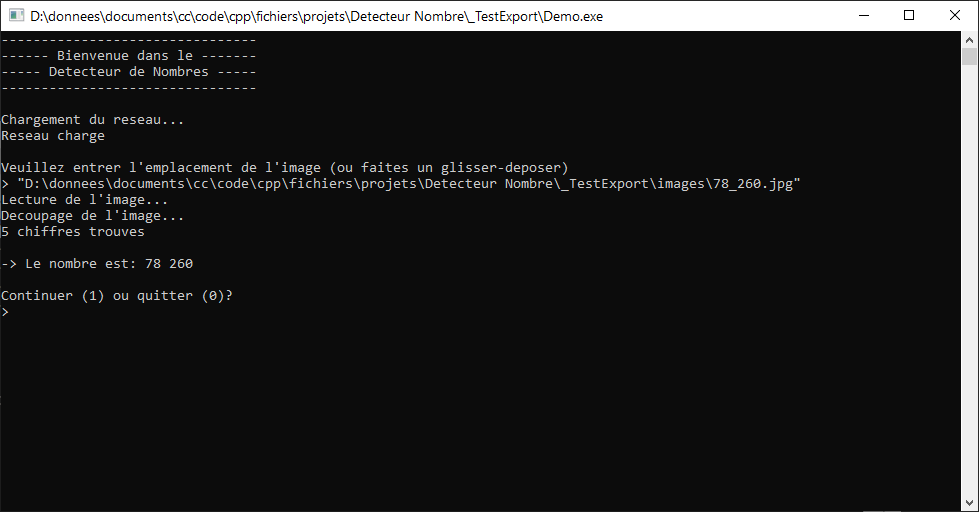
Il pourrait permettre également aux personnes mal voyantes de se repérer dans la vie de tous les jours.

Finalement ce programme est plus adapté à cette dernière fonction car il est entraîné exclusivement pour l’écriture manuscrite.

# Comment l’utiliser ?

Nous pouvons l’utiliser avec deux façons, il y a deux programmes, le premier se nomme Demo.

Il fait la même chose que l’autre programme mais plus rapidement et simplement, il suffit juste de glisser-déposer l’image et d’appuyer sur *entrée* : 



Le deuxième programme permet d’entraîner l’intelligence artificielle avec d’autres architectures de réseau de neurones et aussi de prédire juste un seul chiffre.

# Comment ça marche ?

Ce projet a été fait en deux fois, la première étant les maths et la deuxième la programmation.

# Maths

Les maths étant très complexes, nous allons résumer.

Tout d’abord parlons de comment fonctionne une intelligence artificielle en général, elle prend en entrée des paramètres, les passe dans une grande fonction puis retourne le résultat.

Ici l’entrée est une matrice de 784 valeurs car chaque image a une taille de 28 \* 28 soit 784 pixels. La sortie est une matrice de 10 valeurs, ce n’est pas une seule valeur car il est mieux de prédire des probabilités, c’est-à-dire que le 4e élément de la matrice est la probabilité que ça soit un 4, l’indice possédant la valeur maximale de la matrice est la prédiction.

Nous allons définir une fonction d’erreur, plus l’erreur est élevée, moins la prédiction est bonne, le but est de diminuer cette erreur. Pour trouver le minimum d’une fonction, nous utilisons l’algorithme de la descente de gradients. Un gradient regroupe toutes les dérivées partielles d’une fonction par rapport à chaque paramètre que nous appellerons poids.

Nous utilisons du calcul matriciel exclusivement pour alléger le code, les maths ont été faites sans matrice puis il était mieux de tout regrouper.

Juste avant, parlons du réseau de neurones. Il est composé de plusieurs couches, une couche effectue une seule fonction. Le réseau s’utilise de deux manières, il faut prédire un résultat ou ajuster les poids pour mieux prédire un nouveau résultat.

Prédiction : Il suffit juste de calculer le résultat de la première couche puis de l’envoyer à la suivante pour qu’à son tour elle puisse continuer ce processus jusqu’à la dernière, le résultat final étant la prédiction.

Entraînement : On donne une image avec le bon chiffre associé et en fonction de la différence entre la prédiction et le bon chiffre, on modifie plus ou moins les poids. Nous faisons le parcours en sens inverse, on modifie d’abord les derniers poids puis on remonte jusqu’aux premiers car l’erreur se calcule à la fin.

Pour calculer toutes les dérivées nous utilisons le théorème de dérivation des fonctions composées :

Le y ici est l’erreur, le x un poids et u une fonction d’une couche par exemple.

On peut ainsi propager les matrices de dérivées dans toutes les couches du réseau.

# Algorithme

L’algorithme est scindé en deux parties, la première est l’entraînement de l’intelligence artificielle et la dernière, la prédiction d’un résultat.

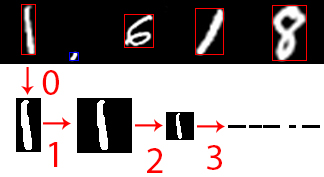
# Prédiction

L’algorithme de prédiction se décompose en deux étapes :

* La découpe de l’image : on trouve quels éléments semblent être des chiffres puis on les traite pour qu’ils aient la bonne taille et les bonnes proportions.
* La prédiction des chiffres : c’est exactement la partie maths, il y a en plus le chargement, l’enregistrement du réseau de neurones et aussi des images de test et d’entraînement.

Découpe de l’image :

Le réseau de neurones a besoin d’une entrée de taille 1 \* 784 pour envoyer en sortie un vecteur de 10 dimensions.



Premièrement on découpe l’image en plusieurs éléments, les éléments trop petits sont enlevés (en bleu). Après on traite le chiffre :

0 : On applique un seuil, soit le pixel est blanc, soit il est noir.

1 : On espace l’image pour qu’elle soit carrée afin de conserver les proportions.

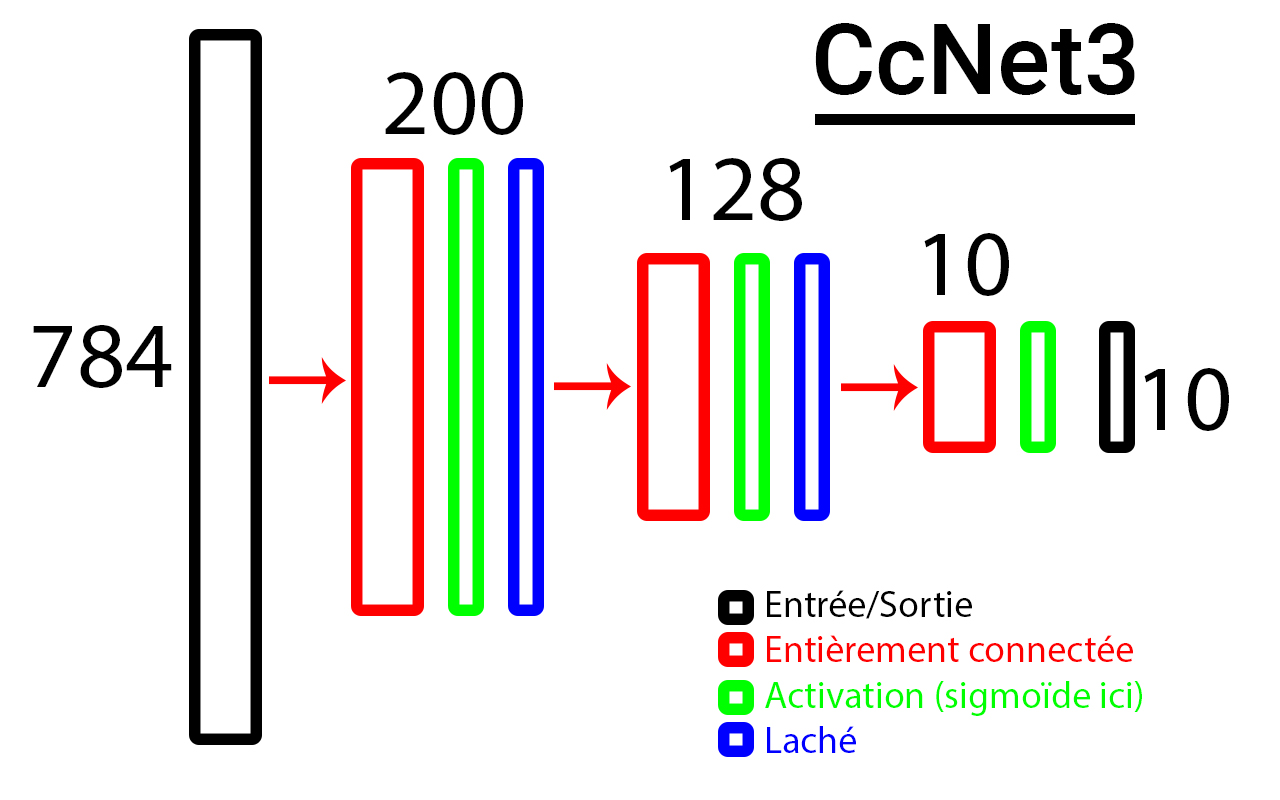
2 : On réduit les dimensions jusqu’à avoir une image de 28 \* 28 pixels avec du max pooling, c’est un algorithme qui garde le pixel le plus lumineux d’une zone.

3 : On aplatit l’image ce qui donne une matrice de 1 \* 784 éléments (car 28 \* 28 = 784).

Il reste juste à envoyer au réseau le chiffre traité.

Réseau de neurones :

Le réseau utilisé en prédiction est nommé CcNet3, il a trois types de couches en plus de celles d’entrée / sortie :



Détails de chaque type de couche :

* Entièrement connectée : couche qui possède des poids, on lui apprend quelle entrée doit être plus ou moins grande pour changer le résultat en sortie.
* Activation : on passe chaque résultat dans une fonction (ici la fonction sigmoïde qui renvoie des nombres entre 0 et 1 exclus).
* Lâché : cette couche permet de ne pas faire surapprendre le réseau, c’est-à-dire pour qu’il soit assez flexible (un pixel changé d’une image peut changer totalement le résultat sans cette couche par exemple).

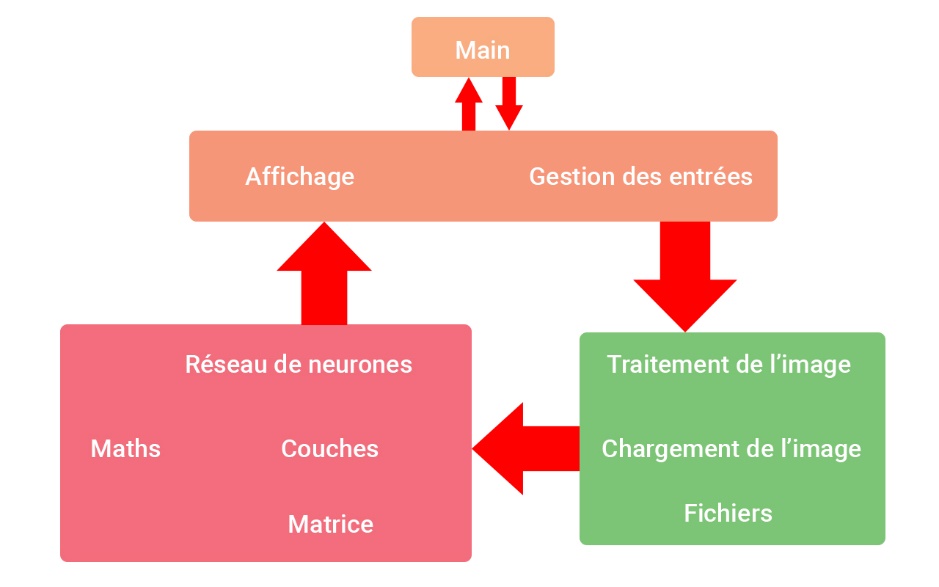
# Entraînement

L’entraînement a été fait sur la base de données MNIST, elle est composée de 70 000 chiffres écrits à la main dont 60 000 pour l’entraînement et 10 000 pour tester si l’algorithme fonctionne bien.

Le réseau de prédiction CcNet3 a réussi à prédire plus de 96% de ces chiffres correctement après une dizaine de minutes d’entraînement.

# Structure

Comme l’algorithme est long il faut le décomposer en plusieurs éléments, voici un schéma de sa structure :



* Le schéma est ordonné, plus on est bas, plus l’élément est bas niveau donc moins il utilise d’autres éléments.

# Notes

Il faut respecter quelques conditions pour que tout fonctionne correctement :

* Les chiffres de l’image doivent être assez grands (au minimum 28\*28px)
* Les chiffres à prédire doivent être écrits à la main, à l’ordinateur le tracé n’a pas la même texture qu’à la main donc le réseau a du mal à correctement prédire les chiffres.
* Certains chiffres sont mieux reconnus que d’autres, par exemple le 6 est souvent confondu avec un 5.
* Les chiffres doivent être droits, pas de rotation car l’algorithme n’est pas prévu pour ça.

# Sources

Dataset MNIST par Yann Lecun :

* <http://yann.lecun.com/exdb/mnist/>

Pour lire les fichiers d’images il a fallu utiliser la bibliothèque stb\_image :

* <https://github.com/nothings/stb>