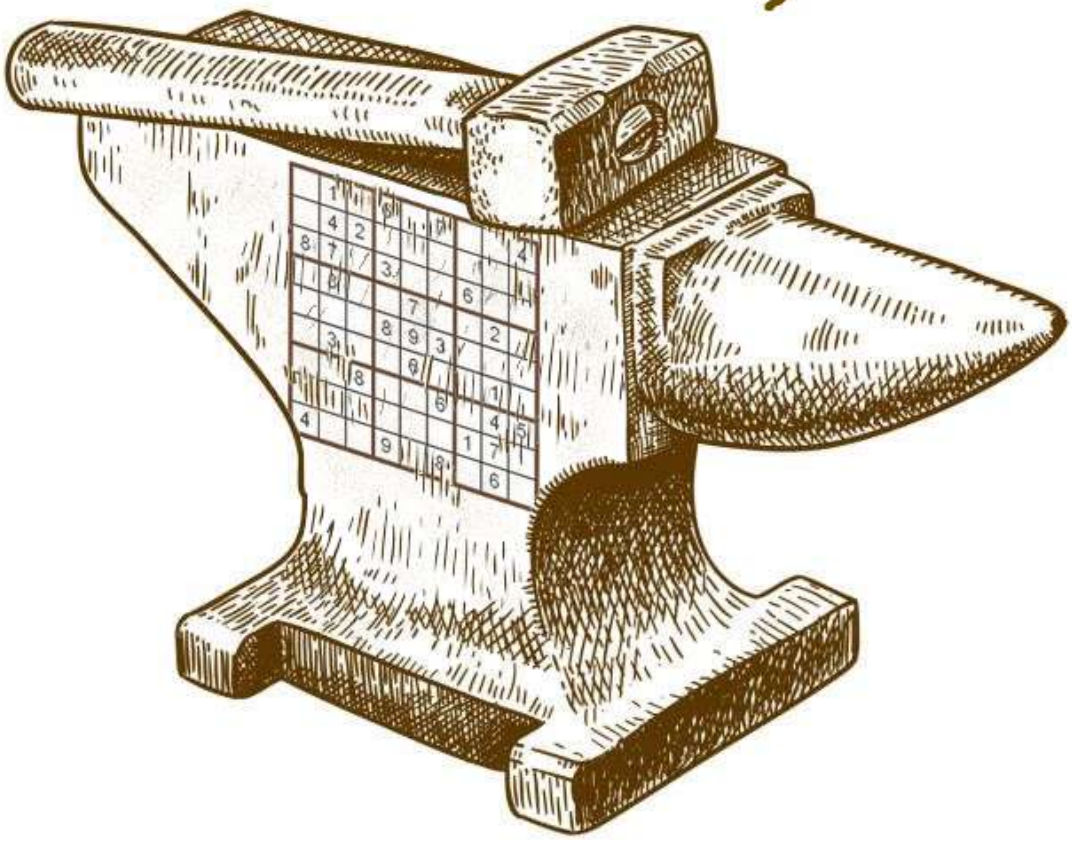
OCR - Soutenance 1

Novembre 2023

Sudoku Forgers



Victor Gardère, Thomas MORIN, Brian PERRET, Ronan Leboucher

Table des matières

1 Groupe

1.1 Présentation du groupe

En charge du valeureux projet que représente cet OCR, notre groupe nommé Sudoku Forgers est composée de : Brian Perret (chef de groupe), Thomas Morin, Ronan Leboucher et Victor Gardère.

1.2 Répartition des charges

Dans ce groupe, nous avons décidé de se séparer les tâches selon l'envie et la curiosité de chacun vis-à-vis des méthodes que l'on devait implémenter pour le bon fonctionnement du projet.

On en arrive donc à :
- Brian, chef du groupe, s'occupe de la binarisation de l'image, du solveur, l'affichage de la grille de sudoku, du réseau de neurones, ainsi que la gestion des entrées de l'utilisateur.
- Ronan qui est en charge de la détection de la grille.
- Victor qui a charge la détection et le nettoyage des cellules de la grilles ainsi qu'aide Ronan dans la détection de la grille.
- Thomas qui est en charge de la rotation, de l'homographie et de l'interpolation bilinéaire ainsi que la détection des cases de la grille avec l'aide de Victor et de Ronan.

1.3 Etat d'avancement du projet

Aujourd'hui, nous avons implémenté la grande majorité des étapes pour cette première soutenance. Nous arrivons à dérouler le programme sur sa quasi-intégralité. Il ne manque plus que la reconnaissance des caractères via le réseau de neurones pour faire le processus dans son intégralité. Globalement, les différentes étapes fonctionnent correctement, mais demande encore quelques ajustements que nous réglerons d'ici la prochaine soutenance.

2 Prétraitement

2.1 Les filtres appliqués

2.1.1 Nuance de gris

La première opération que l'on fait sur notre image originelle est le passage en nuance gris. Cette opération permet ainsi de se passer des informations de couleurs qui, dans le cadre de la reconnaissance de chiffre, ne sont pas quelque chose d'utile. En outre, cela permet de grandement réduire la complexité des futures opérations.

Il existe de nombreuses formules, s'adaptant à divers problèmes, tels que :

$$Gris = (0,299 \times Rouge) + (0.587 \times Vert) + (0.114 \times Bleu)$$

Or, cette formule est mise en relation avec la vision humaine et donc n'est pas la plus efficace pour garder un maximum d'informations utiles pour le traitement d'image.

C'est pour cela qu'on utilise plutôt celle-ci :

$$Gris = \left(\frac{1}{3} \times Rouge\right) + \left(\frac{1}{3} \times Vert\right) + \left(\frac{1}{3} \times Bleu\right)$$

Cela nous donne ainsi les résultats suivants :

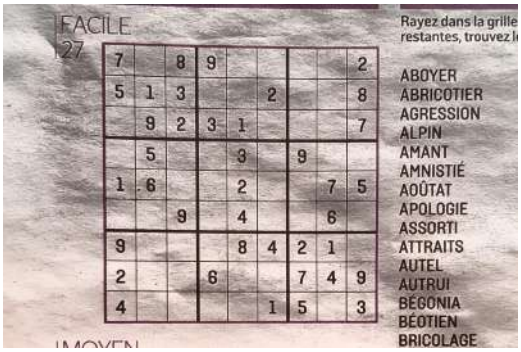


FIGURE 1 – Image originelle

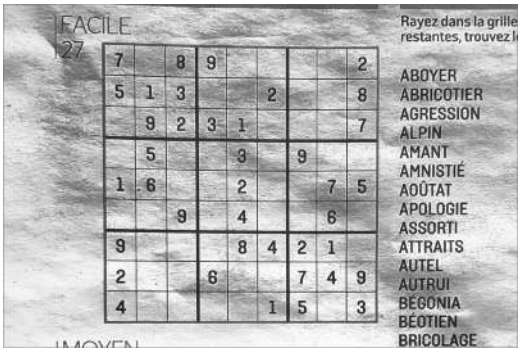


FIGURE 2 – Nuance de gris

2.1.2 Contraste

La prochaine étape du processus est l'augmentation des contrastes de l'image. Cela a pour objectif d'augmenter la différence de gris sur l'image et ainsi mieux faire ressortir les traits.

Cela nous donne ainsi les résultats suivants :

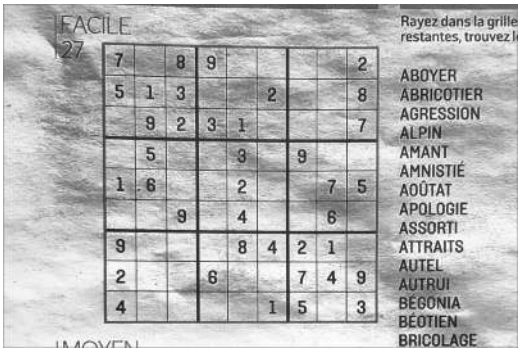


FIGURE 3 – Nuance de gris

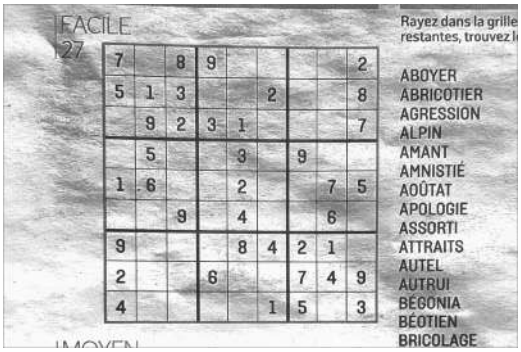


FIGURE 4 – Augmentation du contraste

2.1.3 Normalisation de l'éclairage

A la suite de cela, nous procédons à la normalisation de l'éclairage de l'image. Cette correction permet de supprimer les variations d'éclairage entre deux endroits de l'image. Cela permet ainsi de

rendre les caractères plus distincts les uns des autres.

Pour cette opération, nous utilisons la formule ci-dessous :

$$P_{n_i} = \frac{Pixel_i}{P_{max} \times 255}$$

$$P \in [0\,;255];$$

P_{max} : Valeur max de l'image

P_n : Pixel normalisé

P_i : Pixel à l'index i

Cela nous donne ainsi les résultats suivants :

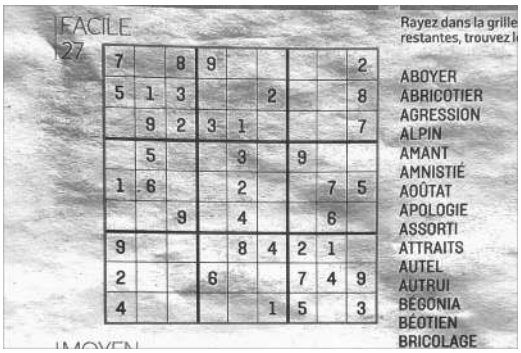


FIGURE 5 – Augmentation du contraste

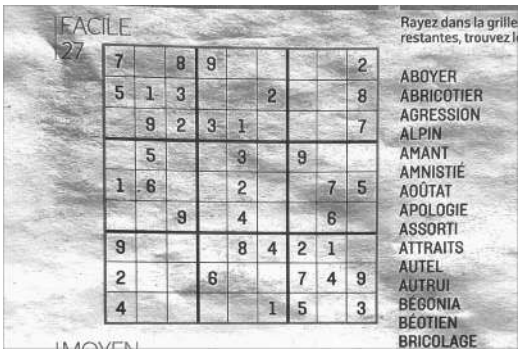


FIGURE 6 – Normalisation de l'éclairage

2.1.4 Médian

Le filtre médian, est un filtre linéaire qui permet de rendre l'image moins bruitée tout en préservant les contours. Pour cela, il atténue les variations soudaines de luminosité à des endroits très localisés.

Voici un exemple de fonction du filtre médian : Soit la matrice des pixels voisins du pixel de valeur n (ici 111)

$$\begin{pmatrix} 5 & 6 & 7 \\ 6 & \mathbf{111} & 8 \\ 7 & 8 & 9 \end{pmatrix}$$

Ensuite, on trie par ordre croissant les valeurs dans un tableau, tel que :

$$\begin{pmatrix} 5 & 6 & 6 & 7 & \mathbf{7} & 8 & 8 & 9 & 111 \end{pmatrix}$$

On prend la valeur médiane de ce tableau, ici 7 Et ainsi, cela nous donne :

$$\begin{pmatrix} 5 & 6 & 7 \\ 6 & \mathbf{7} & 8 \\ 7 & 8 & 9 \end{pmatrix}$$

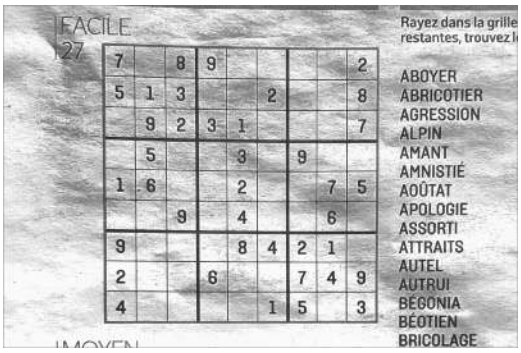


FIGURE 7 – Normalisation de l'éclairage

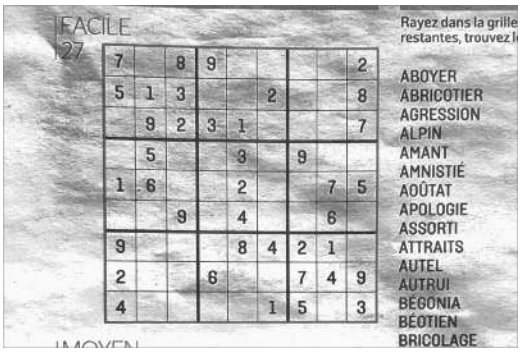


FIGURE 8 – Filtre médian

2.1.5 Flou Gaussien

L'étape suivante est de flouter très légèrement l'image pour enlever les minuscules détails de l'image qui ne nous intéressent pas. Grâce à cela, ce filtre permet de réduire le bruit et de lisser l'image.

Son principe de fonctionnement est le suivant :

$$I_{\text{flou}}(x, y) = \sum_i \sum_j I(x - i, y - j) \cdot G(i, j)$$

$I_{\text{flou}}(x, y)$: est la valeur du pixel dans l'image floutée à la position. (x, y)
$I(x - i, y - j)$: est la valeur du pixel dans l'image d'origine à la position. $(x - i, y - j)$
$G(i, j)$: est la valeur du noyau Gaussien à la position. (i, j)

Et la formule du noyau Gaussien :

$$G(i,j) = \frac{1}{2\pi\sigma^2} e^{-(i^2+j^2)/(2\sigma^2)}$$

i et j : parcourent les coordonnées du noyau Gaussien.

σ : est l'écart-type du noyau Gaussien, qui contrôle la quantité de flou.

Un écart-type plus grand entraîne un flou plus important.

2.2 Binarisation

L'étape de la binarisation, est l'étape la plus importante parmi toutes ces opérations. En effet, c'est elle qui va dire si le pixel devient un pixel blanc ou un pixel noir.
Pour cela, nous utilisons un seuil tel que :

$$\begin{cases} \text{Valeur du pixel} < \text{seuil} & \Rightarrow \text{Blanc} \\ \text{Valeur du pixel} \geq \text{seuil} & \Rightarrow \text{Noir} \end{cases}$$

Ainsi, tout l'enjeu, est de trouver le meilleur seuil pour ne garder que les pixels utiles.

2.2.1 Méthode de Otsu

Le premier type d'algorithme, est une méthode de seuillage dite "global". Cet algorithme classe les pixels parmi deux types :

— Les pixels au premier-plan.

— Les pixels à l'arrière-plan.

C'est-à-dire qu'elle ne va effectuer qu'une seul mesure global de l'image pour déterminer le seuil optimal de l'image et ainsi déterminer dans quelle classe se situe un pixel.

Pour déterminer le seuil optimal, on commence par faire l'histogramme des valeurs des pixels de l'image, tel que :

Ensuite, il suffit de trouver le meilleur emplacement entre deux pics de l'image, qui séparera au mieux l'image en deux classes.

La formule pour trouver le seuil optimal est :

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

$\sigma_w^2(t)$: La variance intraclasse à un seuil. t.

$\omega_1(t)$: Le poids de la classe du premier-plan. t).

$\sigma_1^2(t)$: La variance de la classe du premier-plan. t.

$\omega_2(t)$: Le poids de la classe de l'arrière-plan. t).

$\sigma_2^2(t)$: La variance de la classe de l'arrière-plan. t.

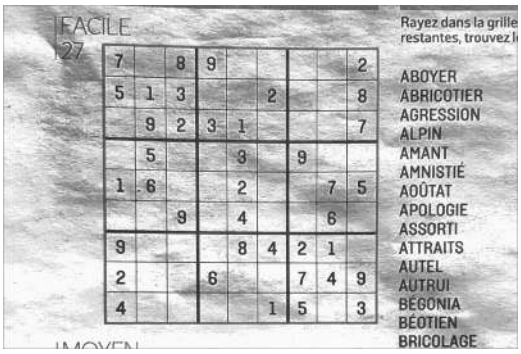


FIGURE 9 – Filtre médian

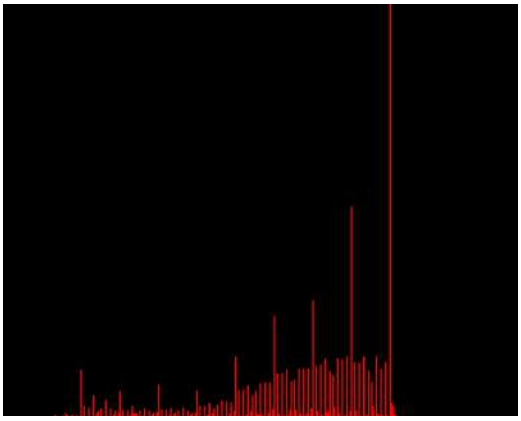


FIGURE 10 – Histogramme de l'image

Voici le résulat après l'utilisation de la méthode d'Otsu :

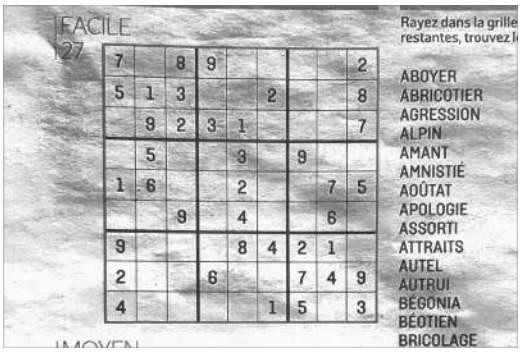


FIGURE 11 – Filtre médian

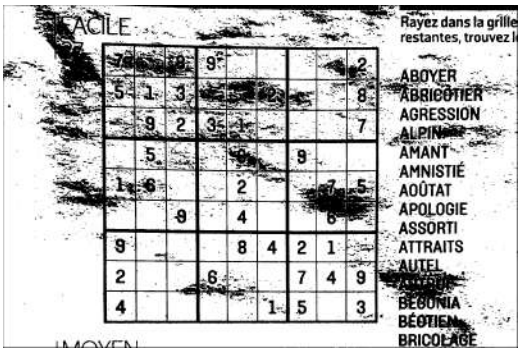


FIGURE 12 – Méthode d'Otsu

Comme on peut le voir, cette méthode est très simple, ce qui pose un problème. En effet, celle-ci perd de son efficacité dès qu'il y a des zones de l'image plus foncées que d'autres. (ombre, vignettage, etc...)

2.2.2 Seuil adaptatif

En revanche, les méthodes de binarisation via l'utilisation d'un seuil adaptatif règle ce problème. En effet, au lieu d'effectuer un seul seuil, nous divisons l'image en multiples zones. A la suite de cela, il nous suffit de calculer le seuil pour chacune de ces zones.

Pour calculer la taille d'une zone, nous calculons le niveau de bruit dans l'image. Plus une image est bruitée, plus il faut de nombreuses zones différentes, et inversement.

Pour calculer le niveau de bruit d'une image, il suffit de calculer la moyenne des valeurs dans une fênetre, et de compter le nombre de pixels qui se trouvent bien au-dessus de cette moyenne.

La taille de la fenêtre et le seuil pour considérer qu'il s'agit d'un pixel parasite sont déterminés à l'avance avec des tests.

Cette méthode étant très performante, nous n'avons pas besoin d'effectuer une méthode de détermination du seuil très poussée telle que la méthode d'Otsu, qui si est répétée de multiples fois, peu causer un fort coût de calcul.
C'est pour cela que nous utilisons simplement les valeurs moyennes dans chacune des zones pour déterminer les seuils respectifs.

Voici le résultat de la binarisation avec une méthode de seuillage adaptatif :

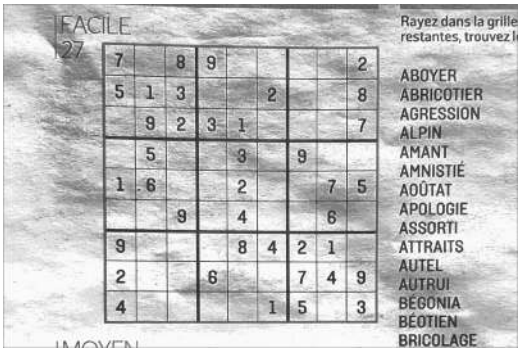


FIGURE 13 – Filtre médian

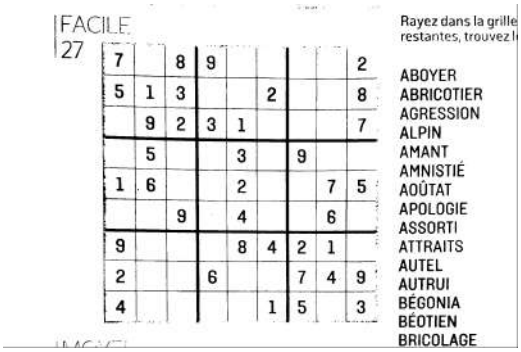


FIGURE 14 – Seuil adaptatif

2.3 Un autre filtre

2.3.1 Morphologie

Pour finir avec ce processus, nous appliquons un filtre de morphologie. Cela permet de séparer deux objets légèrement collés, ainsi que de compléter les espaces vides quand il y en a.

Cette opération se décompose en deux filtres quasi-identiques.

— Filtre de dilatation
— Filtre d'érosion

Principe de fonctionnement du filtre de dilatation :

Soit la matrice des pixels voisins de pixels de valeur n (ici 1). La valeur 0 représente un pixel de couleur blanche, et 1, un pixel de couleur noire.

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Si l'un des voisins du pixel est de couleur noire, alors le pixel devient noir, tel que :

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Voici le résulat de la dilatation :

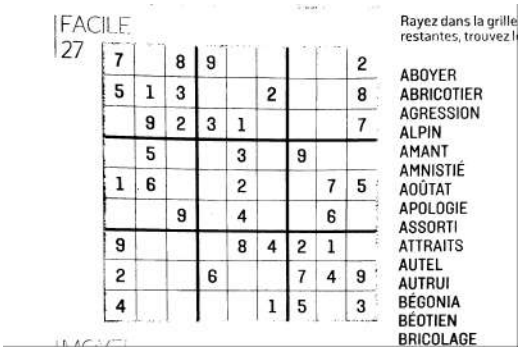


FIGURE 15 – Seuil adaptatif

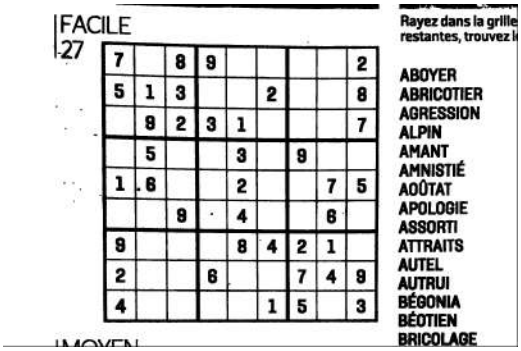


FIGURE 16 – Dilation

Principe de fonctionnement du filtre d'érosion :

Soit la matrice des pixels voisins de pixels de valeur n (ici 1). La valeur 0 représente un pixel de couleur blanche, et 1, un pixel de couleur noire.

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Si l'un des voisins du pixel est de couleur blanc, alors le pixel devient blanc, tel que :

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & \mathbf{0} & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Voici le résulat de l' érosion :

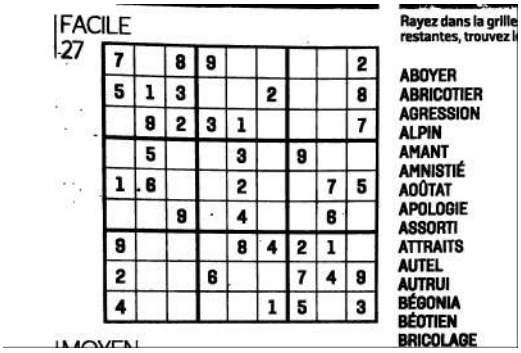


FIGURE 17 – Dilation

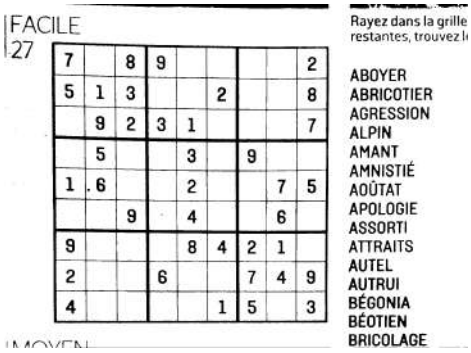


FIGURE 18 – Erosion

2.4 Avant, Après

Un comparatif avant/après du processus.

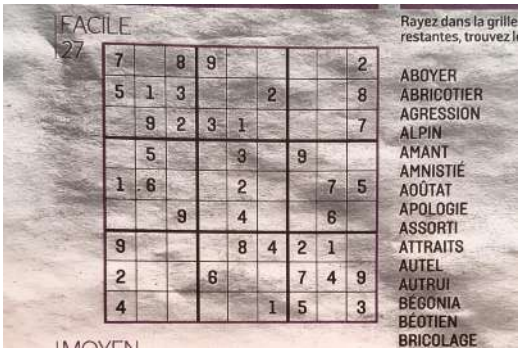


FIGURE 19 – Image original

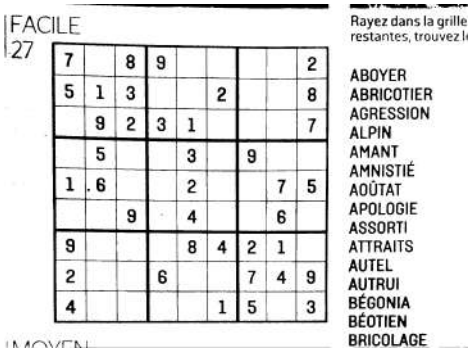


FIGURE 20 – Image final

2.5 Rotation de l'image

Il existe plusieurs méthodes de rotation d'images. Nous avons opté pour la méthode qui nous donnera le meilleur résultat. Comme nous l'avions pensé, une méthode plus simple nous donnerait un résultat de plus basse qualité avec des pixels qui n'existent plus à cause de la simplification et de l'imprécision du calcul de la méthode.

Nous avons donc utilisé une méthode du nom de shearing qui consiste à prendre chaque pixel de l'image puis à appliquer 3 multiplications successives par des matrices de shearing.
Le shearing est une transformation qui va consister à décaler chaque point de l'image en fonction d'un certain angle.

Notre rotation fait de 3 shearing consécutifs pour avoir une perte de qualité minimum :

Nous avons donc le calcul suivant avec θ notre angle en radian :

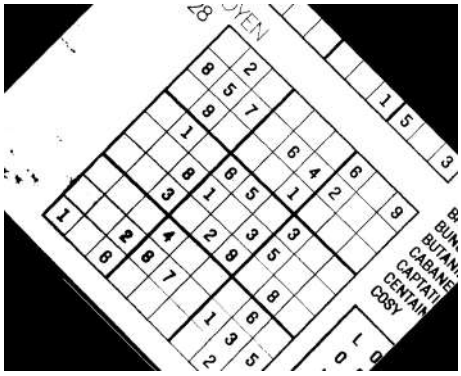


FIGURE 21 – Rotation de 45°

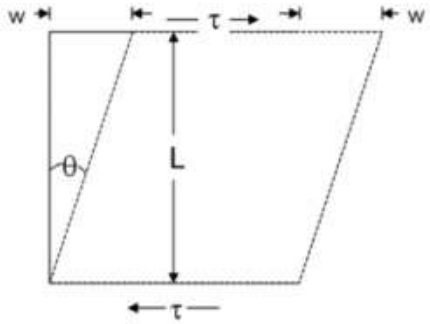


FIGURE 22 – Illustration de la rotation par shearing

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & tan(\theta/2) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ sin(\theta) & 1 \end{pmatrix} \begin{pmatrix} 1 & -tan(\theta/2) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Après avoir calculé nos nouvelles coordonnées, ils nous faut leur ajouter un décalage pour pouvoir bien replacer l'image en son centre.

La perte de qualité après rotation est minime, cependant il y a un peu de crénelage (c'est-à-dire qu'il y a des sortes d'escaliers) sur l'image. Mais cela n'empêche en rien le bon fonctionnement des fonctions qui suivront la rotation.

2.6 Homographie

La transformation de perspective ou homographic transform en anglais, va nous permettre de "normaliser" notre grille, c'est-à-dire la remettre aux bonnes dimensions, la "redresser" , si l'angle de la photo originale n'est pas correcte, et la tourner en même temps. Pour appliquer cette transformation, nous devons d'abord trouver les coefficients de la matrice de transformation. En effet pour chaque transformation, nous avons besoin d'un matrice 3*3 pour appliquer une transformation linéaire, que ce soit une rotation ou une réduction d'échelle. Soient (x, y) les coordonnées d'un coin

FIGURE 23 – Rotation par shearing

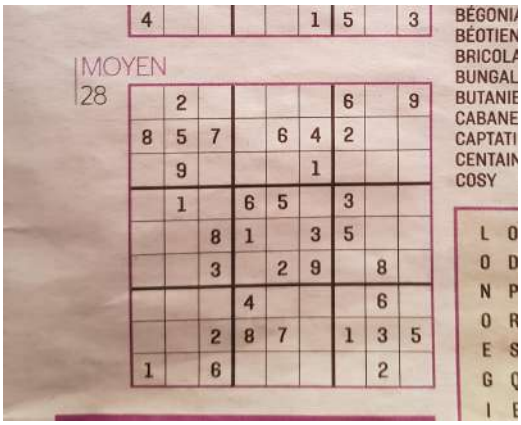


FIGURE 24 – Avant rotation

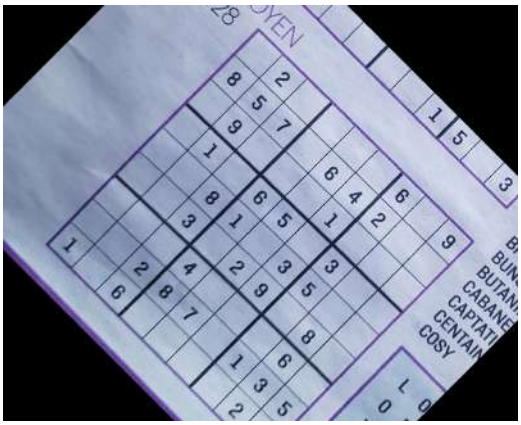


FIGURE 25 – Après rotation

de l'image d'arrivée, (x', y') les coordonnées d'un coin de l'image de départ et a, b, c, d, e, f, g, h nos coefficients de matrice, nous avons le calcul suivant :

$$\begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Où $w = gx + hy + 1$, le problème étant qu'on ne veut pas le garder, car notre image est en 2 dimensions alors nous allons réécrire notre matrice pour faire apparaître un 1 à la place de w dans notre matrice de fin. En remaniant notre expression, nous avons :

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \frac{\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}{\begin{pmatrix} g & h & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}$$

On peut donc déduire les coordonnées de x' et y' :

$$x' = \frac{ax + by + c}{gx + hy + 1}$$

$$y' = \frac{dx + ey + f}{gx + hy + 1}$$

Après simplification nous avons ces deux expressions :

$$x' = ax + by + c - x'gx - x'hy$$

$$y' = dx + ey + f - y'gx - y'hy$$

On remarque que cela ressemble à un produit d'une matrice par un vecteur si on ajoute tous les termes :

$$x' = ax + by + c + 0d + 0e + 0f - x'gx - x'hy$$
$$y' = 0a + 0b + 0c + dx + ey + f - y'gx - y'hy$$

Nous avons donc ces matrices :

$$\begin{pmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \\ & & & & & . & & \\ & & & & & : & & \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ . \\ : \end{pmatrix}$$

Ici, nous pouvons utiliser une unique matrice pour calculer les 4 coins de notre image en même temps.

Soit B la matrice contenant les coordonnées de nos coins, λ la matrice contenant nos coefficients et A l'autre matrice.

Nous avons :

$$A\lambda = B$$
$$A^T A\lambda = A^T B$$
$$\lambda = (A^T A)^{-1} A^T B$$

Une fois nos coefficients calculés, il ne nous reste qu'à appliquer la matrice à chaque pixel de l'image.

2.7 Interpolation bilinéaire

L'interpolation bilinéaire ou filtrage bilinéaire ou encore mappage de texture bilinéaire est une méthode de rééchantillonnage en vision par ordinateur qui va nous permettre de redimensionner une image avec une hauteur et une largeur donnée.

L'interpolation bilinéaire est réalisée en utilisant une interpolation linéaire d'abord dans une direction, puis à nouveau dans une autre direction. Dans notre cas, tout d'abord deux interpolations linéaire sur l'axe X puis une interpolation linéaire sur l'axe des Y.

Bien que chaque étape soit linéaire dans les valeurs échantillonnées et dans la position, l'interpolation bilinéaire dans son ensemble n'est pas linéaire mais quadratique dans l'emplacement de l'échantillon.

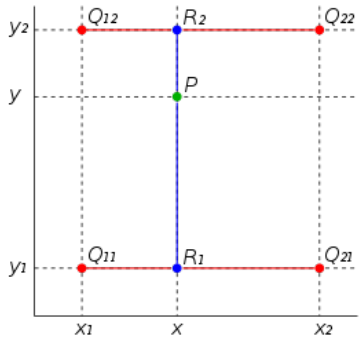


FIGURE 26 – Illustration de l'interpolation bilinéaire
Les quatre points commençant par Q sont les points existants, et le point P est le point dont on cherche la valeur par interpolation.

Nous avons utilisé cette méthode en la remaniant, c'est-à-dire que :
Pour chaque pixel de l'image finale, nous avons effectué un calcul pour nous permettre d'avoir les coordonnées d'un pixel équivalent sur l'image de départ. Or, ces coordonnées peuvent avoir des chiffres après la virgule donc pour cela nous regarderons les pixels avec la valeur entière de ces coordonnées ainsi que ceux avec la valeur supérieure. Ce qui nous donne quatre pixels autour de ces coordonnées.
Avec ceci, nous pouvons connaître la couleur du pixel que l'on souhaite avoir sur l'image de retour, pour cela on va appliquer un pourcentage à cette couleur pour avoir quelque chose de plus homogène en termes de couleur sur l'image. Ce pourcentage est calculé en fonction de la proximité du pixel avec un des quatres pixels calculés précédemment.

Le pourcentage est appliqué sur chaque interpolation linéaire que l'on fait, ainsi on évite des calculs trop compliqués car le pourcentage vers un pixel peut être de "p" et dans l'autre sens il sera de "$1 - p$".

Lors d'un rétrécissement de l'image, celle-ci perd de la qualité, ce qui est attendu. Alors que dans le cas contraire, on voit apparaître une perte de qualité inattendue sur les lignes dans l'image avec des sortes de pixels qui alternent entre noir et blanc, ce qui est dû a l'approximation du calcul du pixel sur l'image de départ.

2.8 Détection de la grille

Pour arriver à détecter la grille, l'algorithme qui présente la plus grande efficacité et la plus petite marge d'erreur est l'algorithme nommé "Blob Detection". Cet algorithme consiste à parcourir les groupes de pixels adjacents ayant la meme couleur (les "blobs"). A chaque nouveau blob détecté, on compare sa taille avec la taille maximale enregistrée pour ainsi obtenir le plus gros blob. Sur une image sur laquelle on a appliqué de la réduction de bruit, ce blob correspond donc à la grille.
Il est important d'appliquer cet algorithme à une image binarisée (avec uniquement des pixels noirs ou blancs) car cela optimisera son temps d'execution et surtout evitera de créer des erreurs pour les étapes qui suivront la détection de la grille.
En plus du parcours classique de l'image, il a donc fallu implémenter un algorithme de parcours des pixels voisins efficace, tout en évitant de repasser deux fois par le meme pixel, et donc pour cela, nous avons tout simplement utilisé une matrice de la même taille que notre image, avec un 1 pour representé un pixel déja visité et un 0 sinon, assez simplement.

2.9 Détection des coins de la grille

Afin d'obtenir les coordonnées des coins de la grille de sudoku, nous allons par courir chaque pixel de l'image binarisé dont il ne reste que la grille de sudoku après la détection par blob ensuite nous allons enregistrer les coordonnées des 4 pixels noirs les plus éloignés les uns des autres jusqu'à la fin du parcours de l'image. Ces 4 points sont ensuite renvoyés dans une liste d'entier.

2.10 Détection des cases de la grille

Pour récupérer chaque cases de la grille, nous allons diviser la grille par 9 dans chaque axe, ce qui nous fait 81 cases. Pour chacune des 81 cases nous prennons les coins de la case et lui appliquons une homographie pour sortir une image de $28x28$ que nous mettons dans une liste avec la position de la case dans la grille pour pouvoir reconstruire la grille à notre fin.

2.11 Extraction des chiffres présents les cases

Avec notre liste de cases de la grille récupérées, nous la trions pour enlever les cases où ils n'y a pas de chiffres. Pour les cases restantes dans la liste, pour chacune d'entre elles, nous prennons l'image avec son plus grand blob qui est le chiffres présent sur l'image. Donc, à la fin, nous avons une liste d'images avec seulement son chiffre.

3 Solveur de sudoku

3.1 Validité de la grille

Avant de chercher à résoudre, on vérifie d'abord que la grille est valide. C'est-à-dire, qu'elle respecte les règles du sudoku.

— Il existe une seule occurrence d'un nombre par rangée.
— Il existe une seule occurrence d'un nombre par colonne.
— Il existe une seule occurrence d'un nombre dans la sous-zone de la grille.

3.2 Résolution de la grille

Pour résoudre la grille de sudoku, nous utilisons un algorithme de backtracking. Celui-ci parcourt la grille à la recherche d'une case vide et y place une valeur possible. Pour chacune de ces valeurs, l'algorithme teste récursivement si une solution valide peut-être construite à partir de cette valeur. Si cela échoue, alors il passe à la valeur possible suivante, et ainsi de suite. S'il n'y a plus de valeur possible et qu'il a échoué, alors la grille est impossible. Sinon la grille est résolue.

Dans le meilleur des cas, cet algorithme est de complexité linéaire. Dans le pire des cas, cet algorithme est de complicité exponentielle.

Ainsi pour une grille de sudoku de 9x9, cela est quasiment instantané, même sur une grille difficile. En revanche, pour une grille de sudoku de 16x16, le temps de calcul d'une grille complexe peut vite se faire ressentir sur le temps d'exécution.

3.3 Affichage de la grille résolue

Une fois la grille résolue, il est possible de convertir les données de la grille en une image. De nombreux paramètres (police, taille, bordure, ...) permettent de choisir quel style de grille on souhaite avoir en sortie.

Voici le résultat de la conversion en image de la grille de sudoku :

5	4	3	9	2	1	8	7	6
2	1	9	6	8	7	5	4	3
8	7	6	3	5	4	2	1	9
9	8	7	4	6	5	3	2	1
3	2	1	7	9	8	6	5	4
6	5	4	1	3	2	9	8	7
7	6	5	2	4	3	1	9	8
4	3	2	8	1	9	7	6	5
1	9	8	5	7	6	4	3	2

FIGURE 27 – Grille 9x9

0	5	1	2	6	B	7	4	C	F	E	3	9	D	A	8
6	E	C	9	0	D	8	3	B	1	A	4	7	2	5	F
7	B	8	F	1	2	9	A	5	D	0	6	4	3	C	E
A	4	3	D	5	C	F	E	8	2	9	7	0	B	1	6
5	8	0	7	B	1	2	6	3	A	4	C	F	9	E	D
D	1	B	4	8	E	3	C	9	6	2	F	A	7	0	5
2	6	F	C	A	7	5	9	0	E	D	B	8	4	3	1
3	9	E	A	F	0	4	D	7	8	1	5	B	C	6	2
4	3	2	6	9	8	E	0	A	7	5	D	C	1	F	B
1	D	5	0	7	3	6	F	4	C	B	E	2	8	9	A
C	F	9	B	2	A	D	1	6	0	3	8	E	5	7	4
8	A	7	E	C	4	B	5	1	9	F	2	3	6	D	0
E	C	6	1	4	F	0	7	D	B	8	9	5	A	2	3
9	7	A	8	3	5	1	2	E	4	6	0	D	F	B	C
B	2	D	5	E	6	C	8	F	3	7	A	1	0	4	9
F	0	4	3	D	9	A	B	2	5	C	1	6	E	8	7

FIGURE 28 – Grille 16x16

4 Réseau de neurones

Le réseau de neurones nous permet d'effectuer des tâches complexes qui pour un algorithme classique seraient très complexes à mettre en œuvre.

Pour cette première soutenance, nous avons conçu une première version de notre réseau de neurones capable d'apprendre la porte logique "ou exclusive".

4.1 Configuration de réseau

Le réseau de neurones que nous avons mis en place est composé de 3 couches :

— La première couche, composé de 2 neurones prend simplement les deux entrées de la porte logique.

— La second couche (couche caché) est composé de minimum 3 neurones.

— La dernière couche, composé de 2 neurones, qui correspondent réspectivement aux chiffres 0 et 1.

Le choix de deux neurones pour la couche de sortie s'explique principalement pour préparer la version 2 du réseau de neurones ou chaque neurone de la dernière couche correspondra respectivement au chiffre de 0 à 9.

De plus, il est aussi possible d'encoder les chiffres en binaire, mais encoder en base 10 donne de meilleur résulta.

4.2 Propagation avant

La propogation avant consiste a propager l'entré initial du réseau à travers toutes les couches dites "cachées" jusqu'a la dernière couche.

Pour effectuer cette opération nous utilisons la formule suivant :

$$N_i^L = \sigma\left(N_i^{L-1} \times W_i^L + B_i^L\right)$$

σ : La fonction d'activation sigmoid

N_i^L : La valeur du neurones à l'indice i de la couche L

N_i^{L-1} : La valeur du neurones à l'indice i de la couche précédent la couche L

W_i^L : Est la matrice des poids du neurone i de la couche L

B_i^L : Est le biais du neurone i de la couche L

Pour la dernière couche, nous faisons quasiment la même chose, mais nous utilisons la fonction d'activation softmax.

$$N_i^L = Softmax\left(N_i^{L-1} \times W_i^L + B_i^L\right)$$

Grâce à cette étape, il nous est possible de connaître les valeurs de sortie du réseau.

4.3 Apprentissage du réseau

Pour que le réseau retourne les valeurs que l'on souhaite pour notre application. Nous devons procéder à une étape d'apprentissage.

Concrètement, nous regardons si les valeurs de sorties sont celles attendues. Si elles sont différentes, nous calculons la différence entre, ce sont deux valeurs telles que :

$$Error_i = N_i^{Lo} - Expected_i$$

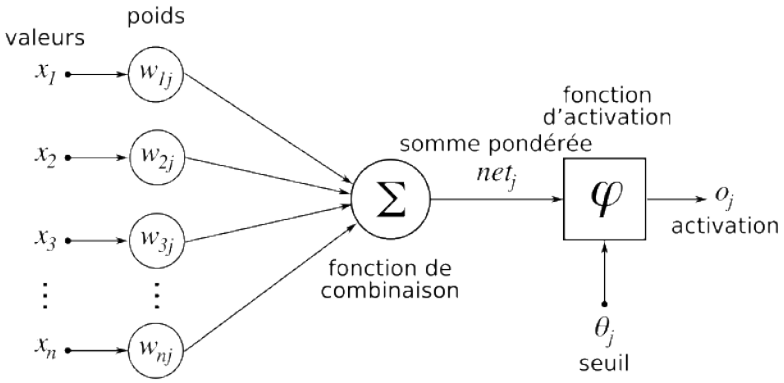


FIGURE 29 – Illustration de la propagation en avant sur un neurone

$Error_i$: Est l'erreur calculer pour le neurone i de la dernière couche

N_i^{Lo} : La valeur du neurones à l'indice i de la dernière couche

$Expected_i$: La valeur attendu pour le neurone i de la dernière couche

Ainsi, en effectuer une propagation arrière de l'erreur, nous pouvons ajuster légèrement* le biais et les poids de chaque neurones. Cela a pour effet de faire converger notre réseau de neurones vers le résultat attendu.

*En fonction du taux d'apprentissage que l'on a indiqué.

4.4 Amélioration du réseau

Cette première version de notre réseau de neurones, nous a permis d'explorer et de tester diverses choses pour ainsi, faire une deuxième version, plus robuste (qui converge plus vite et avec moins de neurones dans notre cas).

5 Conclusion de la première soutenance

La conclusion de cette première soutenance est globalement positive. Nous avons fait une grande partie des fonctions dont nous avons besoin pour l'application final. Ainsi, pour la prochaine soutenance, il nous restera plus qu'à faire l'interface graphique, la deuxième version du réseau de neurones pour reconnaître des caractères ainsi que quelque amélioration de certaines fonctions pour avoir des résultats toujours plus intéressants.