Université Côte d'Azur

Mémoire de Master 1
Base de données MNIST

Auteurs :
Lucas Boiteau
Maxime Larcanché
Benjamin Faustini

Coordonnateur : M. Roland Diel
Tuteur : M. André Galligo

Juin 2020

Table des matières

Table des figures

L'intelligence artificielle n'est plus seulement un programme de recherche confiné aux laboratoires ou à une application précise. Elle va devenir une des clés du monde à venir.

Cédric Villani

Introduction

L'émergence d'une nouvelle technologie suscite en nous bien souvent crainte et admiration : la voiture autonome, les assistants vocaux ou encore les drones, n'en sont que quelques exemples. Toutes ces innovations ont bouleversé notre société et nous le devons à l'émergence de l'intelligence artificielle. En l'espace d'une décennie, les réseaux de neurones se sont retrouvés au coeur de notre quotidien. L'ingénieur français Yann LeCun (prix Turing 2018) participa à cette révolution en s'associant avec deux canadiens : Geoffrey Hinton et Yoshua Bengio. En 2003, ils démarrent ensemble "un programme de recherche visant à remettre au goût du jour les réseaux neuronaux, et à améliorer leurs performances afin de raviver l'intérêt de la communauté scientifique".
Railleur, M.LeCun qualifiera cette alliance de "conspiration de l'apprentissage profond", en référence aux moqueries qu'ils subissaient alors de la part de la majorité des informaticiens.
Néanmoins, l'avenir leur a donné raison : en 2012 des étudiants de Geoffrey Hinton présentent à l'occasion d'un concours le premier réseau de neurones à plusieurs couches (Conv-Net) qui transformera en profondeur notre société et réhabilitera M.LeCun.

Ce dernier, a tenu à jour un site internet dédié à la reconnaissance de chiffres manuscrit (MNIST database) que nous allons étudier. Dans un premier temps, nous vous présenterons une méthode classique de prédiction de chiffres. Puis, nous vous ferons part de notre tentative de création d'un système de prédiction "humainement" compréhensible et fiable. Enfin, nous parlerons de notre réseau de neurones.

Geoffrey Hinton

Yann LeCun

Yoshua Bengio

Chapitre 1

Première approche

1.1 Quelques rappels

La méthode des k plus proche voisins (k-nn ou KNN) est une méthode d'apprentissage supervisé. Nous pouvons la résumer à l'aide du dessin ci-dessous :

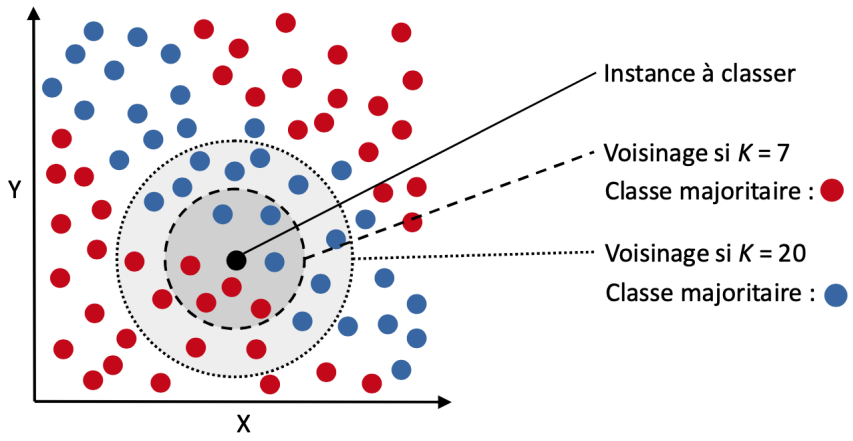


FIGURE 1.1: Schéma de la méthode KNN

On se fixe un nombre de voisins K autour d'une instance à classer. Comme nous pouvons le voir, ce nombre a une influence sur la classe prédite : si $K = 7$, l'algorithme prédira rouge tandis que si $K = 20$, il prédira bleu. La mesure des distances se fait ici à l'aide de la distance euclidienne. Ce procédé a démontré son efficacité pour certains problèmes où la frontière décisionnelle entre les classes est très irrégulière, par exemple pour l'analyse d'images satellite.

1.2 Découverte de la base de données MNIST

La database MNIST est composée de 70 000 images de 28x28 pixels représentants des chiffres manuscrits. L'ensemble d'apprentissage comporte 60 000 éléments et l'ensemble de test en contient 10 000.
Voici une représentation, tirée de Wikipédia, de chiffres de notre base de données :

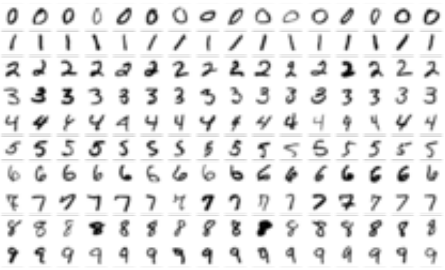


FIGURE 1.2: Un exemple de chiffres de la base MNIST

1.3 Observations des données

Nous avons exploité et amélioré un travail disponible sur un site internet public (Kaggle). Pour ce programme, nous avons utilisé les librairies numpy, pandas, sklearns et scipy.
Tout d'abord, nous avons converti les données au format CSV. La première colonne de celui-ci se nomme "label", il s'agit du nombre associé à l'image : ce sera notre classe à prédire. Les autres colonnes sont des nombres allant de 0 à 255 qui permettent de coder la couleur noire pour chaque pixel. En effet, ces derniers sont codés sur un octet soit $2^8 = 256$ valeurs possibles pour chacun d'entre eux.
Ensuite, nous avons défini les ensembles d'apprentissages et de tests dans deux dataframes à l'aide des fichiers téléchargés sur le site web de M. LeCun. Afin de pouvoir observer une image, nous allons utiliser des tableaux numpy. La i-ème ligne de X_train contient ainsi toutes les données colorimétriques associées au i-ème label de y_train. La fonction X_test permettra de tester notre système de prédiction. Nous avons dû changer la formule par rapport à celle utilisée par l'auteur du programme, puisque celui-ci avait utilisé la base de données disponible sur Kaggle contrairement à nous. Cela nous posait un problème de dimensions au moment de tester.
Voici une ligne au hasard de X_train (la quarantième), que nous avons préalablement transformé en une matrice 28x28 (dimensions de l'image) afin de la comparer au label associé. On observe bien la correspondance entre les deux.

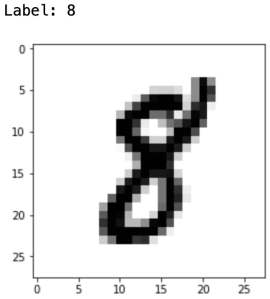


FIGURE 1.3: Une première image

1.4 Sélection d'un modèle

Avant de choisir d'appliquer une méthode KNN, il convient de valider cette méthode de prédiction. La validation croisée semble ici pertinente. Ainsi, nous allons choisir de manière assez classique, $K = 5$ et $n = 42$. Nous avons donc regardé les différentes matrices de confusion ainsi que les performances pour chaque échantillon. Nous obtenons des résultats particulièrements convaincants avec des taux de précision avoisinants les 100%. Les résultats du premier échantillon sont les suivants :

Classification report for Fold 1:

	precision	recall	f1-score	support
0	0.982	0.992	0.987	1185
1	0.956	0.994	0.975	1349
2	0.987	0.970	0.978	1192
3	0.969	0.964	0.967	1227
4	0.983	0.969	0.976	1169
5	0.952	0.960	0.956	1084
6	0.978	0.989	0.984	1184
7	0.968	0.972	0.970	1253
8	0.989	0.929	0.958	1171
9	0.955	0.971	0.963	1190
accuracy			0.972	12004
macro avg	0.972	0.971	0.971	12004
weighted avg	0.972	0.972	0.972	12004

FIGURE 1.4: Performance du premier échantillon

Confusion Matrix for Fold 1:

```
[[1176    0    0    0    0    3    4    1    1    0]
 [   0 1341    0    0    3    0    0    4    1    0]
 [   8   10 1156    1    0    1    1   10    2    3]
 [   1    4    6 1183    0   18    1    9    1    4]
 [   0   11    0    0 1133    0    3    3    0   19]
 [   3    3    2   10    1 1041   15    1    4    4]
 [   4    3    0    0    0    6 1171    0    0    0]
 [   0   14    2    1    6    1    0 1218    0   11]
 [   3   14    4   17    5   20    2    4 1088   14]
 [   2    3    1    9    5    3    0    8    3 1156]]
```

FIGURE 1.5: Matrice de confusion du premier échantillon

Tous les autres résultats sont similaires. Cette étape comporte un temps de calcul élevé, mais elle permet de s'apercevoir de la faible proportion d'erreurs : par exemple seuls 9 zéros sont prédits de manière incorrecte sur un total de 1185. Nous pouvons donc accepter et retenir le modèle.
Il reste néanmoins à déterminer les meilleurs paramètres à considérer pour notre système de prédiction.
Nous avons comparé les résultats pour $K \in \{3, 4, 5, 6, 8, 10\}$ d'abord avec des poids uniformes (tous les voisins ont le même poids) puis selon la distance (les points les plus proches ont un poids plus élevé). Cette étape de comparaison est elle aussi très longue a exécuter (264.6 minutes). Le meilleur score (97%) est obtenu pour une répartition selon la distance avec $K = 4$. Il convient donc ce choix de paramètres à notre ensemble de test. Sur le même principe que sur l'ensemble d'apprentissage, nous avons fait apparaître l'image correspondant à un indice aléatoire de X_test et nous avons pu observer que label prédit est presque systématiquement correct. De plus, le raisonnement erratique de la machine est en général plutôt compréhensible. Voici trois erreurs d'interprétations dont nous pouvons comprendre la confusion, du fait de la ressemblance des tracés entre chiffres écrits et prédits :

FIGURE 1.6: Quelques mauvaises interpretations

1.5 Data Augmentation

Afin d'augmenter la précision de notre algorithme, nous avons implémenté de nouvelles images dans notre ensemble d'apprentissage. En effet, nous pouvons obtenir 8 nouveaux chiffres manuscrits simplement en déplaçant l'image originale d'un pixel dans une direction (haut, bas, gauche, droite) en la tournant ou en la zoomant. C'est le processus dit de *Data Augmentation*. En appliquant la même classification KNN que précédemment, le taux de succès passe de 97.14% à 98.02%. Ainsi, nous pouvons affirmer que :

Le taux d'erreur de notre système de prédiction de chiffres manuscrits est de l'ordre de 2%.

Tous nos résultats sont consignés dans un fichier au format CSV.

Afin d'affiner l'interprétation des résultats, nous avons dressé un comparatif chiffre par chiffre. Nous pouvons alors nous apercevoir que la progression la plus fulgurante est de 2% et concerne les prédictions de 8.

Label	Taux de succès KNN	Taux de succès Data Augmentation
0.0	99.285714	99.489796
1.0	99.735683	99.735683
2.0	96.414729	97.286822
3.0	96.435644	97.821782
4.0	96.741344	97.657841
5.0	96.636771	97.757848
6.0	98.747390	98.851775
7.0	96.689387	97.176241
8.0	94.455852	96.611910
9.0	95.936571	97.621407

FIGURE 1.7: Taux de succès pour chaque chiffre

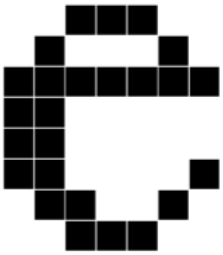
Une question légitime peut alors se poser : quelle méthode de Data Augmentation a eu le plus d'impact sur le taux de succès de prédictions ?
Nous avons dressé des tableaux similaires au précédent afin de répondre à cette problématique. Les deux rotations appliquées aux images sont les plus intéressantes d'un point de vue de minimisation du nombre d'erreurs : le taux de mauvaises réponses passe de 2.86% à 2.23%. Pour le chiffre 8 la précision augmente de 1,44% ce qui n'est pas négligeable. Étonnamment, le fait de zoomer sur les images provoque l'effet inverse : le taux d'erreur passe à 2.88% soit une différence par rapport à la classification k-nn de 0.02%. Peut-être pouvons nous expliquer cela par le manque de lisibilité pouvant apparaitre lors d'un zoom avant sur l'image.

Chapitre 2

Mise en place d'un système de prédiction artisanal

2.1 Un peu d'histoire

Vers la fin du vingtième siècle, lorsque l'on zoomait sur un texte affiché sur un écran d'ordinateur, les différents caractères semblaient êtres composés de carrés. Ce phénomène est appelé *pixellisation*. Avec un ordinateur contemporain, les lettres apparaissent en toutes circonstances agréables à l'oeil, cette évolution est due aux courbes de Bézier.



1980

Aujourd'hui

Elles ont été inventées par un ingénieur français, Pierre Bézier, travaillant pour Renault dans les années 1960. Son objectif était de traduire simplement mathématiquement des dessins complexes comme par exemple les hélices d'avions, les coques de bateaux ou encore les carrosseries. À cette époque, ce type de courbes était tracé à main levé du fait de l'impossibilité pour un ordinateur de les représenter.

2.2 Définitions

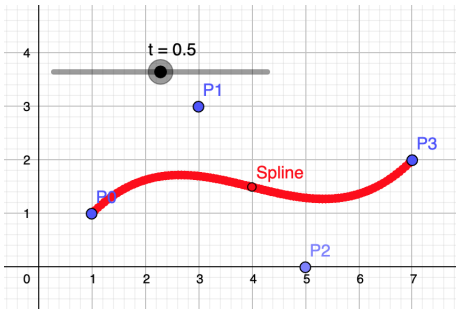
Dans le plan, une spline de Bézier cubique (resp. quadratique, resp. linéaire) est un arc de courbe paramétré, défini par un ensemble ordonné de 4 (resp. 3, resp. 2) points dits *de contrôle* qui constituent *le polygone de contrôle*, $(P0, P1, P2, P3)$.
L'arc est décrit par le point $P(t)$ quand le paramètre t parcourt le segment $]0, 1[$:

$$P(t) = (1-t)^3 P0 + 3t(1-t)^2 P1 + 3t^2(1-t)P2 + t^3 P3$$

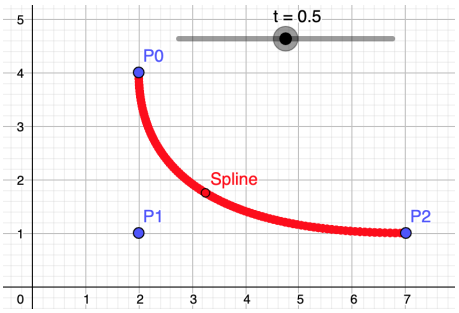
$$\text{(resp. } P(t) = (1-t)^2 P0 + 2t(1-t)P1 + t^2 P2 \text{)}$$

$$\text{(resp. } P(t) = (1-t)P0 + tP1 \text{)}$$

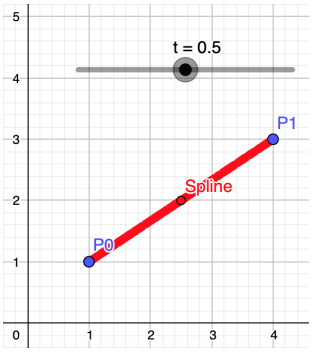
Sur Geogebra, nous avons représenté les trois splines évoquées :



Cubique



Quadratique



Linéaire

2.3 Stratégie adoptée

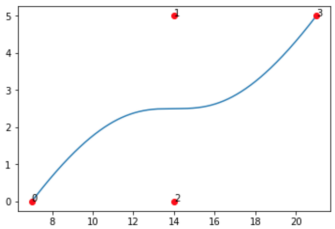
Afin de créer notre programme de prédiction et le comparer aux autres méthodes existantes, nous avons opté pour une approche simple. Les images MNIST sont fournies ajustées, normalisées centrées, telles que les chiffres apparaissent en noir, sur un carré de 28 pixels de côté. Cependant, les chiffres peuvent être floutés, penchés ou très déformés.
Nous allons découper l'image en trois parties de 28x9 pixels. Préalablement, pour chaque chiffre, un modèle aura été établi ce qui permettra de créer un encodeur. L'idée est de calculer le minimum de la somme des écarts entre pixels entre l'image fournie dans l'ensemble d'apprentissage de la database et notre modèle. Par exemple, un huit peut-être vu comme une composition d'une spline quadratique concave dans la partie haute, d'un croisement de deux splines linéaires pour la partie centrale, et enfin d'une spline quadratique convexe pour la partie basse. En créant des patrons pour chacune des parties de l'image, nous devrions parvenir à obtenir une similitude entre ceux-ci et les images représentants des 8.

2.4 Mise en place de l'algorithme

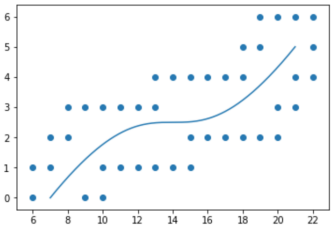
Une spline, en tant que telle, n'est qu'une courbe et ne correspond donc pas à un tracé pixellisé. Le premier défi a été de comprendre comment transformer une courbe en une image binaire. Une distance dans le plan a alors été requise. En effet, en calculant tous les points équidistants d'un tracé, on peut définir un *halo* autour de celui-ci et le discrétiser en chacun des pixels d'une image. Notre choix s'est porté sur la distance d provenant de la norme L_∞ définie pour deux points de $\mathbb{R}^2$ A et B comme :

$$d(A, B) = max(|\ x_A - x_B\ |, |\ y_A - y_B\ |)$$

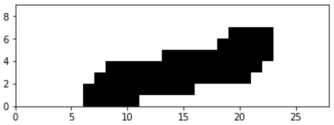
Géométriquement, on cherche le plus petit carré centré en M qui touche la courbe C, $d(M, C)$ est alors le coté de ce carré. Sans plus tarder, voici un exemple de transformation d'une spline cubique :



Étape 1 : Définition de la spline



Étape 2 : Création du halo



Étape 3 : Résultat

FIGURE 2.1: Processus de pixellisation

2.5 Bilan de l'expérience

L'objectif que nous nous étions initialement fixé (atteindre une bonne performance pour une partie seulement de l'ensemble d'apprentissage) n'est malheureusement pas atteint. En utilisant les 100 premiers éléments de l'ensemble d'apprentissage, le taux de précision global est de 14%, et ce quelque soit l'épaisseur du halo choisie. Notre système de prédiction ne peut être retenu dans ces conditions comme acceptable. Avec plus de temps, nous pourrions probablement faire diminuer le taux d'erreurs de notre système, mais nous avons préféré abandonner. Ce choix fut motivé par notre conviction que même en augmentant les données fournies à l'encodeur, nous ne parviendrions jamais à concurrencer la méthode KNN exposée en chapitre 1 en terme de vitesse d'exécution et de justesse.

Chapitre 3

Réseau de neurones artificiels

3.1 Définitions

3.1.1 Réseau de neurones

Un réseau de neurones (*neural networks* en anglais), est un système informatique matériel dont le fonctionnement est calqué sur celui du cerveau humain. Il s'agit d'une variété de technologie Deep Learning (apprentissage profond), qui fait partie du Machine Learning. Chaque neurone est considéré comme un point du réseau, il reçoit, une ou plusieurs informations en entrée, et renvoie une information en sortie. À la différence d'un réseau de serveur internet, les informations échangées dans un réseau de neurones sont simples : elles correspondent à seulement une intensité de signal, à l'image des échanges électrochimiques de notre cerveau.

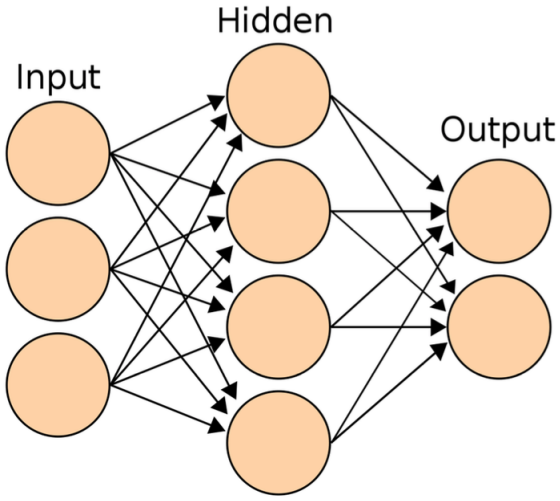


Figure 3.1: Schéma d'un réseau de neurones à plusieurs couches

Un réseau repose généralement sur plusieurs couches de neurones, s'activant les unes après les autres. La première couche, dite *couche d'entrée* ou *input layer*, reçoit le signal brut (les données). Puis viennent les couches centrales du réseau, comme nous ne voyons pas les échanges d'informations entrant et sortant de leurs neurones, elles ont été baptisées *couches cachées* (*hidden layers*). La dernière couche émet le signal final, d'où son appellation d'*output layers*.

De manière générale, le nombre de neurones en entrée est défini en fonction de l'information traitée, celui des outputs layers, dépend lui du nombre de réponses possibles que le réseau peut renvoyer.

Par exemple, si à partir d'une photo, notre réseau doit faire la différence entre un chat et un chien, les données pourront être le nombre de pixels de la photo et les sorties seront : "chat" ou "chien".

Une fois que l'on connaît le nombre d'input et d'output, on peut définir le nombre de couche cachées ainsi que le nombre de neurones associés. Cependant, obtenir un réseau optimisé répondant à l'objectif visé est encore bien souvent compliqué, les problèmes de sur ou de sous-apprentissage sont fréquents.

Dans le cadre d'un *Full-Connected Network*, c'est-à-dire lorsque chaque neurone est connecté à l'ensemble des neurones de la couche précédente, le neurone reçoit plusieurs valeurs d'entrées, chacune possédant un poids. Il doit alors calculer ce que l'on appelle la valeur *d'agrégation* de la couche, c'est à dire la valeur qu'un neurone calcule avant d'être transmise à la fonction d'activation. Elle comporte un biais, qui peut être considéré comme le seuil d'activation de notre neurone. Celui-ci permet de palier à une activation du neurone dûe à un bruit résiduel. L'image de cette valeur d'agrégation par la fonction d'activation nous donne la sortie du neurone. Elle est comprise entre 0 et 1, où 0 signifie que le neurone est totalement inactif, et 1, que le neurone est totalement actif. Il existe plusieurs types de fonctions d'activation comme la tangente hyperbolique, la sigmoïde ou encore la rectification linéaire. L'ensemble de ce processus est nommé *feed-forward*.

Le but du réseau lors de l'apprentissage est de trouver les valeurs adéquates des poids de connexions entrantes ainsi que des biais pour chacun des neurones du réseau. Mathématiquement, on peut exprimer le calcul de la valeur d'activation entre chaque couches par l'équation matricielle suivante :

$$a^1 = \sigma(Wa^0 + b)$$

$$\begin{pmatrix} a_0^1 \\ \dots \\ a_k^1 \end{pmatrix} = \sigma(\begin{pmatrix} W_{00} & \dots & W_{0n} \\ \dots & \dots & \dots \\ W_{k0} & \dots & W_{kn} \end{pmatrix} \begin{pmatrix} a_0^0 \\ \dots \\ a_n^0 \end{pmatrix} + \begin{pmatrix} b_0 \\ \dots \\ b_k \end{pmatrix})$$

Où :

- a^1 correspond à la valeur d'activation de la couche calculée
- a^0 correspond à la valeur d'activation de la couche précedente
- $W \in M_{k,n}(\mathbb{R})$ représente l'ensemble des poids de connexions entre deux couches de neurones
- b est le biais de chaque neurone
- σ est la fonction d'activation

Par la suite, afin d'indiquer au réseau dans quelle mesure il est éloigné du résultat souhaité, nous utiliserons une fonction coût, C, définie comme suit : $C = \sum_{i=0}^{n}(t_i - z_i)^2$, où les t_i représentent les valeurs de sorties des neurones, et les z_i les valeurs attendues. La moyenne de cette fonction sur un grand nombre d'images, retiendra donc notre attention.

Au lancement de l'apprentissage, on attribue des valeurs aléatoires pour chaque paramètres du réseau. Ensuite, ce dernier utilisera l'algorithme de descente du gradient en N dimensions (avec N, nombre total de poids et de biais modifiables) afin d'optimiser le choix des poids et des biais. Ce processus de modification en partant des output layer et remontant couches par couches, est intitulé *backpropagation*.

C'est donc grâce à la répétition de feed-forward et de backpropagation durant l'apprentissage que l'on obtient un réseau de neurone optimal pour le problème rencontré.

3.1.2 Convolutions

Les réseaux de neurones à convolution (*Convolutional Neural Networks*, *CNN* ou *ConvNet* en anglais) sont régulièrements invoqués lorsque l'on entend que les limites scientifiques ont été repoussées grâce à des techniques d'apprentissage profond. Aujourd'hui, ils sont capables d'apprendre à trier des images par catégories avec dans certains cas, de meilleurs résultats qu'après un triage manuel. Le CNN compare les images pixellisées fragment par fragment, qui sont appelés *caractéristique* ou *features*. En trouvant des caractéristiques qui se ressemblent plus ou moins, le CNN devient plus précis qu'un système qui comparerait la globalité des images. Chaque feature peut-être vue comme une image de taille réduite. Cependant, lorsqu'une image se présente au ConvNet, il ne sait pas si les caractéristiques seront présentes et où elles pourraient être. Il va donc chercher à les trouver dans n'importe quelle position. En calculant dans toute l'image si une caractéristique est présente, nous faisons un filtrage. Les mathématiques que nous utilisons pour réaliser cette opération sont appelés *une convolution*, de laquelle les réseaux de neurones à convolution tiennent leur nom. Il existe divers moyen de convoluer, on peut prendre le maximum du noyau formé, la somme ou encore, procéder comme ci-dessous.

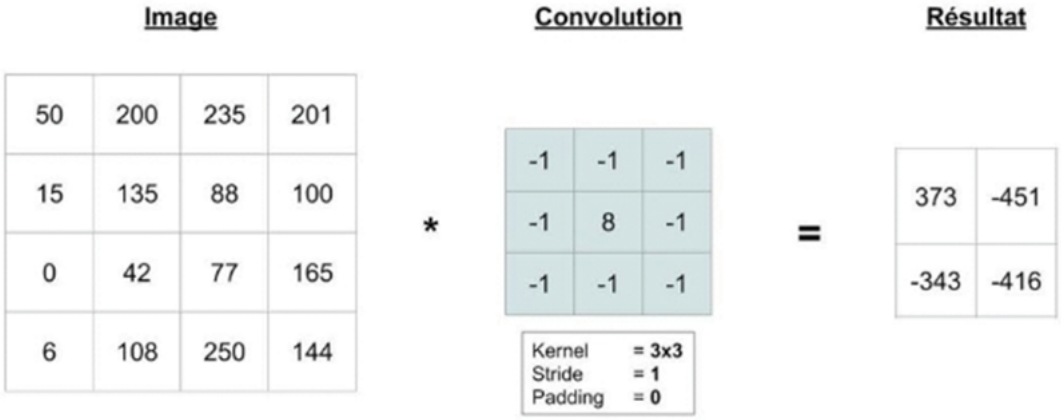


FIGURE 3.2: Un exemple de convolution

Afin d'expliciter la résolution du calcul, observons le détail du calcul du premier élément de la dernière matrice. On a :

$$77*(-1)+42*(-1)+0*(-1)+88*(-1)+135*8+15*(-1)+235*(-1)+200*(-1)+50*(-1) = 373$$

Pour l'anecdote, la matrice de convolution de l'exemple est souvent utilisée pour des détection de contours.

Contrairement aux méthodes traditionnelles, les features ne sont pas pré-définies selon un formalisme particulier, mais sont apprises par le réseau lors la phase d'entraînement. Les noyaux des filtres désignent les poids de la couche de convolution. Ils sont initialisés puis mis à jour par rétropropagation du gradient. C'est là que les réseaux de neurones convolutifs trouvent leur principal attrait : ils sont capables de déterminer, seul, les éléments discriminants d'une image, en s'adaptant au problème posé. Pour reprendre un exemple précédent, si nous devons comparer les chats et les chiens, les features automatiquement définies peuvent décrire la forme des oreilles ou des pattes.

3.2 Notre réseau de neurones

3.2.1 Description du programmme

Dans le cadre de notre projet, nous voulions étudier les splines présentent dans les parties haut milieu et basse de l'écriture manuscrite des chiffres de la base de données MNIST. L'objectif a été d'obtenir de meilleurs résultats qu'au chapitre précédent. À ce titre, nous avons donc découpé les images en 3 parties, afin que celle-ci soit équitable et limiter la perte de données„ nous avons cette fois redimensionné les images en des carrés de 30 pixels de longueur. Par la suite, nous avons codé 4 réseaux. Les 3 premiers, similaires dans leurs nombres de hidden layers, reçoivent en entrée un tiers (haut milieu ou bas) de l'image, celle-ci est convolué plusieurs fois, avec des noyau 3x3. Puis, dans un réseau full-connected, on passe d'une couche de 120 neurones, à 64, pour obtenir en output le nombre de patterns attendus selon la partie de l'image (8 pour le haut, 10 au milieu et enfin 8 pour le bas). Enfin le quatrième réseau, récupère en entrée les 26 coefficients des patterns, puis une couche de 10 neurones (chacun représentant un label de chiffre) pour ensuite ne posséder qu'une sortie, le label de la prédiction. Les 3 premiers réseaux utilisent la rectification linéaire (ReLu) comme fonction d'activation. Celle-ci est définie comme : $f(x) = max(0, x)$.
Nous avons également utilisé la fonction d'activation treshold, qui est similaire à Relu à la différence que l'on peut choisir la valeur seuil. Le quatrième utilise quant à lui la fonction d'activation LogSoftmax, qui est interprétée par la formule pour tout $z = (z_1, ..., z_K) \in \mathbb{R}^K$:

$$LogSoftmax(z_j) = \log(\frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}), \forall j \in \{1, ..., K\}$$

D'autre part, lors de l'apprentissage, plutôt que de répéter le processus de feed-forward et de back-propagation pour chaque image, qui pourrait conduire à de nombreuse erreurs dans l'algorithme de descente du gradient, nous allons procéder par batch (lot) de 64 images.

3.2.2 Résultats

Nous pouvons constater que le modèle apprends vite, un *epoch* (cycle d'apprentissage) dure une quinzaine de secondes ce qui induit un temps total de 5min20. Le taux d'accuracy atteint 97,5%pour un coût de 0,001.

Par la suite, lorsque que l'on affiche les valeurs des outputs de nos premiers réseaux (correspondant au pattern de chaque partie des chiffres) on remarque qu'effectivement, ce sont les même patterns qui sont activés sur des chiffres ayant la même spline sur la même partie. Par exemple, en analysant la spline quadratique que l'on peut retrouver dans la partie haute du 0, du 8 et du 9, on remarque que le pattern 7* est celui qui s'active principalement. On peut donc conjecturer que celui-ci est de cette forme. Malheureusement, n'étant pas parvenus à l'afficher, nous ne pouvons pas être certain de cette conjecture.
(* : à chaque réapprentissage, il se peut que le numéro du pattern change, mais le principe reste inchangé)

Conclusion

Ce projet a été très enrichissant tant sur le plan personnel que professionnel. Nous avons découvert pour la première fois la base de données MNIST ainsi que l'importance qu'elle a eu sur notre société. Aujourd'hui, plus que jamais, l'intelligence artificielle est au centre des recherches de grands groupes. Par exemple, des chercheurs de Google ont récemment publié un article présentant une nouvelle solution d'Intelligence Artificielle capable d'imiter une caractéristique de l'esprit humain : la cécité d'inattention. Il s'agit de la capacité de notre cerveau d'occulter volontairement certains détails d'une situation pour se concentrer sur l'essentiel. Cette aptitude a été mise en valeur lors du bien célèbre test du gorille invisible où les participants devaient regarder attentivement une vidéo où deux équipes de basket, se lançaient un ballon. Ils devaient alors compter le nombre de passes entre les membres de l'une d'entre elles. Pendant la partie, une personne déguisée en gorille traversait la scène. On demandait ensuite aux participants combien de passes ils avaient comptées et s'ils avaient vu quelque chose qui sortait de l'ordinaire. Environ 50 % d'entre eux n'avaient pas vu passer le gorille. Les travaux de Google s'inscrivent ainsi dans leur poursuite d'imitation de l'esprit humain. Leur solution est capable de trier les éléments entrants selon leur pertinence. Le système a été entraîné sur des environnements issus du monde du jeu vidéo (VizDoom et CarRacing). Les premiers résultats sont encourageants, leur production est capable de s'adapter à son environnement de manière très rapide, avec beaucoup moins de paramètres que d'autres programmes comparables. Ces travaux permettront peut-être aux prototypes issus de l'intelligence artificielle de prendre des décisions plus rapide dans un environnement complexe.

Enfin, nous souhaitons remercier notre tuteur M.André Galligo qui malgré le contexte très particulier que nous avons traversé, a su être à notre écoute pour nous guider tout au long de notre travail.

Bibliographie

[1] www.wikipedia.fr.

[2] Sciences et Vie Hors-Série, *I.A. Les 10 ans qui ont tout changé*, Mars 2020.

[3] https ://pjreddie.com/projects/mnist-in-csv/.

[4] https ://www.kaggle.com/gauthampughazh/digit-recognition-using-knn.

[5] http ://yann.lecun.com/exdb/mnist/

[6] http ://lyceeenligne.free.fr/IMG/pdf/16_bezier.pdf

[7] https ://www.imo.universite-paris-saclay.fr/ perrin/CAPES/geometrie/BezierDP.pdf.

[8] Nicolas Pasquier, *Cours de Data Mining et Machine Learning M1IM*

[9] https ://www.youtube.com/watch ?v=Ccxd6qzqFms

[10] https ://www.youtube.com/watch ?v=2pNjW-2944Y

[11] https ://www.lebigdata.fr/reseau-de-neurones-artificiels-definition

[12] Extrait du livre de Cyril de Sousa Cardoso, Emmanuelle Galou, Aurore Kervella et Patrick Kwok, *Data Power*

[13] https ://www.miximum.fr/blog/introduction-au-deep-learning-2/

[14] https ://medium.com/@CharlesCrouspeyre/comment-les-réseaux-de-neurones-à-convolution-fonctionnent-b288519dbcf8

[15] https ://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5083336-decouvrez-les-differentes-couches-dun-cnn

[16] https ://ai.googleblog.com/2020/06/using-selective-attention-in.html ?m=1

[17] Post LinkedIn de Guillaume Besson

Annexes

Code du chapitre 1

```
# Librairies utilisees

import numpy as np # Linear algebra
import pandas as pd # For data manipulation
import json
import os
import matplotlib.pyplot as plt # For visualization
from sklearn.neighbors import KNeighborsClassifier # For modelling
from sklearn.model_selection import cross_val_score, GridSearchCV, StratifiedKFold
# For evaluation and hyperparameter tuning
from sklearn.metrics import confusion_matrix, classification_report # For evaluatio
n
from scipy.ndimage import shift, rotate, zoom # For data augmentation
from IPython.display import FileLink # For downloading the output file

#Conversion des données au format CSV

def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28*28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image)+"\n")
    f.close()
    o.close()
    l.close()

convert("train-images-idx3-ubyte", "train-labels-idx1-ubyte",
        "mnist_train.csv", 60000)
convert("t10k-images-idx3-ubyte", "t10k-labels-idx1-ubyte",
        "mnist_test.csv", 10000)


train_df = pd.read_csv("mnist_train.csv") #ensemble d'apprentissage
test_df = pd.read_csv("mnist_test.csv") #ensemble de test

train_df.info() # 60 000 entrees comme indique sur le site officiel , les colonnes
representent les pixels
test_df.info() # 10 000  ------------------""---------------------
```

```
X_train = train_df.iloc[:, 1:].values
y_train = train_df.iloc[:, 0].values
X_test = test_df.iloc[:, 1:].values
y_test = test_df.iloc[:,0].values

# visualisons un nombre

some_digit = X_train[40]


some_digit_image = some_digit.reshape(28, 28)


print(f"Label: {y_train[40]}")

plt.imshow(some_digit_image, cmap="binary")
plt.show()


#Cross validation

stratified_fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for fold, indices in enumerate(stratified_fold.split(X_train, y_train)):

    X_train_, y_train_ = X_train[indices[0]], y_train[indices[0]]
    X_test_, y_test_ = X_train[indices[1]], y_train[indices[1]]

    estimator = KNeighborsClassifier()
    estimator.fit(X_train_, y_train_)
    predictions = estimator.predict(X_test_)

    print(f"Classification report for Fold {fold + 1}:")
    print(classification_report(y_test_, predictions, digits=3), end="\n\n")

    print(f"Confusion Matrix for Fold {fold + 1}:")
    print(confusion_matrix(y_test_, predictions), end="\n\n")

    del X_train_
    del X_test_
    del y_train_
    del y_test_

#Choix des parametres

grid_params = {
    "weights": ['uniform', 'distance'],
    "n_neighbors": [3, 4, 5, 6, 8, 10]
}
estimator = KNeighborsClassifier()
grid_estimator = GridSearchCV(estimator, # Base estimator
                              grid_params, # Parameters to tune
```

```
                               cv=stratified_fold, # cross-validation stratergy
                               verbose=2, # Verbosity of the logs
                               n_jobs=-
1) # Number of jobs to be run concurrently with -1 meaning all the processors



# Fitting the estimator with training data
grid_estimator.fit(X_train, y_train)

print(f"Best Score: {grid_estimator.best_score_}", end="\n\n")
print("Grid Search CV results:")
results_df = pd.DataFrame(grid_estimator.cv_results_)
results_df

estimator = KNeighborsClassifier(n_neighbors=4, weights='distance')
estimator.fit(X_train, y_train)
predictions = estimator.predict(X_test)

# visualisons un nombre
i=40
some_digit = X_test[i]
some_digit_image = some_digit.reshape(28, 28)
print("Label prédit:")
print(predictions[i])
plt.imshow(some_digit_image, cmap="binary")
plt.show()

def shift_in_one_direction(image, direction):
    """
    Shifts an image by one pixel in the specified direction
    """
    if direction == "DOWN":
        image = shift(image, [1, 0])
    elif direction == "UP":
        image = shift(image, [-1, 0])
    elif direction == "LEFT":
        image = shift(image, [0, -1])
    else:
        image = shift(image, [0, 1])

    return image


def shift_in_all_directions(image):
    """
    Shifts an image in all the directions by one pixel
    """
    reshaped_image = image.reshape(28, 28)

    down_shifted_image = shift_in_one_direction(reshaped_image, "DOWN")
    up_shifted_image = shift_in_one_direction(reshaped_image, "UP")
```

```
    left_shifted_image = shift_in_one_direction(reshaped_image, "LEFT")
    right_shifted_image = shift_in_one_direction(reshaped_image, "RIGHT")

    return (down_shifted_image, up_shifted_image,
            left_shifted_image, right_shifted_image)


def rotate_in_all_directions(image, angle):
    """
    Rotates an image clockwise and anti-clockwise
    """
    reshaped_image = image.reshape(28, 28)

    rotated_images = (rotate(reshaped_image, angle, reshape=False),
                      rotate(reshaped_image, -angle, reshape=False))

    return rotated_images


def clipped_zoom(image, zoom_ranges):
    """
    Clips and zooms an image at the specified zooming ranges
    """
    reshaped_image = image.reshape(28, 28)

    h, w = reshaped_image.shape

    zoomed_images = []
    for zoom_range in zoom_ranges:
        zh = int(np.round(h / zoom_range))
        zw = int(np.round(w / zoom_range))
        top = (h - zh) // 2
        left = (w - zw) // 2

        zoomed_images.append(zoom(reshaped_image[top:top+zh, left:left+zw],
                                  zoom_range))

    return zoomed_images

def alter_image(image):
    """
    Alters an image by shifting, rotating, and zooming it
    """
    shifted_images = shift_in_all_directions(image)
    rotated_images = rotate_in_all_directions(image, 10)
    zoomed_images = clipped_zoom(image, [1.1, 1.2])

    return np.r_[shifted_images, rotated_images, zoomed_images]

X_train_add = np.apply_along_axis(alter_image, 1, X_train).reshape(-1, 784)
y_train_add = np.repeat(y_train, 8)
```

```
print(f"X_train_add shape: {X_train_add.shape}")
print(f"y_train_add shape: {y_train_add.shape}")

X_train_combined = np.r_[X_train, X_train_add]
y_train_combined = np.r_[y_train, y_train_add]

del X_train
del X_train_add
del y_train
del y_train_add

print(f"X_train_combined shape: {X_train_combined.shape}")
print(f"y_train_combined shape: {y_train_combined.shape}")

cdata_estimator = KNeighborsClassifier(n_neighbors=4, weights='distance')
cdata_estimator.fit(X_train_combined, y_train_combined)
cdata_estimator_predictions = cdata_estimator.predict(X_test)

results = {'Label': y_test, 'Predictions KNN': predictions, 'Predictions Data Augme
ntation':cdata_estimator_predictions}
results_df =pd.DataFrame(data=results)
results_df.to_csv('Results.csv', index=False)
FileLink('Results.csv')

results_df

error_rate1 = 0
error_rate2 = 0

for i in range(9999):
    if results_df['Label'][i] - results_df['Predictions KNN'][i] != 0:
        error_rate1 += 1
    if results_df['Label'][i] - results_df['Predictions Data Augmentation'][i] != 0
:
        error_rate2 += 1
error_rate_knn = (error_rate1/100)
error_rate_DataAug = (error_rate2/100)
print(f"Pour la méthode KNN le taux de précision est de: {100-
error_rate_knn}% soit un taux d'erreur de: {error_rate_knn}%")
print(f"En utilisant la Data Augmentation, le taux de précision devient: {100-
error_rate_DataAug}% soit un taux d'erreur de: {error_rate_DataAug}%")

tot_chiffre = np.zeros([10,1])

for i in range(10):
    for j in range(9999):
        if results_df['Label'][j] == i:
            tot_chiffre[i,0] += 1


# calcul des taux de précision par chiffres
```

```
prec_knn_chif = np.zeros([10])
prec_da_chif = np.zeros([10])


for i in range(10):
    for j in range(9999):
        if results_df['Label'][j] == results_df['Predictions KNN'][j] == i:
            prec_knn_chif[i] += 1

        if results_df['Label'][j] == results_df['Predictions Data Augmentation'][j]
 == i:
            prec_da_chif[i] += 1

for i in range(10):
    prec_knn_chif[i] = (prec_knn_chif[i]/tot_chiffre[i])*100
    prec_da_chif[i] = (prec_da_chif[i]/tot_chiffre[i])*100

Chiffres = [0,1,2,3,4,5,6,7,8,9]

res = np.zeros([10,3])
for i in range(10):
        res[i,0] += Chiffres[i]
        res[i,1] += prec_knn_chif[i]
        res[i,2] += prec_da_chif[i]

res_df = pd.DataFrame(data=res)
res_df.columns = ['Label', 'Taux de succès KNN', 'Taux de succès Data Augmentation'
]
res_df.to_csv('Res.csv', index=False)
FileLink('Res.csv')
res_df

#shifted
def alter_image_shifted(image):
    """
    Alters an image by shifting, rotating, and zooming it
    """
    shifted_images = shift_in_all_directions(image)
    #rotated_images = rotate_in_all_directions(image, 10)
    #zoomed_images = clipped_zoom(image, [1.1, 1.2])

    #return np.r_[shifted_images, rotated_images, zoomed_images]
    return np.r_[shifted_images]

X_train_add_shifted = np.apply_along_axis(alter_image_shifted, 1, X_train).reshape(
-1, 784)
y_train_add_shifted = np.repeat(y_train, 4)

print(f"X_train_add_shifted shape: {X_train_add_shifted.shape}")
print(f"y_train_add_shifted shape: {y_train_add_shifted.shape}")
```

```
X_train_combined_shifted = np.r_[X_train, X_train_add_shifted]
y_train_combined_shifted = np.r_[y_train, y_train_add_shifted]

#del X_train
del X_train_add_shifted
#del y_train
del y_train_add_shifted

print(f"X_train_combined_shifted shape: {X_train_combined_shifted.shape}")
print(f"y_train_combined_shifted shape: {y_train_combined_shifted.shape}")

cdata_estimator_shifted = KNeighborsClassifier(n_neighbors=4, weights='distance')
cdata_estimator_shifted.fit(X_train_combined_shifted, y_train_combined_shifted)
cdata_estimator_predictions_shifted = cdata_estimator_shifted.predict(X_test)

results_shifted = {'Label': y_test, 'Predictions KNN': predictions, 'Predictions Da
ta Augmentation Shifted':cdata_estimator_predictions_shifted}
results_df_shifted =pd.DataFrame(data=results_shifted)
results_df_shifted.to_csv('Results_shifted.csv', index=False)
FileLink('Results_shifted.csv')

results_df_shifted

error_rate1 = 0
error_rate2 = 0

for i in range(9999):
    if results_df_shifted['Label'][i] - results_df_shifted['Predictions KNN'][i] !=
 0:
        error_rate1 += 1
    if results_df_shifted['Label'][i] - results_df_shifted['Predictions Data Augmen
tation Shifted'][i] != 0:
        error_rate2 += 1
error_rate_knn = (error_rate1/100)
error_rate_DataAug = (error_rate2/100)
print(f"Pour la méthode KNN le taux de précision est de: {100-
error_rate_knn}% soit un taux d'erreur de: {error_rate_knn}%")
print(f"En utilisant la Data Augmentation Shifted, le taux de précision devient: {1
00-error_rate_DataAug}% soit un taux d'erreur de: {error_rate_DataAug}%")


# calcul des taux de précision par chiffres
prec_knn_chif_shifted = np.zeros([10])
prec_da_chif_shifted = np.zeros([10])


for i in range(10):
    for j in range(9999):
        if results_df_shifted['Label'][j] == results_df_shifted['Predictions KNN'][
j] == i:
            prec_knn_chif_shifted[i] += 1
```

```
        if results_df_shifted['Label'][j] == results_df_shifted['Predictions Data A
ugmentation Shifted'][j] == i:
            prec_da_chif_shifted[i] += 1

for i in range(10):
    prec_knn_chif_shifted[i] = (prec_knn_chif_shifted[i]/tot_chiffre[i])*100
    prec_da_chif_shifted[i] = (prec_da_chif_shifted[i]/tot_chiffre[i])*100

Chiffres = [0,1,2,3,4,5,6,7,8,9]

res_shifted = np.zeros([10,3])
for i in range(10):
        res_shifted[i,0] += Chiffres[i]
        res_shifted[i,1] += prec_knn_chif_shifted[i]
        res_shifted[i,2] += prec_da_chif_shifted[i]

res_df_shifted = pd.DataFrame(data=res_shifted)
res_df_shifted.columns = ['Label', 'Taux de succès KNN', 'Taux de succès Data Augme
ntation Shifted']
res_df_shifted.to_csv('Res_Shifted.csv', index=False)
FileLink('Res_Shifted.csv')
res_df_shifted

#rotated
def alter_image_rotated(image):
    """
    Alters an image by shifting, rotating, and zooming it
    """
    #shifted_images = shift_in_all_directions(image)
    rotated_images = rotate_in_all_directions(image, 10)
    #zoomed_images = clipped_zoom(image, [1.1, 1.2])

    #return np.r_[shifted_images, rotated_images, zoomed_images]
    return np.r_[rotated_images]

X_train_add_rotated = np.apply_along_axis(alter_image_rotated, 1, X_train).reshape(
-1, 784)
y_train_add_rotated = np.repeat(y_train, 2)

print(f"X_train_add_rotated shape: {X_train_add_rotated.shape}")
print(f"y_train_add_rotated shape: {y_train_add_rotated.shape}")

X_train_combined_rotated = np.r_[X_train, X_train_add_rotated]
y_train_combined_rotated = np.r_[y_train, y_train_add_rotated]

#del X_train
del X_train_add_rotated
#del y_train
del y_train_add_rotated

print(f"X_train_combined_rotated shape: {X_train_combined_rotated.shape}")
```

```
print(f"y_train_combined_rotated shape: {y_train_combined_rotated.shape}")

cdata_estimator_rotated = KNeighborsClassifier(n_neighbors=4, weights='distance')
cdata_estimator_rotated.fit(X_train_combined_rotated, y_train_combined_rotated)
cdata_estimator_predictions_rotated = cdata_estimator_rotated.predict(X_test)


results_rotated = {'Label': y_test, 'Predictions KNN': predictions, 'Predictions Da
ta Augmentation Rotated':cdata_estimator_predictions_rotated}
results_df_rotated =pd.DataFrame(data=results_rotated)
results_df_rotated.to_csv('Results_rotated.csv', index=False)
FileLink('Results_rotated.csv')

results_df_rotated

error_rate1 = 0
error_rate2 = 0

for i in range(9999):
    if results_df_rotated['Label'][i] - results_df_rotated['Predictions KNN'][i] !=
 0:
        error_rate1 += 1
    if results_df_rotated['Label'][i] - results_df_rotated['Predictions Data Augmen
tation Rotated'][i] != 0:
        error_rate2 += 1
error_rate_knn = (error_rate1/100)
error_rate_DataAug = (error_rate2/100)
print(f"Pour la méthode KNN le taux de précision est de: {100-
error_rate_knn}% soit un taux d'erreur de: {error_rate_knn}%")
print(f"En utilisant la Data Augmentation Rotated, le taux de précision devient: {1
00-error_rate_DataAug}% soit un taux d'erreur de: {error_rate_DataAug}%")


# calcul des taux de précision par chiffres
prec_knn_chif_rotated = np.zeros([10])
prec_da_chif_rotated = np.zeros([10])


for i in range(10):
    for j in range(9999):
        if results_df_rotated['Label'][j] == results_df_rotated['Predictions KNN'][
j] == i:
            prec_knn_chif_rotated[i] += 1

        if results_df_rotated['Label'][j] == results_df_rotated['Predictions Data A
ugmentation Rotated'][j] == i:
            prec_da_chif_rotated[i] += 1

for i in range(10):
    prec_knn_chif_rotated[i] = (prec_knn_chif_rotated[i]/tot_chiffre[i])*100
    prec_da_chif_rotated[i] = (prec_da_chif_rotated[i]/tot_chiffre[i])*100
```

```
Chiffres = [0,1,2,3,4,5,6,7,8,9]

res_rotated = np.zeros([10,3])
for i in range(10):
        res_rotated[i,0] += Chiffres[i]
        res_rotated[i,1] += prec_knn_chif_rotated[i]
        res_rotated[i,2] += prec_da_chif_rotated[i]

res_df_rotated = pd.DataFrame(data=res_rotated)
res_df_rotated.columns = ['Label', 'Taux de succès KNN', 'Taux de succès Data Augme
ntation Rotated']
res_df_rotated.to_csv('Res_Rotated.csv', index=False)
FileLink('Res_Rotated.csv')
res_df_rotated

#zoomed
def alter_image_zoomed(image):
    """
    Alters an image by shifting, rotating, and zooming it
    """
    #shifted_images = shift_in_all_directions(image)
    #rotated_images = rotate_in_all_directions(image, 10)
    zoomed_images = clipped_zoom(image, [1.1, 1.2])

    return np.r_[zoomed_images]

X_train_add_zoomed = np.apply_along_axis(alter_image_zoomed, 1, X_train).reshape(-
1, 784)
y_train_add_zoomed = np.repeat(y_train, 2)

print(f"X_train_add_zoomed shape: {X_train_add_zoomed.shape}")
print(f"y_train_add_zoomed shape: {y_train_add_zoomed.shape}")

X_train_combined_zoomed = np.r_[X_train, X_train_add_zoomed]
y_train_combined_zoomed = np.r_[y_train, y_train_add_zoomed]

#del X_train
del X_train_add_zoomed
#del y_train
del y_train_add_zoomed

print(f"X_train_combined_zoomed shape: {X_train_combined_zoomed.shape}")
print(f"y_train_combined_zoomed shape: {y_train_combined_zoomed.shape}")

cdata_estimator_zoomed = KNeighborsClassifier(n_neighbors=4, weights='distance')
cdata_estimator_zoomed.fit(X_train_combined_zoomed, y_train_combined_zoomed)
cdata_estimator_predictions_zoomed = cdata_estimator_zoomed.predict(X_test)

results_zoomed = {'Label': y_test, 'Predictions KNN': predictions, 'Predictions Dat
a Augmentation Zoomed':cdata_estimator_predictions_zoomed}
results_df_zoomed =pd.DataFrame(data=results_zoomed)
```

```
results_df_zoomed.to_csv('Results_zoomed.csv', index=False)
FileLink('Results_zoomed.csv')

results_df_zoomed

error_rate1 = 0
error_rate2 = 0

for i in range(9999):
    if results_df_zoomed['Label'][i] - results_df_zoomed['Predictions KNN'][i] != 0
:
        error_rate1 += 1
    if results_df_zoomed['Label'][i] - results_df_zoomed['Predictions Data Augmenta
tion Zoomed'][i] != 0:
        error_rate2 += 1
error_rate_knn = (error_rate1/100)
error_rate_DataAug = (error_rate2/100)
print(f"Pour la méthode KNN le taux de précision est de: {100-
error_rate_knn}% soit un taux d'erreur de: {error_rate_knn}%")
print(f"En utilisant la Data Augmentation Zoomed, le taux de précision devient: {10
0-error_rate_DataAug}% soit un taux d'erreur de: {error_rate_DataAug}%")


# calcul des taux de précision par chiffres
prec_knn_chif_zoomed = np.zeros([10])
prec_da_chif_zoomed = np.zeros([10])


for i in range(10):
    for j in range(9999):
        if results_df_zoomed['Label'][j] == results_df_zoomed['Predictions KNN'][j]
 == i:
            prec_knn_chif_zoomed[i] += 1

        if results_df_zoomed['Label'][j] == results_df_zoomed['Predictions Data Aug
mentation Zoomed'][j] == i:
            prec_da_chif_zoomed[i] += 1

for i in range(10):
    prec_knn_chif_zoomed[i] = (prec_knn_chif_zoomed[i]/tot_chiffre[i])*100
    prec_da_chif_zoomed[i] = (prec_da_chif_zoomed[i]/tot_chiffre[i])*100

Chiffres = [0,1,2,3,4,5,6,7,8,9]

res_zoomed = np.zeros([10,3])
for i in range(10):
        res_zoomed[i,0] += Chiffres[i]
        res_zoomed[i,1] += prec_knn_chif_zoomed[i]
        res_zoomed[i,2] += prec_da_chif_zoomed[i]

res_df_zoomed = pd.DataFrame(data=res_zoomed)
```

```
res_df_zoomed.columns = ['Label', 'Taux de succès KNN', 'Taux de succès Data Augmen
tation Zoomed']
res_df_zoomed.to_csv('Res_Zoomed.csv', index=False)
FileLink('Res_Zoomed.csv')
res_df_zoomed

#recherche d'erreurs
for i in range(250):
    if results_df['Label'][i] - results_df['Predictions KNN'][i] != 0:
        print(i)
```

Code du chapitre 2

```python
import numpy as np
import pandas as pd
from scipy.special import comb
from matplotlib import pyplot as plt

# Définition de la fonction P(t) qui calcule les valeurs de la spline
def courbe_bezier(points_controle, disc=1000):
    n = len(points_controle)
    t = np.linspace(0.0,1.0,disc)

    #Ici on sépare les x des y en créant deux arrays
    x = np.zeros(n, dtype=float)
    y = np.zeros(n, dtype=float)
    for i in range(0,n):
        x[i] = points_controle[i,0]
        y[i] = points_controle[i,1]


    tab_poly = np.zeros((n,disc), dtype=float)
    if n == 2:
        tab_poly[0,] = 1-t
        tab_poly[1,] = t
    if n == 3:
        tab_poly[0,] = (1-t)**2
        tab_poly[1,] = 2*t*(1-t)
        tab_poly[2,] = t**2
    if n == 4:
        tab_poly[0,] = (1-t)**3
        tab_poly[1,] = 3*t*((1-t)**2)
        tab_poly[2,] = (3*(t**2))*(1-t)
        tab_poly[3,] = t**3

    xres = np.dot(x,tab_poly)
    yres = np.dot(y,tab_poly)

    return xres, yres

def Halo_Quadratique(point1,point2,point3,dis):

    points_controle = np.array([ point1,point2,point3 ])
    x_controle = np.array([p[0] for p in points_controle])
    y_controle = np.array([p[1] for p in points_controle])

    xres, yres = courbe_bezier(points_controle, disc=1000)
   # plt.plot(xres, yres)

# Cette partie du code sert à afficher les points de contrôle
# "ro" signifie red circle
    #plt.plot(x_controle,y_controle, "ro")

# Cette petite partie de code sert simplement à numéroter les points de contrôle
```

```
  #  for num in range(len(points_controle)):
  #      plt.text(points_controle[num][0], points_controle[num][1], num)

   # plt.show()

    distance = np.zeros([28,9])   #On utilise une grille 9*28 et on va calculer la
distance infinie par rapport à la spline

    for i in range(28):
        for j in range(9):
            test=[]
            for k in range(len(xres)):
                test.append(max(abs(xres[k] - i),abs(yres[k] - j)))
            distance[i,j] = min(test)
    new_dist = distance.round()

    xdist=[]
    ydist=[]
    pixels = np.zeros([28,9])
    for i in range(28):
        for j in range(9):
            if new_dist[i,j] == dis:
                xdist.append(i)   #abscisse du point de distance d de la spline
                ydist.append(j)
                pixels[i,j] = 255  # on colorie le pixel correspondant en noir
            if new_dist[i,j] == 0:
                pixels[i,j] = 255

    #plt.plot(xres, yres)
    #plt.scatter(xdist,ydist)
    #plt.figure()
    #plt.imshow(pixels.T, cmap='binary', extent=[0,28,0,9],origin='lower')
    return(pixels)

def Halo_Cubique(point1,point2,point3,point4,dis):

    points_controle = np.array([ point1,point2,point3,point4 ])
    x_controle = np.array([p[0] for p in points_controle])
    y_controle = np.array([p[1] for p in points_controle])

    xres, yres = courbe_bezier(points_controle, disc=1000)
    #plt.plot(xres, yres)

# Cette partie du code sert à afficher les points de contrôle
# "ro" signifie red circle
    #plt.plot(x_controle,y_controle, "ro")

# Cette petite partie de code sert simplement à numéroter les points de contrôle
    #for num in range(len(points_controle)):
     #   plt.text(points_controle[num][0], points_controle[num][1], num)
```

```
    #plt.show()
    distance = np.zeros([28,9])   #On utilise une grille 9*28 et on va calculer la
distance infinie par rapport à la spline

    for i in range(28):
        for j in range(9):
            test=[]
            for k in range(len(xres)):
                test.append(max(abs(xres[k] - i),abs(yres[k] - j)))
            distance[i,j] = min(test)
    new_dist = distance.round()

    xdist=[]
    ydist=[]
    pixels = np.zeros([28,9])
    for i in range(28):
        for j in range(9):
            if new_dist[i,j] == dis:
                xdist.append(i)   #abscisse du point de distance d de la spline
                ydist.append(j)
                pixels[i,j] = 255  # on colorie le pixel correspondant en noir
            if new_dist[i,j] == 0:
                pixels[i,j] = 255

    #plt.plot(xres, yres)
    #plt.scatter(xdist,ydist)
    #plt.figure()
    #plt.imshow(pixels.T, cmap='binary', extent=[0,28,0,9],origin='lower')
    return(pixels)


def Halo_Lineaire(point1,point2,dis):

    points_controle = np.array([ point1,point2 ])
    x_controle = np.array([p[0] for p in points_controle])
    y_controle = np.array([p[1] for p in points_controle])

    xres, yres = courbe_bezier(points_controle, disc=1000)
    #plt.plot(xres, yres)

# Cette partie du code sert à afficher les points de contrôle
# "ro" signifie red circle
    #plt.plot(x_controle,y_controle, "ro")

# Cette petite partie de code sert simplement à numéroter les points de contrôle
    #for num in range(len(points_controle)):
   #     plt.text(points_controle[num][0], points_controle[num][1], num)

   # plt.show()
```

```
    distance = np.zeros([28,9])   #On utilise une grille 9*28 et on va calculer la
distance infinie par rapport à la spline

    for i in range(28):
        for j in range(9):
            test=[]
            for k in range(len(xres)):
                test.append(max(abs(xres[k] - i),abs(yres[k] - j)))
            distance[i,j] = min(test)
    new_dist = distance.round()

    xdist=[]
    ydist=[]
    pixels = np.zeros([28,9])
    for i in range(28):
        for j in range(9):
            if new_dist[i,j] <= dis:
                xdist.append(i)   #abscisse du point de distance d de la spline
                ydist.append(j)
                pixels[i,j] = 255  # on colorie le pixel correspondant en noir
            if new_dist[i,j] == 0:
                pixels[i,j] = 255

    #plt.plot(xres, yres)
    #plt.scatter(xdist,ydist)
    #plt.figure()
    #plt.imshow(pixels.T, cmap='binary', extent=[0,28,0,9],origin='lower')
    return(pixels)

Halo_Lineaire([14,5],[14,0],1)

Halo_Quadratique([7,0],[14,5],[21,0],1)

Halo_Cubique([7,0],[14,5],[14,0],[21,5],1)

#Conversion des données au format CSV pour récuperer des chiffres MNIST

def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28*28):
            image.append(ord(f.read(1)))
        images.append(image)
```

```
    for image in images:
        o.write(",".join(str(pix) for pix in image)+"\n")
    f.close()
    o.close()
    l.close()

convert("train-images-idx3-ubyte", "train-labels-idx1-ubyte",
        "mnist_train.csv", 60000)
convert("t10k-images-idx3-ubyte", "t10k-labels-idx1-ubyte",
        "mnist_test.csv", 10000)

train_df = pd.read_csv("mnist_train.csv") #ensemble d'apprentissage
test_df = pd.read_csv("mnist_test.csv") #ensemble de test

X_train = train_df.iloc[:, 1:].values
y_train = train_df.iloc[:, 0].values
X_test = test_df.iloc[:, 1:].values
y_test = test_df.iloc[:,0].values

predictions = []
d = 1


for k in range(100):
    X = X_train[k].reshape(28,28)
    part1 =np.zeros([28,9])
    part2 =np.zeros([28,9])
    part3 =np.zeros([28,9])

    for i in range(28):
        for j in range(9):
            part3[i,j] = X[i,j]

    for i in range(28):
        for j in range(9):
            part2[i,j] = X[i,j+9]

    for i in range(28):
        for j in range(9):
            part1[i,j] = X[i,j+18]

    distance1 = np.zeros([28,28])
    res = []

    ############
    #Est-ce 0 ?
    t1 = Halo_Quadratique([9,0],[14,3],[19,0],d)


    t2 = Halo_Lineaire([8,9],[8,0],d)
```

```
    t21 = Halo_Lineaire([20,9],[20,0],d)


    t3 = Halo_Quadratique([8,9],[14,2],[20,9],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = abs(t1[i,j] - part1[i,j])
            distance1[i,j] = min(abs(t2[i,j] - part2[i,j]),abs(t21[i,j] - part2[i,j
]))
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)
    ############
    #Est-ce 1 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Lineaire([14,7],[14,0],d)
    t2 = Halo_Lineaire([14,9],[14,0],d)
    t3 = Halo_Lineaire([14,9],[14,9],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = abs(t1[i,j] - part1[i,j])
            distance1[i,j] = abs(t2[i,j] - part2[i,j])
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)

    ############
    #Est-ce 2 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Quadratique([7,0],[14,7],[21,0],d)
    t2 = Halo_Lineaire([21,9],[7,0],d)
    t3 = Halo_Quadratique([7,9],[14,6],[21,9],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = abs(t1[i,j] - part1[i,j])
            distance1[i,j] = abs(t2[i,j] - part2[i,j])
            distance1[i,j] = abs(t3[i,j] - part3[i,j])
```

```
    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)
    ############
    #Est-ce 3 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Quadratique([9,0],[14,4],[19,0],d)
    t2 = Halo_Cubique([19,9],[14,5],[14,5],[21,0],d)
    t3 = Halo_Quadratique([21,9],[20,6],[12,5],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = abs(t1[i,j] - part1[i,j])
            distance1[i,j] = abs(t2[i,j] - part2[i,j])
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)

    ############
    #Est-ce 4 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Lineaire([9,5],[9,0],d)
    t11 = Halo_Lineaire([18,5],[18,0],d)

    t2 = Halo_Lineaire([9,9],[9,3],d)
    t21 = Halo_Lineaire([9,3],[19,3],d)
    t22 = Halo_Lineaire([19,9],[19,0],d)

    t3 = Halo_Lineaire([19,9],[19,3],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = min(abs(t1[i,j] - part1[i,j]),abs(t11[i,j] - part1[i,j
]))
            distance1[i,j] = min(abs(t2[i,j] - part2[i,j]),abs(t21[i,j] - part2[i,j
]),abs(t22[i,j] - part2[i,j]))
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
```

```
    res.append(somme_ecarts_entre_pixels)

    ############
    #Est-ce 5 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Lineaire([9,0],[9,3],d)
    t11 = Halo_Lineaire([9,3],[18,3],d)

    t2 = Halo_Lineaire([9,9],[9,4],1)
    t21 = Halo_Quadratique([9,4],[14,5],[20,0],d)


    t3 = Halo_Quadratique([20,9],[16,4],[9,4],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = min(abs(t1[i,j] - part1[i,j]),abs(t11[i,j] - part1[i,j
]))
            distance1[i,j] = min(abs(t2[i,j] - part2[i,j]),abs(t21[i,j] - part2[i,j
]))
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)

    ############
    #Est-ce 6 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Lineaire([13,0],[21,5],d)

    t2 = Halo_Quadratique([13,9],[9,5],[9,0],d)
    t21 = Halo_Quadratique([10,5],[17,5],[20,0],d)


    t3 = Halo_Quadratique([9,9],[15,2],[21,9],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = abs(t1[i,j] - part1[i,j])
            distance1[i,j] = min(abs(t2[i,j] - part2[i,j]),abs(t21[i,j] - part2[i,j
]))
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
```

```
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)

    ############
    #Est-ce 7 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Lineaire([8,4],[17,4],d)
    t11 = Halo_Lineaire([17,4],[17,0],d)

    t2 = Halo_Lineaire([17,9],[17,0],d)
    t21 = Halo_Lineaire([12,3],[19,3],d)


    t3 = Halo_Quadratique([9,9],[15,2],[21,9],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = min(abs(t1[i,j] - part1[i,j]),abs(t11[i,j] - part1[i,j
]))
            distance1[i,j] = min(abs(t2[i,j] - part2[i,j]),abs(t21[i,j] - part2[i,j
]))
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)

    ############
    #Est-ce 8 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Quadratique([8,0],[12,4],[17,0],d)


    t2 = Halo_Lineaire([8,9],[17,0],d)
    t21 = Halo_Lineaire([17,9],[8,0],d)


    t3 = Halo_Quadratique([8,9],[14,2],[19,9],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = abs(t1[i,j] - part1[i,j])
            distance1[i,j] = min(abs(t2[i,j] - part2[i,j]),abs(t21[i,j] - part2[i,j
]))
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
```

```
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)

    ############
    #Est-ce 9 ?
    distance1 = np.zeros([28,28])
    t1 = Halo_Quadratique([9,0],[13,3],[17,0],d)

    t2 = Halo_Quadratique([9,9],[13,6],[17,9],d)
    t21 = Halo_Lineaire([17,9],[17,0],d)


    t3 = Halo_Lineaire([17,9],[17,3],d)
    t31 = Halo_Lineaire([8,3],[17,3],d)

    for i in range(28):
        for j in range(9):
            distance1[i,j] = abs(t1[i,j] - part1[i,j])
            distance1[i,j] = min(abs(t2[i,j] - part2[i,j]),abs(t21[i,j] - part2[i,j
]))
            distance1[i,j] = abs(t3[i,j] - part3[i,j])

    somme_ecarts_entre_pixels = 0
    for i in range(28):
        for j in range(28):
            if distance1[i,j] > 0:
                somme_ecarts_entre_pixels += distance1[i,j]
    res.append(somme_ecarts_entre_pixels)


    predictions.append(res.index(min(res)))

recherche_erreur = []
nombre_erreur = 0
for i in range(100):
    recherche_erreur.append(abs(predictions[i] - y_train[i]))
    if recherche_erreur[i] > 0:
        nombre_erreur += 1

taux_erreur = (nombre_erreur/100)*100
print(f"Le taux d'erreur est de: {taux_erreur}")
```

Code du chapitre 3

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#Librairies
import time
import torch
import torchvision

#Raccourcis utilisé
import torch.nn as nn
from torchvision.transforms.functional import crop

from PIL import Image
from IPython.display import display

import pandas as pd


# In[2]:


#Téléchargement de la base de donnees
mnist_dataset = torchvision.datasets.MNIST(".", train=True, download=True)


# In[3]:


display(mnist_dataset[128][0])


# In[4]:


#Labels pour chacune des images
labels = [x[1] for x in mnist_dataset]

for x in range(10):
  print("Number of", x, ":", labels.count(x))


# In[5]:


#test d'un découpage d'une image
X = mnist_dataset[2][0].resize((30,30))
#on redimensionne l'image en 30x30 pour ne pas perdre de données par rapport à un d
écoupage en 3x9x28(27x28)
zones = {}
```

```
zones[0] = X.crop((0,0,30,10))
zones[1] = X.crop((0,10,30,20))
zones[2] = X.crop((0,20,30,30))


# In[6]:


display(zones[0])
display(zones[1])
display(zones[2])


# In[7]:


class APP(nn.Module): # Classe APPrentissage

  def __init__(self,nb_splines=(8,10,8),threshold=0.1):
    super(APP, self).__init__()

    ## on remplit les images de zéros
    self.padding = nn.ZeroPad2d((0,0,1,1))

    ## Première convolution
    self.conv0 = nn.Sequential()
    self.conv1 = nn.Sequential()
    self.conv2 = nn.Sequential()
    self.conv = {0:self.conv0,1:self.conv1,2:self.conv2}
    self.fc0 = nn.Sequential()
    self.fc1 = nn.Sequential()
    self.fc2 = nn.Sequential()
    self.fc = {0:self.fc0,1:self.fc1,2:self.fc2}

    for i in range(3):
        self.conv[i].add_module("conv_1", nn.Conv2d(in_channels = 1, out_channels=
6 , kernel_size = 5))
        self.conv[i].add_module("relu_1", nn.ReLU())

        self.conv[i].add_module("conv_2", nn.Conv2d(in_channels = 6, out_channels=
12 , kernel_size = 5))
        self.conv[i].add_module("max_pool_2", nn.MaxPool2d(2))
        self.conv[i].add_module("relu_2", nn.ReLU())

        self.fc[i].add_module("fc_3", nn.Linear(in_features = 120, out_features = 6
4))
        self.fc[i].add_module("relu_3", nn.ReLU())
        self.fc[i].add_module("dropout_3", nn.Dropout())

        self.fc[i].add_module("fc_4", nn.Linear(in_features = 64, out_features = nb
_splines[i]))
        #self.fc[i].add_module("relu_4", nn.ReLU())
```

```
        self.fc[i].add_module("thresold_4", nn.Threshold(threshold=threshold, value
=0))



    self.output = nn.Linear(sum(nb_splines), 10)

    self.softmax = nn.LogSoftmax(dim=1)

  def forward(self, x):

    x = self.padding(x)

    zones = {}
    zones[0] = x[:,:,:10,:]
    zones[1] = x[:,:,10:20,:]
    zones[2] = x[:,:,20:30,:]


    for i in range(3):
        zones[i] = self.conv[i](zones[i])
        zones[i] = zones[i].view(-1, 12*1*10)
        zones[i] = self.fc[i](zones[i])

        '''
        #print(zones[i].shape)
        max_zones = torch.max(zones[i],dim=0)
        #print(max_zones)

        one = torch.ones(zones[i].shape)
        zero = torch.zeros(zones[i].shape)
        if zones[i].is_cuda:
            one, zero = one.cuda(), zero.cuda()
        #print(zones[i],zones[i].is_cuda,max_zones,one,zero)
        zones[i] = torch.where(zones[i]==max_zones.values,one,zero)
        '''


    x = torch.cat((zones[0],zones[1],zones[2]),1)

    x = self.output(x)
    y = self.softmax(x)

    return y, zones


# In[8]:


## Création d'une instance pour notre réseau
net = APP()
```

```
##  Convertion des images PIL au format tensor
convert = torchvision.transforms.ToTensor()

## En entrée on a alors une image tensor que le dimention en "unsqueeze"
## car PyTorch est habitué à travailler avec des batchs.
x = convert(mnist_dataset[0][0]).unsqueeze(0)

## Application du réseau sur les entrées
net(x)


# In[9]:


## Chargement du Dataset
mnist_dataset = torchvision.datasets.MNIST(".", train=True, transform=convert, down
load=True)

## Pourcentage de validation sur le dataset
validation_split = 0.1

N_val_samples = round(validation_split * len(mnist_dataset))

## Division en 2 sous-ensemble
train_set, val_set = torch.utils.data.random_split(mnist_dataset, [len(mnist_datase
t) - N_val_samples, N_val_samples])

# train et val sont des objets de ces sous-ensemble
print(train_set)
print(val_set)

# On vérifie leur taille
len(train_set) + len(val_set) == len(mnist_dataset)


# In[10]:


##Création un DataLoader à partir de notre ensemble de formation mnist
BATCH_SIZE = 64

mnist_train_dl = torch.utils.data.DataLoader(train_set, batch_size=BATCH_SIZE, shuf
fle=True, num_workers=2)
mnist_val_dl = torch.utils.data.DataLoader(val_set, batch_size=BATCH_SIZE, shuffle=
True, num_workers=2)


# In[11]:


batch = torch.Tensor(next(iter(mnist_train_dl))[0])
```

```
batch = batch.cuda()


# In[12]:


net = APP()

## Déplacement du réseau sur le GPU
net = net.cuda()
net(batch)


# In[13]:


net


# In[14]:


##RE-RUN du CODE pour obtenir un nouveau réseau

LEARNING_RATE = 0.003
MOMENTUM = 0.9

nb_splines = (8,10,8)

net = APP(nb_splines,threshold=0.1)

net = net.cuda()

criterion = nn.NLLLoss()

# Méthode stochastique de descente du gradient
optimizer = torch.optim.SGD(net.parameters(), lr=LEARNING_RATE, momentum=MOMENTUM)


# In[16]:


## Nombre d'époques d'apprentissage
N_EPOCHS = 20

epoch_loss, epoch_acc, epoch_val_loss, epoch_val_acc = [], [], [], []
start_time = time.time()

for e in range(N_EPOCHS):

    print("EPOCH:",e)
```

```
    ### boucle d'entrainement
    running_loss = 0
    running_accuracy = 0
    start_epoch_time=time.time()

    ## Le réseau est mis en mode "entrainement"
    net.train()

    for i, batch in enumerate(mnist_train_dl):

        # Obtenir batch du dataloader
        x = batch[0]
        labels = batch[1]

        # déplacer le batch sur le GPU
        x = x.cuda()
        labels = labels.cuda()

        # Calcul de l'output et les loss
        output = net(x)
        y = output[0]

        loss = criterion(y, labels)

        # Réinitialisation du gradient
        optimizer.zero_grad()

        # Calcul du gradient
        loss.backward()

        # Application d'une étape d'optimisation de l'algorithme de descente pour m
ettre à jour les poids
        optimizer.step()

        with torch.no_grad():
            running_loss += loss.item()
            running_accuracy += (y.max(1)[1] == labels).sum().item()

    print("Training accuracy:", running_accuracy/float(len(train_set)),
        "Training loss:", running_loss/float(len(train_set)))

    epoch_loss.append(running_loss/len(train_set))
    epoch_acc.append(running_accuracy/len(train_set))

    ### Boucle de validation
    ## Le réseau est mis en mode validation
    net.eval()

    running_val_loss = 0
    running_val_accuracy = 0
```

```
    for i, batch in enumerate(mnist_val_dl):

        with torch.no_grad():

            x = batch[0]
            labels = batch[1]


            x = x.cuda()
            labels = labels.cuda()


            output = net(x)
            y = output[0]


            loss = criterion(y, labels)

            running_val_loss += loss.item()
            running_val_accuracy += (y.max(1)[1] == labels).sum().item()

    print("Validation accuracy:", running_val_accuracy/float(len(val_set)),
        "Validation loss:", running_val_loss/float(len(val_set)))

    epoch_val_loss.append(running_val_loss/len(val_set))
    epoch_val_acc.append(running_val_accuracy/len(val_set))

    inter = time.time() - start_epoch_time
    print ('Temps de l Epoch ',e,' en secondes:', inter )


interval = time.time() - start_time
print ('Temps total en secondes:', interval )


# In[17]:


def Pattern (j) :

    ## Chargement du Dataset
    mnist_dataset = torchvision.datasets.MNIST(".", train=False, transform=convert,
 download=True)

    # for label 1
    idx = mnist_dataset.targets== j
    mnist_dataset.targets = mnist_dataset.targets[idx]
    mnist_dataset.data = mnist_dataset.data[idx]

    ##Création un DataLoader à partir de notre ensemble de formation mnist
    BATCH_SIZE = 64
```

```
    mnist_dl = torch.utils.data.DataLoader(mnist_dataset, batch_size=BATCH_SIZE, sh
uffle=False, num_workers=2)

    stats = {}

    for i in range(3):
        stats[i] = pd.DataFrame(columns = range(nb_splines[i]))

    running_loss = 0
    running_accuracy = 0

    for i, batch in enumerate(mnist_dl):

        with torch.no_grad():

            x = batch[0]
            labels = batch[1]


            x = x.cuda()
            labels = labels.cuda()


            output = net(x)
            y = output[0]

            for i in range(3):
                stats[i] = stats[i].append(pd.DataFrame(output[1][i].cpu().detach()
.numpy()))

            loss = criterion(y, labels)

            running_loss += loss.item()
            running_accuracy += (y.max(1)[1] == labels).sum().item()

    print("Pondération des patterns sur le chiffre",j)
    print ("pondération des patterns sur la partie du haut")
    print (stats[0].mean(0))
    print (stats[0].var(0))
    print ("pondération des patterns sur la partie du milieu")
    print (stats[1].mean(0))
    print (stats[1].var(0))
    print ("pondération des patterns sur la partie du bas")
    print (stats[2].mean(0))
    print (stats[2].var(0))


# In[18]:


Pattern(0)
```

```
# In[19]:


Pattern(8)


# In[20]:


Pattern(9)
```