



ISUP 3 FILIÈRE ISDS

Master 2 Ingénierie Mathématique Parcours ISDS

Année 2024-2025

Projet Réseaux Neuronaux

À Paris

Étudiante : KHELID Lilya-Nada

Professeure : VALIBOUZE Annick

Table des matières

I. Présentation succincte des réseaux de neurones.....	2
1.1 Origine et définition des réseaux de neurones.....	2
1.2 Fonctionnement général	2
1.3 Apprentissage supervisé avec le perceptron multicouche (PMC)	3
1.4 Applications typique	3
II. Utilisation de R pour les PMC	4
2.1 Introduction	4
2.2 Méthodologie	4
2.3 Résultats	4
III. Démonstration avec TensorFlow et CIFAR-10	5
3.1 Introduction	5
3.2 Méthodologie	5
3.3 Résultats	6
IV. Utilisation d'une carte de Kohonen avec données Spotify.....	6
4.1 Introduction	6
4.2 Méthodologie	7
4.3 Résultas.....	8
V. Explications des GANs (Generative Adversarial Networks)	9
5.1 Introduction	9
5.2 Exemple pratique : GAN pour MNIST	10
5.3 Exemple pratique : Résultats	12
VI. Explicabilité.....	13
6.1 Le modèle	13
6.2 Xplique	14
6.3 Application	15
6.4 Résultas.....	16
VII. Éthique et réseaux de neurones.....	17
7.1 Les enjeux éthiques	17
7.2 Cadre réglementaire	17
7.3 Vers une IA éthique	17
VIII. Conclusion	18
IX. Bibliographie.....	19

I. Présentation succincte des réseaux de neurones

1.1. Origine et définition des réseaux de neurones

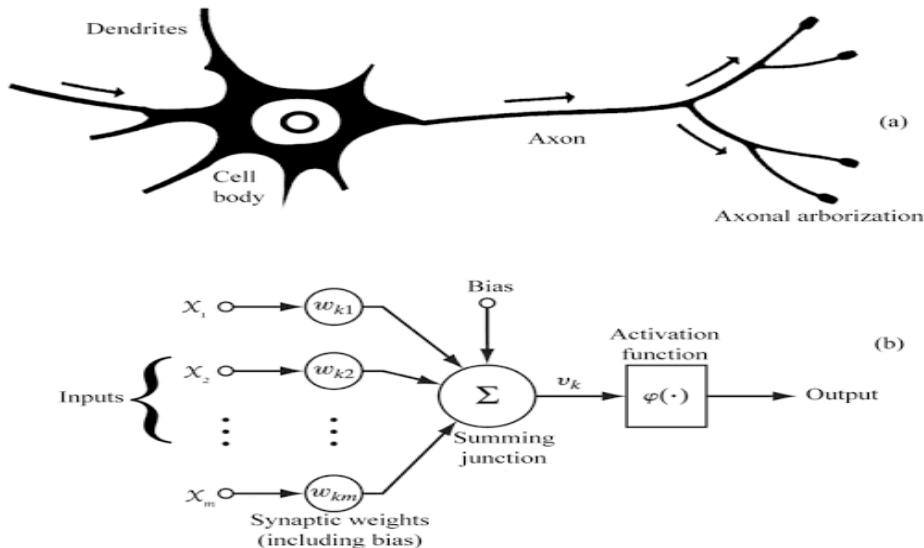


Figure 1 – Mathematical model of a biological neuron

Les réseaux de neurones artificiels (RNA) sont inspirés des réseaux neuronaux biologiques et ont été initialement formalisés par [McCulloch](#) et [Pitts](#) en 1943. Un neurone artificiel reçoit des signaux sous forme de vecteurs d'entrée $x = (x_1, x_2, \dots, x_n)$, de pondérations $w = (w_1, w_2, \dots, w_n)$ à ces entrées, calcule une somme pondérée, puis applique une fonction d'activation pour une sortie :

$$y = f(\sum w_i x_i + b)$$

Où b est le biais, et f la fonction d'activation (par exemple : *sigmoïde, ReLU, ou softmax*).

1.2. Fonctionnement général

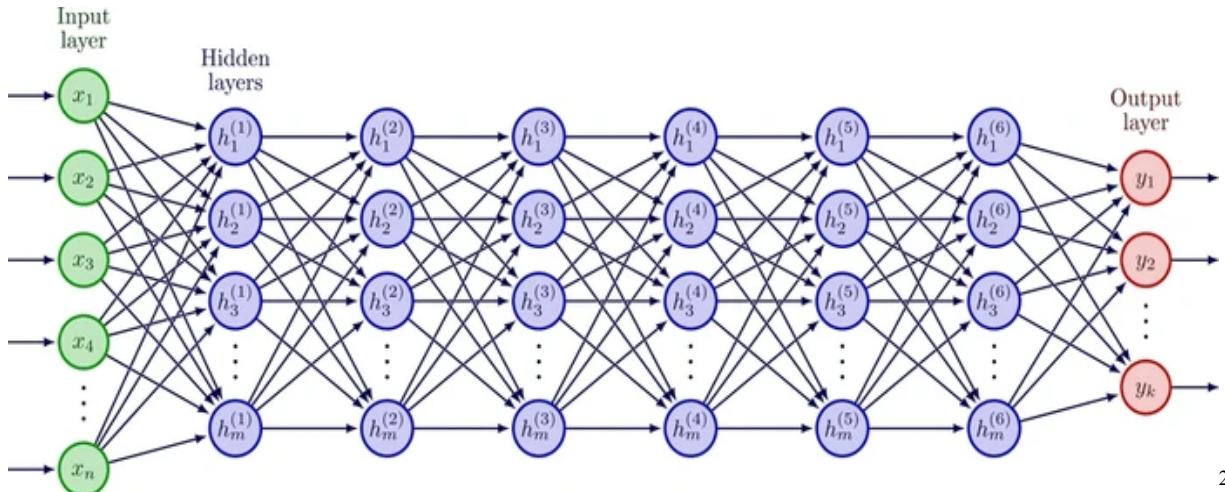
Un réseau de neurones est composé de plusieurs couches de neurones connectés. Ces couches fonctionnent en collaboration pour extraire des informations pertinentes et produire une prédiction. L'architecture classique inclut :

1. Une couche d'entrée : Elle correspond aux données brutes ou aux caractéristiques (features) du problème. Chaque neurone dans cette couche représente une dimension des données d'entrée.
2. Une ou plusieurs couches cachées : Chaque couche transforme les données d'entrée en utilisant une combinaison linéaire des poids et une activation non linéaire. Cette transformation permet au réseau d'apprendre des représentations complexes.
3. Une couche de sortie : Elle produit la décision finale ou la prédiction. Pour un problème de classification avec C classes, la couche de sortie applique souvent une fonction softmax :
$$\hat{y}_i = \exp(z_i) / \sum_j \exp(z_j)$$
, où z_i est l'entrée pondérée du neurone i .

¹ <https://moncoachdata.com/blog/comprendre-les-reseaux-de-neurones/>

1.3. Apprentissage supervisé avec le perceptron multicouche (PMC)

Le perceptron multicouche (PMC) est un réseau neuronal supervisé, utilisé pour résoudre des problèmes de classification ou de régression. Voici les étapes principales de son fonctionnement :



2

Figure 2- Perceptron multicouches

1. Propagation avant :

Chaque couche du réseau de neurones calcule ses activations $h^{\{l\}}$ en utilisant les sorties de la couche précédente $h^{\{l-1\}}$ selon la formule suivante :

$$z_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} W_{ji}^{(l)} h_i^{(l-1)} + b_j^{(l)}, \quad \forall j \in \{1, \dots, n^{(l)}\}$$

$$h_j^{(l)} = f(z_j^{(l)}), \quad \forall j \in \{1, \dots, n^{(l)}\}$$

Où W est la matrice des poids, b est le biais, et f est une fonction d'activation.

2. Calcul de l'erreur :

La fonction de coût mesure l'écart entre la prédiction \hat{y} et la vérité y . Pour un problème de classification, on utilise souvent l'entropie croisée :

Où m est le nombre d'exemples, k le nombre de classes, et $J = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k y_{ij} \log(\hat{y}_{ij})$, où y_{ij} est la vérité pour l'exemple i .

3. Rétropropagation de l'erreur :

On calcule le gradient de la fonction de coût J par rapport aux paramètres W et b , en appliquant la règle de la chaîne. L'objectif est de minimiser J par les poids : $W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta \frac{\partial J}{\partial W_{ij}^{(l)}}$ où η est le taux d'apprentissage.

4. Descente de gradient :

L'optimisation des paramètres se fait de manière itérative jusqu'à ce que la fonction de coût converge vers un minimum. Les poids sont mis à jour de manière itérative pour réduire l'erreur globale.

2 <https://www.picsellia.fr/post/fonctions-dactivation-reseaux-neurones>

1.4. Application typique

Un exemple classique d'utilisation du PMC est la classification d'images. Chaque pixel de l'image est une entrée, et le PMC apprend à associer chaque image à une classe prédéfinie. Une architecture avec plusieurs couches cachées permet au réseau d'extraire des caractéristiques complexes, améliorant ainsi ses performances sur des tâches non linéaires.

II. Utilisation de R pour les PMC

2.1. Introduction

Les perceptrons multicouches (PMC) sont des modèles de réseaux de neurones supervisés utilisés pour des tâches variées telles que la régression et la classification. Cette section illustre deux exemples pratiques : l'approximation d'une fonction continue et la classification des passagers du Titanic. Elle inclut également une optimisation des hyperparamètres pour améliorer les performances du modèle.

2.2. Méthodologie

Deux exemples principaux ont été explorés :

1. Régression avec $\sin(x)$:

- Un PMC a été entraîné pour approximer une fonction bruitée ($y = \sin(x)$).
- Les résultats ont été comparés avec un modèle polynomial pour évaluer les performances.
- Les graphiques illustrent les prédictions du PMC et celles de la fonction réelle.

```
library(nnet)
# --sin--
x <- sort(10 * runif(50))
y <- sin(x) + 0.2 * rnorm(length(x))
data <- data.frame(x, y)

#--PMC--
nn <- nnet(x, y, size = 4, maxit = 100, linout = TRUE)

plot(x, y, col = "blue", pch = 16)
x1 <- seq(0, 10, by = 0.01)
lines(x1, predict(nn, data.frame(x = x1)), col = "red")
lines(x1, sin(x1), col = "green")

#--Comparaison avec un modèle polynomial--
modelPoly <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))
lines(x, predict(modelPoly), col = "orange")
legend(3, 1.3, c("Données bruitées", "sin(x)", "PMC", "Modèle polynomial"),
      lty = c(0, 1, 1, 1), pch = c(16, -1, -1, -1), col = c("blue", "green", "red", "orange"))
```

2. Classification avec le dataset Titanic :

- Les données ont été prétraitées, incluant la gestion des valeurs manquantes et la transformation des variables catégoriques.
- Un PMC a été entraîné pour prédire la survie des passagers.
- Les performances ont été évaluées à l'aide de matrices de confusion (numérique et graphique).

```
library(titanic)
data("titanic_train")

titanic <- titanic_train[, c("Survived", "Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked")]

#--valeurs manquantes--
titanic$Age[is.na(titanic$Age)] <- mean(titanic$Age, na.rm = TRUE)
titanic$Embarked[is.na(titanic$Embarked)] <- "S"

#--mettre des niveaux--
titanic$Survived <- as.factor(titanic$Survived)
titanic$Pclass <- as.factor(titanic$Pclass)
titanic$Sex <- as.factor(titanic$Sex)
titanic$Embarked <- as.factor(titanic$Embarked)

#--split en test et train--
set.seed(123)
train_index <- sample(1:nrow(titanic), size = 0.7 * nrow(titanic))
train_data <- titanic[train_index, ]
test_data <- titanic[-train_index, ]

#--PMC--
model <- nnet(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
              data = train_data, size = 5, decay = 0.01, maxit = 200)
```

3. Optimisation des hyperparamètres :

- La validation croisée a été utilisée pour ajuster le nombre de neurones dans la couche cachée et le paramètre de régularisation.

```
## --tuning--  
tune.model <- tune.nnet(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,  
data = train_data, size = c(3, 5, 7), decay = c(0.001, 0.01, 0.1))
```

- Une heatmap a été générée pour visualiser les performances en fonction des combinaisons d'hyperparamètres.

2.3. Résultats

Les principaux résultats obtenus sont les suivants :

- La précision finale du modèle Titanic après optimisation des hyperparamètres est calculée.

```
## [1] "Précision optimisée : 79.1 %"
```

Figure 3- Précision optimisée

- Une matrice de confusion graphique a été générée pour interpréter les performances du modèle optimisé.

- La heatmap des performances montre les meilleures combinaisons d'hyperparamètres.

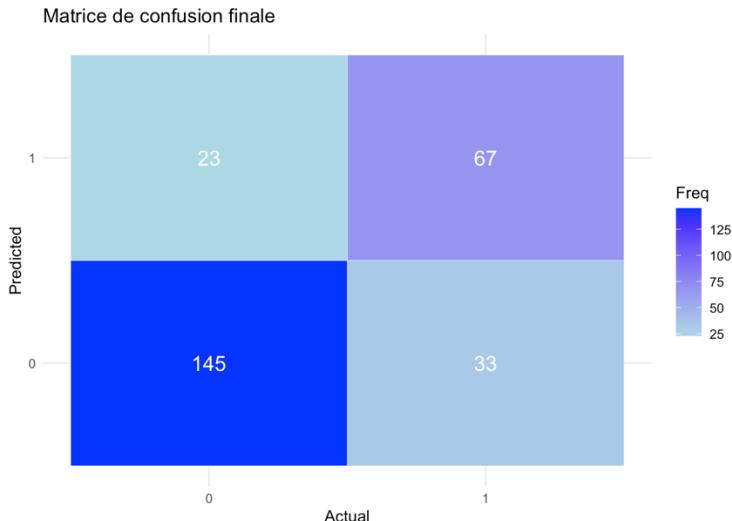


Figure 4- Matrice de confusion

III. Démonstration avec TensorFlow et CIFAR-10

3.1. Introduction

Le dataset CIFAR-10 est composé de 60,000 images couleur de dimensions 32x32 pixels réparties en 10 classes, telles que avion, voiture, chat, chien, etc. Ce dataset est particulièrement adapté pour entraîner et évaluer des réseaux convolutifs (CNN). Dans cette démonstration, nous utilisons TensorFlow et Keras pour construire un CNN capable de classifier ces images.

3.2. Méthodologie

La méthodologie adoptée se divise en plusieurs étapes :

1. Chargement et prétraitement des données :

Les données CIFAR-10 ont été chargées via TensorFlow, et les pixels des images ont été

normalisés dans l'intervalle [0, 1] pour uniformiser les valeurs. Les étiquettes associées ont été converties en encodage one-hot afin de faciliter leur utilisation dans le modèle de classification.

2. Création du modèle CNN :

L'architecture du modèle comprend des couches convolutives suivies de couches de pooling, ainsi que des couches entièrement connectées avec une activation softmax pour produire les sorties. Le modèle a été compilé en utilisant l'optimiseur Adam et la fonction de perte catégorielle croisée pour un apprentissage efficace.

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
```

3. Entraînement et évaluation :

Le modèle est entraîné sur les données d'entraînement et évalué sur un ensemble de test. Les métriques de performance incluent la précision et la perte.

3.3. Résultats

Précision sur le test : 0.70

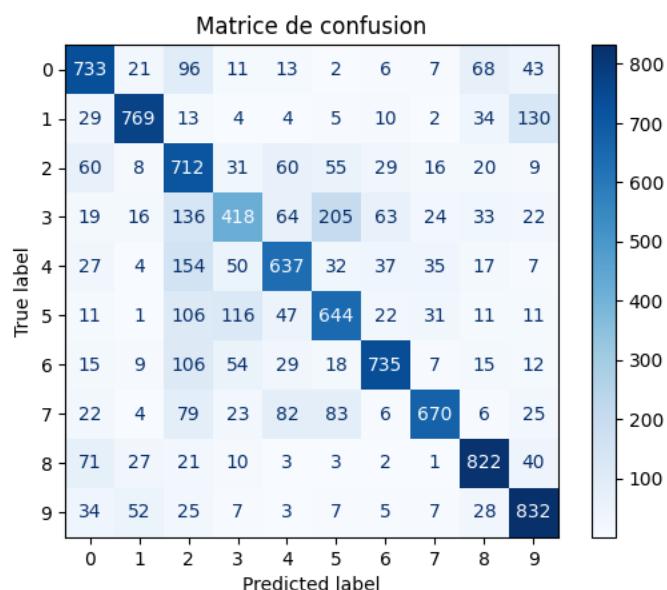


Figure 5- Matrice de confusion

Les réseaux convolutifs sont particulièrement efficaces pour les tâches de classification d'images, comme l'a démontré cette expérimentation avec CIFAR-10. Cependant, les performances du modèle pourraient être améliorées par :

- L'augmentation des données pour générer des variations supplémentaires d'images.
- L'utilisation d'architectures CNN plus complexes, telles que ResNet ou VGG.
- L'optimisation des hyperparamètres comme le taux d'apprentissage et la taille des couches.

Dans un projet de détection de tumeurs sur IRM cérébrales, j'ai testé plusieurs architectures de CNN.
[Lien du GitHub à consulter.](#)

IV. Utilisation d'une carte de Kohonen avec données Spotify

4.1. Introduction

Les cartes de Kohonen, ou cartes auto-organisatrices (*SOM - Self-Organizing Maps*), sont des réseaux de neurones **non supervisés** utilisés pour réduire la dimensionnalité et regrouper des observations similaires en clusters. Pour une démonstration, nous utilisons des caractéristiques audios de chansons issues d'un [dataset Spotify](#) pour analyser et regrouper les morceaux en fonction de leur similarité.

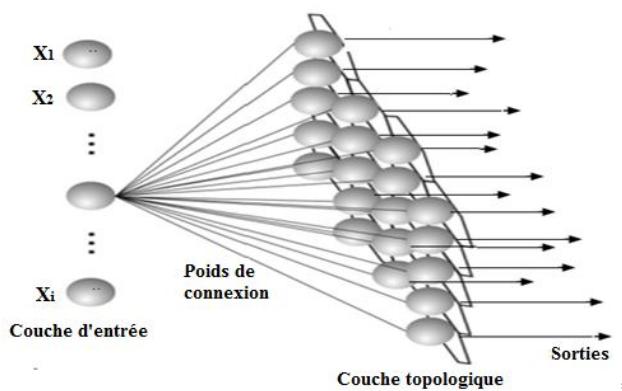


Figure 6 – Carte auto-organisatrice de Kohonen

4.2. Méthodologie

La méthodologie adoptée se divise en plusieurs étapes :

1. Chargement et prétraitement des données :

Les données Spotify, téléchargées depuis un dépôt public, ont été prétraitées en sélectionnant des caractéristiques pertinentes telles que **danceability**, **energy**, **tempo** et **valence**, qui ont ensuite été normalisées dans l'intervalle [0, 1].

2. Création et entraînement de la carte de Kohonen :

Une carte 2D de dimensions 10x10 a été initialisée pour organiser les données dans un espace simplifié. Les poids des neurones ont été ajustés grâce à un processus d'apprentissage non supervisé réalisé sur 100 itérations. À chaque étape, une donnée est sélectionnée, et le neurone dont les poids sont les plus proches de cette donnée (le neurone gagnant) est identifié. Les poids du neurone gagnant et de

```
som = MiniSom(x=10, y=10, input_len=X.shape[1], sigma=1.0, learning_rate=0.1, random_seed=42)
som.random_weights_init(X)
som.train_random(X, num_iteration=100)
```

³ https://www.researchgate.net/figure/Carte-auto-organisatrice-de-Kohonen-Chaque-sphère-symbolise-un-neurone-de-la-couche_fig2_304559519

ses voisins sont ensuite modifiés pour se rapprocher davantage des valeurs de la donnée d'entrée. Ce processus est répété pour toutes les données, permettant de regrouper les éléments similaires et de structurer les informations de manière cohérente sur la carte.

3. Clustering et analyse :

Les chansons ont été regroupées en clusters en fonction des neurones gagnants (winners) identifiés pour chaque donnée. Une carte des distances (U-Matrix) a ensuite été générée pour visualiser les similarités entre les clusters, en mettant en évidence les zones de forte proximité ou de dissimilarité dans les données.

```
clusters = defaultdict(list)
for i, x in enumerate(X):
    winner = som.winner(x)
    clusters[winner].append(titles.iloc[i])
```

4. 3. Résultats

1. Carte des distances (*U-matrix*) :

La carte montre les distances entre les neurones de la SOM. Les zones claires indiquent des distances faibles (similitudes élevées), tandis que les zones sombres représentent des clusters éloignés.

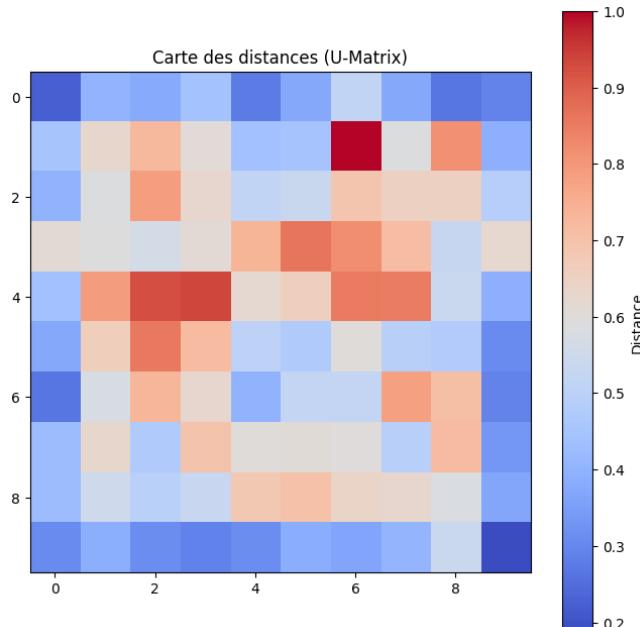


Figure 7 – Carte des distances

2. Clustering de chansons :

Les chansons ont été assignées à différents clusters en fonction de leurs caractéristiques audio telles que **danceability**, **energy**, **tempo** et **valence**. Ensuite les titres des chansons sont regroupées dans chaque cluster et ont été [affichés pour analyse](#).

Les cartes de Kohonen se sont révélées efficaces pour **regrouper les chansons similaires en clusters**. Les clusters obtenus reflètent des similarités audios entre les morceaux, offrant une base pour des applications comme la *recommandation musicale* ou la *catégorisation automatique des chansons*.

Cependant, certaines limites doivent être notées :

- Les résultats dépendent fortement des paramètres de la SOM (taille de la carte, nombre d'itérations, etc.).
- L'absence d'une variable cible (comme un genre musical) rend difficile une validation directe des clusters.

Pour améliorer cette approche, il serait pertinent d'explorer des techniques supervisées ou d'intégrer davantage de caractéristiques pour enrichir la représentation des chansons.

V. Explications des GANs (Generative Adversarial Networks)

5.1. Introduction

Les réseaux génératifs adversariaux (GAN - Generative Adversarial Networks) représentent une avancée majeure dans le domaine de l'intelligence artificielle, introduite par [Ian Goodfellow](#) en 2014. Ces modèles génératifs fonctionnent grâce à deux réseaux neuronaux qui s'entraînent en opposition :

- **Le générateur (Generator)** : il crée des données synthétiques à partir d'un bruit aléatoire, telles que des images ou des textes.
- **Le discriminateur (Discriminator)** : il tente de distinguer les données réelles des données générées par le générateur.

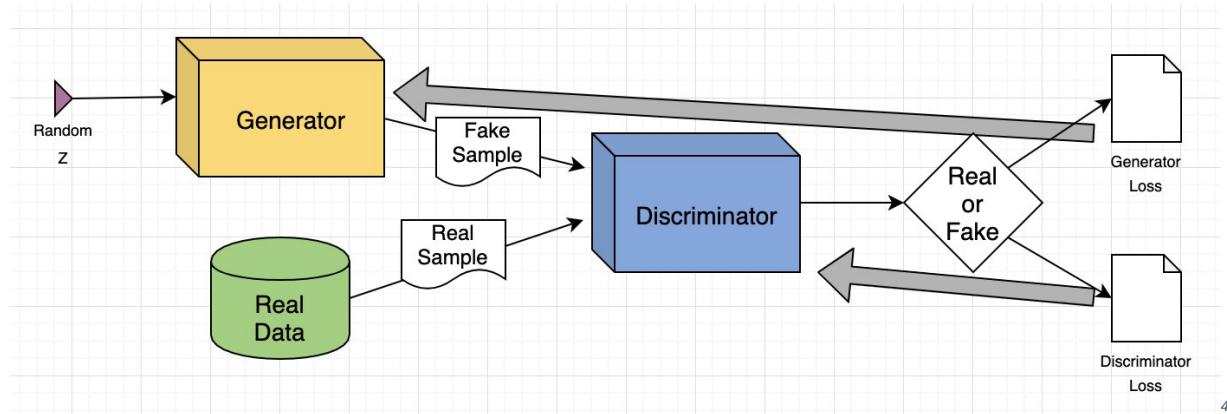


Figure 8 – Architecture GAN

Le générateur et le discriminateur sont entraînés en opposition, avec une fonction de coût basée sur un jeu à somme nulle :

- Le générateur s'améliore pour produire des données qui trompent le discriminateur.
- Le discriminateur apprend à mieux détecter les données synthétiques.

⁴ https://developers.google.com/machine-learning/gan/gan_structure?hl=fr

Applications des GANs :

1. **Génération d'images réalistes** (par exemple, création de visages humains avec StyleGAN⁵).

Une image générée par un StyleGAN qui ressemble de façon trompeuse au portrait d'une jeune femme. Cette image a été générée par une intelligence artificielle basée sur une analyse des portraits.

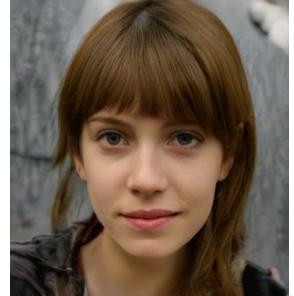


Figure 9 – Exemple StyleGAN⁷

2. **Amélioration de la qualité d'images** comme la *super-résolution*⁶.

3. **Augmentation de données** pour l'entraînement de modèles supervisés et une meilleure robustesse.

4. **Art génératif** : création d'images ou de musiques à partir de styles prédéfinis.

5.2. Exemple pratique : GAN pour MNIST

Dans le cadre d'une prise en main d'un GAN, j'ai commencé par implémenter un GAN simple pour me familiariser avec son fonctionnement, avant de passer à un modèle plus complexe afin d'explorer des architectures avancées sur les [données MNIST](#).

GAN standard :

1. **Générateur** : Transforme un bruit aléatoire en une image de 28x28 pixels.

```
model = tf.keras.Sequential([
    layers.Dense(256, activation='relu', input_dim=100),
    layers.BatchNormalization(),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(28 * 28 * 1, activation='tanh'),
    layers.Reshape((28, 28, 1))
])
```

2. **Discriminateur** : Classifie les images comme réelles ou synthétiques.

```
model = tf.keras.Sequential([
    layers.Flatten(input_shape=(28, 28, 1)),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

3. **Entraînement** : Alternance entre l'amélioration du discriminateur et du générateur.

⁵ StyleGAN est un [réseau antagoniste génératif](#) (GAN) introduit par les chercheurs de [Nvidia](#) en décembre 2018 et rendu disponible en février 2019.

⁶ La [super-résolution](#) désigne le processus qui consiste à améliorer la résolution spatiale, c'est-à-dire le niveau de détail, d'une image ou d'un système d'acquisition.

⁷ https://fr.wikipedia.org/wiki/R%C3%A9seaux_antenagonistes_g%C3%A9n%C3%A9ratifs

Avec :

`activation='relu'` : La fonction ReLU (Rectified Linear Unit) est une fonction d'activation couramment utilisée dans les réseaux de neurones. La formule : $f(x) = \max(0, x)$.

Cette fonction renvoie 0 pour les valeurs négatives et x pour les valeurs positives. Elle est appréciée pour sa simplicité et son efficacité, car elle introduit de la non-linéarité tout en évitant le problème du **vanishing gradient**⁸, souvent rencontré avec des fonctions comme la *sigmoid* ou la '`tanh`'.

`BatchNormalization()` : C'est une technique qui normalise les activations d'un réseau de neurones pour chaque mini-lot (batch) durant l'entraînement. Cette méthode accélère la convergence et améliore la stabilité de l'entraînement.

`layers.Dense()` : Une couche Dense (ou fully connected) est une couche où chaque neurone est connecté à tous les neurones de la couche précédente.

`layers.Flatten()` : La couche Flatten est utilisée pour transformer des données multidimensionnelles en un vecteur unidimensionnel.

GAN Optimisé :

Ces améliorations sont :

- Utilisation de **couches convolutives** dans le **générateur et le discriminateur** pour capturer des motifs complexes.

Générateur :

```
model = tf.keras.Sequential([
    layers.Dense(7 * 7 * 256, use_bias=False, input_dim=100),
    layers.BatchNormalization(),
    layers.LeakyReLU(),
    layers.Reshape((7, 7, 256)),
    layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
use_bias=False),
    layers.BatchNormalization(),
    layers.LeakyReLU(),
    layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False),
    layers.BatchNormalization(),
    layers.LeakyReLU(),
    layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',
use_bias=False, activation='tanh')
])
```

Discriminateur :

```
model = tf.keras.Sequential([
    layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=(28,
28, 1)),
    layers.LeakyReLU(),
```

⁸ Quand les gradients deviennent très petits au cours de la rétropropagation, les poids des couches précédentes sont mis à jour de manière minime, voire pas du tout. Cela ralentit l'apprentissage ou l'empêche complètement

```

        layers.Dropout(0.3),
        layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1)
    ])

```

- Adoption d'une fonction de perte avancée (**BinaryCrossentropy** avec **logits**⁹).
- Ajustement des hyperparamètres (*Learning rate*, initialisation des poids).

Avec :

`layers.LeakyReLU()` : *LeakyReLU* est une fonction d'activation qui permet un faible gradient pour les entrées négatives, évitant les neurones "morts" de ReLU.

5.3. Exemple pratique : Résultats

GAN standard



Amélioration

GAN optimisé

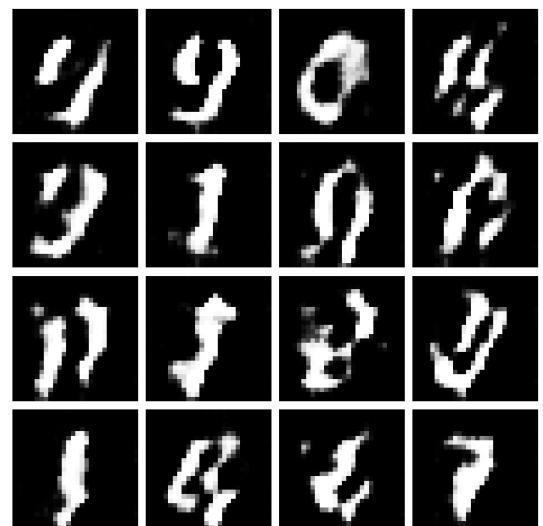


Figure 10 – temps d'exécution 4min

Figure 11 – temps d'exécution 68min

Conclusion :

Les GAN sont une avancée fascinante dans le domaine des réseaux neuronaux, offrant des possibilités incroyables comme la création d'images réaliste ou l'amélioration des données d'entraînement. Bien qu'ils soient puissants, leur mise en œuvre reste complexe et demande une bonne maîtrise des concepts, notamment pour stabiliser l'entraînement. Des modèles comme *StyleGAN* montrent à quel point ces réseaux peuvent être performants, mais ils nous rappellent aussi qu'il reste beaucoup à explorer pour mieux comprendre et exploiter leur potentiel.

⁹ La couche **logits** désigne la couche qui alimente la fonction softmax (ou une autre normalisation similaire). La sortie du softmax correspond aux probabilités pour la tâche de classification, tandis que son entrée provient de la couche logits.

VI. Explicabilité

Dans le cadre d'un [projet de détection de tumeurs cérébrales](#), le package [Xplique](#) à servit a mieux comprendre les décision d'un model CNN sur la classification de tumeurs cérébrales.

6.1. Le modèle

Le modèle CNN est conçu pour classifier des images en quatre catégories : **No tumor**, **Glioma tumor**, **Meningioma tumor**, et **Pituitary tumor**. Les images utilisées sont en niveaux de gris et redimensionnées à (256, 256, 1).

¹⁰

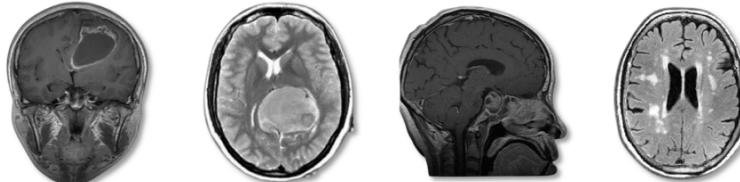


Figure 12 – glioma, meningioma, pituitary, no tumor

Architecture :

```
[  
    layers.Input(shape=(256, 256, 1)), # (256, 256, 3) si on veut RGB  
    # Bloc 1  
    layers.Conv2D(32, (3, 3), activation="relu"),  
    # Bloc 2  
    layers.Conv2D(64, (3, 3), activation="relu"),  
    layers.MaxPooling2D((2, 2)),  
    # Bloc 3  
    layers.Conv2D(128, (3, 3), activation="relu"),  
    # Bloc 4  
    layers.Conv2D(128, (3, 3), activation="relu"),  
    layers.MaxPooling2D((2, 2)),  
    # Couche Fully Connected  
    layers.Flatten(),  
    layers.Dense(64, activation="relu"),  
    layers.Dropout(0.2), # Pour éviter le surapprentissage  
    layers.Dense(4, activation="softmax"), # Multiclass classification  
]  
  
optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]
```

Les images passent par 4 blocs de convolutions avec des filtres (3 * 3) et des activations ReLU, suivis de couches fully connected (Flatten, Dense, Dropout) pour la classification. Le modèle est optimisé avec Adam, utilisant la fonction de perte **sparse categorical crossentropy** et mesurant la précision (**accuracy**).

Le model est particulièrement bon sur les classes no tumor et meningioma, moins pour pituitary et beaucoup moins pour glioma.

¹⁰ <https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset>

6.2. Xplique

Xplique repose sur une architecture modulaire qui s'intègre facilement aux frameworks de deep learning comme TensorFlow et Keras. Son fonctionnement suit une approche en plusieurs étapes, depuis la préparation des données jusqu'à la génération de visualisations explicatives.

Tout d'abord, l'utilisateur fournit une image ou un ensemble d'images, ainsi qu'un modèle d'apprentissage profond pré-entraîné (par exemple, un réseau convolutionnel). Les données sont prétraitées si nécessaire, pour s'assurer qu'elles sont compatibles avec le modèle (redimensionnement, normalisation, etc.).

La deuxième étape consiste à choisir une méthode d'explication parmi celles proposées par Xplique. Ces méthodes se divisent en deux grandes catégories : **les techniques basées sur les gradients** et **les approches par perturbation**. Les techniques basées sur les gradients, comme **Grad-CAM** ou **Integrated Gradients**, exploitent les dérivées des prédictions par rapport aux entrées pour identifier les pixels les plus influents. Les approches par perturbation, comme **l'occlusion ou RISE**, modifient ou suppriment des pixels ou des régions entières pour observer leur impact sur la prédiction du modèle. Ce choix de méthode dépend des besoins de l'utilisateur et de la nature du problème.

Enfin, Xplique génère des visualisations qui permettent d'interpréter les résultats. Par exemple, une carte de chaleur (heatmap) superposée à l'image d'origine indique les zones qui ont le plus influencé la décision du modèle. Ces visualisations peuvent être analysées pour identifier des biais, valider les performances ou explorer comment un modèle comprend ses entrées. En combinant flexibilité et puissance, Xplique facilite l'analyse détaillée des modèles complexes, tout en rendant leurs prédictions plus compréhensibles pour les humains.

11

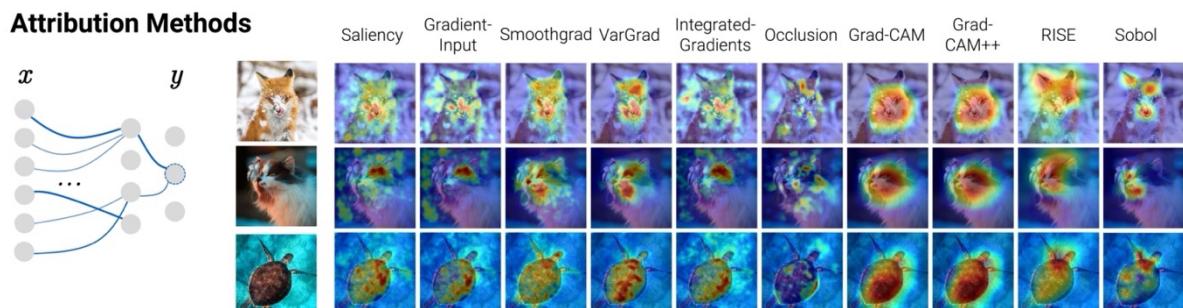


Figure 13 – Méthode d'attributions des 'explainers'

11 <https://github.com/deel-ai/xplique>

6.3. Application

Après avoir chargé le modèle et installé la librairie Xplique, on l'applique sur nos images.

```
model.layers[-1].activation = tf.keras.activations.linear
explainers = [
    GuidedBackprop(model),
    IntegratedGradients(model, steps=80, batch_size=batch_size),
    SmoothGrad(model, nb_samples=80, batch_size=batch_size),
    GradCAM(model),
]
for explainer in explainers:
    print(f"📊 Méthode : {explainer.__class__.__name__}")
    explanations = explainer(X, Y)
    plot_attributions(
        explanations,
        X,
        img_size=2.0,
        cmap='jet',
        alpha=0.4,
        cols=len(X),
        absolute_value=True,
        clip_percentile=0.5
    )
plt.show()
```

Avec :

GuidedBackprop : Permet d'obtenir des visualisations en rétro-propageant uniquement les gradients positifs jusqu'à l'entrée.

IntegratedGradients : Attribue un score d'importance à chaque pixel en intégrant les gradients sur un chemin entre une référence neutre et l'image d'entrée.

SmoothGrad : Réduit le bruit des explications basées sur les gradients en moyennant plusieurs versions perturbées de l'image.

GradCAM : Génère des cartes de chaleur en utilisant les gradients des couches finales pour identifier les zones importantes.

Contraintes :

Les temps d'exécution de chacun de ces explainers sont assez longs, et certains explainers très intéressants, comme Occlusion ou RISE, nécessitent une puissance computationnelle considérable, ce qui nous a contraints à nous en passer. Néanmoins, il est possible d'en voir l'application sur des [notebooks Colab](#), où les codes peuvent être exécutés sans problème.

6.4. Résultats

Images mal prédites

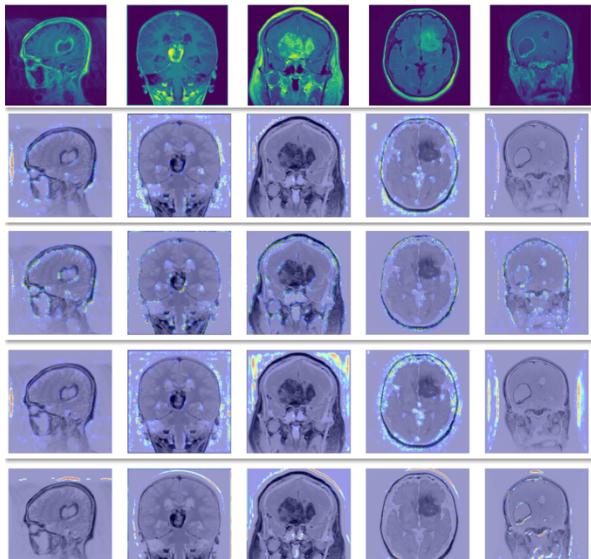


Figure 14 – Gliomes

Images bien prédites

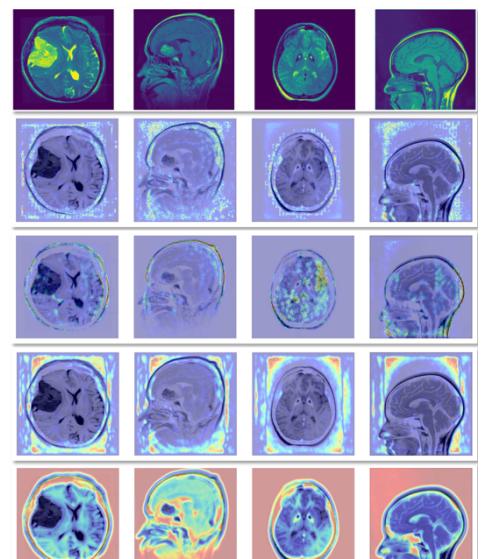


Figure 15 – Méningiomes et sains

On constate que les attributions ne fonctionnent pas correctement sur les images mal prédites. En effet, demander pourquoi le modèle a attribué un label incorrect à une image alors qu'il n'a pas réussi à la classer correctement n'est pas pertinent. Étudions donc les résultats sur les images bien prédites :

- Integrated Gradients : Les cartes produites par Integrated Gradients montrent un détourage précis du cerveau sans apparition de rectangle parasite. La méthode identifie clairement les contours externes du cerveau, mais elle attribue peu d'importance aux structures internes, comme les tumeurs. Cependant, lorsque les pixels internes sont supprimés, les performances du modèle diminuent, ce qui indique qu'ils ont tout de même une certaine influence. Integrated Gradients semble ainsi bien capturer les contours du cerveau, mais reflète potentiellement un biais où les caractéristiques internes sont sous-évaluées.
- SmoothGrad et Guided Backpropagation : Ils offrent des perspectives similaires sur les activations du modèle. Les deux méthodes mettent en évidence un détourage précis du cerveau, souvent accompagné d'une forme rectangulaire récurrente. Cette forme semble jouer un rôle important dans les prédictions du modèle, ce qui suggère un possible biais systématique lié à l'acquisition des IRM ou à des caractéristiques invisibles à l'oeil nu. SmoothGrad, en produisant des cartes d'activations lisses et raffinées, met en avant les contours périphériques, tandis que Guided Backpropagation, avec ses détails texturaux plus fins, montre que ces bords dominent également les activations.
- Grad-CAM : Il met en évidence une forte activation des pixels entourant le cerveau, suggérant que les contours périphériques jouent un rôle clé dans les décisions du modèle. Cependant, les structures internes du cerveau sont également bien colorées, ce qui montre qu'elles influencent significativement les prédictions.

VII. Éthique et réseaux de neurones

Les réseaux de neurones et l'intelligence artificielle (IA) soulèvent des questions éthiques cruciales. Ces technologies, bien qu'elles apportent des solutions innovantes, posent des questions importantes concernant la vie privée, l'équité et la responsabilité. La réflexion sur ces enjeux est essentielle pour garantir une utilisation juste et responsable des modèles dans des contextes réels.

7.1. Les enjeux éthiques

Un des enjeux majeurs concerne la présence de biais dans les modèles. Ces biais sont souvent amplifiés par les données d'entraînement, ce qui peut conduire à des résultats injustes ou discriminatoires. Par exemple, dans les systèmes de reconnaissance faciale, les performances peuvent varier considérablement entre différents groupes démographiques en raison d'un déséquilibre dans les données. Ce problème est accentué par l'impact croissant de l'IA sur l'emploi, où l'automatisation menace certains métiers tout en créant de nouveaux défis pour la formation des travailleurs.

L'utilisation malveillante de ces technologies est également préoccupante. Les **deepfakes**¹², qui exploitent des réseaux génératifs adversariaux (GAN), permettent de créer de faux contenus visuels ou audio avec une qualité impressionnante. Bien que cette technologie ait des applications légitimes, comme la génération d'effets spéciaux dans le cinéma, elle présente des risques élevés de désinformation et de fraude. Enfin, la transparence et l'explicabilité des modèles complexes, tels que les réseaux neuronaux profonds utilisés dans les diagnostics médicaux, restent un défi majeur. Comprendre les décisions prises par ces modèles est souvent difficile, ce qui peut poser des problèmes dans des applications critiques comme la santé ou la justice.

7.2. Cadre réglementaire

Les cadres réglementaires, notamment le **RGPD** en Europe, imposent des règles strictes sur la collecte et l'utilisation des données. La confidentialité et la sécurité des données sont des priorités, mais elles ne suffisent pas à elles seules. L'équité et l'inclusion doivent être intégrées dès la conception des modèles pour éviter toute discrimination. Cela signifie également que les développeurs portent une grande responsabilité dans la conception et la mise en œuvre de modèles éthiques.

La notion de responsabilité est particulièrement importante dans des domaines sensibles comme la santé ou la finance. Dans de tels contextes, il est essentiel de garantir que les résultats des modèles sont **interprétables** et **conformes aux normes éthiques**.

7.3. Vers une IA éthique

Plusieurs stratégies peuvent être mises en œuvre pour garantir une IA éthique. La mise en place d'audits réguliers permet d'identifier les biais et d'évaluer les performances des modèles sur des populations variées. Ces audits doivent être menés de manière indépendante pour garantir une conformité aux principes éthiques. Une autre priorité est le développement de l'IA explicative (XAI). La bibliothèque [Xplique](#), par exemple, permet de fournir des explications claires sur le fonctionnement des modèles, renforçant ainsi la confiance des utilisateurs dans les décisions prises par l'IA. Ce package a été utilisé dans un [projet de détection de tumeurs cérébrales](#) que j'ai réalisé, où il a aidé à interpréter les résultats des modèles complexes de manière pertinente et compréhensible.

¹² Un deepfake est un enregistrement vidéo ou audio réalisé ou modifié grâce à l'intelligence artificielle.

VIII. Conclusion

Ce projet a permis d'explorer différentes facettes des réseaux neuronaux artificiels, mettant en évidence leur potentiel dans divers contextes d'apprentissage automatique. Les points principaux que nous avons abordés incluent :

1. **Compréhension des réseaux supervisés** : Nous avons analysé le Perceptron Multi-Couches (PMC) en termes de structure et de fonctionnement, avec un accent particulier sur son usage pour des tâches de classification et de régression.
2. **Expérimentation logicielle** : Nous avons implémenté et testé des solutions logicielles pratiques, notamment pour démontrer les réseaux neuronaux à travers des applications CNN, des outils comme les cartes auto-adaptatives de Kohonen, et des applications GAN.
3. **Apprentissage non supervisé** : L'examen de paradigmes avancés tels que les GANs nous a permis de mieux comprendre les applications possibles de l'apprentissage non supervisé.
4. **Considérations éthiques** : Les aspects éthiques, y compris les biais algorithmiques, le respect de la vie privée et les impacts sociaux, ont été discutés pour souligner l'importance d'une utilisation responsable de ces technologies.

Ce projet a souligné l'importance des réseaux neuronaux dans l'intelligence artificielle, tout en montrant leur complexité et le besoin d'un cadre rigoureux pour une utilisation optimale. Les perspectives futures visent des modèles avancés, comme les Transformers, pour résoudre des défis plus complexes.

Nous pourrons trouver les applications associées à ce projet dans ce [repository GitHub](#).

VIII. Bibliographie

Références aux ouvrages, articles, documents consultés et utilisés pour le rapport.

- [1] [David Foster \(2019\). Generative Deep Learning. O'Reilly](#)
- [2] [Ian Goodfellow \(2016\). Advances in neural information processing systems](#)
- [3] [Ricco Rakotomalala, Université de Lyon. Carte de Kohonen](#)
- [4] [Explainable AI\(XAI\): A systematic meta-survey of current challenges and future opportunities](#)
- [5] [GitHub : Brain_tumor. Projet personnel.](#)
- [6] [Github : Xplique](#)
- [7] [Cours sur les CNN, Erwan Scornet](#)