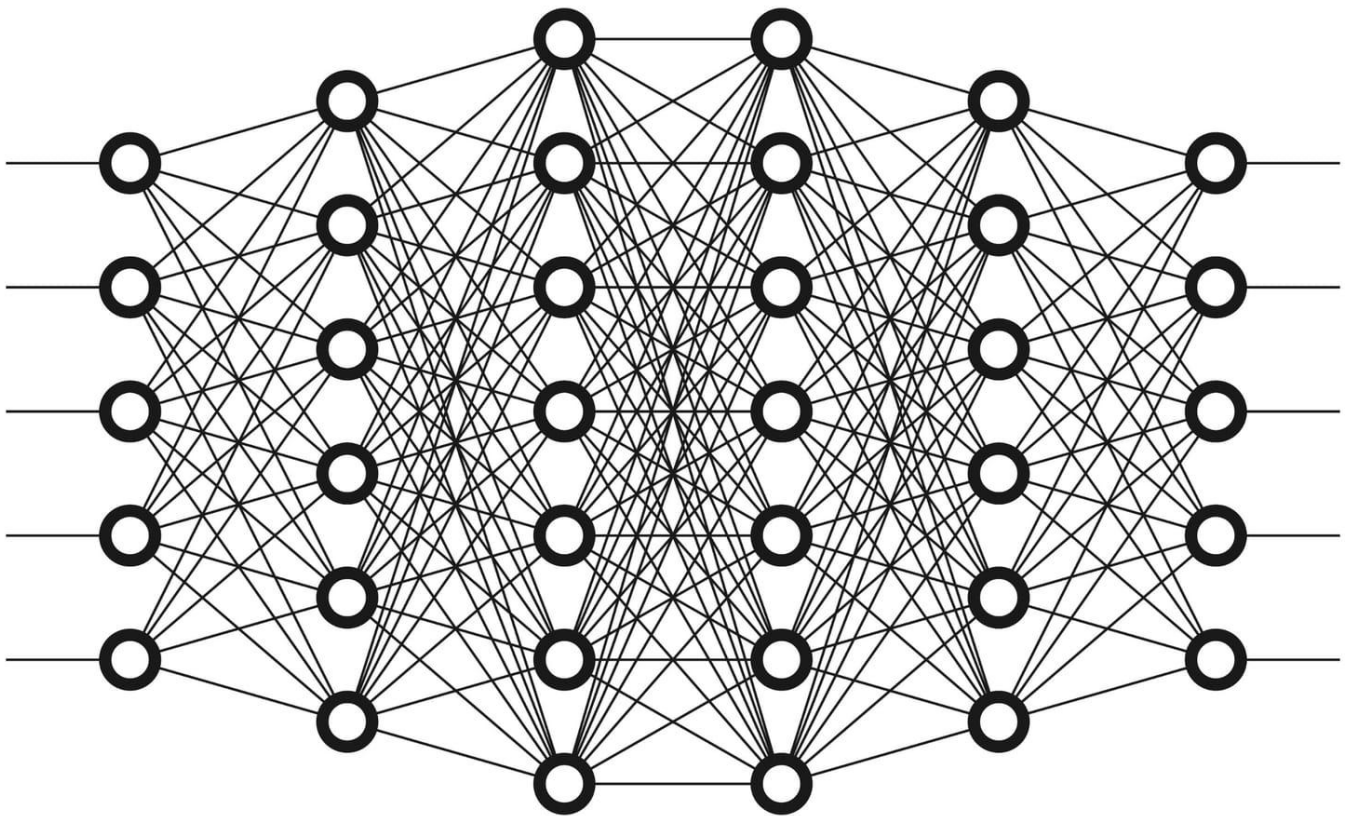


Machine-Learning : Projet-2

Réseaux De Neurones



AL NATOUR Mazen et HERVOUET Léo

ML 2024-2025

Table des matières

Introduction :	3
I. Perceptron Simple	4
A. Stratégie de Développement	4
B. Fonctionnement pour le cas du OU	5
C. Figure de la frontière de décision	6
II. Apprentissage Widrow	7
A. Stratégie de Développement	7
B. Ensemble Test 1	8
1. Graphiques des étapes d'apprentissage	8
2. Erreur en fonction des itérations	9
3. Tests avec initialisations différentes	10
C. Ensemble Test 2	11
1. Graphiques des étapes d'apprentissage	11
2. Erreur en fonction des itérations	13
3. Tests avec initialisations différentes	15
III. Perceptron Multicouche	16
A. Stratégie de Développement	16
B. Résultats	17
IV. Apprentissage Multicouche	18
A. Stratégie de Développement	18
B. Erreur en fonction des itérations	18
C. Droites séparatrices	19
V. Classification par Full-Connected	21
A. Stratégie de Développement	21
B. Comparaison des caractéristiques	22
C. Paramétrage optimal	27
D. Évaluation par classe d'images	28
E. Comparaison avec K-Plus-Proches-Voisins (KppV)	29
VI. Classification par Deep Learning	30
A. Architecture et stratégie d'apprentissage :	30
B. Comparaison des caractéristiques du réseau	31
C. Data Augmentation	33
D. Transfert Learning	34
E. Stratégies pour limiter l'overfitting	35
F. Comparaison Globale des Performances	35
VII. Comparaison Deep vs Full Connected	36
Conclusion :	37

Introduction :

Pour ce projet nous allons voir plusieurs algorithmes d'apprentissage supervisé, à commencer par les modèles les plus simples comme les perceptrons, jusqu'aux modèles les plus complexes impliquant des réseaux neuronaux multicouches et des méthodes de classification plus avancées.

L'objectif est de comprendre le comportement et l'efficacité de ces algorithmes dans des environnements d'apprentissage variés, en analysant l'impact des choix d'initialisation, du nombre d'itérations et des caractéristiques des données sur la convergence des modèles, tout en optimisant les paramètres pour obtenir des performances optimisées.

Nous verrons dans un premier temps un perceptron simple, en décrivant son fonctionnement et en étudiant ses zones de non-décision. Nous verrons ensuite l'apprentissage selon Widrow et le testerons sur différents jeux de données. Pour finir, nous nous intéresserons aux réseaux multicouches dans le cadre de leur évaluation dans des tâches de classification, notamment avec des architectures complètement connectées et en apprentissage profond, en faisant attention aux stratégies d'optimisation, les surapprentissage et aux transferts d'apprentissage.

I. Perceptron Simple

A. Stratégie de Développement

Le perceptron simple qui repose sur une règle d'apprentissage simple. Il consiste à ajuster les poids synaptiques en fonction des erreurs observées lors de la classification des points d'entrée. L'objectif est de trouver une frontière de décision linéaire qui sépare correctement les classes des données.

➤ Implémentation

L'implémentation du perceptron simple commence par la définition d'une fonction `perceptron_simple`. Cette fonction prend en entrée :

- **inputs** : un tableau numpy représentant les points de données à classer.
- **weights** : un tableau numpy contenant les poids synaptiques du neurone, incluant le seuil.
- **activation_function** : un entier qui détermine la fonction d'activation à utiliser (0 pour la fonction signe et 1 pour la fonction tanh).

Le calcul du produit scalaire entre les points d'entrée et les poids est réalisé à l'aide de `np.dot(inputs, weights[1:])`, suivi de l'ajout du seuil (`poids[0]`). Ensuite, en fonction de la fonction d'activation choisie, le résultat est activé avec la fonction signe (`np.sign`) ou tangente hyperbolique (`np.tanh`).

➤ Choix de l'initialisation des poids

Dans l'exemple, les poids initiaux sont définis manuellement : **`weights_OR = np.array([-0.5, 1, 1])`**

Ce vecteur de poids correspond à un biais (le seuil, ici -0.5) et à deux poids associés aux entrées `x1x_1x1` et `x2x_2x2`. Ces poids sont choisis de manière à résoudre le problème du OU, comme illustré ci-dessous.

➤ Fonction d'activation

Le perceptron utilise deux options pour la fonction d'activation : la fonction **signe** ou la fonction **tanh**. Dans ce cas, la fonction signe est utilisée, car elle est bien adaptée à la classification binaire :

- Signe de la sortie : 1 si le résultat est positif, -1 sinon.

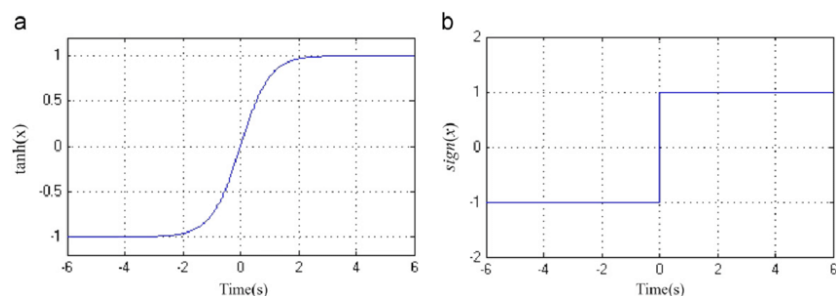


Figure 1 : Différence entre la fonction `tanh()` et `sign()`

➤ Critère d'arrêt

Dans cette implémentation, le perceptron ne continue pas l'apprentissage à travers les itérations. Une exécution unique est réalisée, et les poids sont définis de manière fixe. Un critère d'arrêt classique dans un cas d'apprentissage serait basé sur un nombre d'itérations maximal ou sur une convergence en fonction de l'erreur.

B. Fonctionnement pour le cas du OU

Pour vérifier le fonctionnement du perceptron, nous avons testé l'algorithme sur un ensemble de données correspondant à la fonction logique OU. Les points d'entrée utilisés sont les suivants :

```
the_data_points = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
```

Ce tableau correspond à la table de vérité de la fonction OU, où chaque couple d'entrée est associé à une sortie définie par :

- $0 \text{ OR } 0 = 0$
- $0 \text{ OR } 1 = 1$
- $1 \text{ OR } 0 = 1$
- $1 \text{ OR } 1 = 1$

L'exécution du perceptron sur cet ensemble produit les résultats suivants :

```
results_OR = perceptron_simple(the_data_points, weights_OR, 0)  
print("results_OR : ", results_OR)
```

Sortie obtenue : *results_OR : [-1 1 1 1]*

Ici, les résultats montrent que le perceptron classe correctement les points associés à la sortie du OU :

- Le point $[0, 0]$ est classé comme -1 (faux).
- Les points $[0, 1]$, $[1, 0]$, et $[1, 1]$ sont classés comme 1 (vrai).

C. Figure de la frontière de décision

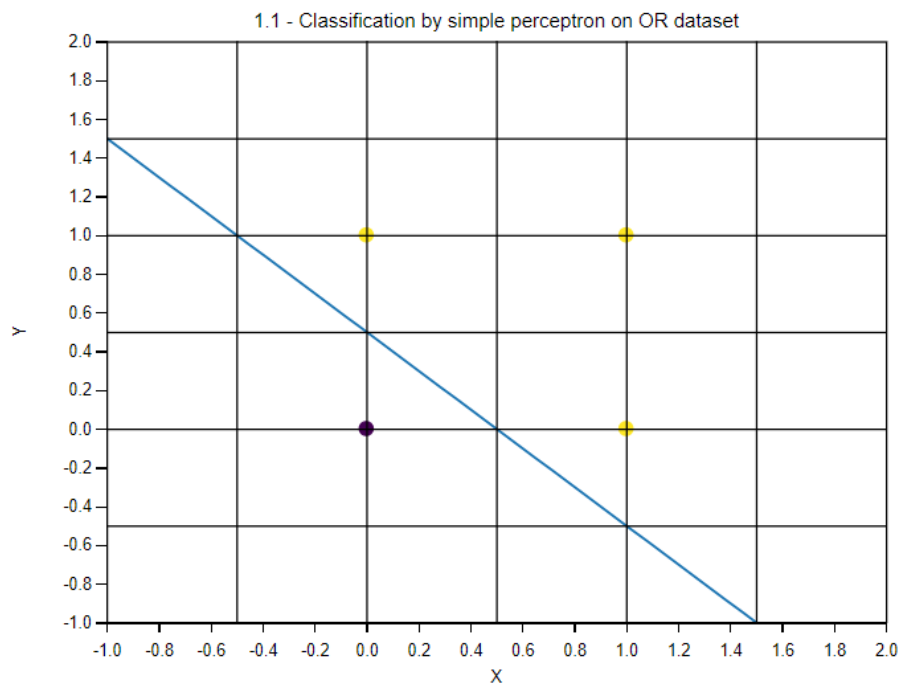


Figure 2 : Classification par perceptron Simple sur l'ensemble de données OR

La figure obtenue (Figure 2) montre les points d'entrée (en violet et jaune) ainsi que la droite de séparation calculée. Voici le code utilisé pour générer le graphique :

```
plot_with_class(  
    the_data_points, weights_OR, results_OR,  
    "1.1 - Classification by simple perceptron on OR dataset",  
    -1, 2  
).show()
```

Le graphique montre une droite de séparation qui divise correctement les points, avec une légère marge pour les points $[0, 1]$, $[1, 0]$, et $[1, 1]$ qui sont classés dans la classe 1. La frontière de décision du perceptron permet de séparer les points correspondant à la sortie 1 de ceux correspondant à la sortie 0. Pour le cas du OU, le perceptron arrive à trouver une solution linéaire, ce qui est attendu puisqu'une seule droite suffit à séparer les points de ce problème. Cela montre que le perceptron simple est capable de résoudre correctement ce type de tâche de classification linéairement séparable.

II. Apprentissage Widrow

A. Stratégie de Développement

Pour implémenter l'algorithme de Widrow-Hoff (ou LMS), nous avons utilisé un perceptron simple afin de classer les points dans deux catégories distinctes. La stratégie adoptée repose sur la mise à jour progressive des poids après un certain nombre d'éléments (**batch_size**), et non à chaque itération. Également, mettre à jour les poids après chaque élément pourrait orienter le modèle dans une mauvaise direction si cet élément est mal classé. Cette approche par lots permet de lisser les fluctuations causées par des données bruitées.

Afin de maximiser l'efficacité de l'apprentissage, à chaque nouvelle **epoch**, les données sont mélangées aléatoirement. Cela évite que le perceptron n'apprenne uniquement sur des éléments d'une seule classe en début d'apprentissage, ce qui pourrait provoquer une convergence prématurée dans une mauvaise direction. Le mélange des données permet donc une exploration plus équilibrée des deux classes, réduisant ainsi le risque d'un apprentissage biaisé.

Pour la mise à jour des poids, nous utilisons la règle suivante :

$$\begin{aligned} \text{nouveau_poids}[i] \\ &= \text{ancien_poids}[i] - \text{learning_rate} \times (-(\text{resultat_désiré} \\ &\quad - \text{resultat_obtenu}) \times \text{dérivé_fonction_activation}(\text{poids} \cdot \text{données}) \times \text{données}[i]) \end{aligned}$$

L'erreur globale est calculée comme la somme des carrés des erreurs pour chaque élément :

$$\text{erreur_globale} = \sum (\text{résultat_désiré} - \text{résultat_obtenu})^2$$

Une erreur nulle signifie que les classes sont correctement séparées.

Pour accélérer l'apprentissage, nous appliquons la fonction `sign()` en sortie du perceptron, car la fonction d'activation utilisée, `tanh`, retourne des valeurs continues entre -1 et 1. Sans cette transformation, il faudrait plus de temps pour que le perceptron converge vers des valeurs discrètes (-1 ou 1), ce qui ralentirait considérablement l'apprentissage.

En ce qui concerne le taux d'apprentissage, nous avons opté pour une valeur de 0,1, qui s'est avérée être un bon compromis entre la rapidité et la précision. Un taux plus élevé ($> 0,1$) peut empêcher le modèle de converger correctement, car les mises à jour seraient trop importantes. À l'inverse, un taux plus bas ($< 0,1$) ralentirait inutilement le processus d'apprentissage sans pour autant améliorer la précision finale.

B. Ensemble Test 1

1. Graphiques des étapes d'apprentissage

Lors de l'entraînement sur le premier ensemble de test, les itérations montrent une convergence rapide. Cela est dû au batch_size de 1, car les éléments sont distincts, et à chaque nouvel élément nous pouvons mettre à jour nos poids. La droite de séparation évolue au fil des itérations, jusqu'à ce qu'elle divise clairement les deux classes.

Voici les résultats obtenus pour une séparation des données sur l'ensemble "Test1" en 5 époques :

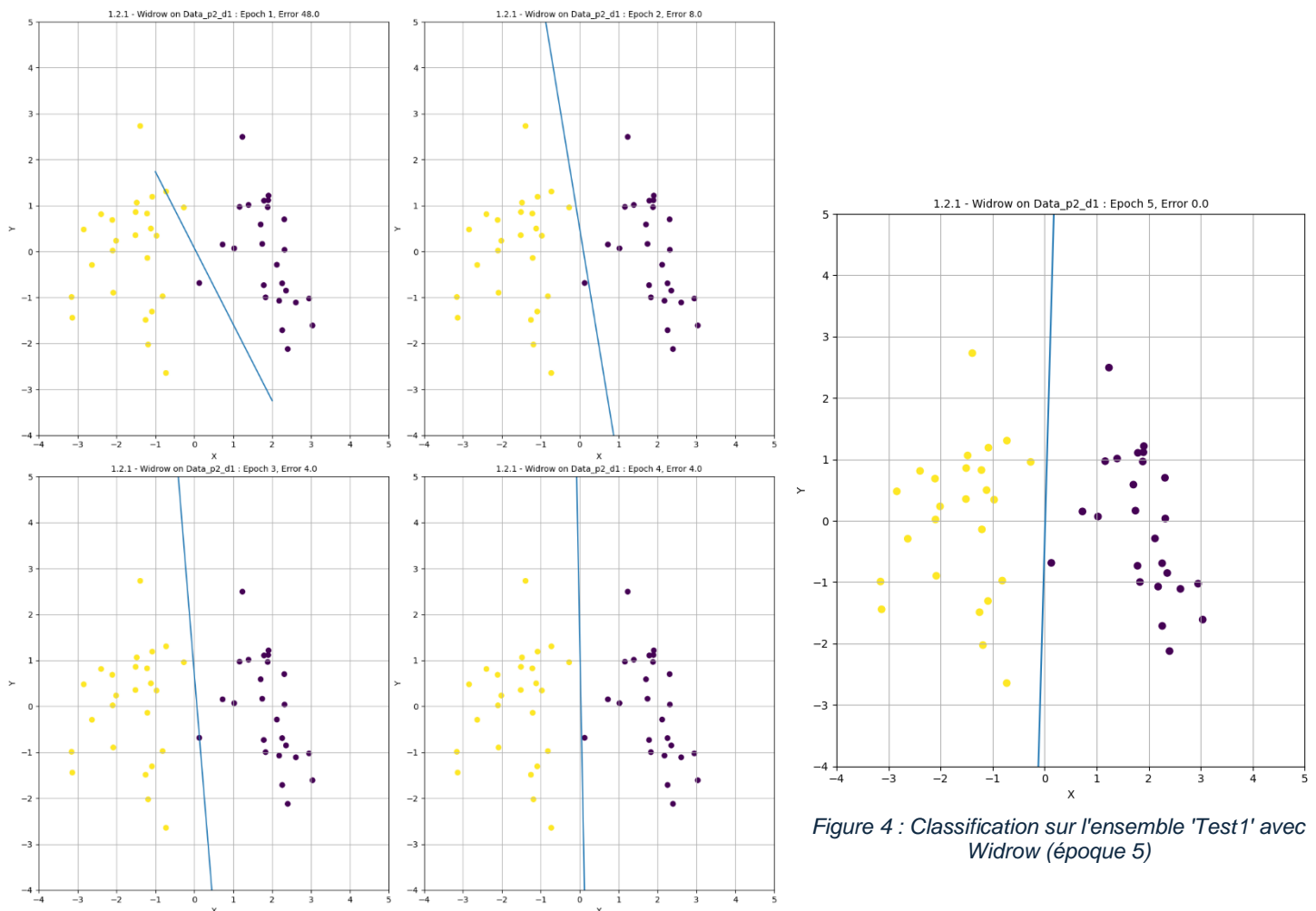


Figure 4 : Classification sur l'ensemble 'Test1' avec Widrow (épisodes 1 à 4)

Comme nous pouvons le voir sur les graphiques ([Figure 4](#) et [Figure 4](#)), les deux classes sont représentées par deux couleurs : le jaune et le violet.

Au départ, à époque 1, la droite est placée aléatoirement ([Figure 4](#)) et au fil des époques, elle s'oriente de façon à couper les deux ensembles en deux, à l'époque 5 ([Figure 4](#)).

Voici en plus grand, la séparation des deux classes ([Figure 5](#)) :

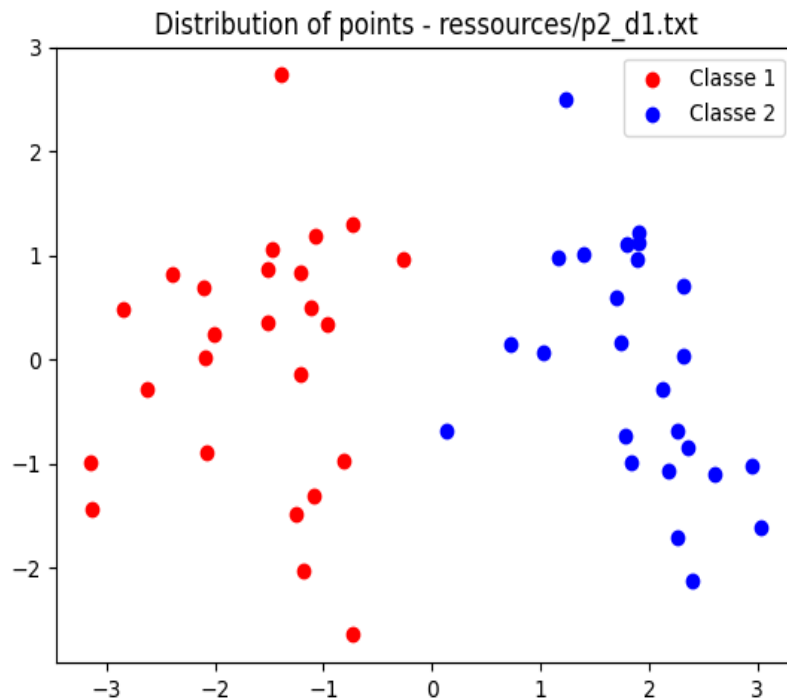


Figure 5 : Séparation des données pour l'ensemble 'Test1'

2. Erreur en fonction des itérations

En analysant l'évolution de l'erreur au fil des itérations, nous observons une diminution rapide de l'erreur. Cela montre que l'algorithme converge efficacement pour cet ensemble de données.

Voici la courbe d'erreur que nous avons obtenue pour les résultats précédants ([Figure 6](#)) :

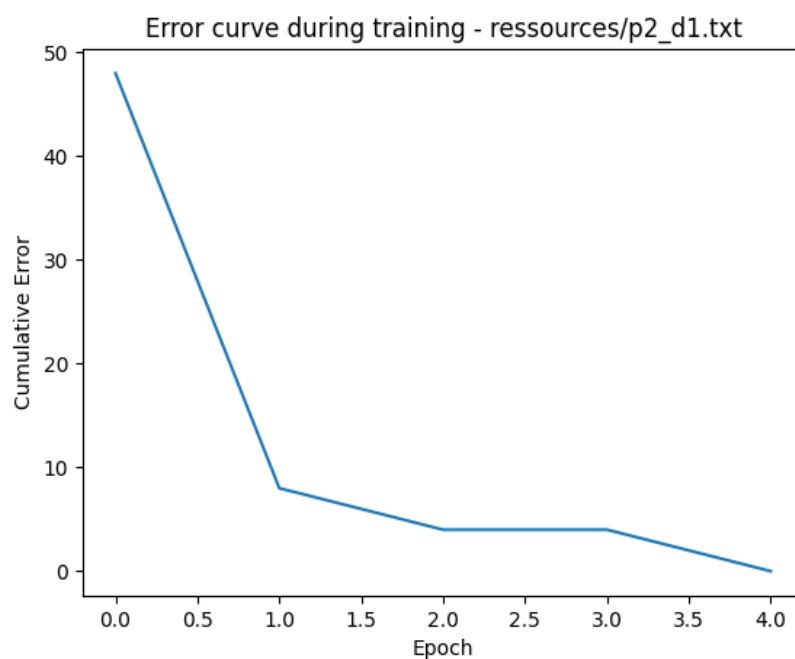


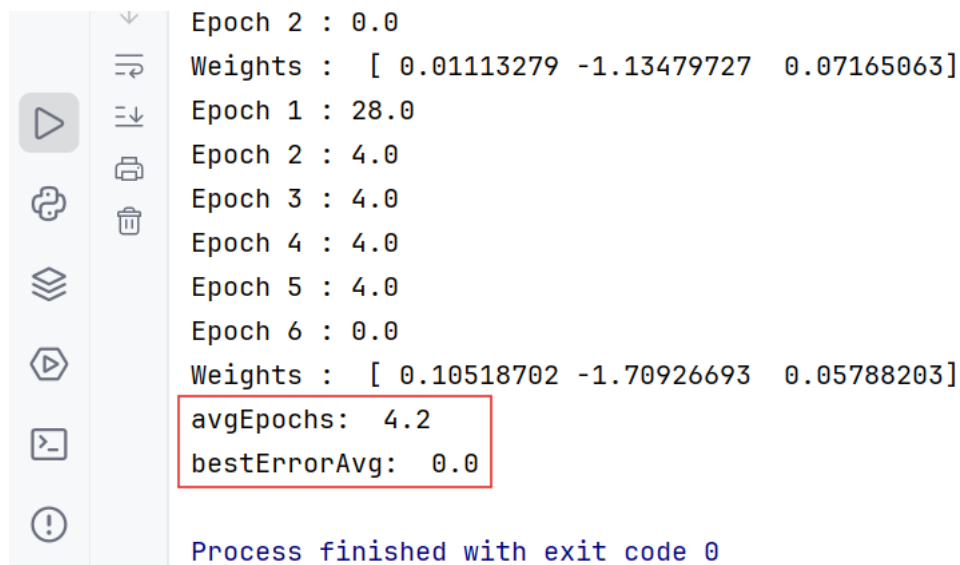
Figure 6 : Courbe d'erreur pour l'ensemble 'Test1' avec Widrow

La courbe commence avec une erreur de 48 (époque 1), ce qui s'explique par le fait que 4 points se trouvent dans la mauvaise classe (**Erreur ! Source du renvoi introuvable.**). Ensuite, plus nous avançons dans le nombre d'époques et plus l'erreur diminue pour finalement atteindre 0, ce qui indique que notre entraînement a trouvé une solution à notre problème de classification et que nos deux classes sont correctement séparées.

3. Tests avec initialisations différentes

Pour tester la robustesse de l'algorithme, nous avons effectué 100 essais avec différentes initialisations des poids (initialisation aléatoire). Dans chaque cas, l'algorithme parvient à converger rapidement vers une solution satisfaisante. En effet, c'est en environ 4 époques que le modèle arrive en moyenne à une séparation correcte des classes. L'algorithme semble donc peu sensible à l'initialisation des poids pour cet ensemble de données.

Voici les résultats que nous obtenons avec notre code pour 100 essais ([Figure 7](#)) :



```
Epoch 2 : 0.0
Weights : [ 0.01113279 -1.13479727  0.07165063]
Epoch 1 : 28.0
Epoch 2 : 4.0
Epoch 3 : 4.0
Epoch 4 : 4.0
Epoch 5 : 4.0
Epoch 6 : 0.0
Weights : [ 0.10518702 -1.70926693  0.05788203]
avgEpochs:  4.2
bestErrorAvg: 0.0

Process finished with exit code 0
```

Figure 7 : Résultat obtenu après 100 essais d'apprentissage avec Widrow sur l'ensemble 'Test1'

Avec 'avgEpochs' qui est le nombre d'époques en moyenne et 'bestErrorAvg' qui est le taux d'erreur à la fin de l'apprentissage.

C. Ensemble Test 2

1. Graphiques des étapes d'apprentissage

Dans le deuxième ensemble de test, la droite de séparation n'arrive pas à séparer complètement les deux classes, ce qui est attendu puisque certains éléments de la classe 2 se trouvent dans la zone de la classe 1. Ces deux éléments rendent impossible une séparation parfaite par une simple droite. Le graphique montre que malgré ces contraintes, l'algorithme tente de minimiser l'erreur autant que possible.

Voici les résultats obtenus pour une séparation des données sur l'ensemble "Test2" en 15 époques :

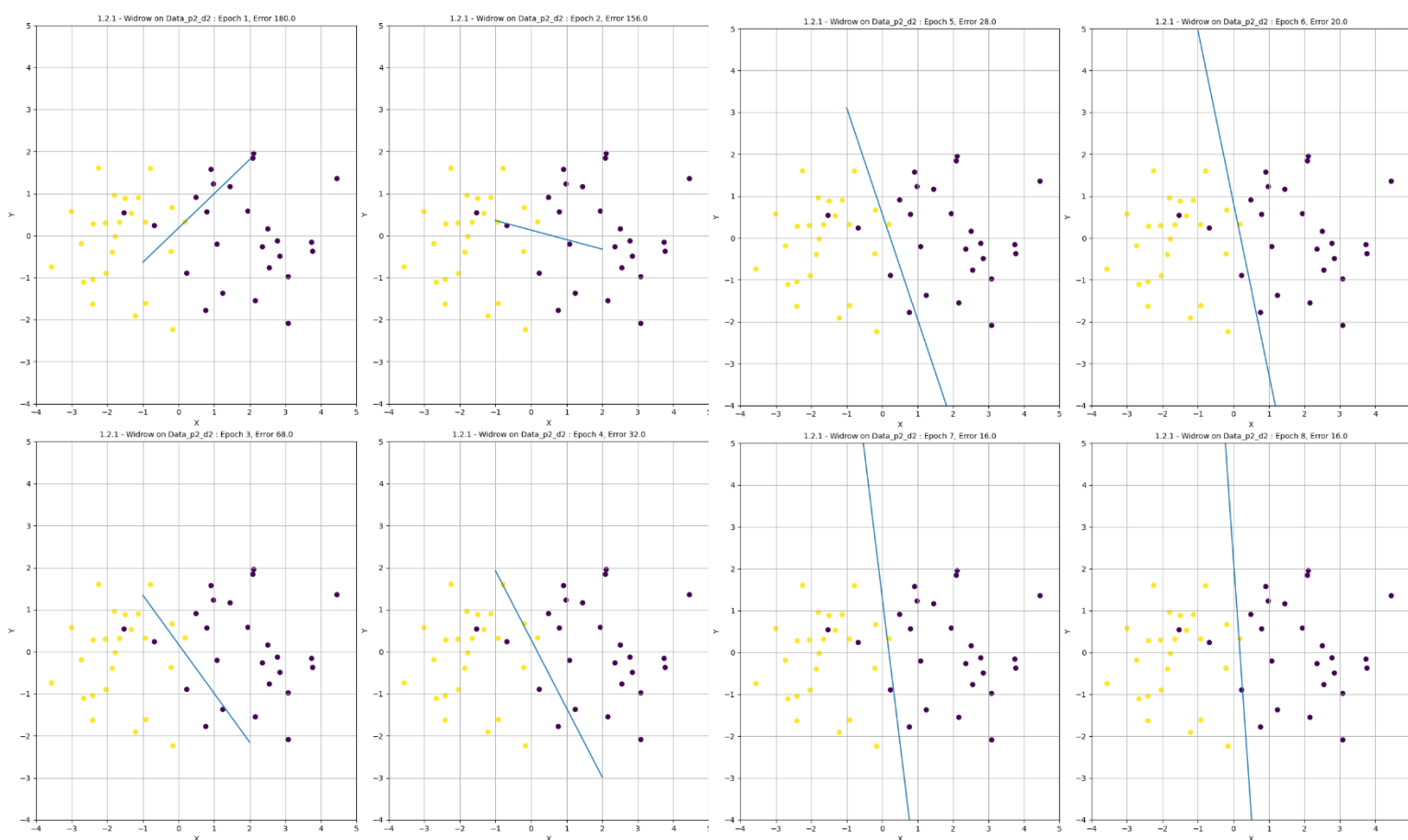


Figure 9 : Classification sur l'ensemble 'Test2' avec Widrow (époches 1 à 4)

Figure 9 : Classification sur l'ensemble 'Test2' avec Widrow (époches 5 à 8)

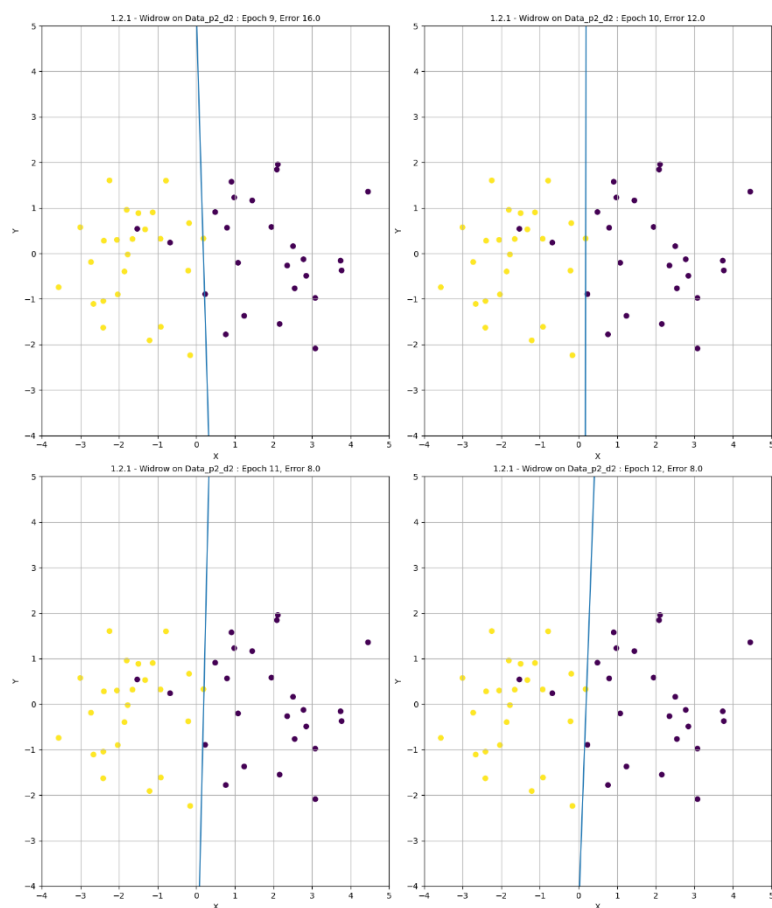


Figure 10 : Classification sur l'ensemble 'Test2' avec Widrow (époques 9 à 12)

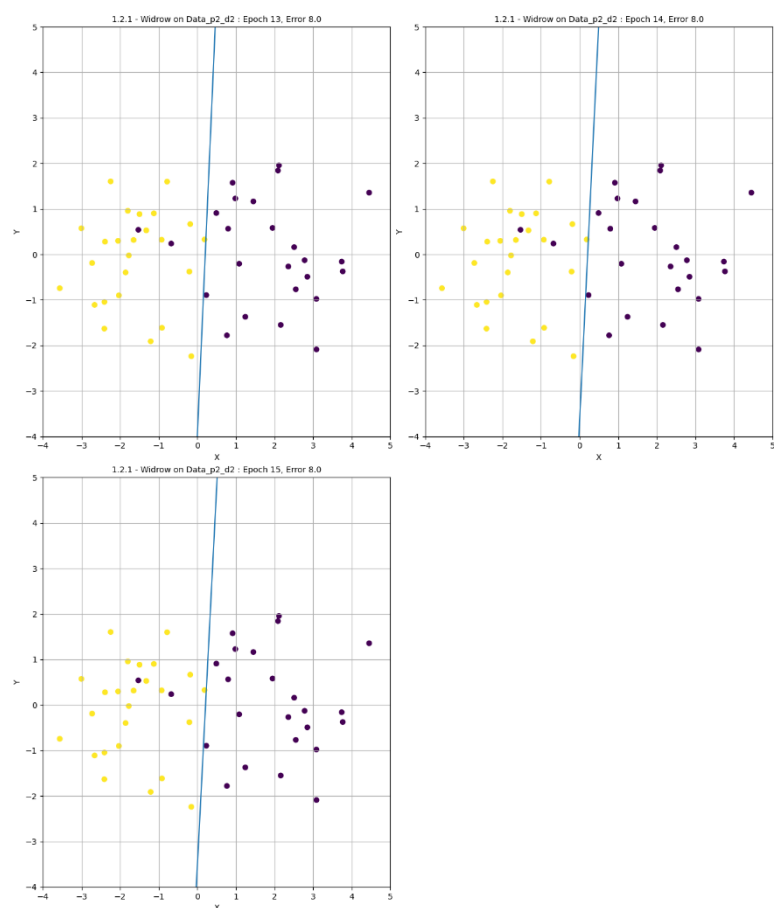


Figure 11 : Classification sur l'ensemble 'Test2' avec Widrow (époques 13 à 15)

Tout d'abord, quand l'apprentissage commence nous pouvons voir que la courbe de séparation est complètement fautive car elle coupe la première classe en deux (Figure 9, époque 1). Ensuite comme pour l'apprentissage sur l'ensemble 'Test1', notre modèle progresse de plus en plus (Figure 9, Figure 9, Figure 10, Figure 11, époques 2 à 14). Et une fois arrivé au bout de l'apprentissage, nous avons bien une droite qui sépare les deux classes en deux avec bien sûr, les deux éléments perturbateurs qui se retrouvent de l'autre côté de la ligne (Figure 11, époque 15).

Voici la répartition de l'ensemble 'Test2' (Figure 12) :

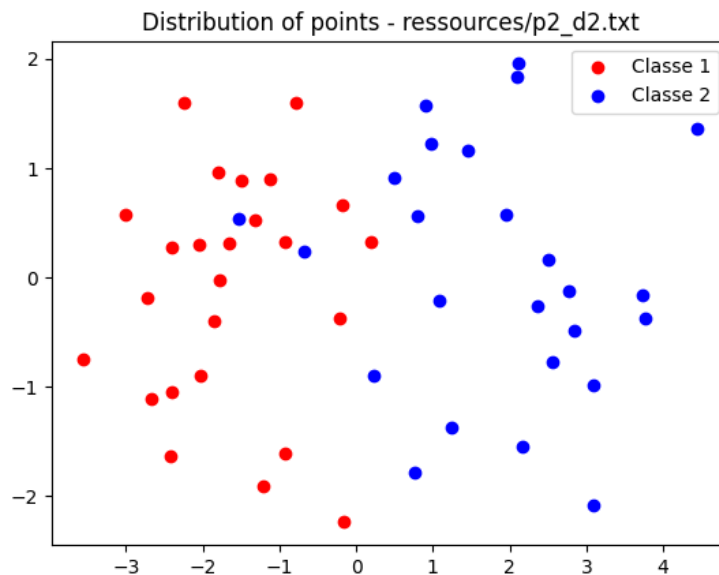


Figure 12 : Séparation des données pour l'ensemble 'Test2'

2. Erreur en fonction des itérations

Contrairement au premier ensemble de test, l'erreur ne parvient jamais à atteindre zéro. La présence d'éléments mal classés empêche une convergence totale. L'erreur se stabilise autour de 8, ce qui correspond aux deux points qui ne peuvent être correctement classés.

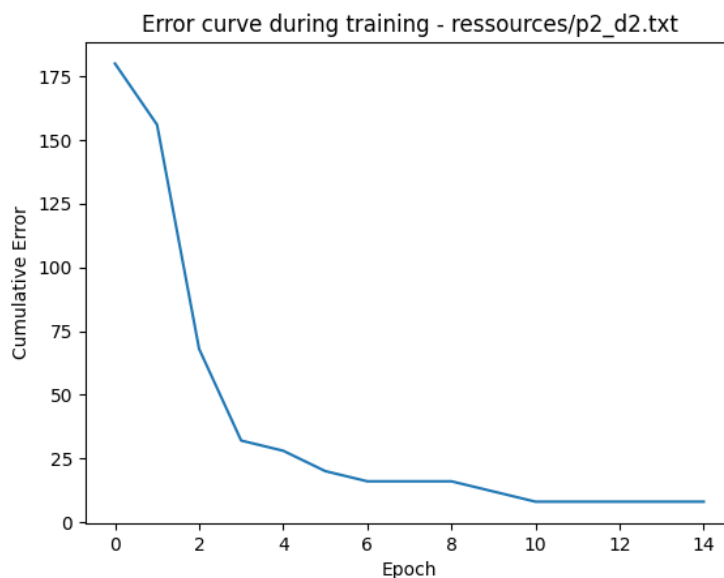
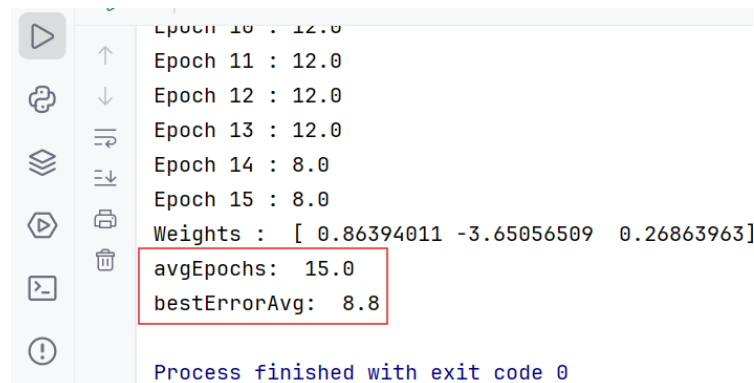


Figure 13 : Courbe d'erreur pour l'ensemble 'Test2' avec Widrow

Cette courbe (Figure 13) nous permet de voir qu'au début, comme vu précédemment, la répartition des classes n'est pas bonne, avec une erreur de 180. Ensuite, le modèle progresse rapidement pour tomber à une erreur de 20 à la 6^{ème} époque. Puis la courbe s'affine jusqu'à la 15^{ème} époque pour obtenir le résultat attendu (une erreur de 8).

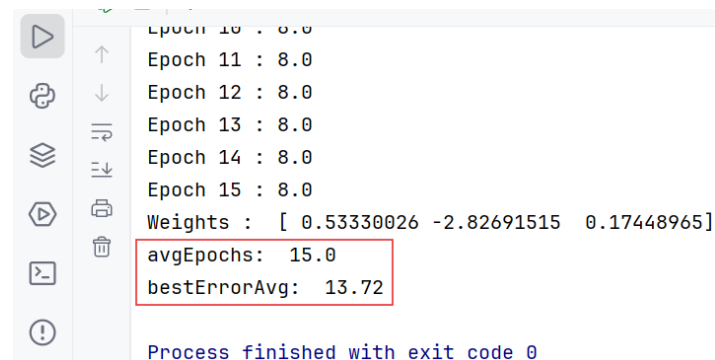
3. Tests avec initialisations différentes

Lorsque nous testons différentes initialisations pour le deuxième ensemble (initialisation aléatoire), l'algorithme met plus de temps à converger. La présence d'éléments perturbateurs et d'un batch_size plus élevée rend l'apprentissage plus sensible à la position initiale de la droite de séparation. Ainsi, l'initialisation de la droite de séparation peut retarder la convergence vers la solution optimale. Cela montre que pour des ensembles plus complexes, l'algorithme de Widrow-Hoff peut nécessiter un ajustement plus fin des paramètres pour assurer une convergence rapide. Dans notre cas il faut en moyenne 15 époques pour que le modèle trouve la meilleure solution (avec une batch_size de 50, Figure 14). Et nous pouvons voir qu'avec un batch_size égale à 1, l'apprentissage est beaucoup moins bon (Figure 15), ce qui s'explique par le fait qu'un élément perturbateur peut venir changer les poids dans la mauvaise direction, ce qui retarde donc l'apprentissage.



```
Epoch 10 : 12.0
Epoch 11 : 12.0
Epoch 12 : 12.0
Epoch 13 : 12.0
Epoch 14 : 8.0
Epoch 15 : 8.0
Weights : [ 0.86394011 -3.65056509 0.26863963]
avgEpochs: 15.0
bestErrorAvg: 8.8
Process finished with exit code 0
```

Figure 14 : Résultat obtenu après 100 essais d'apprentissage avec Widrow sur l'ensemble 'Test2' (époques max 15 et batch_size à 50)



```
Epoch 10 : 8.0
Epoch 11 : 8.0
Epoch 12 : 8.0
Epoch 13 : 8.0
Epoch 14 : 8.0
Epoch 15 : 8.0
Weights : [ 0.53330026 -2.82691515 0.17448965]
avgEpochs: 15.0
bestErrorAvg: 13.72
Process finished with exit code 0
```

Figure 15 : Résultat obtenu après 100 essais d'apprentissage avec Widrow sur l'ensemble 'Test2' (époques max 15 et batch_size à 1)

Avec 'avgEpochs' qui est le nombre d'époques en moyenne et 'bestErrorAvg' qui est le taux d'erreur à la fin de l'apprentissage.

III. Perceptron Multicouche

A. Stratégie de Développement

Dans cette partie, nous avons implémenté un perceptron multicouche simple, composé d'une couche d'entrée, d'une couche cachée, et d'une couche de sortie. L'architecture est définie comme suit :

- **Couche d'entrée** : Cette couche reçoit deux entrées, x_1 et x_2 , représentant les caractéristiques du problème à traiter. Nous y ajoutons un biais, noté 1, pour améliorer la convergence du modèle.
- **Couche cachée** : La couche cachée contient deux neurones. Les poids synaptiques associés à cette couche sont représentés par une matrice w_1 de taille 3×2 , où les trois lignes correspondent aux poids appliqués aux entrées $[1, x_1, x_2]$, et les deux colonnes représentent les deux neurones de la couche cachée. Nous appliquons une **fonction d'activation sigmoïde** à la sortie de chaque neurone de la couche cachée pour introduire de la non-linéarité dans le modèle, ce qui permet de mieux capter des relations complexes entre les variables.
- **Couche de sortie** : La couche de sortie se compose d'un unique neurone. Les poids synaptiques associés sont représentés par un vecteur w_2 de trois éléments (pour le biais et les deux sorties de la couche cachée). Ce neurone de sortie applique également une fonction d'activation sigmoïde à sa somme pondérée pour générer une prédiction.

L'algorithme d'apprentissage suivi est le suivant :

1. Calcul de la somme pondérée des entrées de la couche cachée.
2. Application de la fonction d'activation sigmoïde pour chaque neurone de la couche cachée.
3. Utilisation de ces sorties cachées pour calculer la somme pondérée des entrées du neurone de sortie.
4. Application de la fonction sigmoïde pour produire la sortie finale.

La fonction d'activation sigmoïde est définie par l'équation suivante :

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Cette fonction contraint les valeurs des neurones entre 0 et 1, facilitant ainsi l'apprentissage.

B. Résultats

Nous avons testé ce perceptron multicouche avec les entrées suivants : $\mathbf{x}_1 = 1$ et $\mathbf{x}_2 = 1$, et les poids synaptiques sont initialisés comme suit :

- **Poids de la couche cachée (w_1) :**

$$w_1 = \begin{pmatrix} -0,5 & 0,5 \\ 2,0 & 0,5 \\ -1,0 & 1,0 \end{pmatrix}$$

- **Poids de la couche de sortie (w_2) :**

$$w_2 = (2.0 \quad -1.0 \quad 1.0)$$

Les sorties des neurones de la couche cachée sont :

- **Neurone 1 :**

$$Sortie = \sigma(-0.5 \times 1 + 2.0 \times 1 + (-1.0) \times 1) = \sigma(0.5) \approx 0.622$$

- **Neurone 2 :**

$$Sortie = \sigma(0.5 \times 1 + 0.5 \times 1 + 1.0 \times 1) = \sigma(2.0) \approx 0.881$$

La sortie du neurone de la couche de sortie est ensuite calculée comme suit :

$$Sortie\ finale = \sigma(2.0 \times 1 - 1.0 \times 0.622 + 1.0 \times 0.881) = \sigma(2.259) \approx 0.905$$

Ainsi, la sortie finale du perceptron multicouche pour les entrées $\mathbf{x}_1 = 1$ et $\mathbf{x}_2 = 1$ est environ 0.905.

L'analyse des résultats montre que les sorties de la couche cachée, activées par la fonction sigmoïde, permettent de capturer des relations non-linéaires entre les entrées, et le perceptron multicouche réussit à produire une sortie significative après l'application des poids à la couche de sortie.

IV. Apprentissage Multicouche

A. Stratégie de Développement

- **Choix du Taux d'Apprentissage**

Un taux d'apprentissage de **0.5** a été choisi. Ce choix a été fait après plusieurs essais pour garantir une convergence rapide tout en évitant les oscillations. Un taux trop élevé pourrait entraîner un apprentissage instable, et un taux trop faible ralentirait l'apprentissage.

- **Critère d'Arrêt**

Le critère d'arrêt est basé sur l'erreur cumulée par époque. Nous avons fixé un seuil de **0.01** en dessous duquel l'apprentissage s'arrête. Ce seuil garantit une précision pour la classification XOR tout en minimisant la durée de l'entraînement.

- **Optimisation**

Nous avons utilisé une méthode par lots (batch size de **4**) afin de mettre à jour les poids après avoir accumulé les gradients pour toutes les données d'entrée. Cela permet une optimisation plus stable par rapport à une approche purement stochastique. De plus, la dérivée de la fonction sigmoïde a été utilisée pour calculer les gradients et ajuster les poids.

B. Erreur en fonction des itérations

L'évolution de l'erreur en fonction des itérations est représentée graphiquement ci-dessous, permettant de visualiser la convergence de l'apprentissage.

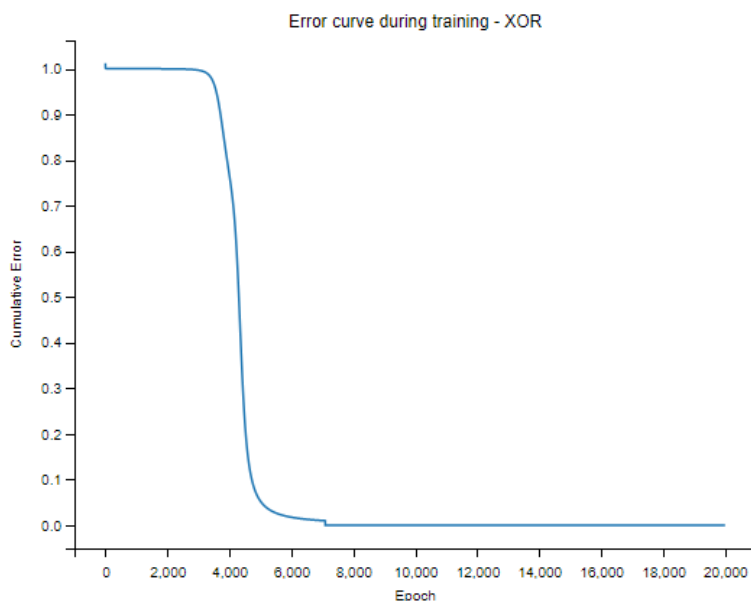


Figure 16 : La courbe d'erreur durant l'entraînement

- **La Convergence**

La courbe montre une décroissance continue de l'erreur jusqu'à atteindre le critère d'arrêt qu'on a fixé précédemment (0.01). Cela montre une bonne convergence de l'apprentissage, permettant de prouver que le réseau parvient à minimiser l'erreur pour la tâche XOR. Le réseau parvient à généraliser correctement après plusieurs milliers d'époques.

➤ Test sur le XOR

Les résultats des tests du réseau sur les différentes entrées de la fonction XOR sont les suivants :

$$x = [0, 0] / y = 0$$

$$x = [0, 1] / y = 1$$

$$x = [1, 0] / y = 1$$

$$x = [1, 1] / y = 0$$

Le réseau est capable de correctement classer les quatre combinaisons d'entrées du problème XOR, comme attendu.

C. Droites séparatrices

• Graphique des Droites Séparatrices

Les trois droites séparatrices obtenues à partir des poids des neurones cachés et de sortie sont illustrées dans la figure ci-dessous :

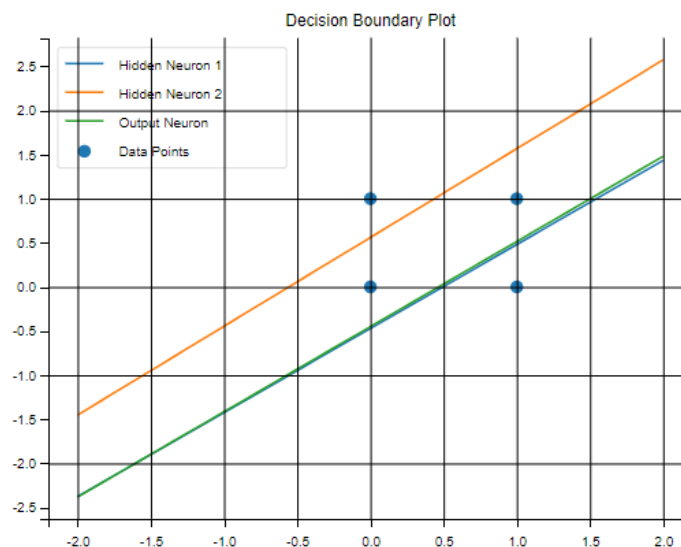


Figure 17 : Graphique des limites de décision

• Rôle dans la Classification XOR

- **Droite Neurone Caché 1** (orange) : Cette droite correspond au premier neurone caché. Son rôle est de séparer les données dans l'espace de manière à faciliter la tâche du neurone de sortie.
- **Droite Neurone Caché 2** (vert) : Cette droite correspond au second neurone caché. Elle est complémentaire à la première droite, permettant une séparation non linéaire des données.

- **Droite du Neurone de Sortie** (bleu) : Cette droite représente la séparation finale effectuée par le neurone de sortie pour classer les points dans l'une des deux catégories XOR (0 ou 1).

Les droites des neurones cachés découpent l'espace des entrées de manière à rendre linéairement séparable un problème initialement non linéairement séparable. Le neurone de sortie effectue ensuite une classification binaire à partir de ces transformations.

V. Classification par Full-Connected

A. Stratégie de Développement

Pour cette partie, une approche **Full-Connected** a été utilisée pour effectuer la classification des images selon les dix catégories définies (**Jungle [J]**, **Plage [P]**, **Monuments [M]**, **Bus [B]**, **Dinosaures [D]**, **Éléphants [E]**, **Fleurs [F]**, **Chevaux [C]**, **Montagne [M]**, **Plats [P]**) [dans cet ordre pour les matrices de confusion].

Le modèle construit est un réseau de neurones **séquentiel** comprenant :

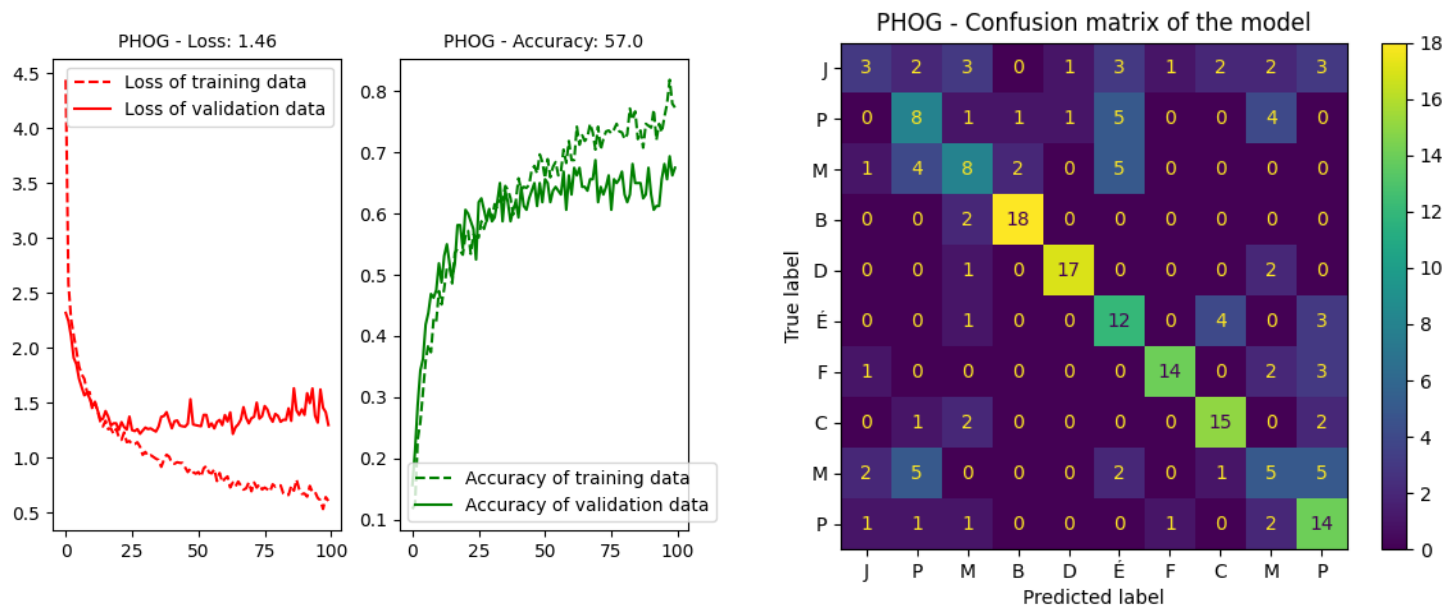
- **Couche d'entrée** : Les descripteurs extraits ou concaténés des images.
- **Architecture du réseau** :
 - Une première couche dense avec **512 neurones** et une fonction d'activation **ReLU**, suivie d'un dropout (**0,5**).
 - Une seconde couche dense avec **256 neurones (ReLU)** suivie d'un dropout (**0,5**).
 - Une troisième couche dense avec **128 neurones (ReLU)**.
 - Une quatrième couche dense avec **64 neurones (ReLU)**.
 - Une couche de sortie avec **10 neurones** et une fonction d'activation **softmax**.

Le réseau est entraîné avec une perte **categorical_crossentropy**, un optimiseur **Adam**, et la métrique d'évaluation est la précision (**accuracy**). Les données sont divisées en **60 % pour l'entraînement**, **20 % pour la validation** et **20 % pour les tests**, avec une validation interne de **20 %** sur l'ensemble d'entraînement. Le nombre d'époques est initialement fixé à **1000**, mais un **arrêt anticipé (EarlyStopping)** est utilisé pour arrêter l'entraînement si la perte de validation n'améliore pas pendant **100** époques. Le modèle a été appliqué à plusieurs descripteurs extraits des images : **PHOG**, **JCD**, **CEDD**, **FCTH**, **Fuzzy Color Histogram**, ainsi qu'une version **concaténée** de tous les descripteurs.

B. Comparaison des caractéristiques

Six types de descripteurs ont été testés pour évaluer leur impact sur la performance du réseau Full-Connected :

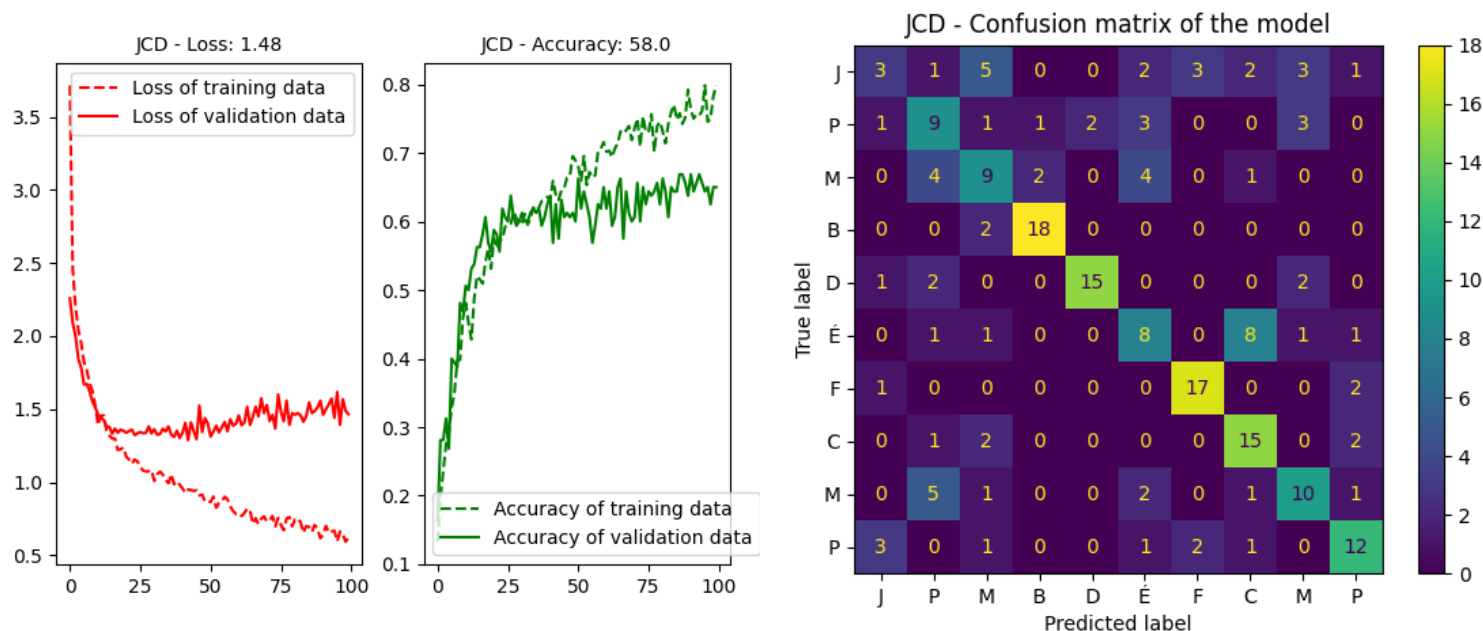
1. PHOG (Pyramid Histogram of Oriented Gradients)



Loss	Accuracy (Full Connected)	Accuracy (KppV)
1.4564	56.99 %	51 %

Le descripteur PHOG capture les caractéristiques structurales et les gradients orientés des images. Bien qu'il permette une classification relativement correcte (57 % de précision), ses performances restent limitées pour des catégories nécessitant davantage de discrimination basée sur les couleurs ou les textures (ex. : *Fleurs* ou *Plage*). Il est plus adapté pour des classes avec des motifs bien définis (ex. : *Bus* ou *Dinosaures*), mais la perte élevée montre qu'il ne capte pas toute la variabilité des données.

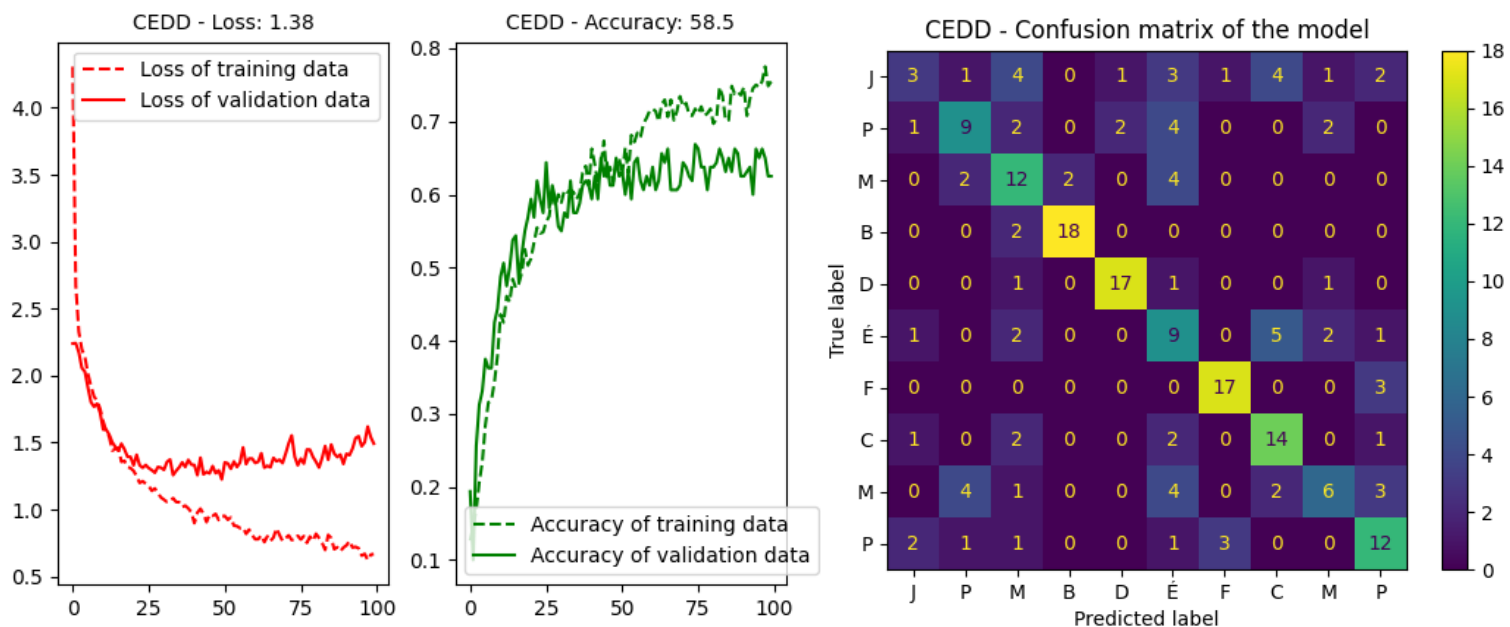
2. JCD (Joint Composite Descriptor)



Loss	Accuracy (Full Connected)	Accuracy (KppV)
1.4758	57.99 %	51 %

Le descripteur JCD combine des informations de couleur et de texture, ce qui le rend pertinent pour des classes visuellement complexes. Malgré cela, sa précision est légèrement inférieure à celle d'autres descripteurs comme *CEDD* ou *Fuzzy Color Histogram* (dans la suite). Les résultats montrent qu'il souffre d'un manque de robustesse face à des classes présentant des similitudes (ex. : *Montagne* et *Plage*).

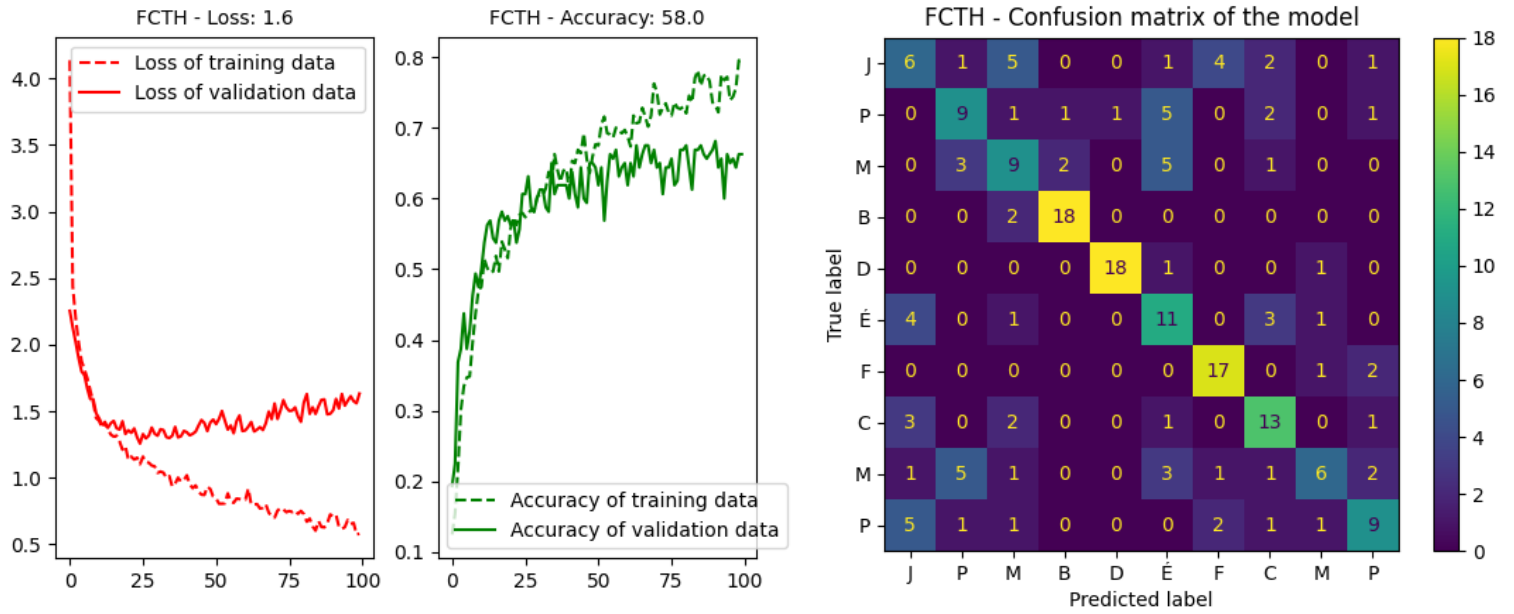
3. CEDD (Color and Edge Directivity Descriptor)



Loss	Accuracy (Full Connected)	Accuracy (KppV)
1.3766	58.49 %	51 %

Le descripteur CEDD intègre des informations sur la couleur et les bords. Avec une précision de 58.49 %, il surpasse légèrement PHOG et JCD. Cela reflète sa capacité à mieux discriminer des classes visuellement distinctes (ex. : *Dinosaures*, *Fleurs*). Cependant, sa perte relativement élevée montre une certaine difficulté à traiter des classes aux motifs plus complexes ou hétérogènes.

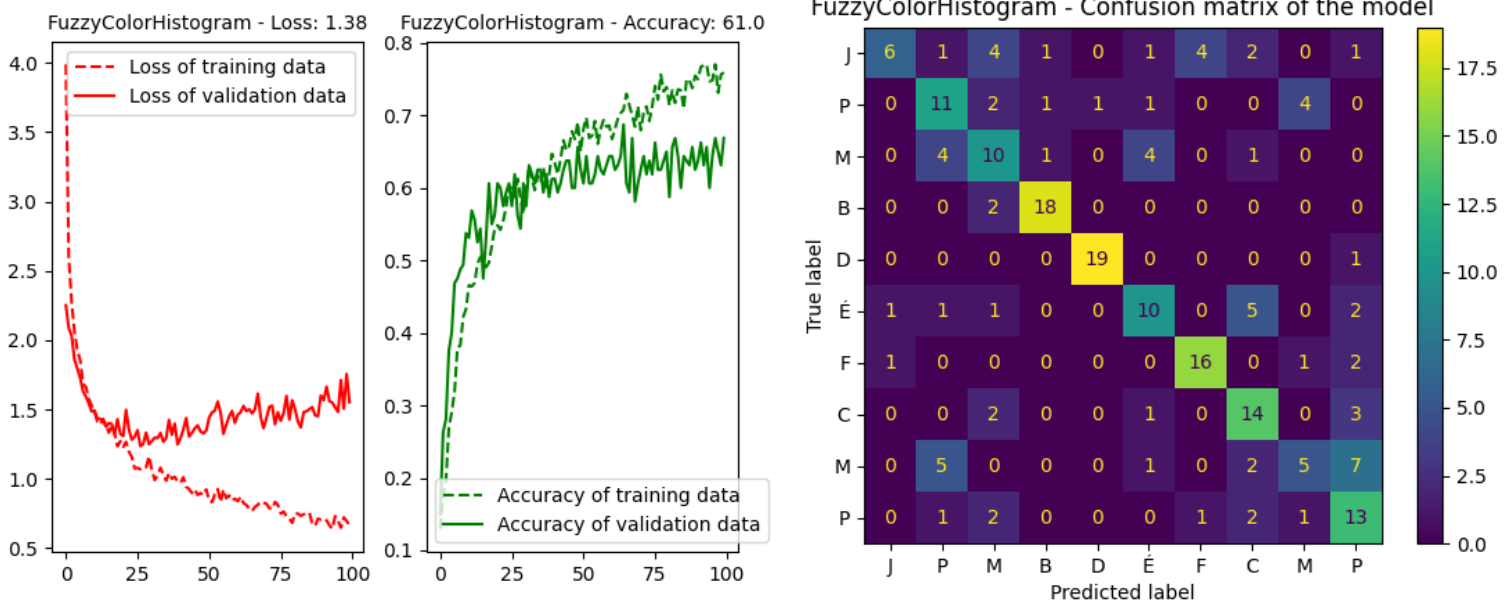
4. FCTH (Fuzzy Color and Texture Histogram)



Loss	Accuracy (Full Connected)	Accuracy (KppV)
1.5989	57.99 %	51 %

Le descripteur FCTH met l'accent sur la combinaison de textures et de couleurs en utilisant une approche floue. Cependant, ses performances sont légèrement inférieures à celles de *CEDD* et *Fuzzy Color Histogram*. Une perte plus élevée (1.5989) indique qu'il a du mal à généraliser, notamment pour des classes aux caractéristiques partagées (ex. : *Montagne* et *Plats*). Cela peut être dû à une moins bonne sensibilité aux subtilités des variations de couleur.

5. Fuzzy Color Histogram



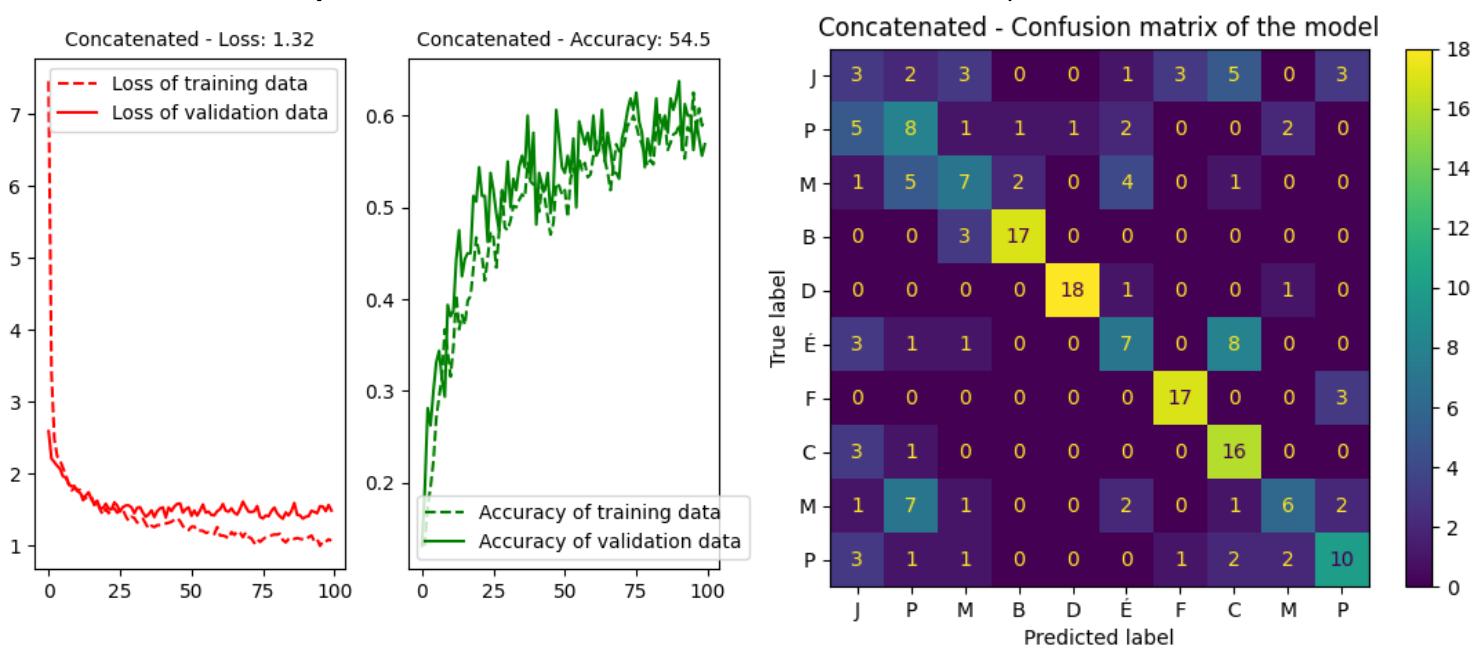
Loss	Accuracy (Full Connected)	Accuracy (KppV)
1.3775	61.00 %	51 %

Ce descripteur est le meilleur parmi tous les candidats, avec une précision de 61 % et une perte relativement faible. Sa force réside dans sa capacité à discriminer efficacement les images en se basant sur les couleurs de manière plus granulaires et sur une approche floue qui capture les nuances. Les performances sont particulièrement bonnes pour des classes comme :

- Dinosaur, où les variations de couleur jouent un rôle.
- Bus, qui se distingue par des tons clairs et homogènes.

Mais, il reste des confusions pour des classes comme *Montagne* ou *Jungle*, où les caractéristiques de couleur peuvent être moins discriminantes.

6. Descripteurs concaténés : Combinaison de tous les descripteurs.



Loss	Accuracy (Full Connected)	Accuracy (KppV)
1.3154	54.50 %	51 %

Les descripteurs concaténés intègrent toutes les caractéristiques extraites (PHOG, JCD, CEDD, FCTH, Fuzzy Color Histogram). Bien que cette approche soit théoriquement plus riche, elle a produit une performance inférieure, avec une précision de seulement 54.50 %. Cela peut être due à une surcharge d'informations non pertinentes ou redondantes qui perturbent l'apprentissage du réseau.

➔ Synthèse des performances :

DESCRIPTEUR	FORCES	FAIBLESSES
PHOG	Capture des motifs et des formes structurées.	Faible discrimination pour des classes basées sur les couleurs.
JCD	Bonne combinaison de couleur et texture.	Moins efficace pour des classes complexes.
CEDD	Bonne performance pour des classes distinctes.	Difficulté avec des motifs hétérogènes.
FCTH	Approche floue pour texture et couleur.	Sensibilité réduite aux nuances.
FUZZY COLOR HISTOGRAM	Excellente discrimination basée sur la couleur.	Limité pour des classes avec peu de variations.
CONCATENES	Plus grande richesse d'informations.	Surcharge et perte d'efficacité.

C. Paramétrage optimal

Nous avons choisi le paramétrage optimal de notre architecture en testant différents hyperparamètres et configurations de descripteurs. Notre choix est basé sur une analyse des meilleurs résultats obtenus avec différentes architectures et descripteurs :

1. Descripteur Optimal :

- **Le Fuzzy Color Histogram** est le descripteur permettant d'obtenir les meilleures performances globales. Son taux de bonne classification de 61 % surpasse celui des autres descripteurs, ceci grâce à la prise en compte des nuances de couleur à un niveau granulaire élevé, qui combine la prise en compte de l'imprécision avec des variations intra-classe. Néanmoins, d'autres descripteurs comme CEDD ou PHOG, même s'ils ont donné de bons résultats sur certaines classes particulières, ne possèdent pas la polyvalence qui caractérise Fuzzy Color Histogram.
- **Architecture du Réseau :**
 - **Couches denses** : Les quatre couches cachées avec respectivement 512, 256, 128 et 64 neurones permettent de capturer la complexité des relations entre caractéristiques.
 - **Régularisation** : Le dropout (0,5) utilisé après les deux premières couches denses joue un rôle déterminant dans la prévention du sur-apprentissage, tangible lors des tests initiaux avec les descripteurs concaténés.
 - **Fonction d'activation** : Les ReLU pour les couches cachées permettent d'obtenir une convergence rapide, sans gradients nuls.

2. Optimisation et Critères d'arrêt :

- Le réseau a été entraîné à l'aide de l'optimiseur Adam, à la fois adaptatif et efficace.
- L'arrêt anticipé (EarlyStopping) relativement strict (100 époques) afin d'éviter toute sur-optimisation au-delà du point de convergence afin d'améliorer la généralisation du modèle.

Ce paramétrage a donné des résultats à la fois solides et cohérents avec une précision supérieure à 60 %, associée à une perte à peu près constante et une généralisation sur les données de test quasiment identiques.

D. Évaluation par classe d'images

Les matrices de confusion générées après entraînement nous ont permis de faire une analyse de la performance du modèle pour chaque classe d'images. Les résultats montrent des variations en termes de précision par classe, en montrant les points forts et les limites.

1. Catégories bien discriminées :

- **Dinosaures** : Les images de cette catégorie ont été correctement classées avec une grande précision, grâce à des caractéristiques distinctives comme la texture et la couleur.
- **Bus** : Cette classe a également été bien reconnue, ses caractéristiques homogènes en termes de couleur et de forme ayant été capturées efficacement par les descripteurs et notre architecture du réseau.

2. Classes confondues :

- **Plats et Montagne** : Ces deux catégories ont montré une confusion notable, attribuée à des similitudes visuelles dans leurs descripteurs respectifs. Par exemple, les tons terreux ou les textures homogènes peuvent prêter à confusion.
- **Plage et Montagne** : Les plages rocheuses ou montagneuses partagent parfois des textures similaires, entraînant des erreurs de classification dans ces cas spécifiques.

3. Variabilité intra-classe :

- **Fleurs** : Bien que cette classe soit visuellement distinctive, une variabilité importante au sein des images (différentes couleurs, types de fleurs, arrière-plans variés) a compliqué leur classification. Cela montre les limites des descripteurs actuels face à des motifs très diversifiés.

4. Confusion inter-catégorie :

- Certaines catégories montrent également des confusions due à des similarités dans leurs descripteurs basés sur la couleur et la texture (plage par exemple). Cette confusion inter-catégorie peut être atténuée à l'avenir en combinant des informations contextuelles ou en intégrant des descripteurs plus spécifiques.

E. Comparaison avec K-Plus-Proches-Voisins (KppV)

Pour évaluer la performance relative de l'approche Full-Connected, l'algorithme de K-Plus-Proches-Voisins (KppV), a été utilisé afin de comparer les 2 approches. Les résultats obtenus mettent en évidence les limites du KppV face à la complexité des données utilisées.

1. Résultats du KppV :

- Avec $k = 5$, l'algorithme KppV a atteint une précision constante de **51 %** pour tous les descripteurs testés.
- Cette précision, bien que respectable pour un modèle non-paramétrique, reste inférieure à celle du modèle Full-Connected, qui a surpassé 60 % avec le descripteur Fuzzy Color Histogram.

2. Forces et Limites du KppV :

- **Forces :**
 - Simplicité et rapidité d'implémentation.
 - Bonne performance pour des données bien séparées dans l'espace des caractéristiques.
- **Limites :**
 - Sensibilité au bruit : KppV repose sur la distance dans l'espace des caractéristiques, ce qui le rend vulnérable aux variations intra-classe et inter-catégorie.
 - Incapacité à voir les relations complexes entre les descripteurs.

3. Comparaison avec Full-Connected :

L'architecture Full-Connected surpasse largement KppV grâce à sa capacité à apprendre des relations non linéaires complexes entre les descripteurs, en intégrant des mécanismes de régularisation et des fonctions d'activation adaptées. Là où KppV échoue à discriminer des catégories visuellement similaires, le réseau Full-Connected réussit à mieux généraliser.

VI. Classification par Deep Learning

A. Architecture et stratégie d'apprentissage :

- **Modèle Simple :**

- ➔ Architecture : Ce modèle est constitué d'une seule couche convolutive avec 32 filtres de taille 3×3 , suivie d'une couche de pooling 2×2 pour réduire la dimensionnalité. Enfin, une couche dense entièrement connectée avec 10 neurones (activation softmax) permet de classer les images en sortie.
- ➔ Objectif : Son rôle est de fournir une référence de base pour évaluer les améliorations des modèles plus complexes. Bien que sa simplicité permette une convergence rapide, il est limité dans sa capacité à extraire des caractéristiques complexes des images, ce qui réduit son efficacité pour des tâches plus exigeantes.

- **Modèle Profond (Deep Model) :**

- ➔ Architecture : Ce modèle intègre deux couches convolutives successives. La première comporte 8 filtres de taille 5×5 , suivis d'une couche de pooling 2×2 , et la seconde utilise 16 filtres de taille 3×3 avec un autre pooling 2×2 . Une couche de Flatten transforme les caractéristiques en vecteur pour les passer à une couche dense finale.
- ➔ Stratégie : L'ajout de profondeur permet au modèle de détecter des motifs plus abstraits. La première couche identifie des caractéristiques simples (bords, textures), tandis que les couches plus profondes capturent des relations plus complexes.

- **Modèle de Transfert Learning :**

- ➔ Base pré-entraînée : Le réseau EfficientNetB0, déjà entraîné sur ImageNet, est utilisé comme point de départ.
- ➔ Architecture Personnalisée : Les couches convolutives de la base sont gelées pour conserver leurs connaissances générales. Une couche de pooling global (GlobalAveragePooling2D) compresse les caractéristiques extraites. Ensuite, une couche dense intermédiaire (64 neurones avec activation ReLU) et une couche softmax en sortie effectuent la classification spécifique.

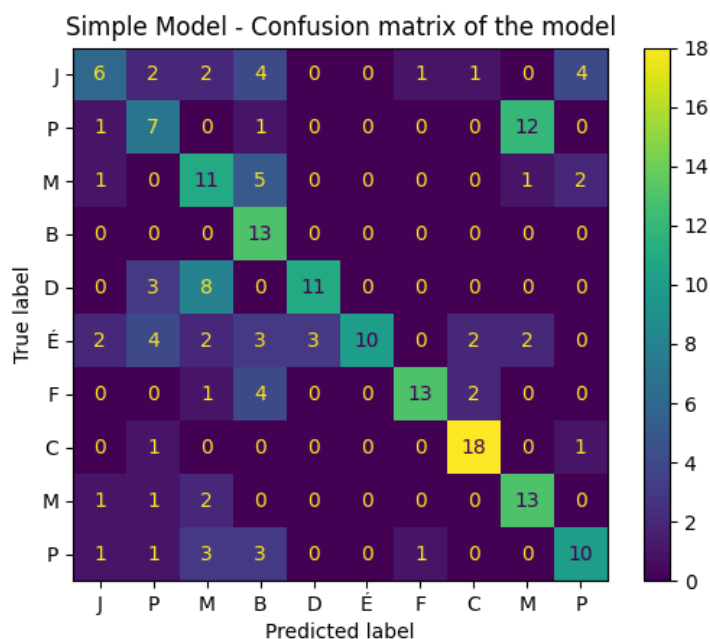
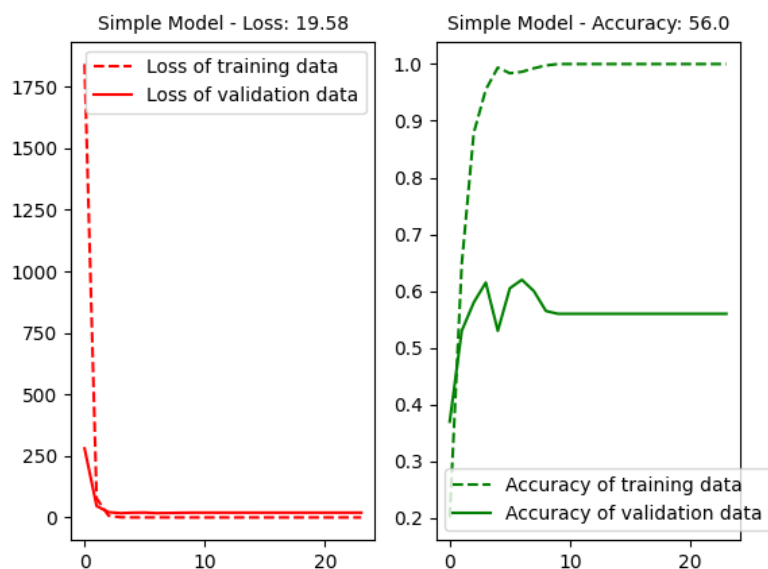
- **Modèle avec Data Augmentation :**

- ➔ Prétraitement : Ce modèle utilise une couche d'augmentation des données pour générer des images artificiellement variées via des rotations aléatoires et des symétries horizontales.
- ➔ Architecture : Inspirée du modèle profond, elle comporte davantage de filtres (32 et 64), optimisant l'utilisation des données augmentées pour une meilleure généralisation.

B. Comparaison des caractéristiques du réseau

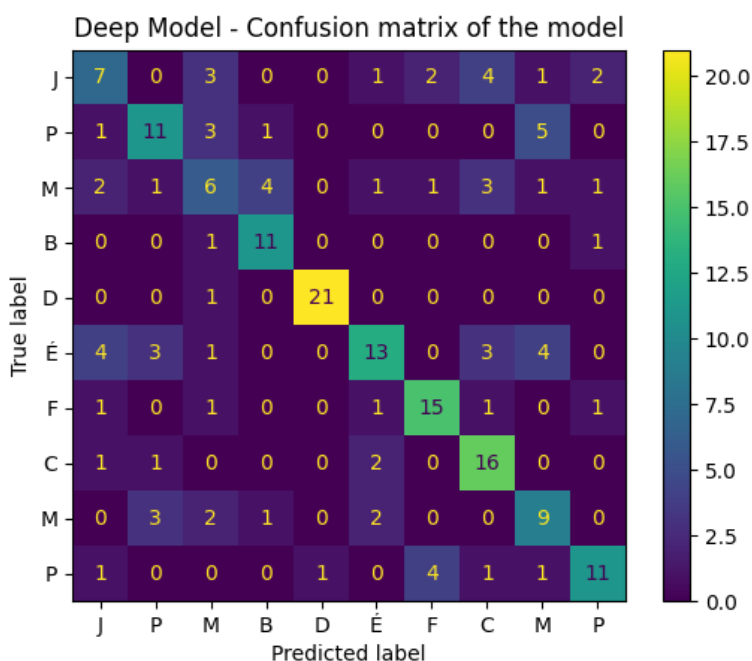
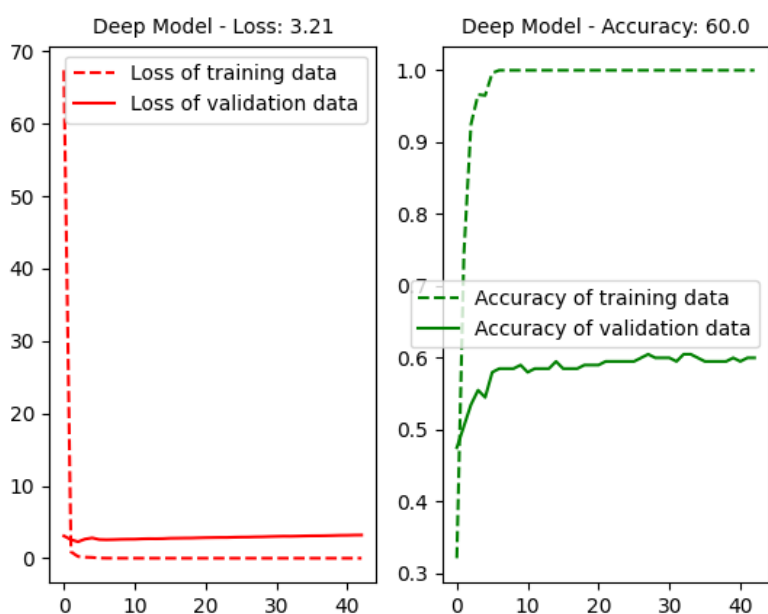
- **Profondeur (Nombre de Couches)**

➔ L'influence de la profondeur est directement observable à travers les performances obtenues par les modèles simples et profonds :



- **Modèle Simple :**

- **Loss** : 19.58. Une perte aussi élevée indique que le modèle a du mal à s'ajuster aux données d'entraînement. Cela peut être attribué à la faible complexité du modèle, qui limite sa capacité à extraire des caractéristiques pertinentes des images.
- **Accuracy** : 56%. Avec une précision légèrement supérieure à la moitié, le modèle simple atteint des résultats modérés, mais ne parvient pas à généraliser efficacement les relations complexes entre les caractéristiques.



- **Modèle Profond (Deep) :**

- **Loss :** 3.21. Une baisse significative par rapport au modèle simple, ce qui montre que le réseau parvient à mieux minimiser l'écart entre ses prédictions et les étiquettes réelles. L'ajout d'une seconde couche convolutive et de davantage de filtres améliore sa capacité à extraire des caractéristiques plus détaillées.
- **Accuracy :** 60%. Bien que l'amélioration ne semble pas spectaculaire, elle témoigne de l'effet bénéfique de la profondeur, qui permet de capturer des relations plus subtiles entre les pixels. Cette précision pourrait encore être optimisée en ajustant les hyperparamètres, comme le taux d'apprentissage ou les fonctions de régularisation.

- **Comparaison et Analyse :**

- ➔ L'écart de précision (4%) entre les deux modèles montre l'impact positif de l'ajout de couches convolutives pour la détection de motifs complexes. Cependant, cela met également en évidence que l'architecture profonde seule ne suffit pas à garantir des performances optimales, en particulier avec un volume de données limité. Une approche complémentaire, comme la Data Augmentation ou le Transfert Learning, pourrait donner de meilleurs résultats.

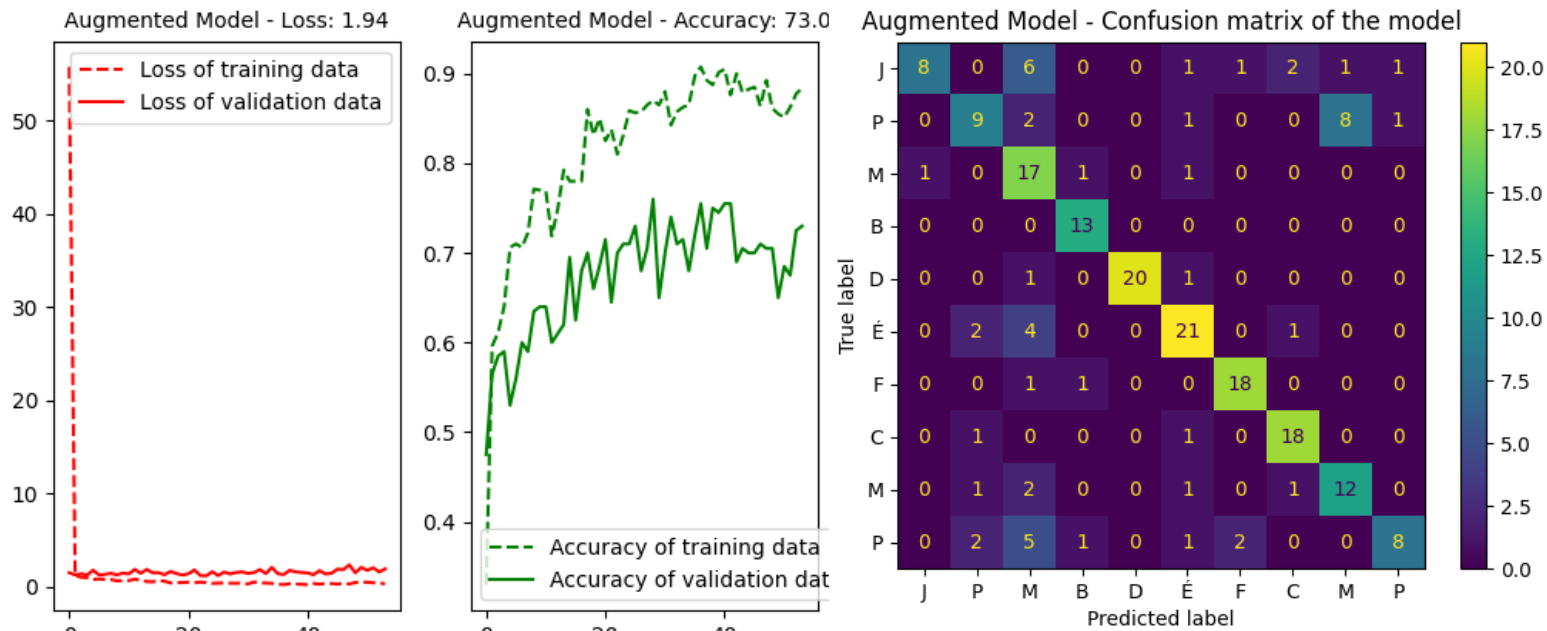
2. Largeur (Nombre de Filtres)

- L'augmentation du nombre de filtres (par exemple, de 8 à 32 ou 64) offre au modèle la possibilité de détecter une plus grande variété de motifs dans les images (bords, textures, structures complexes). Cependant, une largeur excessive peut rapidement conduire à un surentraînement, surtout si les données disponibles ne sont pas suffisantes pour exploiter pleinement cette capacité. Dans le cas du modèle profond avec 16 filtres, l'amélioration des performances est notable, mais l'efficacité diminue au-delà d'un certain point, ce qui souligne l'importance d'un équilibre entre complexité du modèle et volume de données.

3. Taux d'Apprentissage

- Le taux d'apprentissage fixe utilisé avec l'optimiseur Adam a permis une convergence stable, mais d'autres stratégies, comme les taux adaptatifs ou la réduction progressive du taux (learning rate scheduler), pourraient permettre à la convergence d'aller plus vite et ainsi réduire les oscillations en fin d'entraînement.

C. Data Augmentation



- **Mise en Œuvre et Impact :**

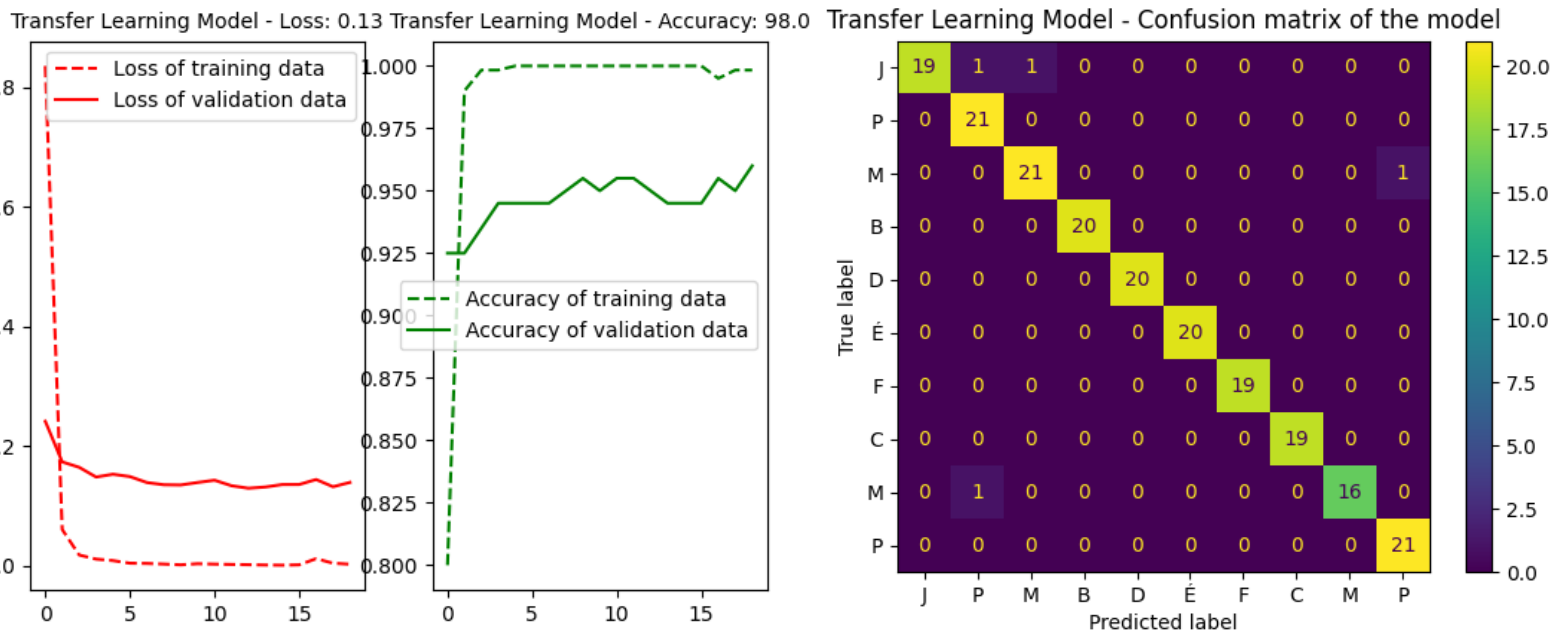
➔ L'augmentation des données a considérablement amélioré les performances du modèle profond :

- **Loss : 1.94.** Une réduction significative par rapport au modèle sans augmentation, ce qui indique que le réseau s'entraîne plus efficacement en exploitant des données diversifiées.
- **Accuracy : 73%.** Ce bond de 13% démontre que la Data Augmentation aide le modèle à mieux généraliser sur les données de test, réduisant ainsi les erreurs liées à un surajustement sur l'ensemble d'entraînement initial.

- **Comparaison avec les Modèles Précédents :**

Par rapport au modèle simple, la précision s'est améliorée de 17%, montrant que la diversification des données d'entraînement aide à compenser les limites de complexité de l'architecture. Par rapport au modèle profond, l'amélioration est de 13%, confirmant que la Data Augmentation est un levier pour optimiser les performances sans augmenter la complexité structurelle du réseau.

D. Transfert Learning



- **Performances Atteintes :**

- **Loss :** 0.13. Cette perte extrêmement faible indique que le modèle pré-entraîné sur ImageNet parvient à minimiser presque parfaitement les écarts entre les prédictions et les étiquettes, grâce aux représentations riches déjà apprises.
- **Accuracy :** 98%. Une très forte précision qui reflète la capacité du transfert d'apprentissage à exploiter efficacement des connaissances préexistantes pour s'adapter rapidement à une nouvelle tâche.

- **Comparaison avec les Autres Modèles :**

- Par rapport au **modèle simple**, la précision a bondi de 42%, ce qui illustre l'écart énorme entre une architecture basique et un réseau sophistiqué comme EfficientNetB0.
- Par rapport au **modèle profond**, l'augmentation de 38% démontre que, même avec une architecture plus complexe, un entraînement à partir de zéro reste limité par la taille des données et la durée d'entraînement.
- Par rapport au **modèle avec Data Augmentation**, le gain de 25% montre que la richesse des représentations pré-entraînées sur ImageNet est bien supérieure à ce qu'un modèle peut apprendre en partant de zéro, même avec des données diversifiées.

- **Analyse des Résultats :**

Le transfert d'apprentissage se distingue comme la solution la plus performante et la plus efficace, en particulier pour des bases de données d'images modestes comme celle utilisée ici. L'approche réduit non seulement le temps d'entraînement, mais permet également d'atteindre une précision presque optimale, idéale pour des applications nécessitant une fiabilité élevée.

E. Stratégies pour limiter l'overfitting

- **Dropout** : l'intégration de Dropout (désactivation aléatoire de neurones) après chaque couche dense pourrait être une stratégie efficace pour limiter l'overfitting.
- **Régularisation** : Une pénalité sur les poids pourrait être ajoutée pour limiter la complexité excessive et prévenir le surentraînement.
- **Validation Croisée** : Utiliser une validation croisée sur 20% des données d'entraînement aide à évaluer la performance du modèle sur des données non vues, réduisant ainsi le risque d'overfitting.
- **Optimisation du Gradient** : Utiliser des techniques comme le clipping des gradients pourrait stabiliser l'entraînement et améliorer les performances finales du modèle.

F. Comparaison Globale des Performances

Modèle	Loss	Accuracy	Commentaires
Simple	19.58	56%	Modèle de référence, simple et rapide, mais limité pour des tâches complexes.
Deep	3.21	60%	Amélioration notable grâce à une profondeur accrue, mais encore limité.
Data Augmentation	1.94	73%	Réduction de l'overfitting et meilleure généralisation sur les données test.
Transfert Learning	0.13	98%	Performances optimales grâce aux connaissances pré-entraînées d'ImageNet.

- Le modèle simple est utile pour des tâches basiques, mais montre rapidement ses limites sur des images plus complexes.
- Le modèle profond et la Data Augmentation apportent des améliorations significatives, mais restent contraints par la taille de l'ensemble d'entraînement.
- Le Transfert Learning surpasse toutes les approches, confirmant son avantage pour des ensembles de données limités en exploitant des réseaux pré-entraînés sur des bases massives comme ImageNet.

VII. Comparison Deep vs Full Connected

- **Performances des Réseaux Convolutifs (CNN) vs. Full-Connected :**

- **CNN** : Les réseaux convolutifs exploitent les propriétés locales des images, capturant ainsi des caractéristiques essentielles comme les bords, les textures et les motifs. Cette approche est particulièrement efficace pour la classification visuelle où les relations spatiales entre les pixels sont cruciales. Les CNN sont capables d'extraire des caractéristiques complexes grâce à leurs différentes couches convolutives et de pooling, ce qui les rend adaptés pour des tâches de détection et de reconnaissance où les informations spatiales sont pertinentes.
- **Full-Connected** : À l'inverse, les architectures Full-Connected (FC) traitent les pixels individuellement sans tenir compte de leurs relations spatiales. Chaque neurone est connecté à tous les autres dans l'image, ce qui augmente la complexité du modèle et le nombre de paramètres à ajuster. Cela peut rapidement conduire à un surentraînement, surtout lorsqu'il y a une quantité limitée de données d'entraînement disponibles. En raison de leur approche globale, les FC nécessitent généralement un nombre beaucoup plus élevé de neurones pour atteindre des performances similaires à celles des CNN, ce qui peut ralentir leur convergence.

- **Impact des Performances :**

- **CNN** montre une meilleure généralisation avec une précision généralement plus élevée par rapport aux architectures Full-Connected. Le CNN est capable de mieux capturer les relations locales et globales dans les images, ce qui se traduit par de meilleures performances sur des données non-standardisées.
- **Full-Connected**, bien qu'efficace pour des tâches basiques, ne peut pas tirer parti des structures et motifs spatiaux présents dans les images. Cela limite sa capacité à surperformer sur des tâches complexes nécessitant une détection de pattern fine. Les réseaux Full-Connected peuvent être limités par l'overfitting plus rapidement en raison de leur architecture trop complexe par rapport à la taille de l'ensemble d'entraînement.

Conclusion :

Ce projet nous aura permis d'explorer en profondeur les principaux algorithmes d'apprentissage supervisé, des modèles simples aux architectures avancées, en mettant en évidence leurs forces, leurs limites et leurs performances pour des contextes variés, tout en analysant l'impact des paramètres et des stratégies d'apprentissage.

Nous aurons montré comment le Perceptron est capable de résoudre des problèmes linéairement séparables et à quel point des approches plus complexes comme le réseau multicouche sont nécessaires pour un cas non linéaire comme le XOR. Les parties autour de l'apprentissage selon Widrow auront permis de montrer combien l'initiation ainsi que le critère de convergence sont importants, tandis que les parties sur la classification Full-Connected et Deep Learning ont montré l'intérêt d'utiliser des méthodes plus avancées telles que la Data Augmentation ou le Transfer Learning pour améliorer les performances.

Ce projet a renforcé notre compréhension de l'apprentissage automatique et amélioré nos compétences en développement et en analyse de modèles, tout en nous préparant à des défis plus complexes.