

Projet 1 : Les réseaux de neurones

Introduction

Le domaine de l'apprentissage automatique et de la vision par ordinateur a connu un essor considérable ces dernières années. Avec l'avènement de nouveaux algorithmes et de nouvelles architectures de réseaux de neurones, il est aujourd'hui possible d'obtenir des résultats impressionnantes dans des tâches de classification, de détection d'objets ou encore de segmentation d'images.

Dans ce premier projet, nous allons explorer différents aspects de l'apprentissage automatique et de la vision par ordinateur en utilisant des réseaux de neurones. Nous allons commencer par la mise en place d'un perceptron simple, puis nous étudierons l'apprentissage de ce modèle en implémentant l'algorithme de Widrow-Hoff. Nous testerons ensuite notre perceptron sur des jeux de données simples.

Ensuite, nous passerons à des modèles plus complexes en développant un perceptron multicouche. Nous mettrons en place l'apprentissage de ce modèle et testerons ses performances sur des jeux de données plus complexes.

Nous aborderons ensuite la discrimination d'une image en utilisant deux approches différentes. Tout d'abord, nous utiliserons une approche basée sur les descripteurs et mettrons en place un système de discrimination basé sur un modèle Full-Connected. Ensuite, nous explorerons l'approche "Deep" basée sur les données et mettrons en place un réseau de neurones convolutionnel pour discriminer une image.

Enfin, nous conclurons sur les résultats obtenus et les limites de ces modèles, ainsi que sur les perspectives dans le domaine de l'apprentissage automatique et de la vision par ordinateur.

Algorithme Perceptron Simple

Ce programme doit évaluer la sortie d'un perceptron simple (1 neurone) pour une entrée élément de \mathbb{R}^2 .

Les paramètres :

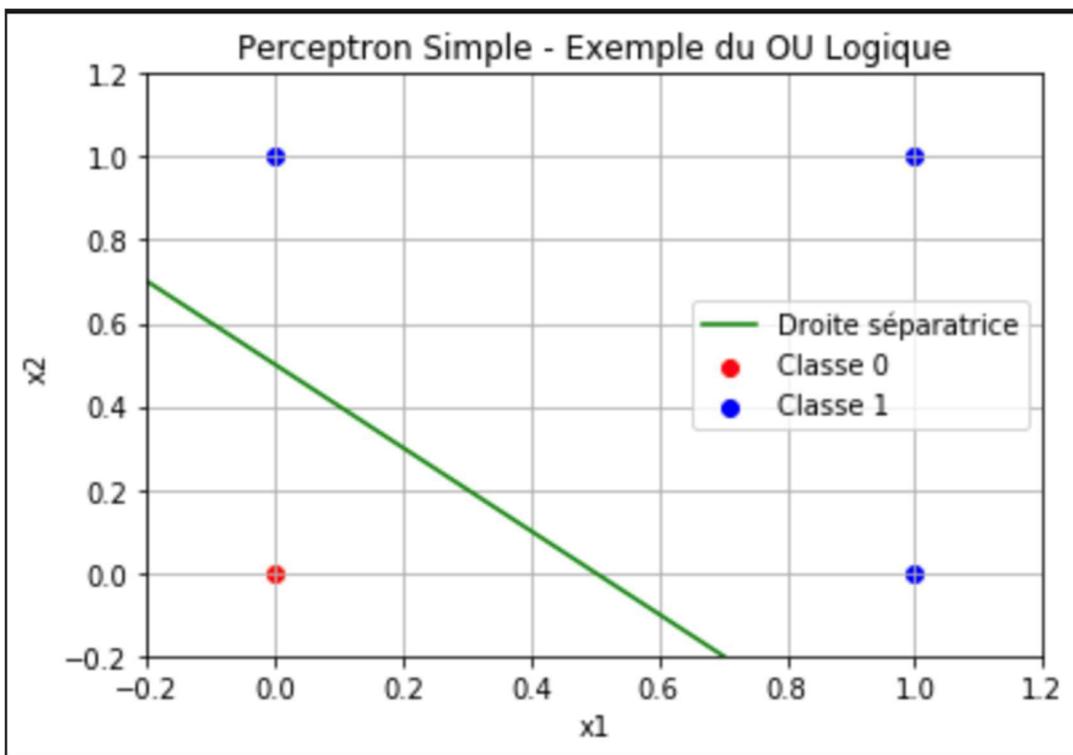
- La variable w contient les poids synaptiques du neurone. C'est un vecteur à 3 lignes. La première ligne correspond au seuil.
- La variable x contient l'entrée du réseau de neurones. C'est un vecteur à 2 lignes.
- La variable $active$ indique la fonction d'activation utilisée.

Le résultat : La variable y est un scalaire correspondant à la sortie du neurone.

OR		
in ₁	in ₂	out
0	0	0
0	1	1
1	0	1
1	1	1

Afin de vérifier que le perceptron fonctionne correctement, nous avons testé toutes les combinaisons d'entrées possibles pour le OU logique : (0, 0), (0, 1), (1, 0), et (1, 1).

Les sorties calculées correspondent aux attentes, confirmant le bon fonctionnement du perceptron pour le OU logique. Cette approche garantit que le perceptron est correctement configuré pour résoudre ce problème linéairement séparable.



La visualisation ci-dessous montre la frontière de décision du perceptron pour le OU logique, ainsi que les points de données.

- **Points de Données :** Les points (0,0), (0,1), (1,0), et (1,1) sont représentés. Le point (0,0) (en rouge) correspond à la sortie -1, tandis que les autres points (en vert) correspondent à la sortie 1.
- **Frontière de Décision :** La ligne bleue représente la frontière de décision du perceptron. Elle est obtenue en réorganisant l'équation du perceptron pour tracer la séparation entre les classes -1 et 1. Cette frontière indique que le perceptron attribuera une sortie 1 pour toutes les entrées se trouvant au-dessus de la ligne, conformément au OU logique.

La séparation montre que le perceptron a correctement appris la logique du OU. En résumé, le perceptron est capable de différencier les classes de manière linéaire, répondant ainsi aux attentes du modèle.

Apprentissage Widrow : ensemble Test 1 / Test 2

Ce programme retourne le vecteur de poids w obtenu par apprentissage selon la règle d'apprentissage utilisant la descente du gradient.

Les paramètres :

- La variable x contient l'ensemble d'apprentissage. C'est une matrice à 2 lignes et n colonnes.
- La variable yd[i] indique la réponse désirée pour chaque élément x[:,i].
- yd est un vecteur de 1 ligne et n colonnes de valeurs +1 ou -1 (classification à 2 classes).
- Epoch : le nombre d'itérations sur l'ensemble d'apprentissage.
- Batch size : le nombre d'individus de l'ensemble d'apprentissage traités avant mise à jour des poids.

Le résultat :

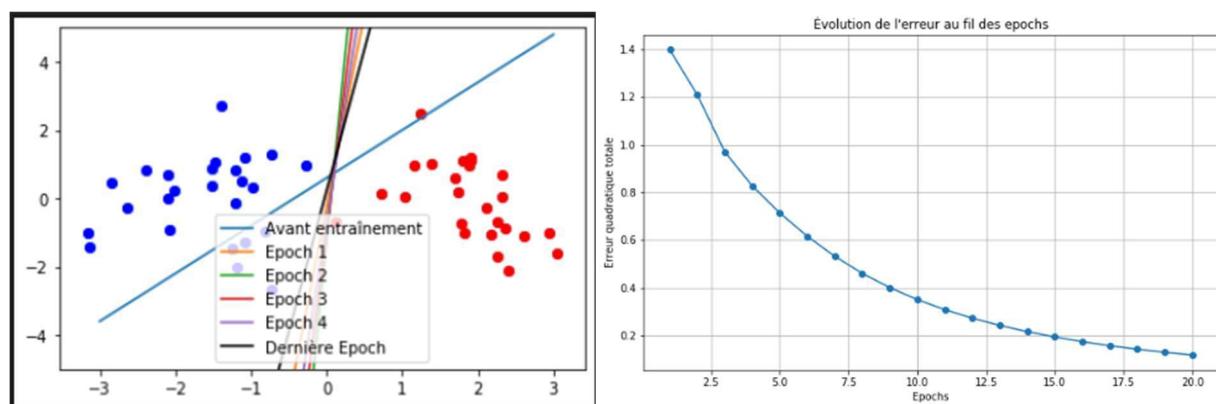
- La variable w contient les poids synaptiques du neurone après apprentissage. C'est un vecteur à 3 lignes. La première ligne correspond au seuil.
- La variable erreur contient l'erreur cumulée calculée pour le passage complet de l'ensemble d'apprentissage à savoir : erreur =

$$\sum_{i=0}^{N-1} (yd(i) - y(i))^2$$

La variable erreur est donc un vecteur de taille égale au nombre d'itération.

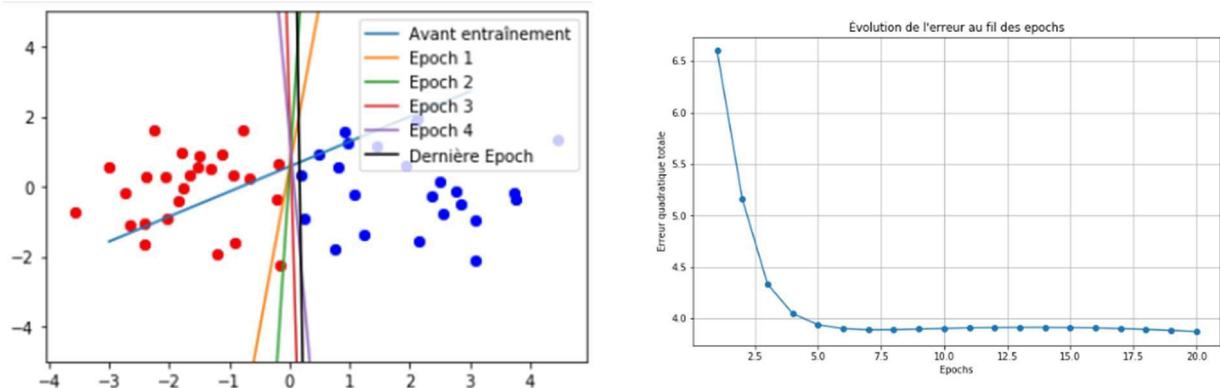
La droite séparatrice et les points d'apprentissage seront affichés à chaque itération (une itération correspond à la présentation de tous les individus de l'ensemble d'apprentissage), ainsi que l'erreur de classification. L'algorithme s'arrêtera dès que l'erreur est nulle, ou au bout de Epoch itérations. Les poids initiaux seront générés de manière aléatoire.

- Pour l'ensemble Test 1 :



On remarque que l'erreur diminue de manière continue pour se rapprocher de 0. Cela se symbolise sur le graphique avec les points où l'on voit la droite de séparation qui s'ajuste au cours des epochs.

- Pour l'ensemble Test 2,



Comme pour le test 1, on remarque que l'erreur diminue de manière continue pour se rapprocher de 0. Cette fois ci, elle semble se rapprocher de 0 plus vite que dans le test précédent. Cela se symbolise encore une fois sur le graphique avec les points où l'on voit la droite de séparation qui s'ajuste au cours des epochs.

Mise en place d'un perceptron Multicouche

Ce programme calcule la sortie d'un perceptron multicouches à 1 neurone sur la couche de sortie et deux neurones sur la couche cachée pour une entrée élément de \mathbb{R}^2 .

Les paramètres :

- La variable $w1$ contient les poids synaptiques des 2 neurones de la couche cachée. C'est une matrice à 3 lignes et 2 colonnes. La première colonne $w1(:,1)$ correspond au 1er neurone de la couche cachée et $w1(:,2)$ correspond au 2ème neurone de la couche cachée.
- La variable $w2$ contient les poids synaptiques du neurone de la couche de sortie. C'est un vecteur à 3 lignes.
- La variable x contient l'entrée du réseau de neurones. C'est un vecteur à 2 lignes.

Le résultat :

- La variable y est un scalaire correspondant à la sortie du neurone.

Lorsque l'on lance la fonction permettant de simuler le perceptron multicouche, on obtient une sortie du réseau égale à 0,91.

Projet Classification

La classification en deep learning est une des applications les plus emblématiques de l'intelligence artificielle moderne. Elle consiste à attribuer des étiquettes ou catégories spécifiques à des données d'entrée, telles que des images, du texte, de l'audio ou des séries temporelles. Cette tâche est au cœur de nombreuses innovations technologiques, allant de la reconnaissance d'objets dans des images à l'analyse de sentiments dans les réseaux sociaux, en passant par la détection de fraudes dans les transactions financières.

Dans le cadre de ce projet, nous avons eu à faire un problème de classification d'images. Pour ce faire, nous disposions d'une base de données de test composée de 10 types d'images (Jungle, Plage, Monuments, Bus, Dinosaures, Eléphants, Fleurs, Chevaux, Montagne et Plats) avec 100 images pour chaque classe. L'objectif est de créer un modèle capable de reconnaître le type d'une nouvelle image inconnue proposé par un utilisateur.

Pour créer ce modèle de classification, nous avons testé 2 approches différentes. La première est une approche basée sur les descripteurs. Un descripteur d'image est une méthode permettant de représenter le contenu visuel d'une image sous une forme numérique. Ces descripteurs extraient des caractéristiques spécifiques de l'image (comme la couleur, la texture ou la forme) afin de permettre des comparaisons ou des analyses. Grâce à ces données, nous allons ensuite pouvoir créer notre réseau de neurones pour classifier les images directement à partir des descripteurs.

La deuxième est approche une approche Deep basée directement sur les images. Dans cette deuxième approche, nous allons construire les descripteurs grâce des couches de convolution. Dans la suite de ce rapport, nous allons vous détailler les méthodes et tests que nous avons mis en place via ces deux approches pour obtenir les modèles les plus performants.

Modèle basée sur les descripteurs

Pour cette première approche, nous avions comme source de données un fichier Excel appelé WangSignatures.xlsx composée de 5 onglets correspondant à 5 descripteurs différents qui sont les suivants :

PHOG (Pyramid of Histograms of Oriented Gradients) : C'est un descripteur qui analyse les formes dans une image en se basant sur les contours et leurs directions. L'image est divisée en plusieurs zones, et les orientations des gradients (les changements d'intensité) dans chaque zone sont résumées dans des histogrammes. Ce descripteur est particulièrement utile pour reconnaître les objets ou comparer des formes dans des images.

JCD (Joint Composite Descriptor) : Le JCD est un descripteur combinant les informations de couleur et de texture pour décrire une image. Il regroupe les données de deux autres descripteurs (CEDD et FCTH) pour produire une signature compacte et rapide à calculer. Il est principalement utilisé pour rechercher des images similaires dans de grandes bases de données.

CEDD (Color and Edge Directivity Descriptor) : CEDD est un descripteur qui mélange les informations sur les couleurs et les contours d'une image. Il découpe l'image en petits morceaux, analyse les couleurs présentes ainsi que la forme des contours, et compile le tout dans un histogramme. Ce descripteur est particulièrement efficace pour analyser les images de manière rapide tout en capturant leur contenu principal.

FCTH (Fuzzy Color and Texture Histogram) : FCTH combine les informations sur les textures et les couleurs d'une image. En utilisant des filtres pour détecter des motifs répétitifs et en associant les couleurs correspondantes, ce descripteur fournit une description visuelle précise. Il est utile pour identifier des images partageant des motifs similaires, même si elles ne sont pas identiques.

Fuzzy Color : Fuzzy Color est un descripteur focalisé sur les couleurs, mais avec une approche plus flexible. Il catégorise les couleurs en grands ensembles (par exemple, rouge clair ou bleu foncé) au lieu d'être très précis, ce qui le rend adapté pour trouver des images avec des teintes similaires même si elles varient légèrement.

Après avoir importé les données des descripteurs dans 5 dataframes différents, nous avons dû créer une liste de labels pour savoir à quelle classe appartient chaque image. Les classes sont regroupées à travers leur numéro qui se trouve dans la première colonne de nos dataframes (par exemple de 200 à 299, ce sont les images Monuments). Nous avons donc créé une fonction qui permet de créer une colonne supplémentaire dans nos dataframe contenant le label de l'image récupéré à partir du numéro de l'image. Après l'application de cette fonction, nous obtenons par exemple le dataframe suivant pour le descripteur PHOG :

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255	label
0	0.jpg	10.0	8.0	7.0	7.0	7.0	7.0	7.0	7.0	6.0	...	3.0	3.0	1.0	3.0	2.0	5.0	5.0	6.0	3.0	Jungle
1	1.jpg	15.0	13.0	12.0	11.0	10.0	10.0	10.0	8.0	9.0	...	3.0	5.0	10.0	12.0	14.0	13.0	12.0	9.0	5.0	Jungle
2	10.jpg	15.0	12.0	12.0	13.0	12.0	13.0	13.0	13.0	13.0	...	4.0	3.0	3.0	5.0	4.0	4.0	8.0	8.0	7.0	Jungle
3	100.jpg	12.0	10.0	6.0	4.0	4.0	3.0	3.0	3.0	3.0	...	2.0	2.0	2.0	3.0	3.0	3.0	4.0	9.0	7.0	Plage
4	101.jpg	15.0	12.0	10.0	9.0	9.0	8.0	8.0	7.0	7.0	...	11.0	12.0	10.0	10.0	10.0	9.0	11.0	10.0	11.0	Plage

Mise en place du modèle

Dans un premier temps, nous avons décidé de créer un réseau de neurones simple et de l'entraîner sur nos 5 descripteurs séparément afin d'obtenir des premiers résultats. Pour ce faire, nous avons réutiliser le modèle donné dans le cours qui comportait 3 couches : une première couche Dense avec 16 neurones, une deuxième couche Dense avec 8 neurones et pour finir une troisième couche Dense avec 10 neurones à savoir le nombre de classe à prédire. Nous avons entraîné notre modèle avec 50 epochs et un batch size à 32.

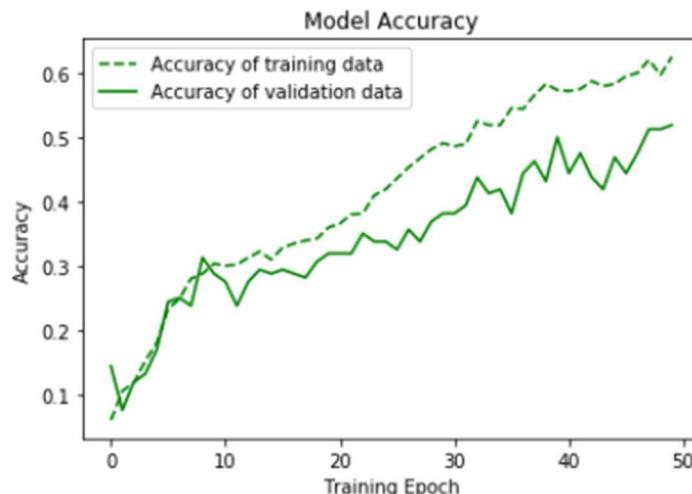
Pour chaque descripteur, nous avons donc en sortie une matrice de confusion, une courbe de l'évolution de la précision en fonction du nombre d'epochs et une courbe de l'évolution de l'erreur en fonction du nombre d'epochs.

PHOG

La matrice de confusion obtenue montre les prédictions du modèle sur le jeu de test, comparées aux véritables étiquettes. Les résultats révèlent que certaines classes, comme Bus, Dinosaures, Fleurs et Monuments, sont relativement bien différencierées, avec un nombre élevé de prédictions correctes sur la diagonale. Cependant, plusieurs classes montrent des confusions importantes (Chevaux, Jungle, Plat).

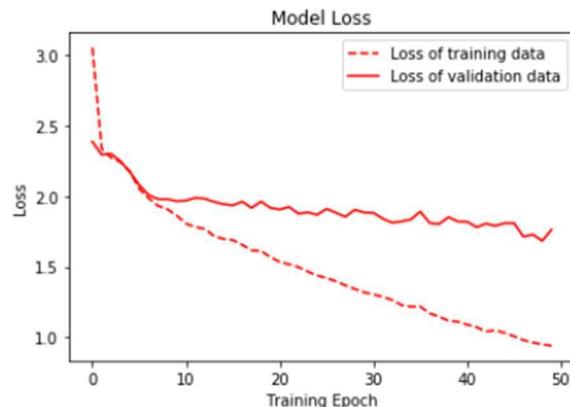
Ces confusions suggèrent que le descripteur PHOG, bien qu'utile pour certaines classes, manque de capacité discriminative pour différencier des catégories visuellement proches. Cela pourrait également indiquer que le modèle n'est pas assez complexe pour capturer les relations entre les caractéristiques.

Matrice de confusion											
Vérités classes	Bus	Chevaux	Dinosaures	Eléphants	Fleurs	Jungle	Montagne	Monuments	Plage		
	Bus	16	0	0	1	0	0	0	3	0	0
	Chevaux	1	4	0	1	6	0	0	2	1	5
	Dinosaures	0	0	14	2	1	0	1	0	2	0
	Eléphants	0	5	1	9	0	1	0	0	3	1
	Fleurs	0	0	1	0	13	1	4	0	0	1
	Jungle	2	2	0	6	1	2	3	1	2	1
	Montagne	0	2	1	1	0	1	7	0	3	5
	Monuments	1	0	1	3	0	1	0	11	3	0
	Plage	1	0	2	2	0	1	4	0	10	0
	Plats	0	1	1	1	3	2	6	2	0	4

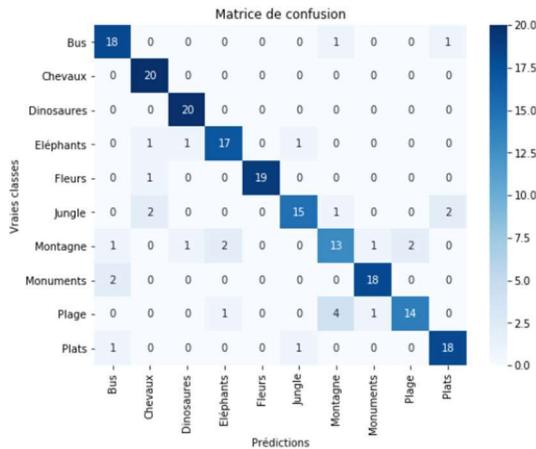


L'évolution de la précision montre que le modèle atteint une précision d'environ 60 % sur les données d'entraînement après 50 époques. Cependant, la précision sur les données de validation reste plus basse, autour de 50-55 %, avec une certaine instabilité au fil des époques. Cette instabilité est un signe que le modèle commence à surapprendre les données d'entraînement sans bien généraliser sur les données qu'il n'a pas vues.

L'évolution de la fonction de perte confirme les observations précédentes. La perte sur les données d'entraînement diminue régulièrement, ce qui montre que le modèle apprend à mieux prédire ces données. Cependant, la perte de validation diminue initialement, puis stagne à un niveau significativement plus élevé que celle de l'entraînement. Ce comportement est typique d'un modèle qui commence à surcharger les données d'entraînement, apprenant des détails spécifiques à ces dernières au détriment de la généralisation.

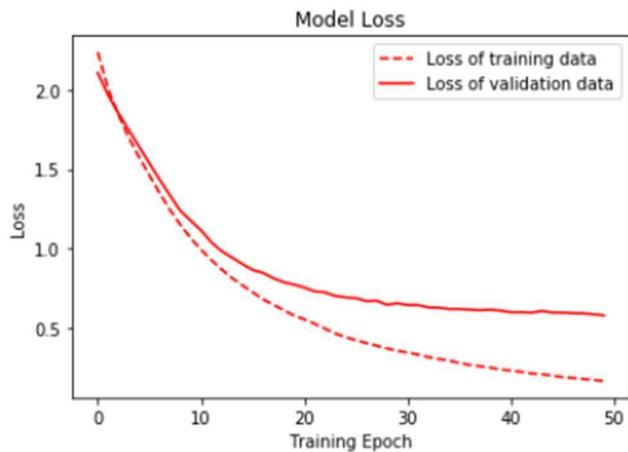
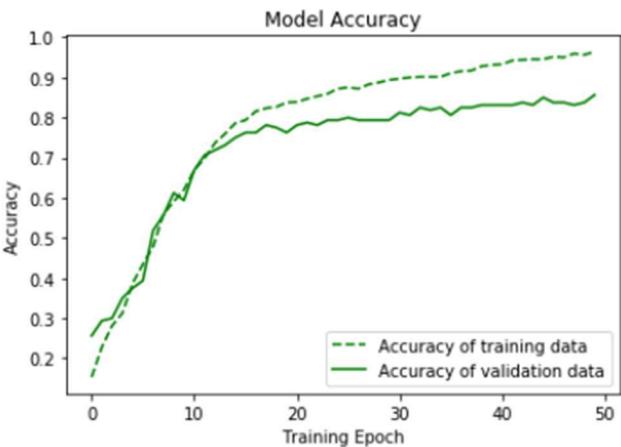


JCD



La matrice de confusion démontre une nette amélioration des prédictions par rapport aux descripteurs PHOG. Plusieurs classes, comme Bus, Fleurs, Monuments et Plats, atteignent une précision quasi-parfaite avec peu de confusions. Cependant, certaines classes continuent de poser des difficultés comme les classes Montagne et Plage. Les classes Chevaux et Dinosaires, quant à elles, sont prédites avec 100 % de précision (aucune confusion).

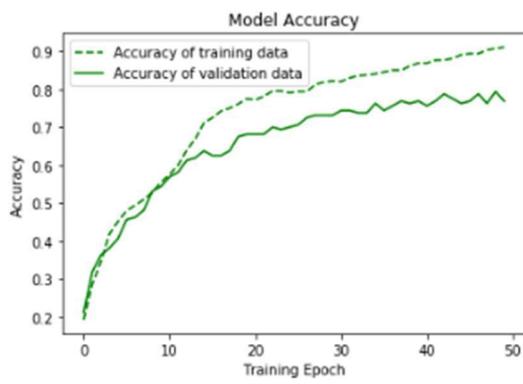
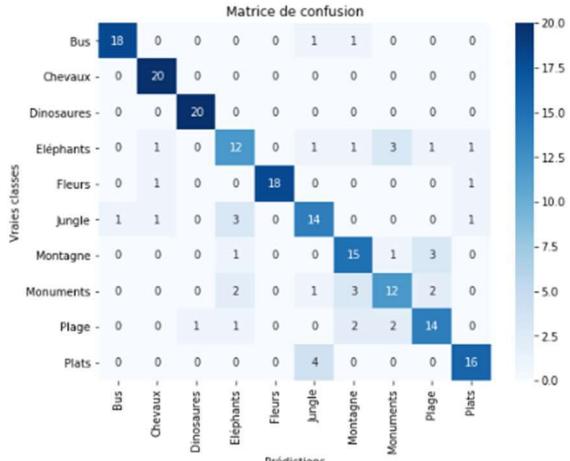
La courbe de précision montre une progression notable avec une précision de validation qui atteint environ 85 % après 50 epochs. La précision d'entraînement, quant à elle, approche 95 %, avec un écart modéré par rapport à la validation. Le descripteur JCD permette au modèle d'apprendre des représentations plus robustes, conduisant à une meilleure généralisation. Cependant, l'écart modéré entre les deux courbes pourrait encore être réduit avec des techniques comme l'augmentation des données ou des ajustements des hyperparamètres.



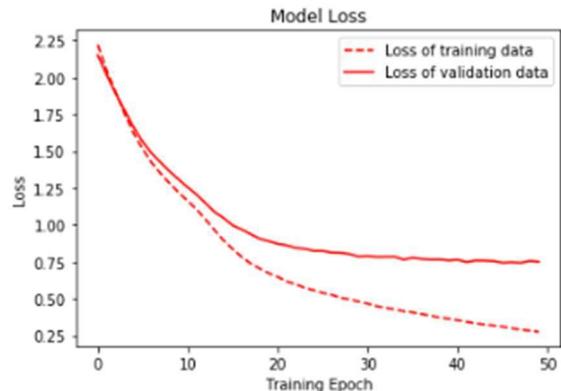
La courbe de perte révèle une diminution continue de la perte d'entraînement et de validation au fil des époques. Une perte finale autour de 0,5 sur la validation est un indicateur clair d'une meilleure performance par rapport au descripteur PHOG. Bien que la perte de validation diminue régulièrement, son écart par rapport à la perte d'entraînement persiste. Cela pourrait être lié à des exemples de validation plus complexes ou à une diversité insuffisante dans les données.

CEDD

Les résultats avec le descripteur CEDD semblent relativement similaires à ceux du descripteur JCD. Sur la matrice de confusion, on remarque que les classes Chevaux et Dinosaures sont parfaitement prédites. Cependant, certaines classes restent moins bien prédites comme la classe Eléphants ou la classe Monuments, qui étaient mieux prédites avec le descripteur précédent.

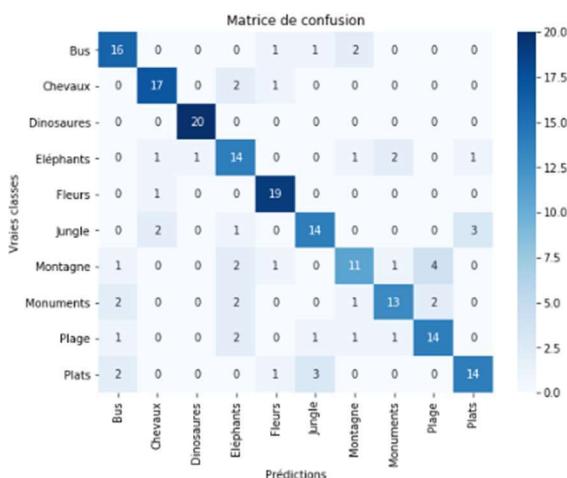


La courbe de précision montre une progression claire, avec une précision de validation qui atteint environ 78 % après 50 epochs, tandis que la précision d'entraînement arrive à 90 %. Cependant, on note encore une fois un léger écart entre les deux courbes qui pourraient être synonyme d'un début de sur-apprentissage.



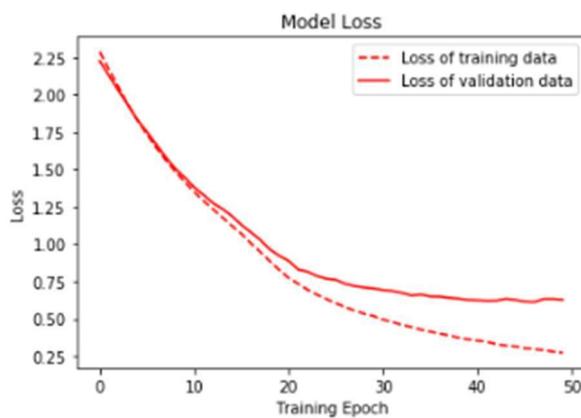
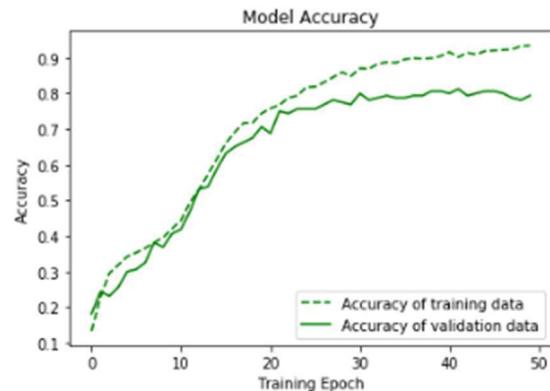
La courbe de perte est ici très similaire à celle du descripteur JCD montré précédemment. On observe une nouvelle fois une diminution continue de la perte d'entraînement et de validation au fil des époques, bien que la courbe de validation semble commencer à stagner. La perte finale atteint une valeur de 0,75 sur la validation ce qui est légèrement inférieur par rapport au descripteur JCD. Bien que la perte de validation diminue régulièrement, son écart par rapport à la perte d'entraînement persiste.

FCTH



Comme pour les deux descripteurs précédents, la classe Dinosaures a été parfaitement prédite par le modèle. On peut commencer à conclure que cette classe est bien distincte des autres, peu importe le descripteur utilisé. On remarque également un très bon score pour la classe Fleurs. Cependant, la classe Montagne n'est ici pas très bien prédite par le modèle avec seulement 11 bonnes prédictions sur 20 images.

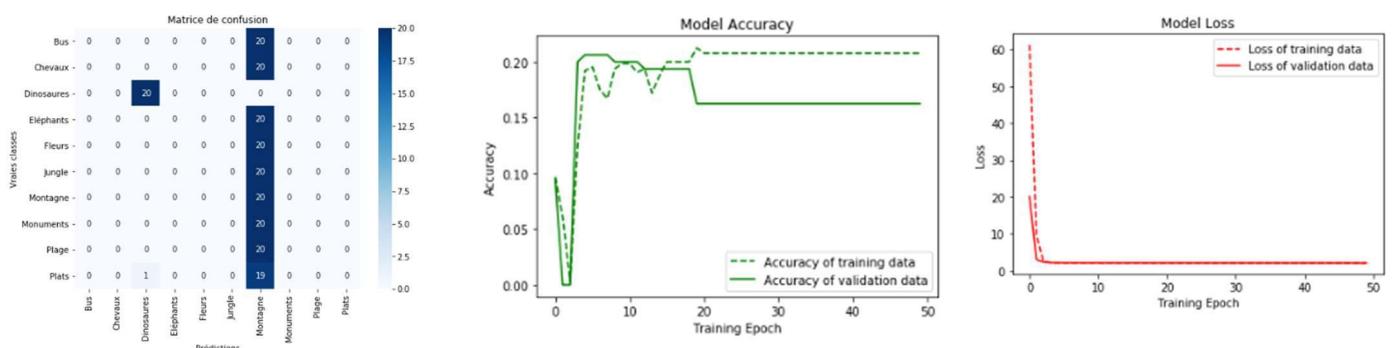
La courbe de précision montre que le modèle atteint une précision de validation d'environ 80 %, avec une précision d'entraînement légèrement supérieure à 90 %. Ces résultats témoignent d'une bonne performance de généralisation. L'écart modéré entre les courbes pourrait être réduit avec une optimisation fine des hyperparamètres ou l'ajout de régularisation.



La courbe de perte montre une diminution continue des pertes pour l'entraînement et la validation. Une telle convergence montre que le modèle est bien entraîné sur les descripteurs FCTH, avec un bon compromis entre complexité et généralisation. La légère stagnation de la perte de validation après l'epoch 30 indique que les performances sont limitées par la richesse des descripteurs ou la complexité du modèle.

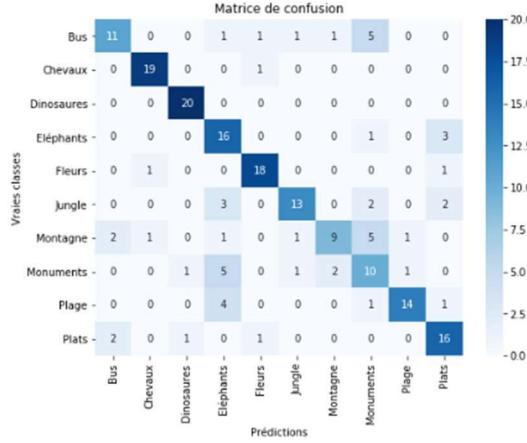
Fuzzy color

Pour le descripteur Fuzzy color, nous avons également testé la classification avec le même modèle que les descripteurs précédents. Cependant, nous avons obtenu de très mauvais résultats que l'on peut observer sur les graphiques ci-dessous.



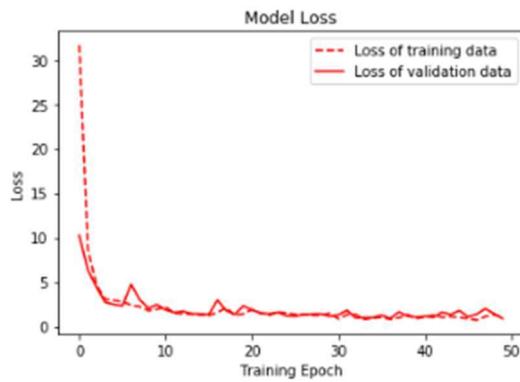
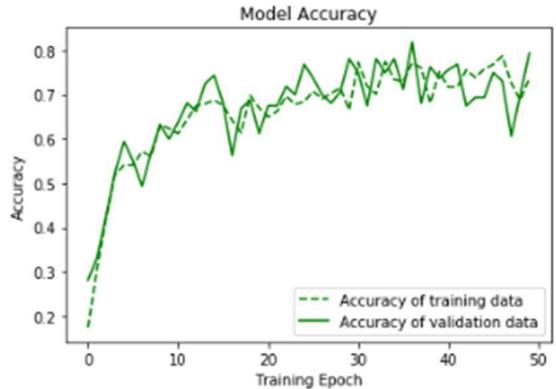
Grâce à la matrice de confusion, on observe que dans quasiment tous les cas, le modèle prédit que l'image fait partie de la classe Montagne. La classe Dinosaires est cependant toujours bien prédite à 100% comme avec les descripteurs précédents. La courbe de précision nous montre que le jeu de données d'entraînement et de validation ne dépasse pas les 20%. Les fluctuations dans les premières époques traduisent une difficulté pour le modèle à trouver un schéma clair dans les données.

Pour pouvoir obtenir des résultats meilleurs avec le descripteur Fuzzy color, nous avons commencé quelques optimisations notamment sur le nombre de neurones par couches. Nous avons testé de changer le nombre de neurones de 16 à 128 pour la première couche et pour la deuxième couche, nous sommes passé de 8 à 64 en gardant le reste des paramètres inchangé.



La matrice de confusion montre une amélioration notable par rapport au modèle précédent, avec une meilleure différenciation des classes. La classe Dinosaures est toujours parfaitement prédite avec 20 échantillons correctement classifiés. Les classes Chevaux, Fleurs et Plats présentent des performances nettement améliorées, avec respectivement 19, 18, et 16 prédictions correctes. Comme pour le descripteur FCTH, la classe Montagne reste la moins bien prédite.

La courbe de précision montre une évolution significative, avec une précision d'entraînement et de validation qui atteint environ 80 %. Les courbes d'entraînement et de validation restent proches, indiquant que le modèle généralise mieux après l'augmentation du nombre de neurones.

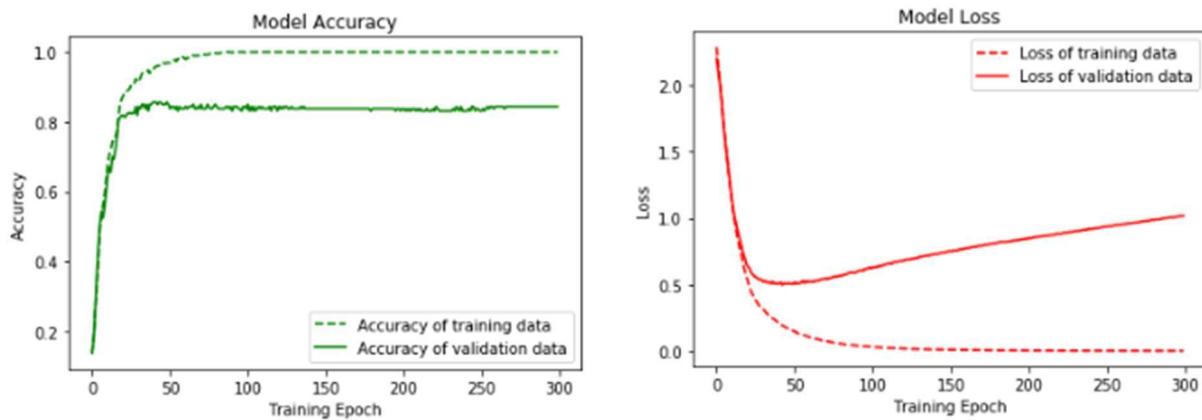


La courbe de perte montre une réduction marquée de la loss sur l'entraînement et la validation. La perte d'entraînement et de validation converge rapidement, atteignant des niveaux faibles après 10 epochs. Les fluctuations de la perte de validation restent présentes, mais elles sont limitées et semblent stabilisées après l'epoch 20.

Avec ces premiers tests sur un modèle relativement simple, nous pouvons tirer des premières conclusions. En effet, nous remarquons que certaines classes sont plus souvent bien prédites que d'autres comme la classe Dinosaures. Chaque descripteur permet au modèle de mieux prédire certaines classes mais cela diffère entre les descripteurs, ce qui signifie que les descripteurs sont complémentaires et qu'il pourrait être intéressant de les combiner pour augmenter les performances du modèle. On note aussi que dans certains cas, on commence à observer du sur-apprentissage même au bout de 50 epochs. Si l'on souhaite augmenter le nombre d'epochs sur l'entraînement du modèle, nous pourrions être amenés à devoir traiter ce sur-apprentissage.

Phase d'optimisation

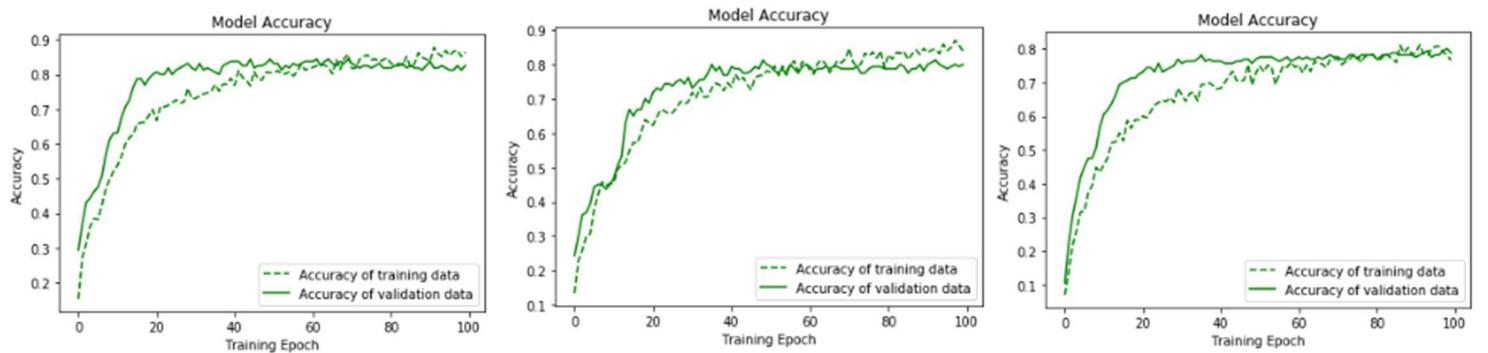
Dans un premier temps, nous avons testé d'augmenter le nombre d'epochs mais comme cela était attendu, nous sommes tombés sur un problème de sur-apprentissage comme on peut l'observer sur le graphique ci-dessous représentant l'évolution de la précision du modèle pour le descripteur JCD mais également sur le graphique montrant la courbe de perte en fonction des epochs pour 300 epochs.



On se rend également compte qu'au bout d'un certains temps, le modèle n'apprend plus et qu'il n'est donc pas nécessaire de fixer un nombre d'epochs aussi importanter, donc pour la suite de nos tests, nous fixerons la variable epochs à 100.

Pour réduire le sur-apprentissage observé dans les cas précédent, nous avons rajouté du dropout sur la première couche dense composée de 16 neurones. Une couche de dropout est une technique de régularisation souvent utilisée dans les réseaux de neurones pour réduire le surapprentissage. Elle consiste à désactiver aléatoirement un certain pourcentage de neurones dans une couche donnée lors de l'entraînement, ce qui force le réseau à ne pas dépendre trop fortement de certains neurones spécifiques.

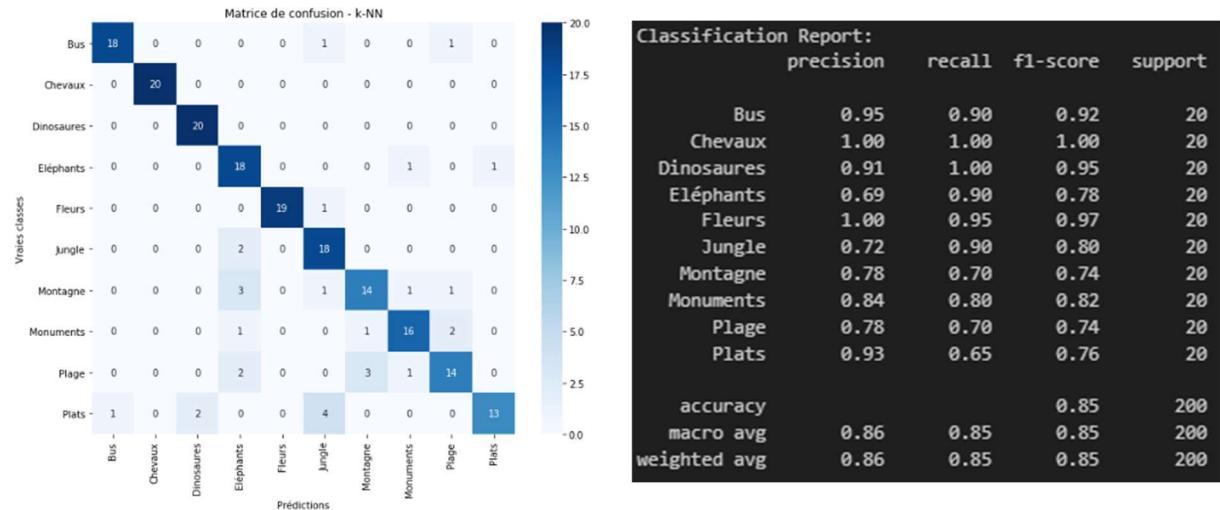
Avec l'ajout de dropout dans notre modèle, nous obtenons de meilleurs résultats avec surtout un surapprentissage qui a été limité. On peut le remarquer sur les graphiques suivants montrant l'évolution de la précision sur l'ensemble d'entraînement et l'ensemble de validation des descripteurs JCD, CEDD et FCTH où les courbes semblent se suivre.



Comparaison avec l'algorithme KPPV

Pour comparer nos résultats, nous avons testé d'utiliser l'algorithme de K plus proches voisins sur nos données des descripteurs qui est une méthode d'apprentissage supervisé utilisée pour les tâches de classification. Il repose sur l'idée que des points proches dans l'espace des caractéristiques appartiennent généralement à la même classe.

En effectuant les tests avec cet algorithme, nous obtenons des résultats qui sont relativement proche de ceux obtenus avec le réseau de neurones. Par exemple, le descripteur JCD obtient une précision de 85%. Il a réussi à classer toutes les images des classes Chevaux et Dinosaures correctement avec également de nombreuses bonnes prédictions sur les classes Bus, Eléphants, Fleurs et Jungle.



Finalement, pour les 5 descripteurs différents, les résultats semblent quasiment identiques à ceux obtenus grâce au réseau de neurones.

Perspectives d'améliorations

Bien que nous ayons réglé ce problème de sur-apprentissage, la précision du modèle n'est pas parfaite et pour être amélioré. En effet, on a pu observer que certains descripteurs sont plus efficaces pour prédire certaines classes comparées à d'autres, il pourrait donc être intéressant de voir les résultats que l'on obtiendrait si l'on fusionnait certains descripteurs notamment JCD, CEDD et FCTH qui semblent être les descripteurs donnant les meilleurs résultats.

Également, il existe un grand nombre d'hyperparamètres que nous aurions pu essayer de modifier comme le nombre d'epochs, le batch size, le nombre de couches ou encore le nombre de neurones par couche. Nous aurions également pu essayer de rajouter de la batch normalisation qui est utilisée pour normaliser les activations d'une couche dans un réseau neuronal.

Modèle basée sur les images

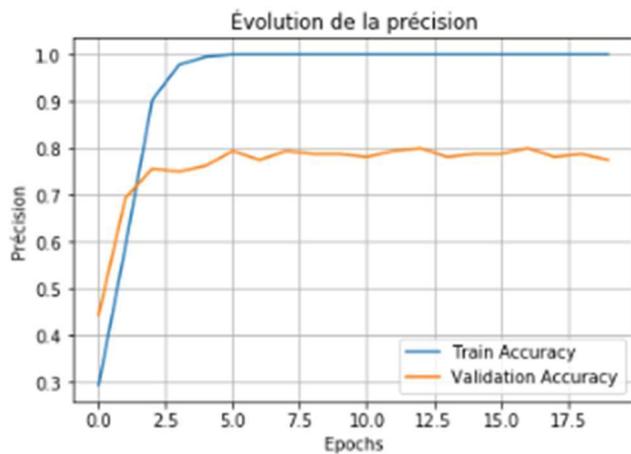
Mise en place du modèle

Le deep learning, basé sur les réseaux de neurones profonds, s'appuie sur des architectures complexes telles que les réseaux convolutionnels. Ces modèles ont la capacité de capturer des caractéristiques hiérarchiques des images, allant des bords et textures aux motifs plus abstraits et spécifiques à une classe. Contrairement aux approches traditionnelles nécessitant une extraction manuelle des caractéristiques, les CNN apprennent directement à partir des données brutes, optimisant ainsi le processus de classification. La différence avec l'approche précédente est que l'on ne fournit plus les valeurs numériques des descripteurs au modèle mais c'est le modèle lui-même qui va calculer ces descripteurs à partir des images brutes directement qui seront l'entrée de notre modèle.

Dans un premier temps, nous sommes partis sur une structure de modèle simple qui est la suivante.

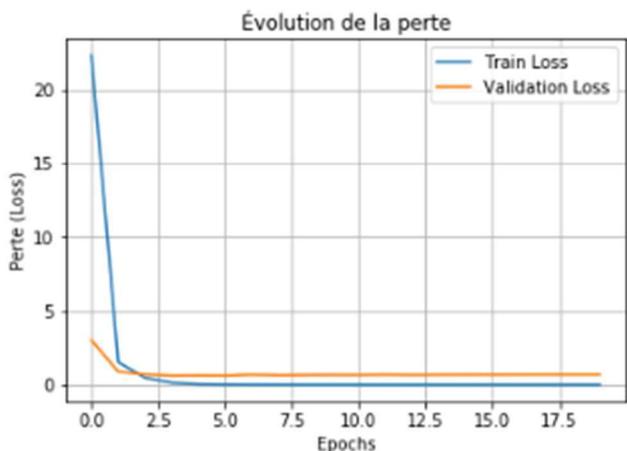
```
# Définition du modèle CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(img_height, img_width, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))
```

En entraînant notre modèle sur 20 epochs, on obtient une précision de 81% qui est pour l'instant similaire à ce que l'on a pu obtenir dans la partie précédente.



En observant l'évolution de la perte, nous pouvons faire le même constat qu'avec la précision. La courbe descend très rapidement mais stagne très vite. On note également un écart entre la courbe d'entraînement et celle de validation ce qui est un autre signe de surapprentissage.

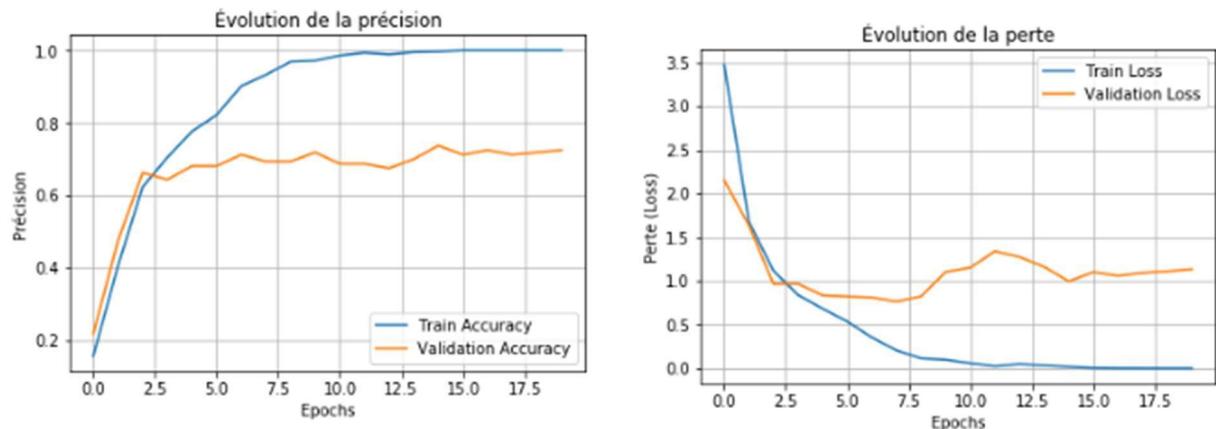
Concernant l'évolution de la précision, on observe que les deux courbes arrivent très rapidement à un niveau de stagnation aux alentours de 5 epochs. On remarque également une différence entre les deux courbes synonymes de sur-apprentissage. Le modèle apprend bien sur les données d'entraînement mais n'arrive pas à généraliser sur les données de tests.



Dans un second temps, nous avons testé une structure plus complexe pour essayer de gommer ce sur-apprentissage avec du Dropout et également d'augmenter la précision de la classification.

```
# Définition du modèle CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Cependant, les résultats n'évoluent pas beaucoup. On observe toujours un sur-apprentissage même s'il est légèrement diminué et la précision ne s'est pas améliorée.



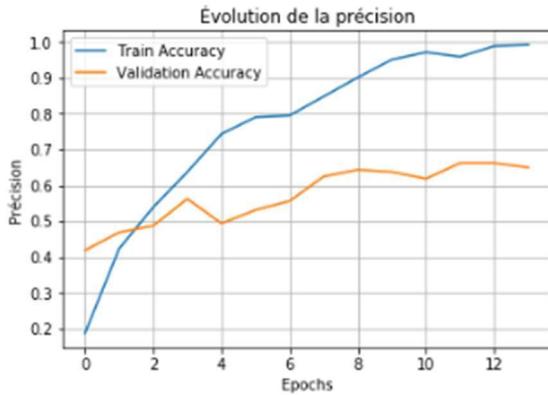
Data augmentation

Afin d'essayer d'améliorer un peu plus notre réseau de neurones, nous avons rajouté une étape de data augmentation. La data augmentation est une technique qui consiste à générer des variations artificielles d'un jeu de données existant. Elle est largement utilisée pour améliorer les performances des modèles, en particulier lorsque le dataset est limité. Les réseaux de neurones convolutifs nécessitent souvent de grandes quantités de données pour éviter le surapprentissage et pour généraliser correctement. Cependant, dans de nombreux cas, collecter et annoter des données à grande échelle peut être coûteux ou impossible. La data augmentation résout ce problème en augmentant artificiellement la diversité des données disponibles par des méthodes de rotation, de translation, de zoom ou encore de renversement horizontal ou vertical.

Dans notre cas, nous avons ajouté les modifications suivantes sur les images :

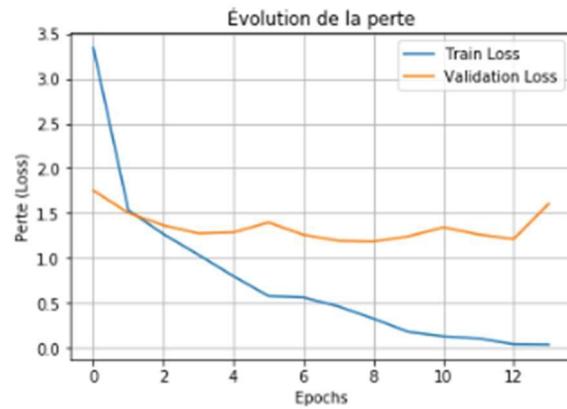
- Pivot autour de leur centre d'un angle compris entre -15° et +15°
- Déplacement horizontal jusqu'à 20% de leur largeur

- Déplacement vertical jusqu'à 20% de leur hauteur
- Déformation en inclinant un axe
- Agrandissement ou réduction jusqu'à 10%
- Retournement horizontal



Même constat pour la courbe de perte où l'on observe un écart entre l'entraînement et la validation.

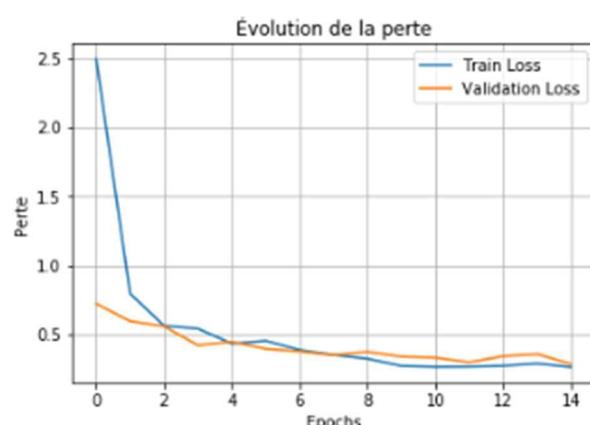
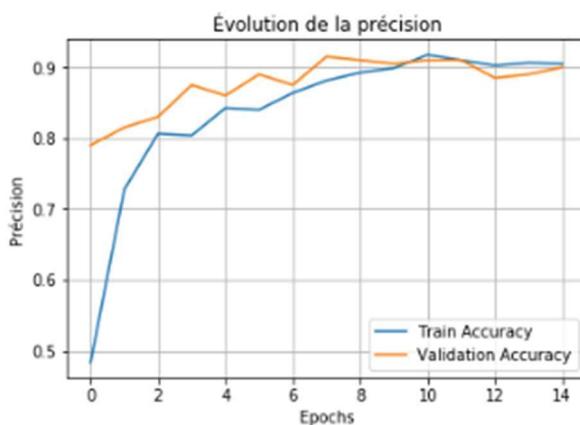
Après avoir fait tourner le modèle en ayant ajouté la data augmentation, on ne note pas d'amélioration notable en ce qui concerne la précision. Il y a toujours un phénomène de sur-apprentissage.



Transfer Learning

Le Transfer Learning est une méthode puissante en apprentissage automatique. Il s'agit d'utiliser un modèle pré-entraîné sur un vaste ensemble de données (comme ImageNet) pour résoudre un nouveau problème, généralement avec un dataset plus petit. L'idée centrale est que les premières couches d'un réseau convolutif apprennent des caractéristiques générales (comme des contours ou des motifs) qui sont transférables à de nombreux problèmes visuels, tandis que les couches supérieures se spécialisent dans une tâche spécifique. Dans notre cas, nous allons utiliser le modèle VGG16 qui a été pré-entraîné sur le dataset ImageNet qui est un dataset composé de plusieurs millions d'images réparties sur plusieurs milliers classes.

Les résultats obtenus grâce au transfer learning sont nettement meilleurs que précédemment.



On observe autant sur l'évolution de la précision que sur l'évolution de la perte qu'il ne semble pas y avoir de sur-apprentissage comparé aux méthodes précédentes. On obtient une précision aux alentours de 90% qui pourrait potentiellement être améliorée en changeant les couches que l'on réentraîne ou en augmentant le nombre d'epochs. Cependant, l'augmentation du nombre d'epochs rendrait le traitement du modèle assez long (déjà 50 minutes pour 15 epochs).

Pour conclure, il paraît logique que le transfer learning donne la meilleure classification car ce modèle a été entraîné sur une base de données très volumineuses qui permet bien distinguer chaque classe, plus qu'en utilisant seulement les données mis à notre disposition pour entraîner notre modèle.