



MASTER 2 : INFORMATIQUE
GÉNIE LOGICIEL
2023

Machine Learning 2
Projet

Etudiants :
CRISTA Allan (21706116)
EFTIMOV David (21914321)
MBELO NDRIAMANAMPY Sandratra (22216525)
ROUQUAIROL Lucas (21608911)

14 décembre 2023

Table des matières

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 2 | Visualisation des données | 3 |
| 3 | Modèle de Base (Baseline) | 4 |
| 3.1 | Modèle baseline Tigre | 5 |
| 3.2 | Modèle baseline Éléphant | 5 |
| 3.3 | Modèle baseline Renard | 6 |
| 3.4 | Évaluation et comparaison des performances | 6 |
| 4 | Amélioration des Modèles | 7 |
| 4.1 | Modele Tiger | 7 |
| 4.2 | Modele Éléphant | 9 |
| 4.3 | Modele Renard | 10 |
| 4.4 | Comparaison différents modèles | 11 |
| 4.5 | Analyse des résultats et conclusion partielle | 12 |
| 5 | Génération de Nouvelles Données | 13 |
| 5.1 | Expérimentation avec Image Data Generator | 13 |
| 5.1.1 | Augmentation avant k-fold | 14 |
| 5.1.2 | Augmentation à chaque fold | 14 |
| 5.1.3 | Augmentation sans k-fold | 15 |
| 5.1.4 | Analyse des résultats | 15 |
| 5.2 | Évaluation sur les modèles et potentielles améliorations | 16 |
| 5.2.1 | Modèle Tigre | 16 |
| 5.2.2 | Modèle Éléphant | 18 |
| 5.2.3 | Modèle Renard | 19 |
| 5.3 | Analyse des résultats et conclusion partielle | 19 |
| 6 | Transfer Learning | 20 |
| 6.1 | Modele Tigre | 21 |
| 6.2 | Modele Éléphant | 22 |
| 6.3 | Modele Renard | 23 |
| 6.4 | Évaluation et comparaison des performances | 23 |
| 7 | Génération d'images avec GAN | 24 |
| 7.1 | Génération d'images de renard | 24 |
| 7.1.1 | Définition du générateur | 24 |
| 7.1.2 | Définition du discriminateur | 24 |
| 7.1.3 | Résultats et Ajustements | 24 |
| 7.1.4 | Prédiction avec un autre modèle | 26 |
| 8 | Analyse des Sorties CNN | 28 |
| 9 | Partie Optionnelle : Autoencodeurs | 29 |
| 9.1 | Coloration d'images | 29 |
| 9.2 | Analyse des résultats | 29 |

1 Introduction

Ce rapport présente les résultats et les analyses d'un projet mené dans le cadre du cours HAI923I. L'objectif principal de ce projet est de développer et d'optimiser des modèles de classification d'images, en utilisant des techniques avancées telles que les réseaux de neurones convolutifs (CNN), le transfer learning, et la génération de nouvelles données via des GANs (Generative Adversarial Networks). Notre projet repose sur trois jeux de données distincts, chacun se concentrant sur une classe d'animaux spécifique : les tigres, les renards et les éléphants par rapport à d'autres animaux.

2 Visualisation des données

Avant de commencer le modèle baseline, nous allons utiliser la réduction de dimension afin de déterminer de potentiels outliers. Voici le résultat des visualisations avec T-SNE.

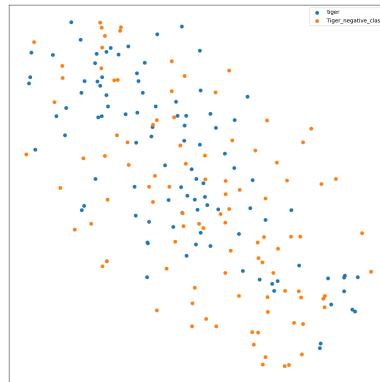


FIGURE 1 – T-SNE entre tigre et autres

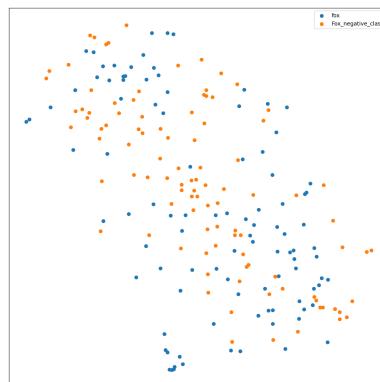


FIGURE 2 – T-SNE entre renard et autres

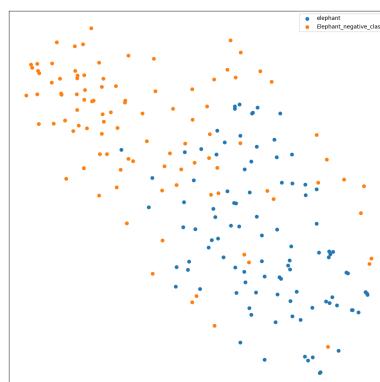


FIGURE 3 – T-SNE entre éléphant et autres

Grâce à ses visualisations, on s'aperçoit que les images d'éléphants sont plus différentes que les images de renards ou de tigres. Ce qui paraît assez logique.

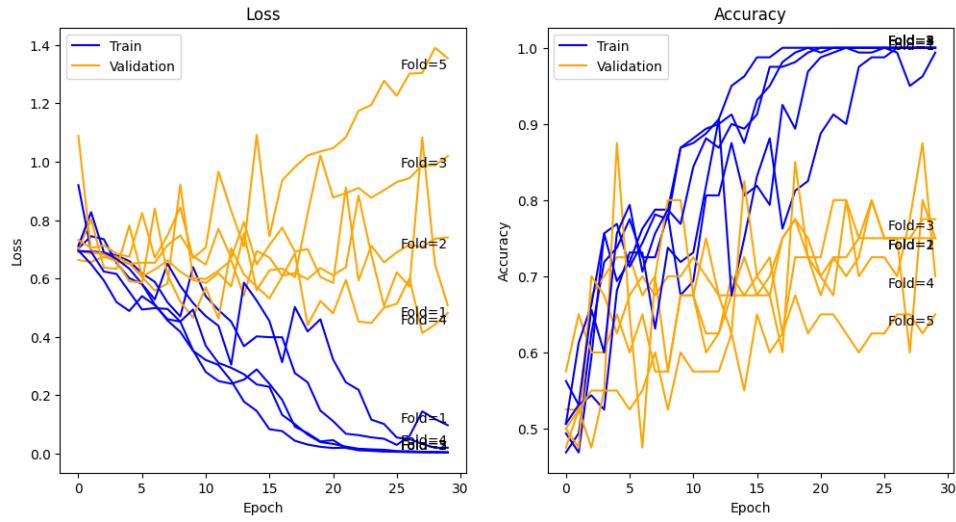
3 Modèle de Base (Baseline)

Pour notre modèle baseline, nous avons choisis simplement une couche de convolution et un couche de max pooling. De cette manière, nous allons appliquer ce modèle sur chacun de nos jeux de données, et observer de potentielles améliorations à apporter.

| Layer (type) | output Shape | Param # |
|---------------------------------------|----------------------|----------|
| <hr/> | | |
| Conv2D_1 (Conv2D) | (None, 122, 122, 32) | 896 |
| Maxpooling2D_1 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| flatten (Flatten) | (None, 119072) | 0 |
| dense (Dense) | (None, 100) | 11907300 |
| dense_1 (Dense) | (None, 1) | 101 |
| <hr/> | | |
| Total params: 11908297 (45.43 MB) | | |
| Trainable params: 11908297 (45.43 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

FIGURE 4 – Modèle de Base

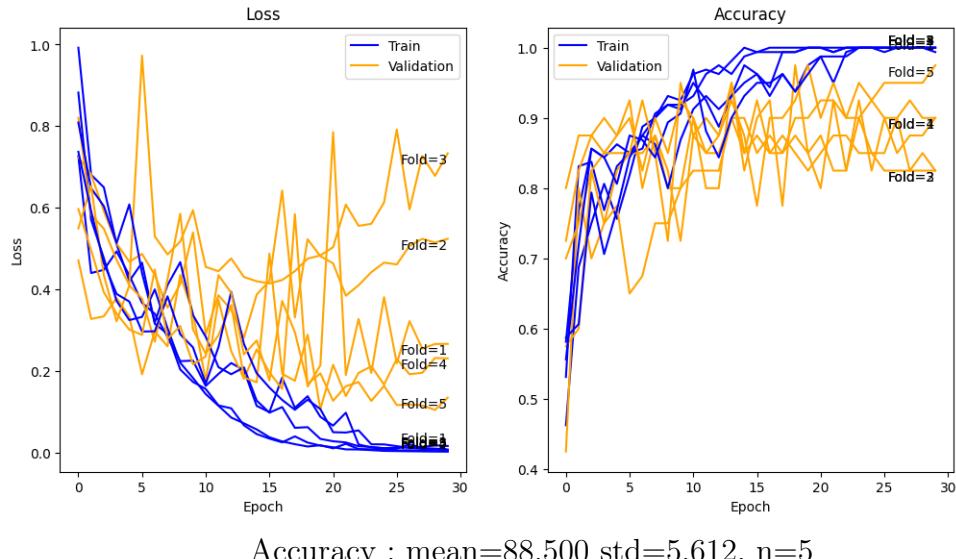
3.1 Modèle baseline Tigre



Accuracy : mean=72.500 std=4.472, n=5

FIGURE 5 – Modèle Tigre

3.2 Modèle baseline Éléphant



Accuracy : mean=88.500 std=5.612, n=5

FIGURE 6 – Modèle Éléphant

3.3 Modèle baseline Renard

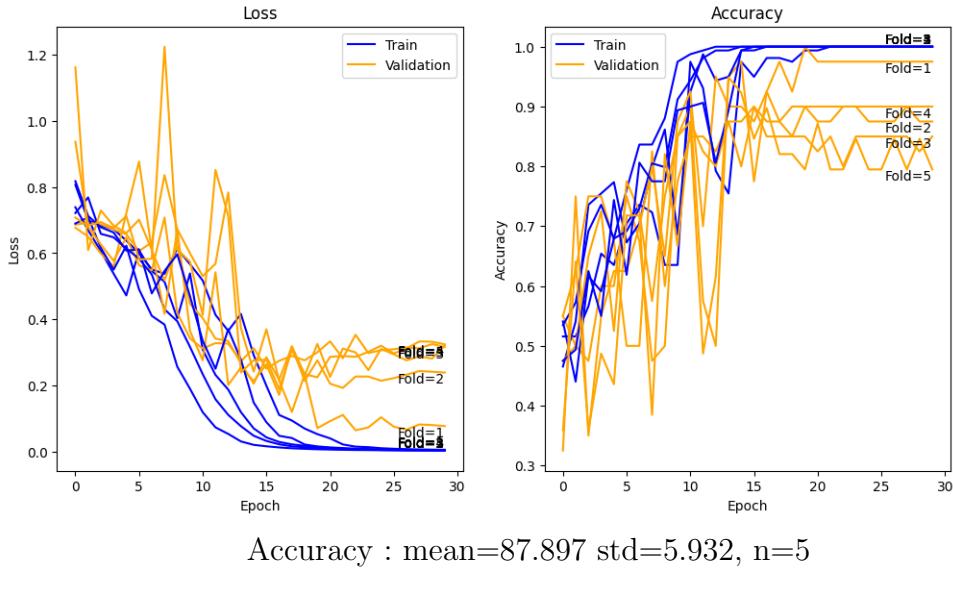


FIGURE 7 – Modèle Renard

3.4 Évaluation et comparaison des performances

Après avoir testé notre jeu de données avec le modèle de base, nous avons constaté que le modèle présente un sur-apprentissage sur tous nos types de données. Il s'avère plus performant avec les données sur les renards, a des performances moyennes avec les données sur les éléphants, mais ne semble pas convenir aux données sur les tigres. Nous devons donc améliorer les performances de ce modèle pour les données sur les tigres.

Recommandations d'Amélioration :

- Introduire de la régularisation ou utiliser la technique de dropout.
- Utiliser l'augmentation de données pour améliorer la généralisation.

4 Amélioration des Modèles

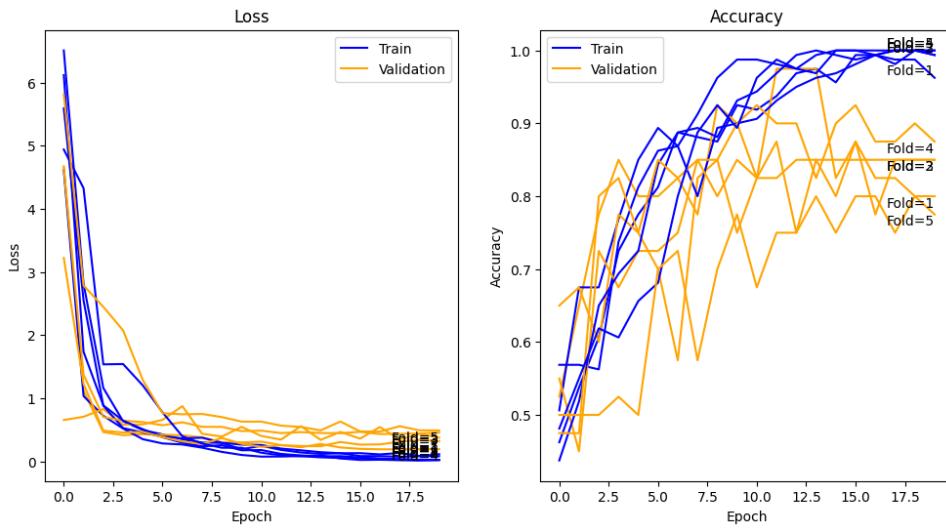
4.1 Modèle Tiger

L'analyse de notre modèle Baseline sur les données de tigres a révélé un surajustement. Des modifications clés ont été apportées pour renforcer la généralisation et corriger le surajustement :

- **Ajout d'une Couche de Pooling** : Une couche de pooling avec un pool size de (1, 1) a été ajoutée avant le dropout. Cela a pour effet de réduire les dimensions des features, tout en conservant les informations essentielles.
- **Couches de Dropout** : Intégration de deux couches de dropout, une après la couche de pooling et une avant la dernière couche dense, pour limiter la corrélation entre les neurones et réduire l'overfitting.
- **Remplacement de l'Optimiseur** : L'optimiseur Adam a remplacé l'optimiseur initial, offrant une meilleure stabilité et efficacité de l'apprentissage grâce à son ajustement automatique du taux d'apprentissage.

| Layer (type) | Output Shape | Param # |
|---------------------------------------|----------------------|----------|
| <hr/> | | |
| Conv2D_1 (Conv2D) | (None, 123, 123, 32) | 416 |
| Maxpooling2D_1 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| Maxpooling2D_2 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| dropout (Dropout) | (None, 61, 61, 32) | 0 |
| flatten (Flatten) | (None, 119072) | 0 |
| dense_36 (Dense) | (None, 100) | 11907300 |
| dropout_1 (Dropout) | (None, 100) | 0 |
| dense_37 (Dense) | (None, 1) | 101 |
| <hr/> | | |
| Total params: 11907817 (45.42 MB) | | |
| Trainable params: 11907817 (45.42 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

FIGURE 8 – Modèle Tigre Amélioré



Accuracy : mean=83.000 std=3.674, n=5

FIGURE 9 – Modèle Tigre Amélioré

Ces ajustements ont conduit à une augmentation significative de la précision, passant de 72.500% à 83%, démontrant ainsi une réduction effective de l'overfitting et une amélioration de la capacité de généralisation du modèle. L'écart type aussi a aussi connu une légère diminution

4.2 Modèle Éléphant

Comme pour les tigres, nous allons ajouter un dropout afin régulariser et nous allons essayer de changer la learning rate pour lisser les courbes et mieux comprendre le comportement du modèle.

| Layer (type) | Output Shape | Param # |
|---------------------------------------|----------------------|----------|
| <hr/> | | |
| Conv2D_1 (Conv2D) | (None, 122, 122, 32) | 896 |
| Maxpooling2D_1 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| dropout_60 (Dropout) | (None, 61, 61, 32) | 0 |
| flatten (Flatten) | (None, 119072) | 0 |
| dense_80 (Dense) | (None, 100) | 11907300 |
| dense_81 (Dense) | (None, 1) | 101 |
| <hr/> | | |
| Total params: 11908297 (45.43 MB) | | |
| Trainable params: 11908297 (45.43 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

FIGURE 10 – Modèle Éléphant Amélioré

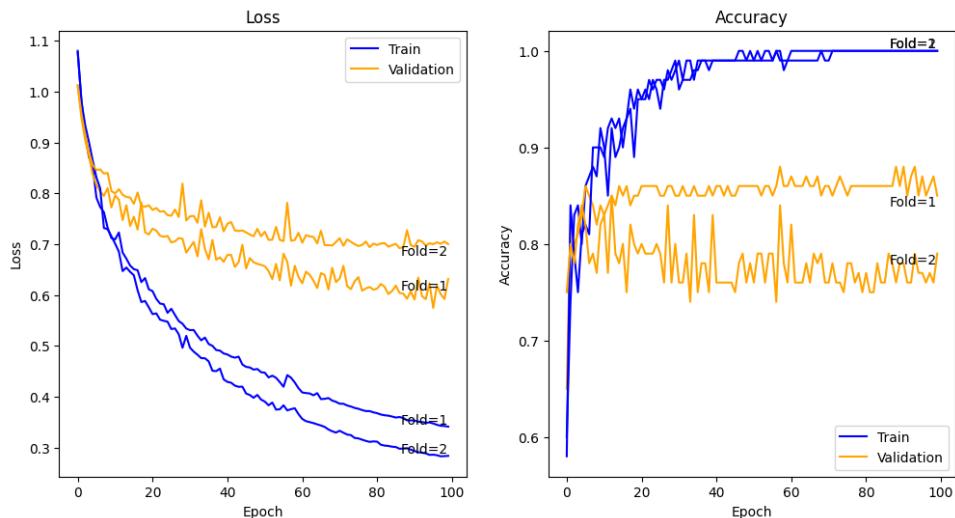


FIGURE 11 – Modèle Éléphant Amélioré

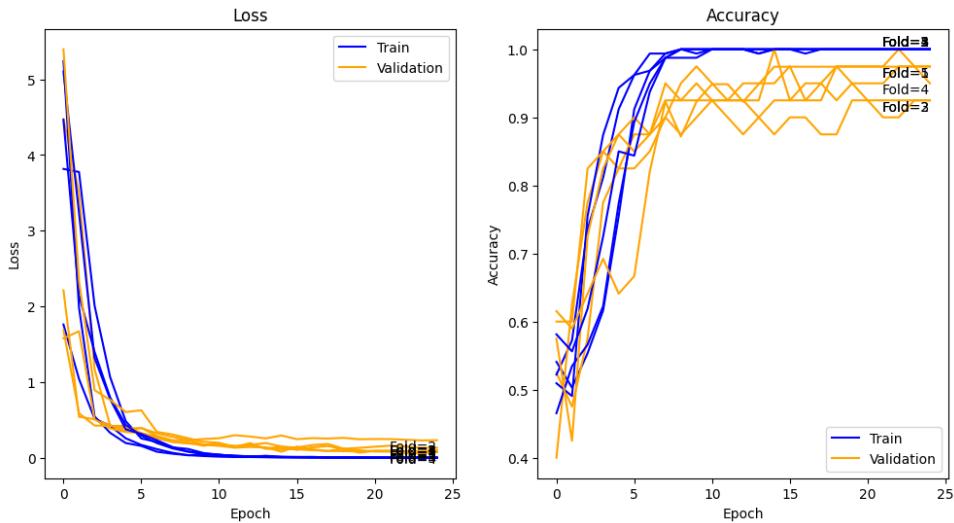
Grâce à la régularisation, on peut observer clairement le sur-apprentissage qui persiste que nous allons corriger plus tard.

4.3 Modèle Renard

Nous avons ajouté un dropout de 0.2 pour résoudre le surapprentissage et nous avons changé l'optimiseur en Adam avec un taux d'apprentissage de 0.0005.

| Layer (type) | Output Shape | Param # |
|---------------------------------------|----------------------|----------|
| <hr/> | | |
| Conv2D_1 (Conv2D) | (None, 122, 122, 32) | 896 |
| Maxpooling2D_1 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| flatten (Flatten) | (None, 119072) | 0 |
| dense_4 (Dense) | (None, 100) | 11907300 |
| dropout_2 (Dropout) | (None, 100) | 0 |
| dense_5 (Dense) | (None, 1) | 101 |
| <hr/> | | |
| Total params: 11908297 (45.43 MB) | | |
| Trainable params: 11908297 (45.43 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

FIGURE 12 – Modèle Renard Amélioré



Accuracy : mean=94.987 std=2.222, n=5

FIGURE 13 – Modèle Renard Amélioré

4.4 Comparaison différents modèles

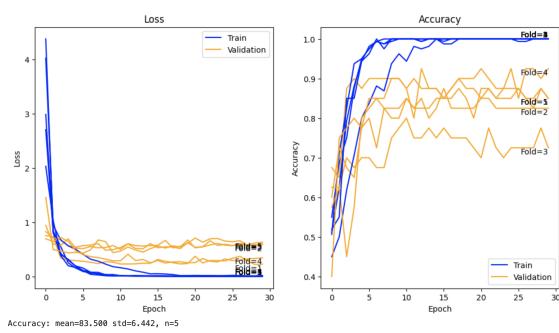


FIGURE 14 – Modèle éléphant sur les tigres

Lors de l'évaluation des modèles pour les éléphants sur les tigres et les renard, nous avons obtenu des taux de précision (accuracy) encourageants. Cependant, il est à noter que le phénomène de surapprentissage (overfitting) est toujours présent, bien que plus prononcé dans les modèles relatifs aux tigres.

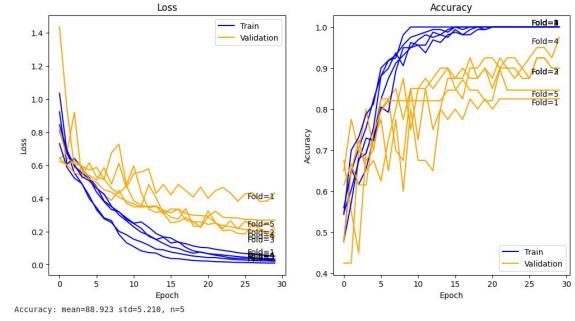


FIGURE 15 – Modèle éléphant sur les renard

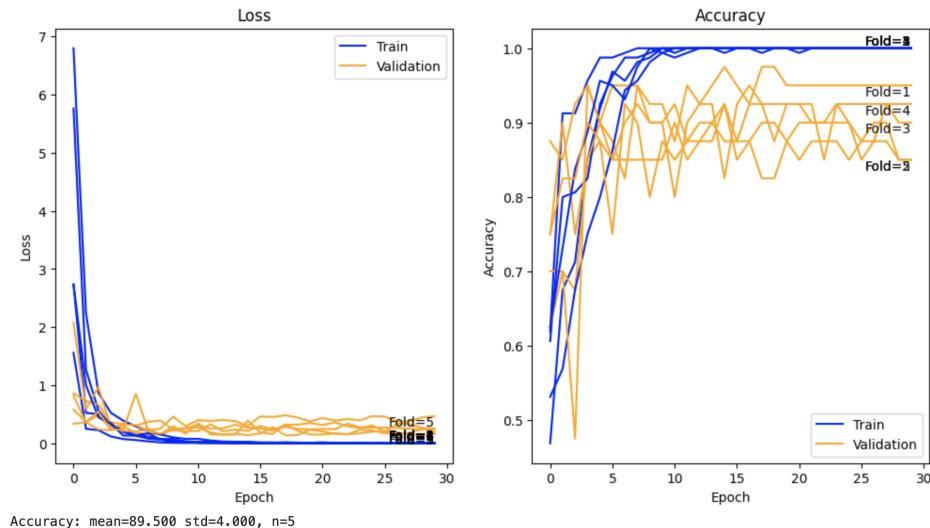


FIGURE 16 – Modèle renard sur les éléphant

Les modèles conçus pour les renards, lorsqu'ils sont testés sur les données concernant les éléphants, affichent de bons résultats. Nous observons également que les performances sont assez similaires entre les deux types de données.

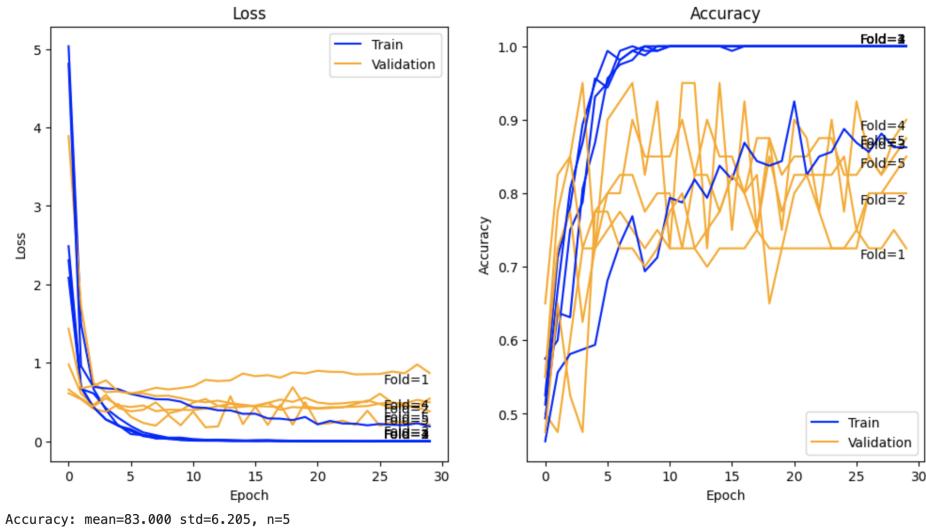


FIGURE 17 – Modèle renard sur les tigres

Nous avons remarqué que le modèle dédié aux renards n'est pas entièrement adapté pour traiter les données relatives aux tigres, malgré sa bonne précision initiale. Cette situation pourrait s'expliquer par le fait que le model baseline étaient déjà satisfaisants pour les données de renards, limitant ainsi les modifications apportées au modèle. Cependant, il est important de souligner que c'étaient les données de renards qui présentaient les performances les moins convaincantes dans notre évaluation initiale.

4.5 Analyse des résultats et conclusion partielle

Nous pouvons voir que nous avons réussi à réduire le sur apprentissage, mais nous n'avons toujours pas une très grande accuracy pour tous les modèles. C'est le maximum que nous pouvons faire sans génération de nouvelles données.

5 Génération de Nouvelles Données

La conclusion générale de nos précédent résultat s'accorde sur le manque de données. Pour palier à cela, nous avons mis en place un générateur de données qui se base sur les données déjà existantes. Voici les paramètres choisis pour la transformations des images :

```
1 ImageDataGenerator(horizontal_flip=True,  
2                         vertical_flip=False,  
3                         rotation_range=10,  
4                         width_shift_range=0.05,  
5                         height_shift_range=0.05,  
6                         fill_mode='wrap',  
7                         zoom_range=[0.8,1.1])
```

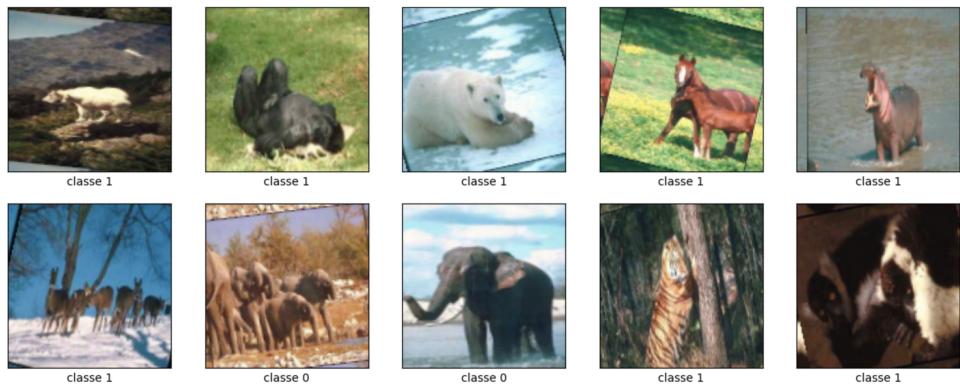


FIGURE 18 – Data augmented examples

5.1 Expérimentation avec Image Data Generator

Avant de présenter les résultats des modèles des différents animaux, nous avons essayé plusieurs méthodes plus ou moins pertinentes :

- Augmentation avant k-fold
- Augmentation à chaque fold
- Augmentation sans k-fold

Tous les résultats suivants sont effectué sur les éléphants avec le modèle amélioré, une régularisation assez forte, 120 époques et un batch size = 32.

5.1.1 Augmentation avant k-fold

On augmente les données globales avant l'entraînement. Le modèle va donc évaluer les tests sur des données qu'il n'a techniquelement jamais vu mais comme la fonction d'augmentation est utilisé avec les données de base, les données testé seront très proche des données de train.

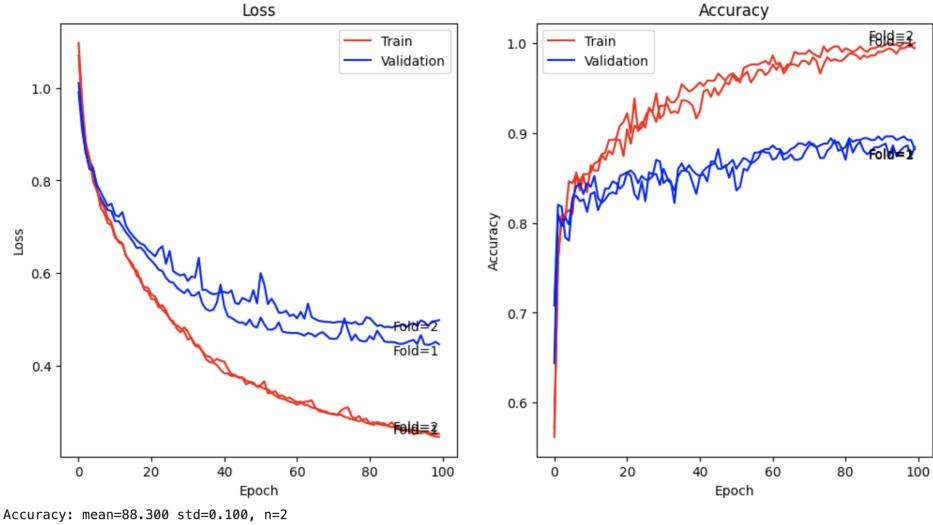


FIGURE 19 – Data augmented examples

5.1.2 Augmentation à chaque fold

On augmente les données des train à chaque folds, cette méthode est assez pertinente pour comparer nos résultat avec les modèles utilisées précédemment, même si techniquelement ce n'est pas vraiment du K-fold.

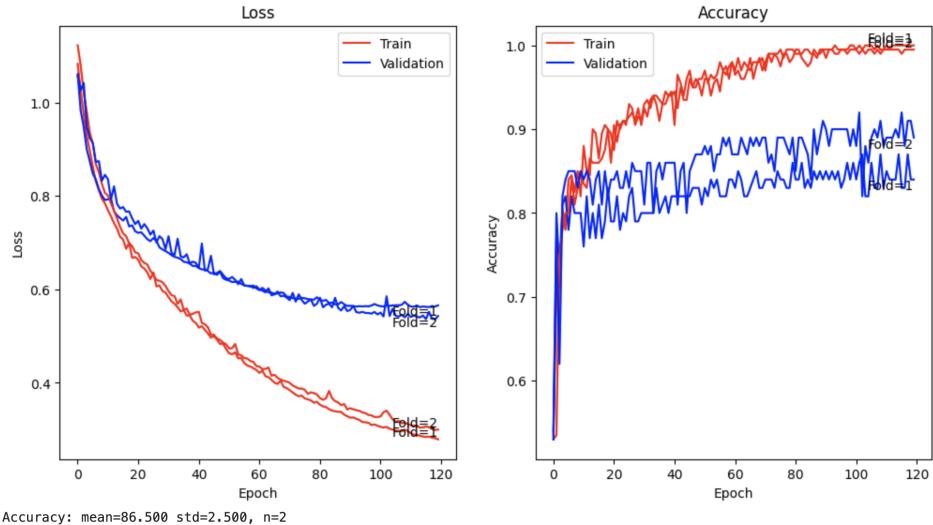


FIGURE 20 – Data augmented examples

5.1.3 Augmentation sans k-fold

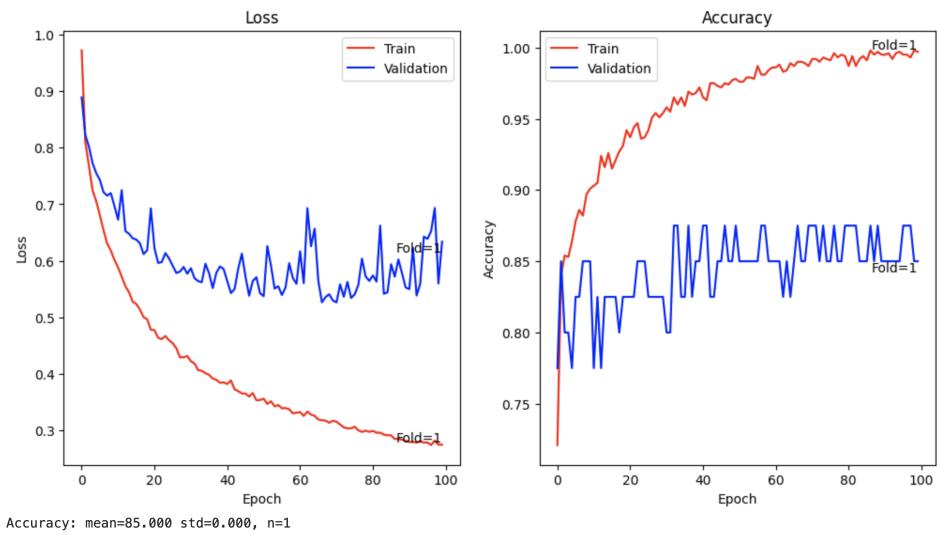


FIGURE 21 – Data augmented examples

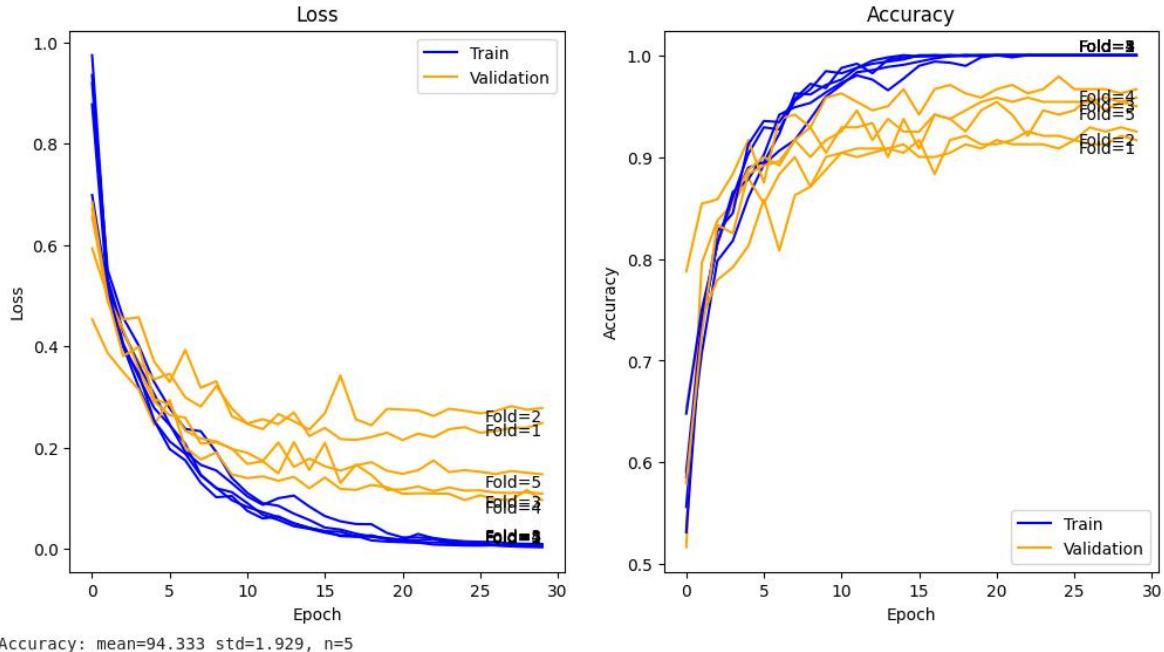
5.1.4 Analyse des résultats

Dans le premier cas, on obtiens des résultats très convaincants. Cependant, le fait d'évaluer le modèle sur les même données augmenter que les test n'est pas représentatif. A l'inverse, les tests avec des données jamais vu par le modèle (même augmenté) donne des résultats moins satisfaisant mais plus proche des capacités réelle du modèle.

5.2 Évaluation sur les modèles et potentielles améliorations

5.2.1 Modèle Tigre

En augmentant notre volume de données à 1000, une amélioration significative de la précision moyenne (94.33%) a été observée, malgré la persistance de légères traces de surapprentissage. L'écart-type réduit (1.929) indique une meilleure cohérence des résultats.



Pour renforcer le modèle, nous avons généré 5000 images supplémentaires, élargissant ainsi notre base de données initiale de 200 images et comme on a beaucoup plus de données nous avons affiner notre . Cette augmentation a favorisé la diversité des exemples d'entraînement et réduit le risque de sur apprentissage.

| Layer (type) | Output Shape | Param # |
|--------------------------------------|----------------------|---------|
| <hr/> | | |
| Conv2D_1 (Conv2D) | (None, 122, 122, 32) | 896 |
| dropout_48 (Dropout) | (None, 122, 122, 32) | 0 |
| Maxpooling2D_1 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| Conv2D_2 (Conv2D) | (None, 59, 59, 64) | 18496 |
| dropout_49 (Dropout) | (None, 59, 59, 64) | 0 |
| Maxpooling2D_2 (MaxPooling 2D) | (None, 29, 29, 64) | 0 |
| flatten (Flatten) | (None, 53824) | 0 |
| dense_48 (Dense) | (None, 128) | 6889600 |
| dense_49 (Dense) | (None, 1) | 129 |
| <hr/> | | |
| Total params: 6909121 (26.36 MB) | | |
| Trainable params: 6909121 (26.36 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

FIGURE 22 – Modèle Tigre améliorer

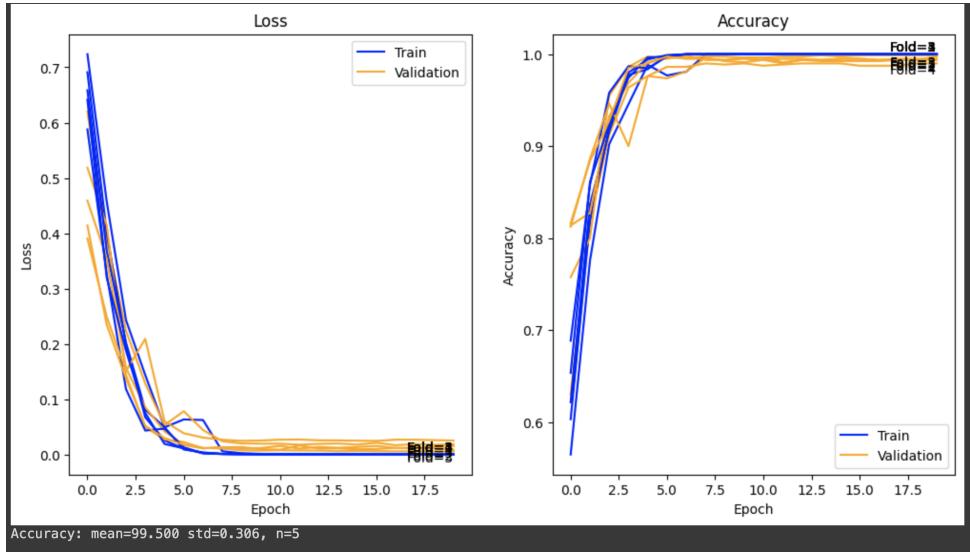


FIGURE 23 – Modèle Tigre Data Generator

L’entraînement sur cet ensemble a produit des résultats prometteurs : une précision moyenne élevée (99.188) avec un faible écart-type (0.306), démontrant une précision accrues du modèle sur divers ensembles de données même si ce sont les images de bases qui se répète avec de légère modification.

Il y a cependant un gros problème dans les deux modèles précédents. La génération des données a été effectuée avant la séparation de les données en train et test. Avec cela, nos données de test ont été corrompues. Pour assurer des données de test valide, nous avons ajusté la fonction d’évaluation pour générer dynamiquement de nouvelles données à chaque validation croisée. Bien que cette méthode ait amélioré les performances du modèle Tigre initial, le modèle modeleIDGTiger récemment crée ne montrait pas d’améliorations significatives.

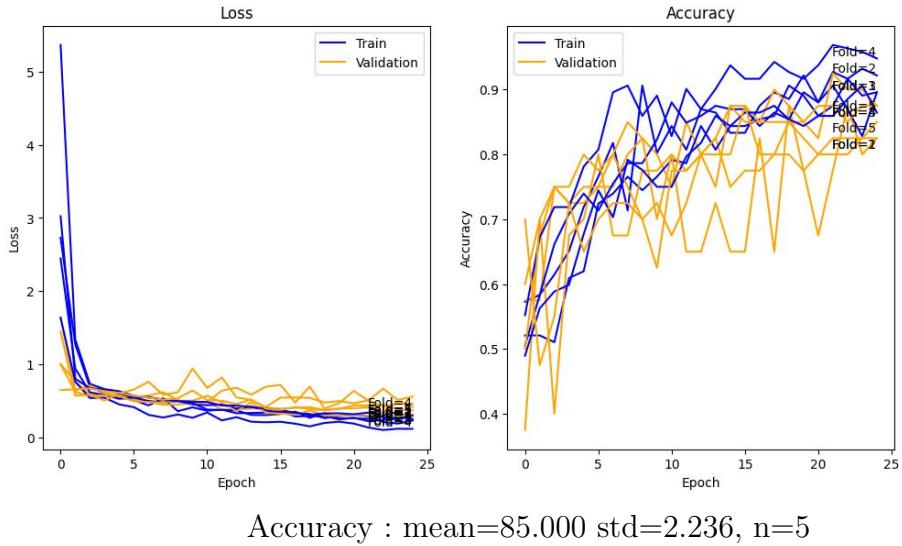


FIGURE 24 – Modèle avec génération d’image Dynamique

5.2.2 Modèle Éléphant

Pour les éléphants, le modèle de amélioré est déjà beaucoup plus performant sur les nouvelles données. Nous avons simplement ajouté une couche de maxpooling et ajusté le dropout. Voici le modèle et les résultats

| Layer (type) | Output Shape | Param # |
|---------------------------------------|----------------------|----------|
| Conv2D_1 (Conv2D) | (None, 122, 122, 32) | 896 |
| Maxpooling2D_1 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| Maxpooling2D_2 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| dropout_33 (Dropout) | (None, 61, 61, 32) | 0 |
| flatten (Flatten) | (None, 119072) | 0 |
| dense_86 (Dense) | (None, 125) | 14884125 |
| dense_87 (Dense) | (None, 1) | 126 |
| <hr/> | | |
| Total params: 14885147 (56.78 MB) | | |
| Trainable params: 14885147 (56.78 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

FIGURE 25 – Modèle éléphant avec Data Generator

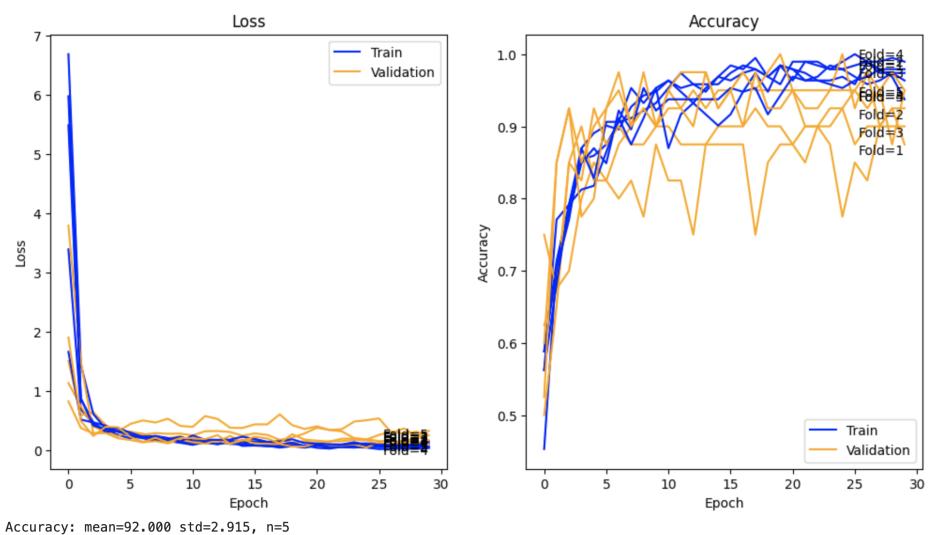


FIGURE 26 – Résultats éléphant avec Data Generator

5.2.3 Modèle Renard

ImageDataGenerator a été utilisé dans chaque fold uniquement sur les données du train. Sans modification apparente du modèle on constate une visible amélioration de la courbe de loss qui peut être traduite par la diminution du sur apprentissage.

| Layer (type) | Output Shape | Param # |
|---------------------------------------|----------------------|----------|
| <hr/> | | |
| Conv2D_1 (Conv2D) | (None, 122, 122, 32) | 896 |
| Maxpooling2D_1 (MaxPooling 2D) | (None, 61, 61, 32) | 0 |
| flatten (Flatten) | (None, 119072) | 0 |
| dense_4 (Dense) | (None, 100) | 11907300 |
| dropout_2 (Dropout) | (None, 100) | 0 |
| dense_5 (Dense) | (None, 1) | 101 |
| <hr/> | | |
| Total params: 11908297 (45.43 MB) | | |
| Trainable params: 11908297 (45.43 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

FIGURE 27 – Modèle du renard avec Data Generator

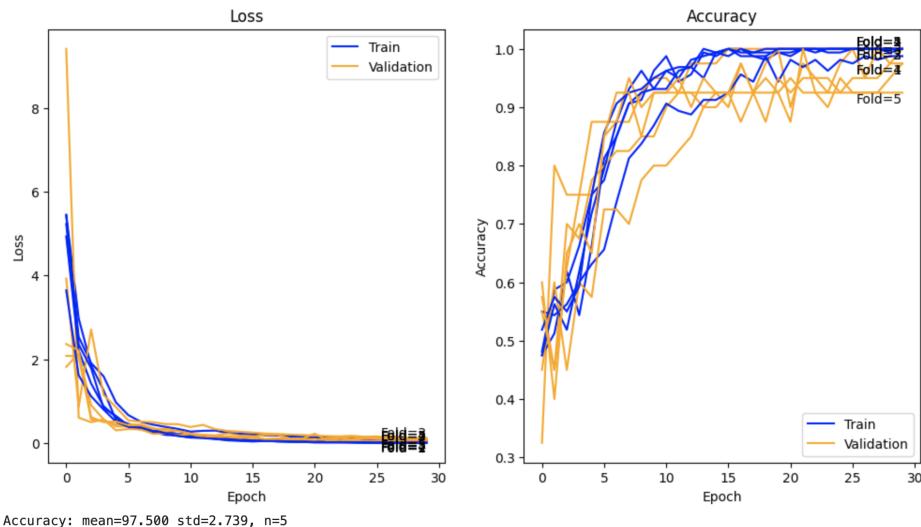


FIGURE 28 – Modèle renard avec Data Generator

5.3 Analyse des résultats et conclusion partielle

Grâce à l'augmentation des données, on s'aperçoit qu'on gagne énormément en précision et en pertinence. Ces observations nous indiquent que le nombre de données est une partie très importante pour les modèles CNN. De plus, le traitement de ces données fait partie du travail nécessaire avant même le choix du modèle.

6 Transfer Learning

Pour l'apprentissage par transfert, nous avons d'abord essayé de créer un modèle en gelant le modèle pré-entraîné, puis un modèle avec un modèle pré-entraîné non gelé (partiellement), en effectuant ces étapes séparément. Nous avons essayé plusieurs modèles pré-entraînés et n'avons pas obtenu les résultats souhaités. Après avoir lu la documentation officielle de Keras, nous avons conclu que ces étapes devaient être réalisées ensemble. Voici ce que nous avons fait au final.

Nous gelons d'abord le modèle Xception et le configurons pour qu'il s'exécute en mode inférence. Nous l'entraînons une fois sur notre ensemble de données. Après cela, nous effectuons un fine-tuning avec le modèle Xception non gelé. Nous l'entraînons à nouveau avec un taux d'apprentissage très faible (1e-5 dans notre cas). Avec cela, nous évitons de détruire les features du modèle xception avec nos features aléatoires générées.

L'exécution a été effectuée à l'aide d'une validation croisée et d'ImageDataGenerator qui a été utilisé dans chaque fold uniquement sur les données du train.

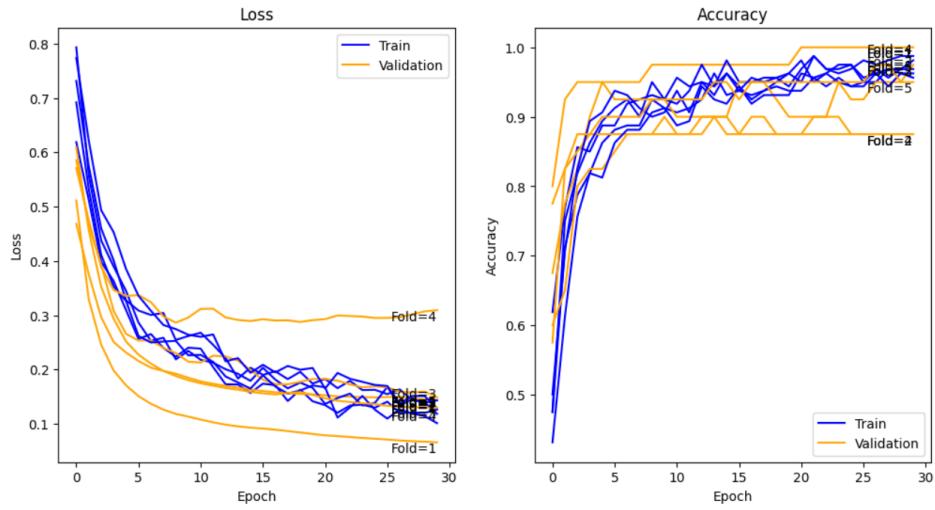
| Layer (type) | Output Shape | Param # | Trainable |
|--|-----------------------|--------------|-----------|
| <hr/> | | | |
| input_18 (InputLayer) | [(None, 124, 124, 3)] | 0 | Y |
| xception (Functional) | (None, 4, 4, 2048) | 2086148 0 | N |
| global_average_pooling2d_8 (GlobalAveragePooling2D) | (None, 2048) | 0 | Y |
| dropout_8 (Dropout) | (None, 2048) | 0 | Y |
| dense_8 (Dense) | (None, 1) | 2049 | Y |
| <hr/> | | | |
| Total params: 20863529 (79.59 MB) | | | |
| Trainable params: 2049 (8.00 KB) | | | |
| Non-trainable params: 20861480 (79.58 MB) | | | |

FIGURE 29 – Modèle transfert gelé

| Layer (type) | Output Shape | Param # | Trainable |
|--|-----------------------|--------------|-----------|
| <hr/> | | | |
| input_20 (InputLayer) | [(None, 124, 124, 3)] | 0 | Y |
| xception (Functional) | (None, 4, 4, 2048) | 2086148 0 | Y |
| global_average_pooling2d_9 (GlobalAveragePooling2D) | (None, 2048) | 0 | Y |
| dropout_9 (Dropout) | (None, 2048) | 0 | Y |
| dense_9 (Dense) | (None, 1) | 2049 | Y |
| <hr/> | | | |
| Total params: 20863529 (79.59 MB) | | | |
| Trainable params: 20809001 (79.38 MB) | | | |
| Non-trainable params: 54528 (213.00 KB) | | | |

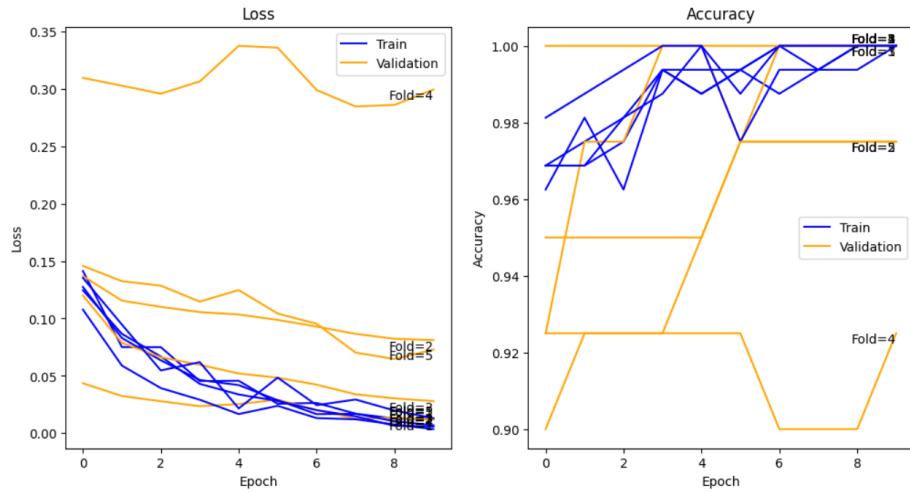
FIGURE 30 – Modèle transfert non gelé

6.1 Modèle Tigre



Accuracy : mean=93.500 std=5.148, n=5

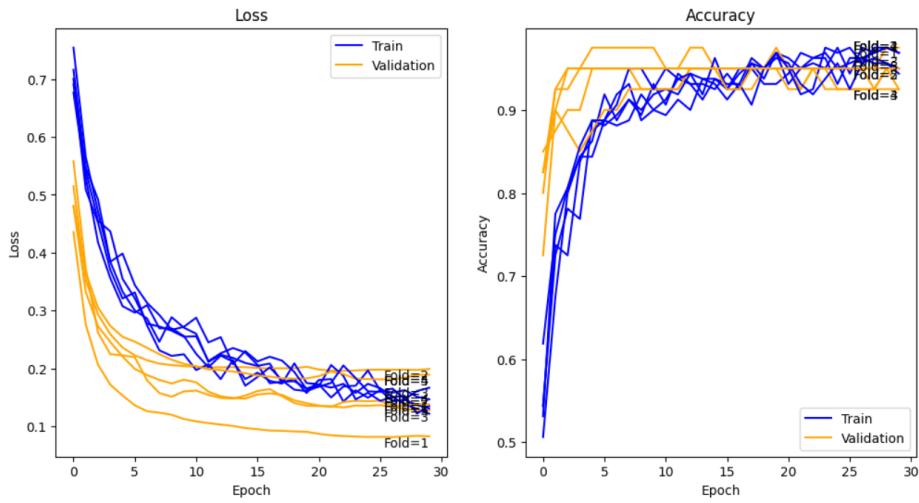
FIGURE 31 – Modèle tigre transfert gelé



Accuracy : mean=97.500 std=2.739, n=5

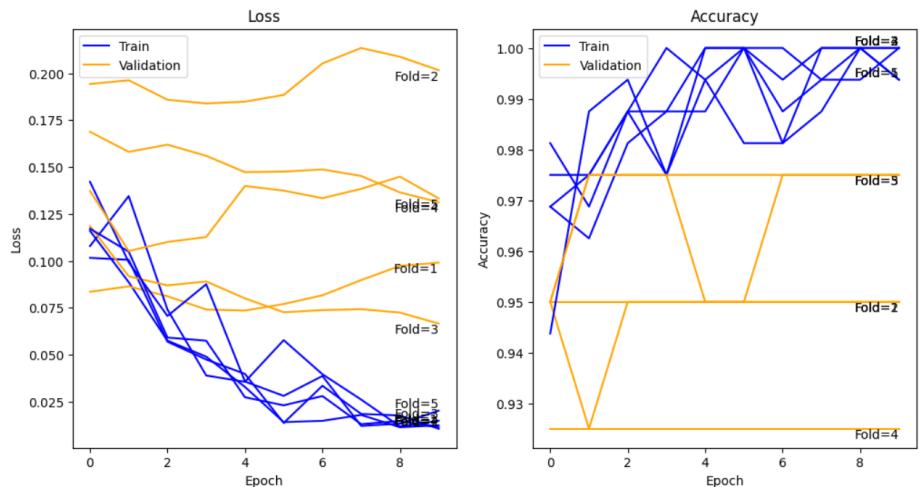
FIGURE 32 – Modèle tigre transfert non gelé (fine-tuning)

6.2 Modèle Éléphant



Accuracy : mean=94.500 std=1.871, n=5

FIGURE 33 – Modèle transfert éléphant gelé



Accuracy : mean=95.500 std=1.871, n=5

FIGURE 34 – Modèle transfert éléphant non gelé (fine-tuning)

6.3 Modèle Renard

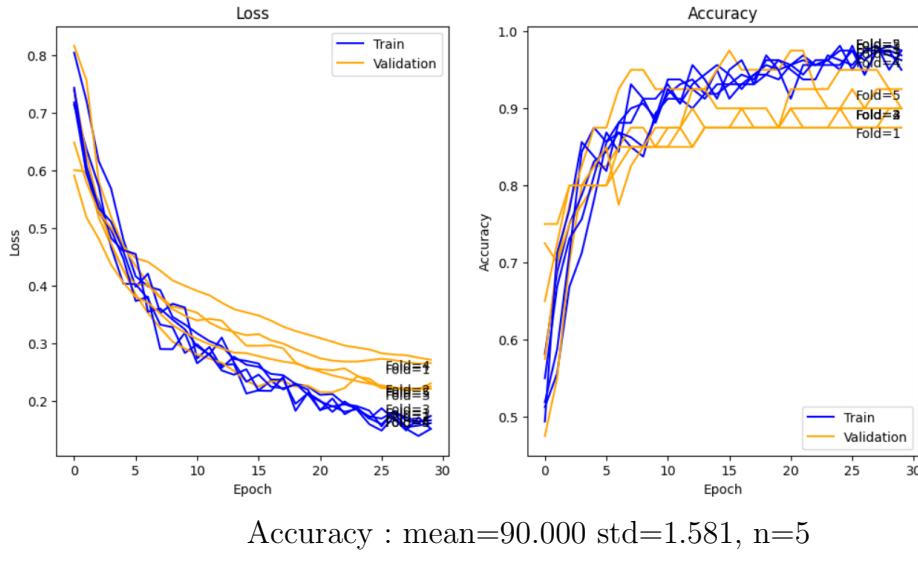


FIGURE 35 – Modèle transfert renard gelé

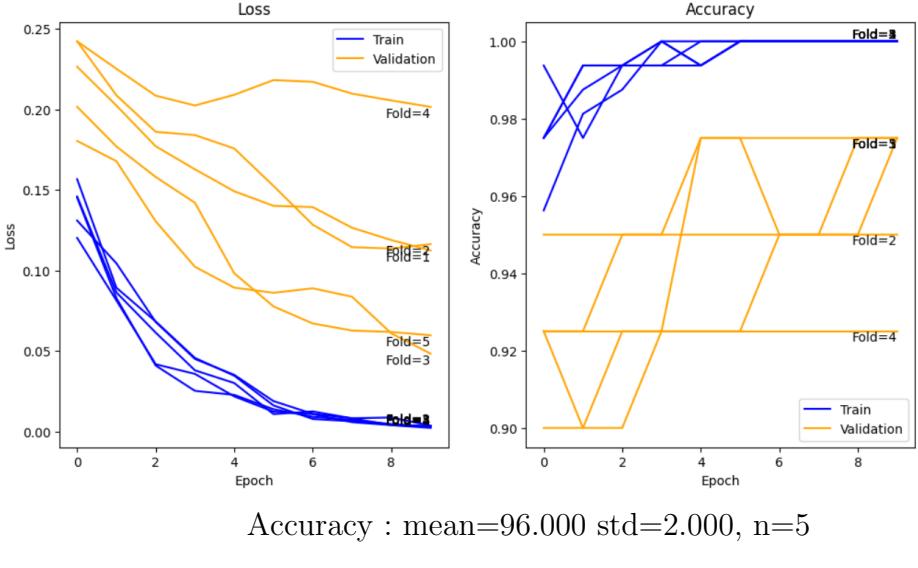


FIGURE 36 – Modèle transfert renard non gelé (fine-tuning)

6.4 Évaluation et comparaison des performances

Nous pouvons observer un modèle qui est le plus performant sur chaque ensemble de données. Parce que ce modèle était déjà très performant, nous n'avons pas créé de modèle distinct pour chaque ensemble de données afin de gagner du temps. Avec des travaux supplémentaires, nous pouvons voir un grand potentiel d'amélioration.

7 Génération d'images avec GAN

7.1 Génération d'images de renard

D'apres ce que l'on a vu precedement, l'entraînement d'un modele de classification d'images necessite beaucoup de donnees pour pouvoir avoir des resutat optimal, Dans cette partie notre objectif est de générer des images de renard à partir de notre dataset grâce a GAN.

7.1.1 Définition du générateur

Les images générées doivent suivre le même format que celles de notre dataset : des images carrées en couleur de 128 pixels de côté. Nous définissons donc le générateur suivant :

| Layer (type) | Output Shape | Param # |
|---|-----------------------|----------|
| <hr/> | | |
| dense (Dense) | (None, 369024) | 37271424 |
| reshape (Reshape) | (None, 31, 31, 384) | 0 |
| up_sampling2d (UpSampling2D) | (None, 62, 62, 384) | 0 |
| conv2d (Conv2D) | (None, 62, 62, 384) | 1327488 |
| batch_normalization (Batch Normalization) | (None, 62, 62, 384) | 1536 |
| leaky_re_lu (LeakyReLU) | (None, 62, 62, 384) | 0 |
| up_sampling2d_1 (UpSampling2D) | (None, 124, 124, 384) | 0 |
| conv2d_1 (Conv2D) | (None, 124, 124, 384) | 1327488 |
| batch_normalization_1 (Batch Normalization) | (None, 124, 124, 384) | 1536 |
| leaky_re_lu_1 (LeakyReLU) | (None, 124, 124, 384) | 0 |
| conv2d_2 (Conv2D) | (None, 124, 124, 3) | 10371 |
| activation (Activation) | (None, 124, 124, 3) | 0 |
| <hr/> | | |
| Total params: 39939843 (152.36 MB) | | |
| Trainable params: 39938307 (152.35 MB) | | |
| Non-trainable params: 1536 (6.00 KB) | | |

FIGURE 37 – Générateur

7.1.2 Définition du discriminateur

Le discriminateur est défini comme l'inverse du générateur. Son rôle est de diminuer sa zone d'analyse et d'aplatir l'information avant d'évaluer si une image est vraie ou fausse.

7.1.3 Résultats et Ajustements

Les premières générations avec le GAN ont produit des images floues et indistinctes, ressemblant davantage à des mélanges abstraits de plusieurs images de notre dataset qu'à des représentations claires de renards. Après analyse, il est apparu que le dataset initial était problématique, contenant non seulement un nombre insuffisant d'images mais aussi des images de renards peu nettes et parfois même difficilement reconnaissables.

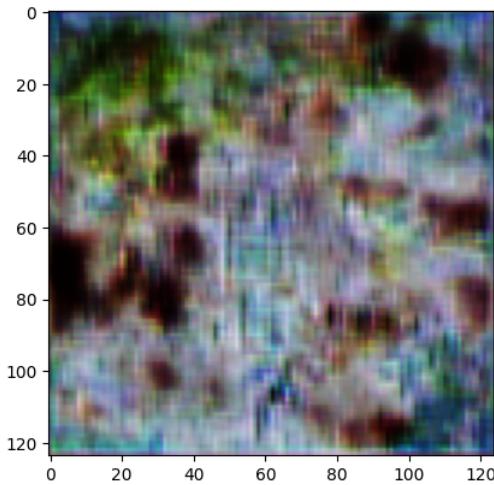


FIGURE 38 – Image générée avec l'utilisation de toutes les images de renard

Pour surmonter les difficultés rencontrées, nous avons entrepris un nettoyage minutieux de notre dataset. Ce processus a consisté à éliminer les images de faible qualité et à ne retenir que des clichés représentatif des renards de profil dans la neige. Suite à ce nettoyage, nous avons conservé une sélection restreinte d'une dizaine d'images. Pour pallier la réduction de notre ensemble de données, nous avons mis en œuvre l'outil ImageDataGenerator. Cette stratégie nous a permis d'augmenter notre jeu de données à 1000 images, en générant de nouvelles variantes à partir de notre sélection restreinte. Grâce à ces ajustements, nous avons nettement amélioré la qualité des images produites par le GAN lors des tests ultérieurs.

On se rend compte que plus nous avons d'époques, plus les images deviennent plus claires et nous obtenons de meilleurs résultats. Malheureusement, avec la limite de Google Colab, nous ne pouvons lancer que 250 époques à chaque fois pour chaque compte Gmail. Par conséquent, pour avoir le plus d'époques possible, nous avons enregistré le poids du modèle à chaque 10ème époque de l'entraînement. De plus, nous avons essayé de continuer l'entraînement pour réussir à le faire sur 1000 époques, afin d'obtenir les résultats suivants :

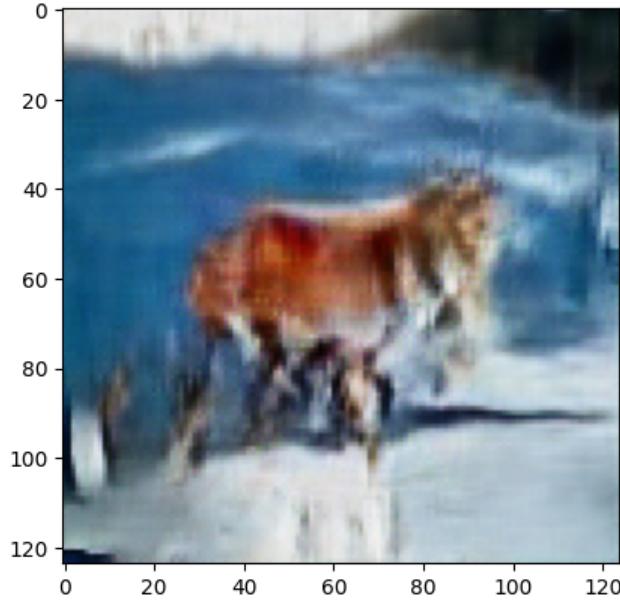


FIGURE 39 – Renard obtenu après 1000 époque sur grâce au images sélectionné

7.1.4 Prédiction avec un autre modèle

Pour évaluer la performance de notre modèle ainsi que l’efficacité de notre générateur d’images GAN (Generative Adversarial Network), nous avons sélectionné dix images de manière aléatoire parmi les plus récentes produites par le GAN. Ces images ont été soigneusement rassemblées dans un dossier dédié.

En ce qui concerne le choix du modèle à tester, nous avons opté pour un de nos meilleur modèle ,celle basé sur le Transfer Learning. Nous avons pris en compte le fait que le modèle initial utilisé pour le Transfer Learning avait été entraîné sur une variété de données. Cela rendait son utilisation plus pertinente pour évaluer la qualité des images générées par notre GAN. L’idée sous-jacente était que, grâce à son expérience préalable sur un large éventail de données, ce modèle serait mieux à même de juger la crédibilité des images générées.



FIGURE 40 – Prédiction du modèle modeleTLFox les images du GAN

Nos observations ont clairement montré que le modèle était susceptible d’être trompé par les images générées par le GAN. En détail, le modèle attribue à chaque image testée une valeur

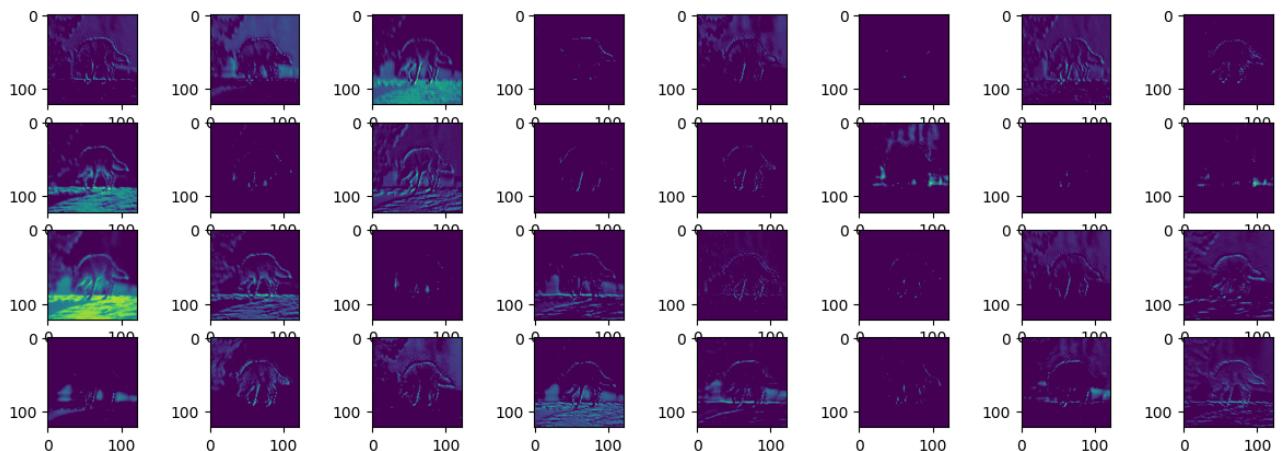
située entre 0 et 1. Une valeur inférieure à 0.5 classe l'image dans la catégorie 'Fox', tandis qu'une valeur supérieure ou égale à 0.5 la classe en tant que 'Non Fox'. L'aspect remarquable ici est que nombre de ces images issues du GAN ont été catégorisées comme 'Fox'. Cela suggère fortement que, même avec un temps d'entraînement relativement limité, notre GAN est capable de générer des images assez réalistes pour imiter les caractéristiques spécifiques que le modèle a appris à identifier.

8 Analyse des Sorties CNN

Voici un exemple de ce que donne la seule souche CNN utilisant le modèle renard :



Conv2D_1



9 Partie Optionnelle : Autoencodeurs

9.1 Coloration d'images

Pour la partie colorisation d'images en noir et blanc de tigres, nous avons adopté une stratégie basée sur l'utilisation d'un auto-encodeur. Tout au long du projet, il est apparu évident que la quantité limitée de données était un défi majeur, nous poussant ainsi à accorder une attention particulière à la préparation des données. Cette étape cruciale a été réalisée à l'aide d'un ImageDataGenerator, un outil efficace pour augmenter notre jeu de données en générant de nouvelles images à partir des existantes par diverses transformations.

Le cœur de notre approche reposait sur l'auto-encodeur, composé de deux parties principales : l'encodeur et le décodeur. L'encodeur avait pour rôle de compresser les images en noir et blanc en une représentation de dimension réduite, tandis que le décodeur s'employait à reconstruire ces images en versions colorisées, en se basant sur les caractéristiques apprises lors de l'entraînement.

9.2 Analyse des résultats

Les résultats de la colorisation varient selon le nombre d'époques d'entraînement. Avec 200 époques, les images colorées sont approximatives. À 800 époques, les résultats sont plus réalistes et montrent une meilleure adaptation du modèle. Avec 1500 époques, la colorisation est significativement améliorée, bien que des erreurs subsistent. L'augmentation du volume de données d'entraînement semble être bénéfique pour la précision de la colorisation.

Les figures ci-dessous présentent les images colorisées à différents stades d'époques d'entraînement :



FIGURE 41 – Coloration après 200 époques.



FIGURE 42 – Coloration après 800 époques.

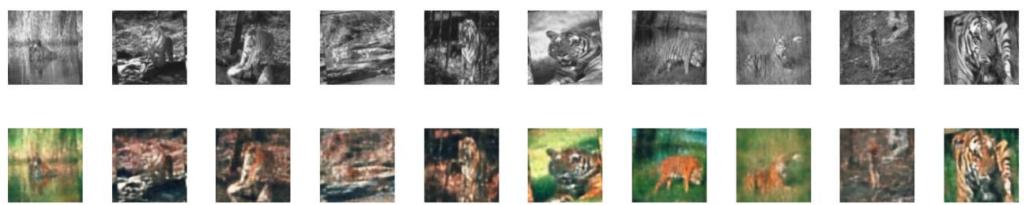


FIGURE 43 – Coloration après 1500 époques.

10 Conclusion

Au fil du temps, nous avons progressivement amélioré nos modèles. Dès le départ, la taille limitée de notre ensemble de données et les choix d'optimisation étaient des défis majeurs. Néanmoins, nos modèles améliorés ont atteint une précision et un ajustement satisfaisants. L'emploi de l'ImageDataGenerator a apporté des améliorations supplémentaires, bien que nous ayons rapidement rencontré ses limites en raison de la similitude des données générées. L'adoption de l'apprentissage par transfert a marqué une nette amélioration. Dans des contextes comme le nôtre, caractérisés par des ensembles de données restreints, l'utilisation de modèles pré-entraînés pour le transfert learning s'avère être une stratégie plus efficace.