



FACULTÉ DES SCIENCES
MONTPELLIER



IMPLÉMENTATION D'UN ALGORITHME DE DEEP LEARNING

Alexandre Cauty 22115525 - Manon Ramaroson 22102547 - Dorine Fontaine 22100559

Groupe 3

Abstract

Dans le cadre du projet de Machine Learning, nous avons eu l'opportunité de travailler sur des algorithmes de Deep Learning. Le but étant d'obtenir un modèle pour chacun des animaux qui puissent prédire si l'image de test correspond à l'animal recherché ou pas. Une grande partie est consacrée à pouvoir générer des images, plus précisément sur les renards.

Université de Montpellier

Contents

1	Introduction	2
2	Création de baseline	2
2.1	Visualisation des données	2
2.2	Les baselines	3
3	Optimisation	4
3.1	Baseline Améliorée Éléphant	4
3.2	Baseline Améliorée Tigre	6
3.3	Baseline Améliorée Renard	7
4	ImageDataGenerator	9
4.1	Modèle Éléphant	9
4.2	Modèle Tigre	10
4.3	Modèle Renard	11
5	Transfer Learning	12
5.1	Modèle Éléphant	12
5.2	Modèle Tigre	12
5.3	Modèle Renard	13
6	Generative Adversarial Network Fox	14
6.1	Prédiction sur les sorties du GAN	15
7	Modèle le plus complexe : contenu des sorties des CNN	16
8	Conclusion	17

1 Introduction

Dans cette étude, notre objectif est de construire des modèles de classification binaire capables d'identifier, à partir d'une image donnée, si celle-ci correspond à l'animal recherché ou pas ; nous allons rechercher un renard, un éléphant ou un tigre. Pour ce faire, notre travail se décomposera en plusieurs étapes :

1. Création d'une baseline pour chaque animal : Nous débuterons en construisant un modèle de base pour effectuer nos premières prédictions. Les résultats obtenus nous guideront dans les étapes suivantes de notre travail.
2. Amélioration du modèle : Nous envisagerons des stratégies d'amélioration des modèles, incluant la recherche d'hyperparamètres, l'application de techniques de régularisation, l'augmentation des données etc...
3. Exploration des modèles de transfert learning : Nous examinerons l'utilisation de techniques de transfert learning pour capitaliser sur des modèles pré-entraînés et les adapter à notre tâche spécifique.
4. Génération d'images de renards via GAN : Enfin, nous explorerons la génération d'images de renards à l'aide de GAN pour ensuite les tester sur notre meilleur modèle de prédiction de renards.

Cette approche nous permettra d'explorer et de perfectionner nos modèles pour atteindre des performances optimales dans l'identification des trois animaux ciblés.

2 Crédation de baseline

2.1 Visualisation des données

Avant de débuter le projet, nous voulons visualiser la distribution des données dans leur ensemble. Cette étape nous permettra de mieux appréhender la nature des données et d'analyser comment elles se regroupent.

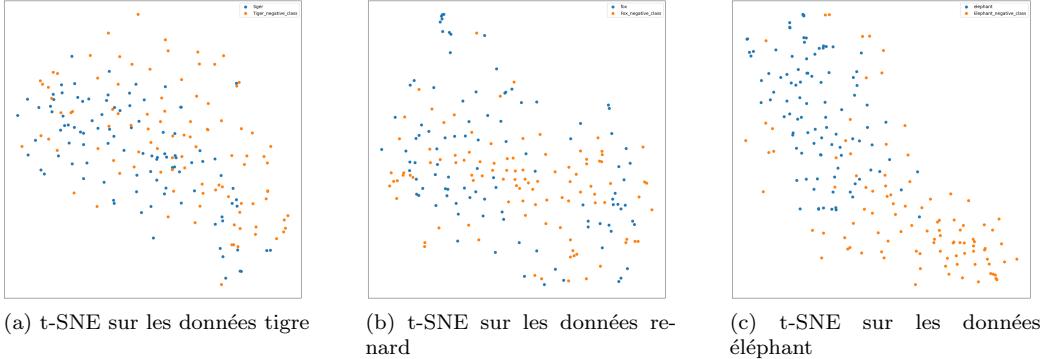


Figure 1: Visualisation des jeux de données grâce à la technique t-SNE.

Que peut on en déduire ? On peut remarquer que pour les données 'tiger' et 'fox', il n'y a pas de séparation nette entre les deux classes. Pour ces données, il va être plus difficile pour le classifieur de différencier les images, cependant pour les données éléphant on peut remarquer que les données se distinguent plus.

Les jeux de données de renard et tigre sont mélangés, on sait d'avance que le model va sans doute trop se concentrer sur certains patterns communs, il nous faudra ainsi mettre des couches de dropout pour éviter que le modèle se focalise trop sur certains détails.

2.2 Les baselines

Nous avons donc créé la Baseline à partir des notebooks du module de Machine Learning 2, nous pouvons observer sur la Figure 2 la structure du modèle utilisé pour chaque jeu de données.

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
Conv2D_1 (Conv2D)	(None, 122, 122, 32)	896
Maxpooling2D_1 (MaxPooling 2D)	(None, 61, 61, 32)	0
flatten (Flatten)	(None, 119072)	0
dense_2 (Dense)	(None, 100)	11907300
dense_3 (Dense)	(None, 1)	101

Total params: 11908297 (45.43 MB)
Trainable params: 11908297 (45.43 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 2: Summary de la baseline.

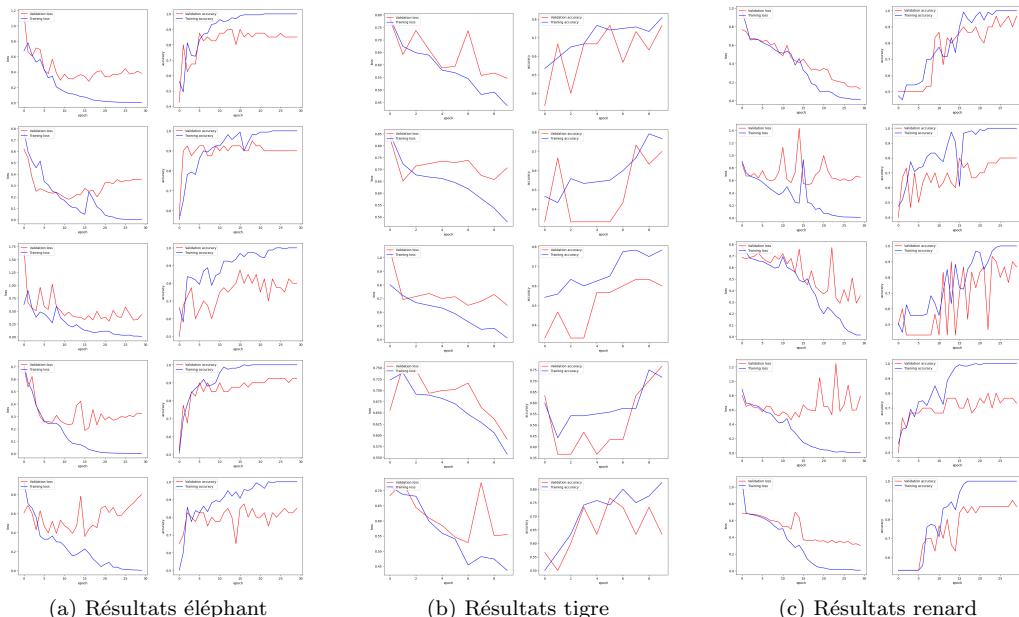


Figure 3: Résultats des 5 folds pour le modèle Baseline

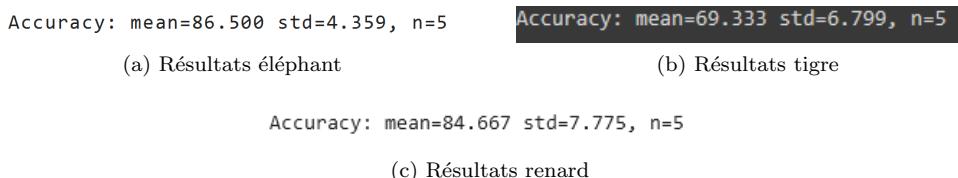


Figure 4: Moyenne d'accuracy, écart-type sur les 5 folds pour chaque jeu de données

Interprétation des données : Dans la majorité des cas de la Figure 3 nous avons du surapprentissage. Pour les trois animaux nous rencontrons des scénarios similaires :

1. La validation loss augmente et la training loss diminue, le modèle apprend trop bien sur les données d'entraînement.
2. La validation accuracy est toujours plus faible que la training accuracy, le modèle performe bien sur les données d'entraînement, mais ne généralise pas bien sur les nouvelles données. Il est en train de surapprendre les caractéristiques spécifiques de l'ensemble d'entraînement.

Nous remarquons que l'écart-type moyen (voir Figure 4) des jeux de données est élevé donc que les accuracy s'éloignent de celle moyenne, les résultats ne sont pas assez stables.

3 Optimisation

Tout d'abord nous recherchons maintenant la meilleure combinaison d'hyper-paramètres pour optimiser le modèle Baseline, pour ce faire nous utilisons un random search sur plusieurs hyper-paramètres. Après avoir ciblé la meilleure combinaison avec random search, nous utilisons un grid search pour essayer des combinaisons en prenant pour base celle trouvée précédemment avec le random search.

Nous souhaitons évaluer les hyper-paramètres suivants : la fonction d'activation d'une des couches du réseau de neurones, le batch_size, l'optimizer et son learning_rate.

Afin de prévenir le surapprentissage un EarlyStopping sur la loss de validation a été ajouté, grâce à cela nous pouvons arrêter le nombre d'epochs plus tôt en cas d'augmentation de validation loss.

3.1 Baseline Améliorée Éléphant

Pour les meilleurs hyper-paramètres nous effectuons un random search puis un grid search comme expliqué précédemment. Cependant cette expérience génère les meilleurs hyper-paramètres pour un jeu de données d'entraînement et de validation précis, c'est pourquoi nous effectuons une validation croisée sur 5 folds pour faire cette expérience sur chacun des folds et ainsi obtenir 5 meilleurs hyper-paramètres.

Les 5 combinaisons obtenues (voir Figure 5) ont les mêmes résultats pour la fonction d'activation et le batch_size. Pour les deux paramètres learning_rate et optimizer nous prendrons la valeur ayant la majorité, 4/5 des combinaisons avaient le même learning_rate (égal à 0.001) et 3/5 des combinaisons optent pour l'optimizer adam (les autres ont utilisés adamax).

Nous considérons donc que la meilleure combinaison est celle égale aux combinaisons 1, 3 et 5.

```
{'activation': 'relu', 'batch_size': 8, 'learning_rate': 0.001, 'optimizer': 'adam'}
{'activation': 'relu', 'batch_size': 8, 'learning_rate': 0.01, 'optimizer': 'adamax'}
{'activation': 'relu', 'batch_size': 8, 'learning_rate': 0.001, 'optimizer': 'adam'}
{'activation': 'relu', 'batch_size': 8, 'learning_rate': 0.001, 'optimizer': 'adamax'}
{'activation': 'relu', 'batch_size': 8, 'learning_rate': 0.001, 'optimizer': 'adam'}
```

Figure 5: Combinaisons des meilleurs hyper-paramètres pour 5 folds.

Tout d'abord en comparant la baseline avec celle améliorée (Modèle 1 du tableau 1) on se rend compte que la loss validation est plus basse pour celle améliorée et que le surapprentissage est nettement moins présent (voir Figure 6).

En testant les méthodes de régularisations sur le jeu de données de l'éléphant (via le tableau 1), nous constatons que pour certains modèles l'accuracy a augmenté et tous les modèles présentent toujours du surapprentissage.

Les meilleurs modèles obtenus sont donc 1 et 6 (s'arrêtant grâce à l'EarlyStopping) car on ne détecte pas d'overfitting.

En augmentant le filtre de la couche convolution (explication dans la partie Renard qui suit) de 32 à 128 nous obtenons de meilleurs résultats pour la base améliorée sans régularisation : Accuracy

Numéro	Régularisation	Moyenne accuracy	Ecart-type	Overfitting	Epochs Max
1	Base améliorée	90.50	8.124	Oui	30
2	Dropout (0.1)	91.0	4.637	Oui	40
3	BatchNormalization	83.0	5.788	Oui	18
4	KernelRegularization (L2 = 0.1)	84.50	7.483	Non	30
5	Dropout + BatchNormalization	74.0	12.903	Oui	14
6	Dropout + KernelRegularization	86.50	4.062	Non	30
7	BatchNormalization + KernelR	56.50	14.967	Oui	14
8	Dropout + BatchN + KernelR	54.50	16.462	Oui	25

Table 1: Résultats de la baseline améliorée éléphant (lr=0.001 et 5 folds)

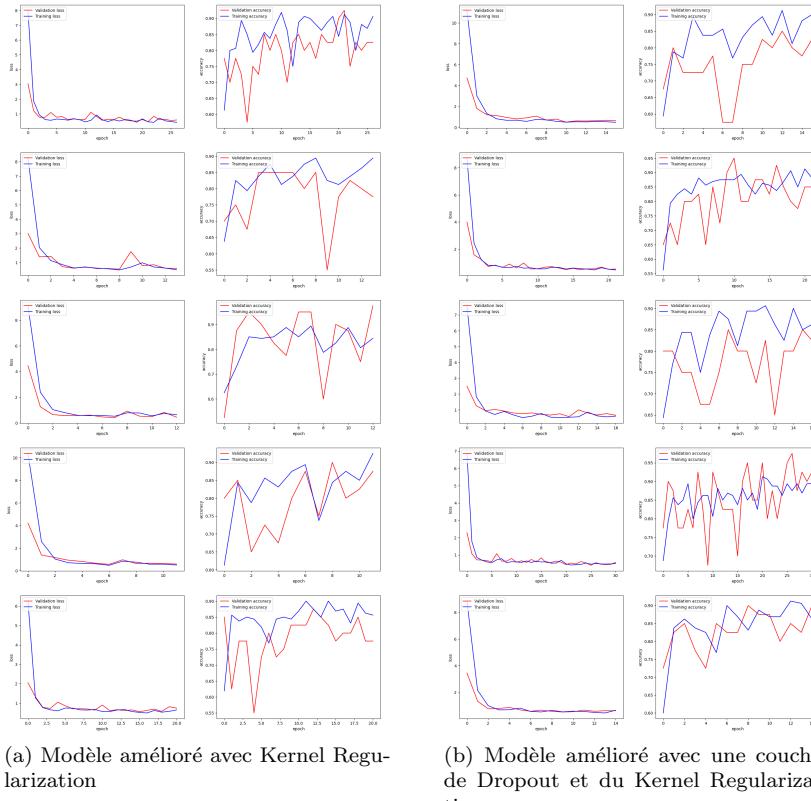


Figure 6: Résultats des 5 folds pour les modèles améliorées d'éléphant

moyenne 88.110 et écart type à 1.832 mais des valeurs de loss qui ne sont pas plus basses ; pour le modèle avec une couche de dropout nous avons un écart-type qui se creuse pour une accuracy similaire.

3.2 Baseline Améliorée Tigre

Les hyperparamètres retournés par le GridSearch pour la meilleure baseline sont : Optimizer Adam, learning rate 0,001 et batch size 8.

	Moyenne accuracy	Ecart-type	Overfitting
Dropout	81.33	9.092	Oui
BatchNormalization	66.00	10.83	Oui
KernelRegularization	76.66	9.88	Oui
Dropout + BatchNormalization	58.00	14.07	Oui
Dropout + KernelRegularization	74.00	4.42	Oui
BatchNormalization + KernelRegularization	78.66	3.39	Oui
Dropout + BatchNormalization + KernelRegularization	84.66	4.52	Oui

Table 2: Résultats de la baseline améliorée tigre pour 30 epochs et 5 folds

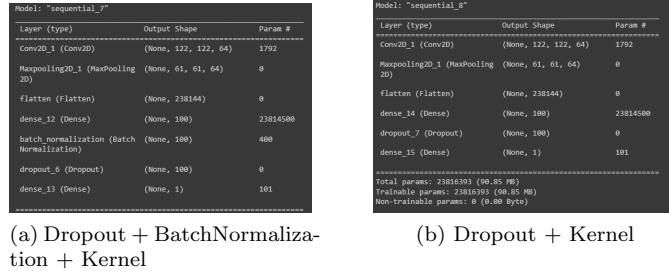


Figure 7: Summary des meilleures baselines améliorées pour le tigre.

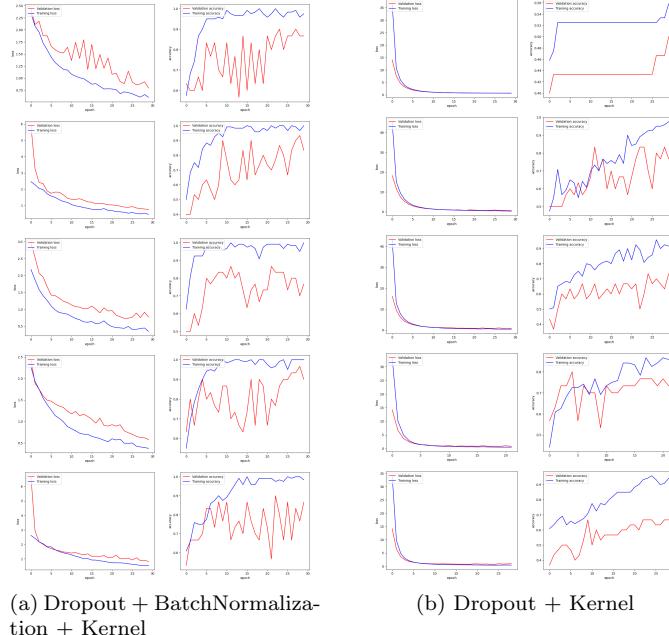


Figure 8: Résultats des meilleures baselines améliorées pour le tigre.

Interprétation des résultats : Nous pouvons voir que le modèle baseline est moins performant que le modèle baseline améliorée. Grâce à la modification de certains hyperparamètres et l'application de la régularisation. Néanmoins le surapprentissage est toujours présent dû au manque d'images.

3.3 Baseline Améliorée Renard

Pour améliorer le modèle renard, on cherche les hyperparamètres les plus optimaux avec un GridSearch. Nous obtenons sur la figure 9 un résultat de ces meilleurs hyperparamètres.

```
{'activation': 'relu', 'dropout_rate': 0.1, 'batch_size': 8, 'learning_rate': 0.001, 'optimizer': 'adamax'}
```

Figure 9: Meilleur paramètre

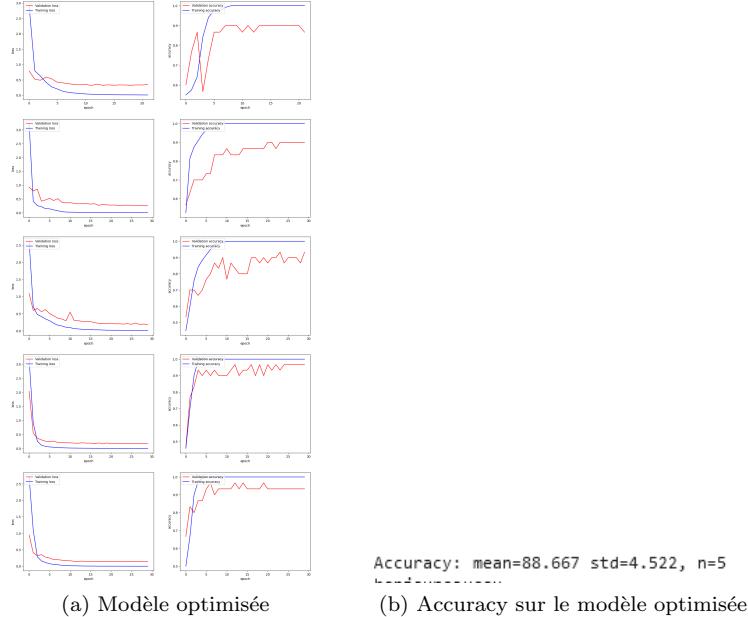


Figure 10: Résultats du modèle optimisé renard.

Il semble que les améliorations apportées au modèle ont entraîné une petite augmentation de l'accuracy de 82 % à 89 % et une réduction de l'écart-type. Cependant, malgré cette amélioration, certains signes indiquent que le modèle présente encore des problèmes de surapprentissage.

Pour améliorer le modèle et éviter le surapprentissage, on peut ajouter en plus des meilleurs hyperparamètres des techniques de régularisation du modèle. Les résultats des tests des méthodes de régularisation sur notre modèle sont présentés dans le tableau 3.

Numéro	Régularisation	Moyenne accuracy	Ecart-type	Overfitting	Epochs
1	Base améliorée	88.667	4.522	oui	30
2	Dropout (0.1)	95.333	4.522	oui	30
3	BatchNormalization	90.667	5.333	oui	30
4	KernelRegularization (L2 = 0.1)	77.333	16.111	oui	30
5	Dropout + BatchNormalization	86.000	11.813	oui	30
6	Dropout + KernelRegularization	80.000	6.667	oui	30
7	BatchNormalization + KernelR	84.000	17.563	oui	30
8	Dropout + BatchN + KernelR	87.333	6.464	oui	30

Table 3: Résultats de la baseline améliorée renard (5 folds)

On peut donc en déduire que la meilleure méthode pour notre modèle est le drop out. De plus, nous avons constaté qu'en faisant varier le filtre de la couche Conv2D, nous pouvons voir de nettes améliorations de l'accuracy du modèle, même si il subsiste encore du surapprentissage (figure 12). L'augmentation de filtres permet au modèle de se complexifier, d'améliorer sa capacité à différencier les classes, et de lui permettre d'apprendre des représentations plus riches des données.

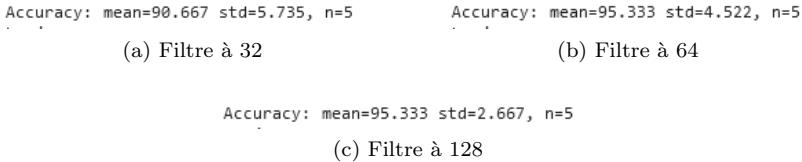


Figure 11: Résultats de l'accuracy lorsqu'on fait les filtres et avec drop out

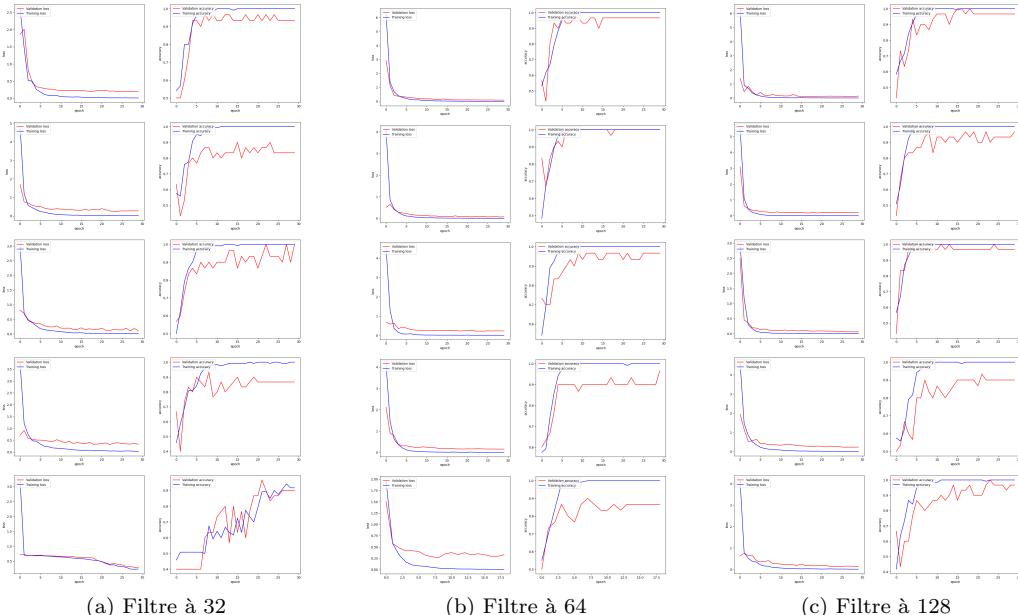


Figure 12: Résultats des modèles après variations des filtres avec drop out

Ce que nous pouvons entreprendre maintenant c'est la génération de données avec DataImageGenerator.

4 ImageDataGenerator

Pour éviter le surapprentissage, nous avons mis en place des techniques de régularisation du modèle. Cependant, malgré ces ajustements, le surapprentissage persiste, principalement en raison d'un manque de données d'entraînement. Afin de palier à ce problème, nous avons utilisé l'outil ImageDataGenerator, permettant de générer de nouvelles images à partir des données existantes (exemple sur la Figure 14). Pour tous les modèles, nous avons pris les paramètres de ImageDataGenerator indiqués sur la Figure 13.

```
datagen = ImageDataGenerator(shear_range=0.2, zoom_range=0.2, horizontal_flip=True, fill_mode='nearest', rotation_range=20)
```

Figure 13: Paramètres de ImageDataGenerator.

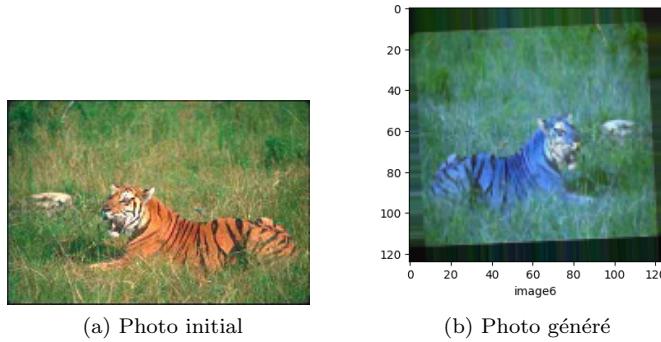


Figure 14: Résultat de Image Data Generator

Les changements effectués sur les images sont divers, on peut retrouver :

1. La rotation : L'image peut être pivotée d'un certain angle, soit dans le sens horaire, anti-horaire ou à des angles aléatoires.
2. La translation : Déplacement de l'image dans différentes directions (haut, bas, gauche, droite).
3. Le miroir : Réflexion horizontale ou verticale de l'image.
4. La déformation : Légères distorsions ou modifications de la géométrie de l'image.
5. Changement de luminosité, contraste ou teinte : Altération des niveaux de luminosité, du contraste ou de la teinte de l'image.

Nous avons un jeu de données relativement petit, donc le mieux serait d'envisager des transformations plus diverses pour augmenter sa taille de manière significative sans introduire de redondance. Cependant il faut s'assurer que les transformations ne perturbent pas trop les caractéristiques importantes des images. Des transformations qui altèrent trop la forme ou les traits caractéristiques des animaux pourraient rendre les données moins utiles. Pour éviter que la redondance s'installe trop et augmente le surapprentissage, nous avons décidé de générer seulement 5 images pour une image réelle.

Une fois que nous avons enrichi notre jeu de données, nous l'utilisons pour évaluer si nos modèles ont présenté des améliorations.

4.1 Modèle Éléphant

En utilisant les modèles améliorés (avec ou sans régularisation) avec les données réelles et générées nous obtenons vite de l'underfitting où la courbe de loss train est toujours en train de décroître n'ayant pas atteint d'état stable, mais celle de validation augmente et s'arrête donc à cause de l'EarlyStopping au bout de moins d'une dizaine d'epochs. En enlevant l'EarlyStopping (avec 50 epochs) on obtient du

surapprentissage car la loss train se stabilise mais la validation loss ne fait qu'augmenter. Le modèle 3 (du Tableau 4) utilise beaucoup plus d'images que le modèle 2 car on génère 10 données pour une donnée réelle, l'écart-type a réduit mais l'underfitting est présent. Le modèle 5 utilise une taille de batch = 32 contrairement aux autres modèles (égale à 8). Malgré la réduction de l'accuracy on constate la réduction du std avec les données générées, donc des folds plus stables.

Num	Type	Moyenne accuracy	Ecart-type	Overfitting	Epochs
1	Améliorée	90.50	8.124	Oui	30
2	Améliorée (5 données générées/réelle)	85.822	2.036	underfitting	10
3	Améliorée (10 données générées/réelle)	88.110	1.832	underfitting	20
4	Améliorée et régularisée	91.0	4.637	Oui	40
5	Améliorée et régularisée (données générées)	85.091	2.105	Oui	13

Table 4: Résultats d'ImageDataGenerator pour 5 images générées à partir d'une image réelle éléphant 5 folds

4.2 Modèle Tigre

Nouvelle approche pour diminuer le surapprentissage : l'augmentation de données (ImageDataGenerator) Utilisation de la baseline améliorée.

	Moyenne accuracy	Ecart-type
Dropout + ImageDataGenerator	82.00	4.00
Dropout + Kernel + ImageDataGenerator	74.00	5.735

Table 5: Résultats avec ImageDataGenerator pour 200 epochs et 5 folds

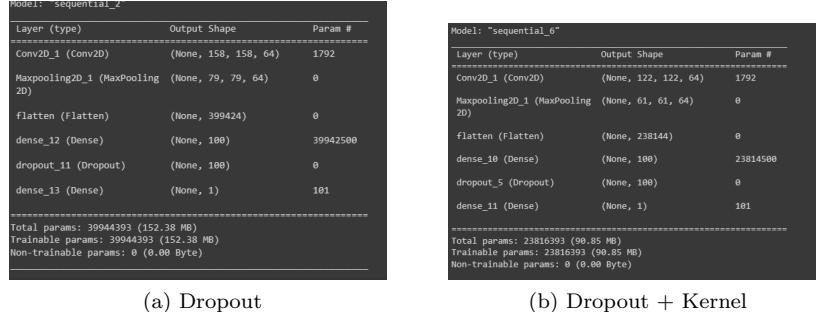


Figure 15: Summary des meilleures baselines pour le tigre.

Interprétation des résultats : Nous pouvons voir que le modèle ImageDataGenerator est plus performant que le modèle améliorée et la baseline de base. On retrouve des résultats proches à la baseline améliorée. Néanmoins le surapprentissage est moins important. On peut en conclure qu'il est important d'analyser ces différents facteurs pour comprendre pourquoi un modèle performe mieux ou moins bien dans une tâche de classification.

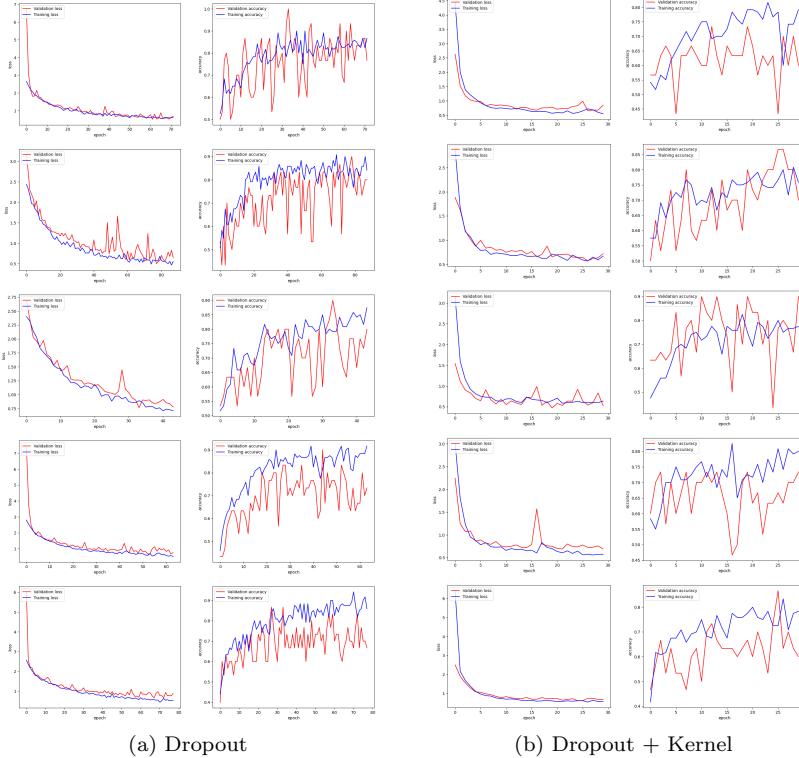


Figure 16: Résultat ImageDataGenerator pour le tigre avec 200 epochs et 5 folds.

4.3 Modèle Renard

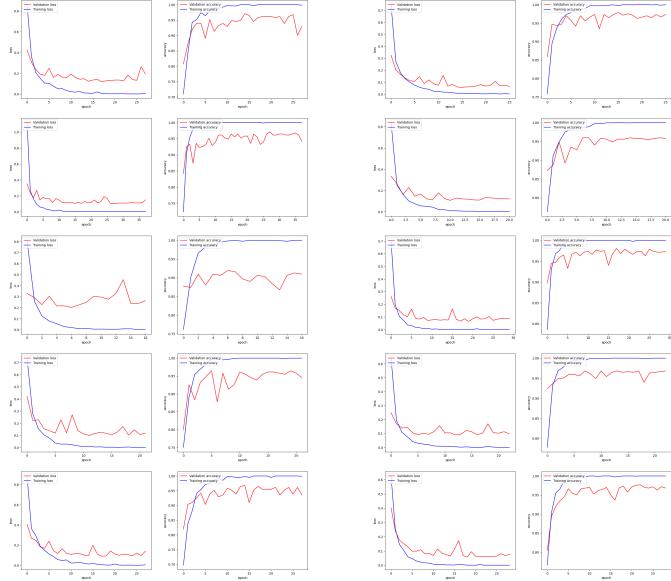
Nous avons créé successivement cinq, puis dix, puis vingt fois photos par image. Nous avons observé que plus le nombre d'images augmentait, meilleures étaient les performances du modèle, ce qui pour l'instant est logique (voir figure 17). Il est important de préciser que les 3 modèles améliorés ont une couche de dropout.

Num	Type	Moyenne accuracy	Ecart-type	Overfitting	Epochs
1	Amélioré	88.667	4.522	oui	30
2	Amélioré (10 données générées/image réelle)	93.230	1.257	léger	100
3	Amélioré (20 données générées/image réelle)	96.822	0.579	non	100

Table 6: Comparaison avec les résultats antérieurs

Il semble que le modèle ait montré des signes de surapprentissage (fort sur la photo a de la figure 17 et léger pour la photo b) sur les données générées bien que nous ayons mis en place du drop out.

Explication plausible au surapprentissage : Cela est probablement dû au fait que les données introduites sont similaires à celles d'origine et présente peu différences.



(a) 10 photo générée pour une image

(b) 20 photo générée pour une image

Figure 17: Résultat ImageDataGenerator pour le renard avec 100 epochs et 5 folds.

Accuracy: mean=93.230 std=1.257, n=5

(a) 10 photo générée pour une image

Accuracy: mean=96.822 std=0.579, n=5

(b) 20 photo générée pour une image

Figure 18: Résultat des accuracy pour le renard avec 100 epochs et 5 folds.

5 Transfer Learning

Pour le transfer learning, nous avons testé plusieurs modèles entraînés que proposent Keras correspondant à des CNN. Le transfer learning est une technique d'apprentissage dans laquelle un modèle développé pour une tâche particulière est réutilisé comme point de départ pour un modèle sur une deuxième tâche.

Voici les modèles que proposent Keras :

- VG16
- VGG19
- Resnet
- ResnetV2
- MobileNet
- MobileNetV2

5.1 Modèle Éléphant

Le tableau 7 récapitule les différents modèles utilisés pour le transfer learning. En utilisant les données générées on se retrouve avec le même problème d'underfitting pour 300 epochs (même en augmentant le learning rate), il faudrait sans doute augmenter le nombre d'epochs si on voulait vraiment étudier avec les données générées.

5.2 Modèle Tigre

Les hyperparamètres sont les suivants : Compilateur Adam et learning rate à 0,001.

	MobileNetV2	ResnetV2	Resnet	VGG19	VGG16
Accuracy moyenne	94.500	96.000	93.500	93.500	94.000
Ecart-type	2.915	2.550	2.550	4.637	5.148
Overfitting	Oui	Oui	Oui	Oui	Oui
Nombre d'epochs	300	300	300	300	300

Table 7: Résultats pour le transfer learning éléphant ($lr = 0.001$, dropout = 0.2, batch = 32)

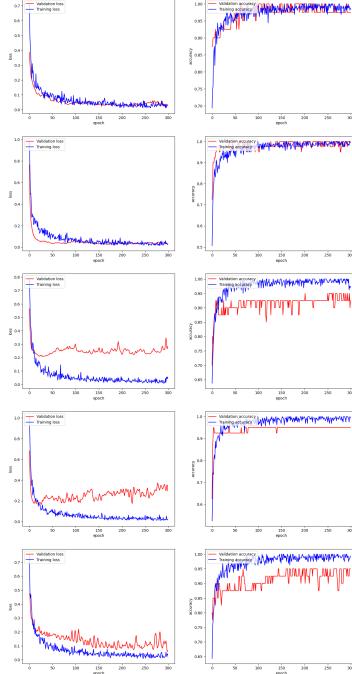


Figure 19: Modèle éléphant avec ResnetV2 et batch taille 32

	MobileNetV2	ResnetV2	VGG19	VGG16
Accuracy moyenne	96.500	97.00	89.50	87.00
Ecart-type	3.742	1.00	4.30	9.92
Nombre d'epochs	20	100	20	20

Table 8: Résultats pour le transfer learning de tigre

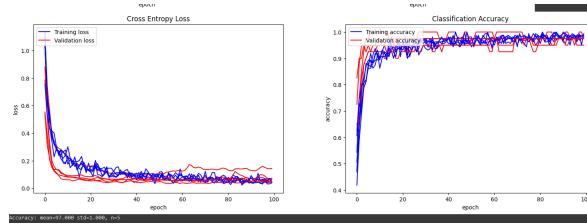


Figure 20: Meilleur résultat transfer learning tigre ResnetV2

5.3 Modèle Renard

Sur la figure 9, nous affichons un récapitulatif des différents résultats que nous obtenons en utilisant les différent modèles de transfert learning.

Nous pouvons ainsi déduire à partir de ces test que le modèles qui convient le plus a notre situation est le VGG19.

Interprétation des résultats : les courbes de loss convergent vers des valeurs basses, notam-

	MobileNetV2	ResnetV2	VGG19	VGG16
Accuracy moyenne	82.000	83.000	98.500	92.500
Ecart-type	5.788	3.674	2.000	4.743
Overfitting	oui	oui	non	non
Nombre d'epochs	100	100	100	100

Table 9: Résultats selon le modèle pris pour le transfer learning de renard

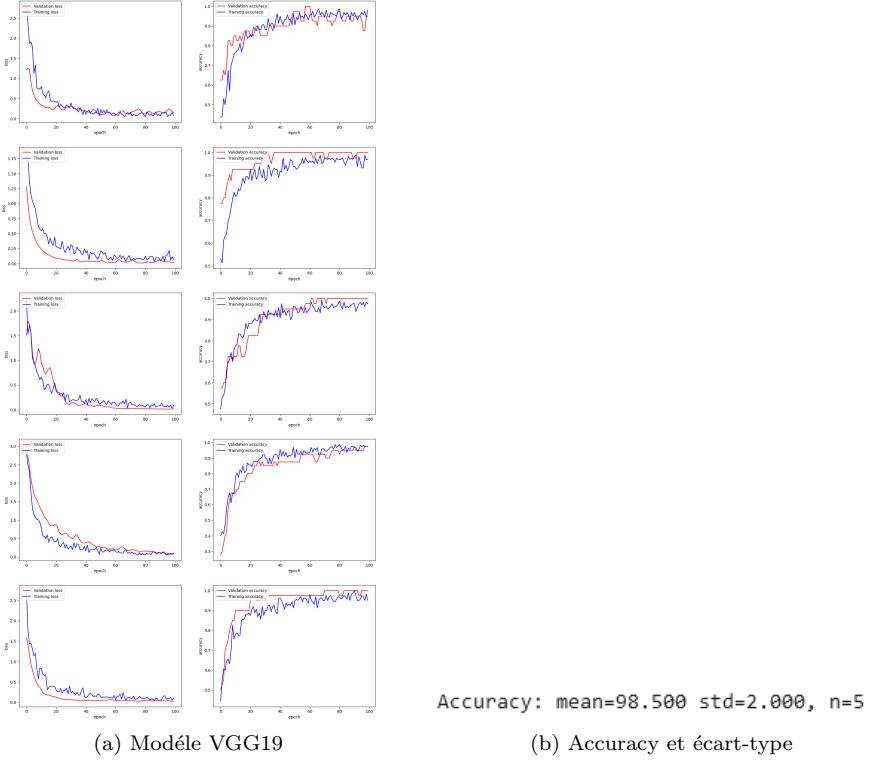


Figure 21: Résultat du modèle VGG19

ment une loss de validation qui se rapproche de 0, cela indique que le modèle apprend de manière approfondie sans présenter du surapprentissage. De plus, les courbes d'accuracy de validation et de test sont similaires et proches, cela suggère une bonne capacité de généralisation du modèle.

6 Generative Adversarial Network Fox

Malgré un entraînement prolongé (5 500 epoch) du GAN sur le jeu de données renards (100 images), les résultats obtenus ne présentent pas une ressemblance satisfaisante avec des renards réels (image a, figure 22). Nous avons décidé de créer un jeu de données de renard avec des nouvelles données (104 images) possédant une meilleure représentation du renard (comparaison des exemples : c et d de la Figure 22). Cependant, même avec cet ensemble de données, les résultats du GAN n'ont pas réussi à générer des images ressemblant à des renards (image b de la Figure 22). Il semble que malgré nos efforts pour diversifier et améliorer la qualité des données, les caractéristiques complexes des renards n'ont pas été correctement capturées par le GAN.

Malgré ces résultats, nous avons quand même souhaité tester ces images avec le meilleur modèle pour la détection de renards.

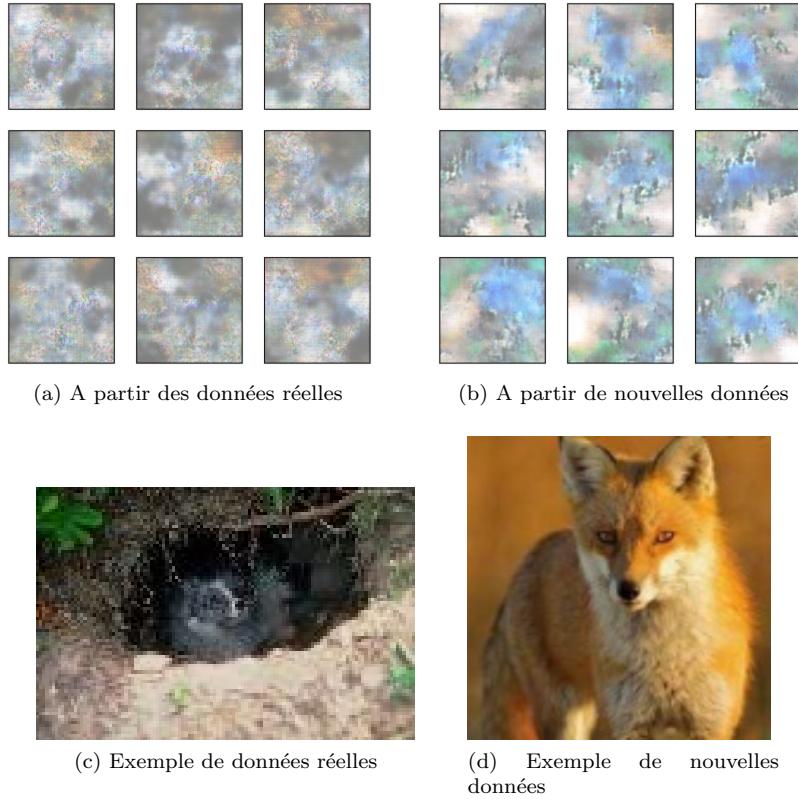


Figure 22: GAN avec les images de renard.

6.1 Prédiction sur les sorties du GAN

Après la génération de ces images, nous avons entrepris de les découper et de les redimensionner afin de les tester sur notre modèle. Une fois cette tâche accomplie, nous avons lancé le modèle sur ces images générées pour observer les résultats (voir figure 23).

```

Image : /content/drive/MyDrive/test_final/image_at_epoch_9980_part_2_2.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_0_0.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_0_1.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_0_2.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_1_0.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_1_1.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_1_2.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_2_0.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_2_1.png, Classe prédictive : 0
Image : /content/drive/MyDrive/test_final/image_at_epoch_9990_part_2_2.png, Classe prédictive : 0

```

Figure 23: Extrait des prédictions sur les photos générées

Interprétation des résultats : Le modèles à correctement classé la dernières images du GAN.

7 Modèle le plus complexe : contenu des sorties des CNN

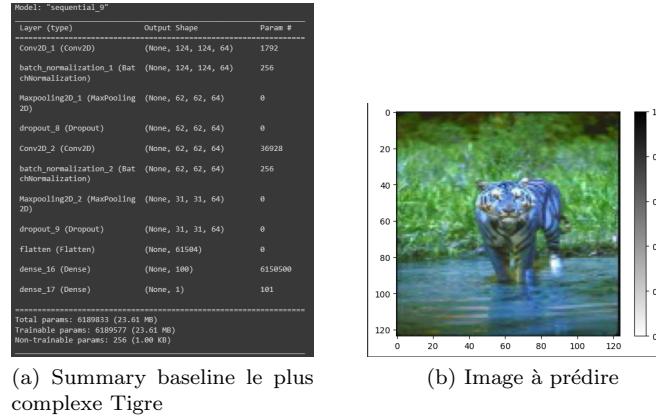


Figure 24: Modèle le plus complexe tigre.

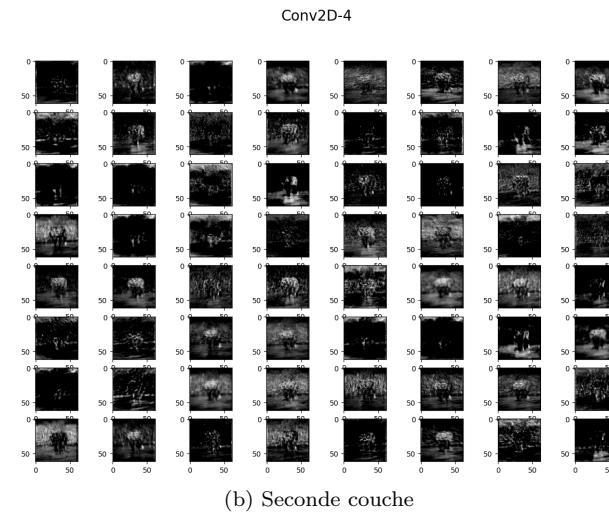
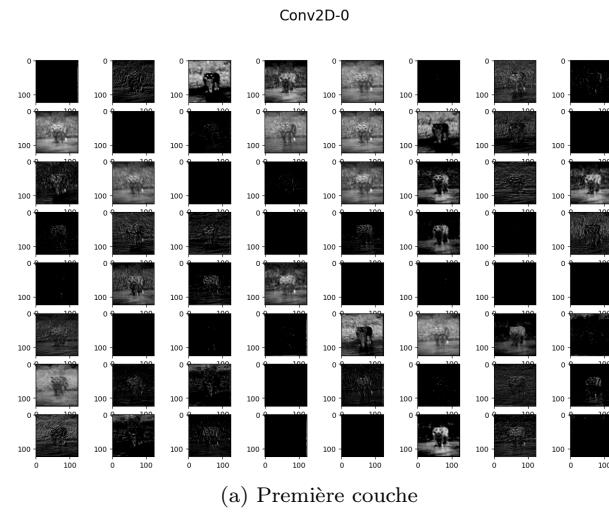


Figure 25: Résultats le plus complexe Tigre.

8 Conclusion

Ce projet met en évidence les défis liés à la capacité de nos modèles à bien se généraliser. Nous avons eu des difficultés à obtenir des résultats satisfaisants et à assurer une bonne généralisation de nos modèles.

Pour améliorer nos résultats, nous avons exploré plusieurs méthodes avancées telles que la régularisation (car le surapprentissage est omniprésent), la génération de données avec ImageDataGenerator et le transfert learning.

Globalement, nos résultats ont été satisfaisants bien que le surapprentissage persiste sur la globalité des modèles, nous avons pu trouver plusieurs modèles où la régularisation a fonctionné notamment grâce à des couches de Dropout et à du Kernel Regularization. Nous pensons qu'en introduisant davantage de diversité dans les données et en augmentant leur volume, nous pourrons obtenir de meilleurs résultats comme cela a été montré avec ImageDataGenerator (surtout pour le renard). Les résultats obtenus avec les modèles de Transfer Learning ont des scores très satisfaisants pour tous les animaux, cependant les loss ne sont pas aussi basses que souhaités, les courbes n'ont pas les allures attendues et l'overfitting est toujours présent.

Les meilleurs modèles de ce projet :

- Elephant : Base améliorée dropout + KernelRegularization
- Tigre : ImageDataGenerator 18 images générées et dropout
- Renard : ImageDataGenerator 20 images générées et dropout

La partie GAN a eu moins de succès étant donné le peu de données mis à disposition pour générer des données malgré un nombre d'epochs qui nous a paru important (10 000).

Cette expérience a été très enrichissante et bénéfique à notre apprentissage dans le deep learning. Certes beaucoup de nouvelles notions ont été introduites et difficiles à utiliser. Le temps d'exécution a freiné notre avancé sur de nombreux modèles. Il aurait été intéressant de faire plus d'expériences sur les GANs, notamment en utilisant un cGAN (conditional GAN) afin de pouvoir étiqueter les images des renards (corps, tête ou encore par races de renards).