

Universités de Montpellier

Faculté des Sciences de Montpellier

Département d'informatique

Machine Learning 2 - 2023/2024

Classification d'images



Réalisé Par :

Wided MEFLAH (22214601)

Encadré par :

Mr. Pascal PONCELET

Mr. Salim HAFID

1 Introduction	2
2 Modèle Baseline	2
2.1 Elephant vs autres animaux	3
2.2 Fox vs autres animaux	4
2.3 Tiger vs autres animaux	4
2.4 Analyse des résultats	4
3 Améliorations du modèle	5
3.1 Elephant vs autres animaux	5
3.2 Fox vs autres animaux	6
3.3 Tigre vs autres animaux	7
3.4 Analyse des résultats :	8
4 Image Data Generator	8
4.1 Elephant vs autres animaux	8
4.2 Fox vs autres animaux	9
4.3 Tiger vs autres animaux	9
4.4 Analyse des résultats	10
5 Approche Transfer Learning	10
5.1 Elephant vs autres animaux	12
5.2 Fox vs autres animaux	12
5.3 Tiger vs autres animaux	13
5.4 Analyse des résultats	13
6 GAN	14
6.1 Mise en place	14
6.2 Analyse des résultats	14
6.3 Prédiction	15
7 Visualisation des features d'un modèle	15
8 Coloration d'images	17
8.1 Mise en place	17
8.2 Analyse des résultats	18
9. Conclusion	18

1 Introduction

Dans le cadre de ce projet, nous devons créer trois modèles qui classifient des images en fonction d'un animal donné en se basant sur les trois jeux de données fournis :

- Éléphant vs. autres animaux
- Renard vs. autres animaux
- Tigre vs. autres animaux

Pour commencer l'étude, nous avons instinctivement utilisé des méthodes classiques vues en M1 en lançant le classifieur SVM sur le jeu de données tigre, Le traitement des données était plus facile étant donné que ces dernières sont des images, et nous avions simplement à prendre la matrice des pixels de l'image et l'aplatir.

Cependant, même avec les hyperparamètres, la précision n'excède pas la barre des 62%, ce qui est quasiment de l'ordre d'une classification aléatoire. Nous en avons donc conclu que les méthodes classiques ne sont pas assez performantes dans la classification d'images. Nous devons donc passer à la puissance supérieure en termes d'extraction de caractéristiques avec les CNN.

2 Modèle Baseline

Dans cette section, nous présentons le modèle baseline utilisé pour les trois jeux de données, ainsi que les résultats obtenus.

Le réseau de neurones choisi est composé d'une seule couche de convolution combinée au Max-Pooling de taille 2x2 pour l'extraction des caractéristiques.

Nous avons choisi des filtres de taille 3x3 ,L'activation choisie est ReLU.

Après le flatten, nous avons utilisé une couche dense avec une activation sigmoid.

La figure suivante illustre le schéma du réseau :

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(124, 124, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

Nous avons lancé le modèle baseline sur les trois jeux de données ,un ensemble de tests étaient effectués afin de :

- choisir le nombre d'epoch optimale qui correspond au point où les performances convergent et commencent à se détériorer sur l'ensemble de validation.
- choisir la valeur de batch size qui offre un bon compromis entre la vitesse d'entraînement et les performances du modèle.
- choisir la valeur de learning rate adapté qui permet la mise à jour des poids la plus adaptée.

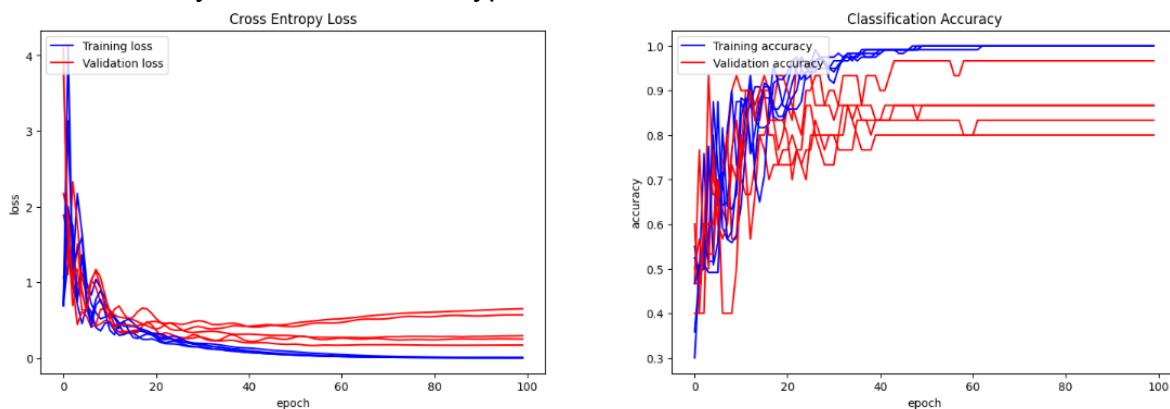
A la fin des tests nous avons utiliser les paramètres suivants:

- nb-epoch : 100
- k-fold : 5
- batch size : 64
- optimiseur : adam
- learning_rate : 0.001

Ci-dessous, les résultats obtenus :

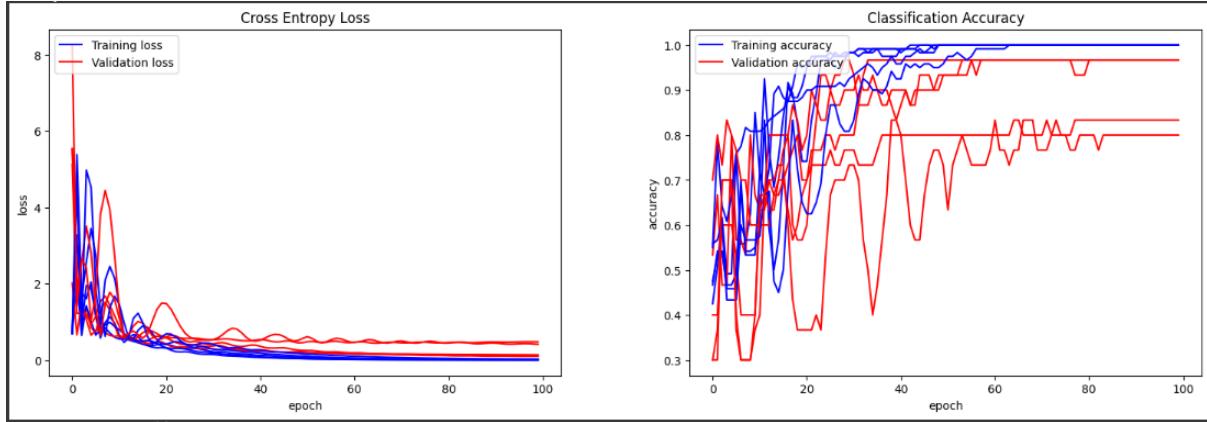
2.1 Elephant vs autres animaux

Jeu de données : Kfold sur les données de base — Modèle : baseline model — value-Accuracy: 86.667% , écart-type : 8



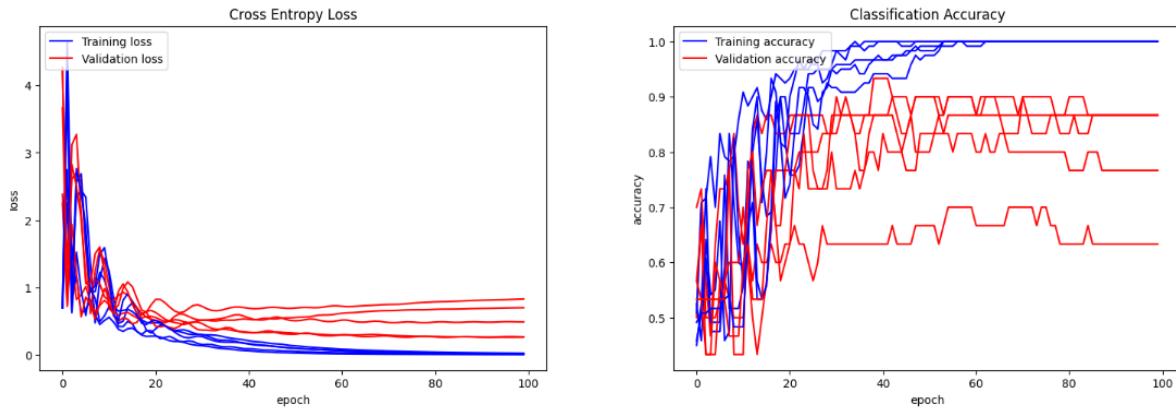
2.2 Fox vs autres animaux

Jeu de données : Kfold sur les données de base — Modèle : baseline model — value-Accuracy : 87% , écart-type : 9.189



2.3 Tiger vs autres animaux

Jeu de données : Kfold sur les données de base — Modèle : baseline model — value-Accuracy : 80% , écart-type = 6.992



2.4 Analyse des résultats

Nous remarquons que le modèle de base présente une bonne accuracy sur les trois jeux de données. Cependant, nous observons également des différences entre les trois ensembles, montrant que Fox est le plus performant, suivi par Elephant, puis Tiger. Cette disparité est justifiée par le niveau de difficulté variable des environnements des photos dans chacun des jeux de données.

Autrement, en examinant les courbes de loss ainsi que celles de l'accuracy, nous constatons que le modèle présente un overfitting. Les courbes de validation

montrent une augmentation de la perte, tandis que les courbes d'entraînement tendent vers 0 pour la perte et 1 pour la précision. Nous avons modifié nos paramètres avec des valeurs significatives mais le problème persiste toujours.

Bien que le modèle de base présente une bonne accuracy, il nécessite des améliorations pour surmonter ce surapprentissage.

3 Améliorations du modèle

Afin de progresser et d'améliorer les résultats obtenus précédemment par le modèle baseline, nous avons testé différentes approches.

3.1 Elephant vs autres animaux

Nous avons commencé par tenter d'améliorer la partie d'extraction des caractéristiques de nos CNN en expérimentant avec plusieurs combinaisons de couches Conv2D et MaxPooling2D, en utilisant différentes configurations de paramètres observées dans divers exemples. Après analyse, nous avons choisi d'adopter un schéma comportant deux couches de convolutions pour que notre modèle apprenne plus de features, suivies chacune d'une couche de max pooling.

et afin de surmonter le surapprentissage, nous avons décidé d'introduire une couche de Dropout avec un taux de 0.5. Cela a pour objectif de forcer le réseau à ne pas trop dépendre de caractéristiques spécifiques apprises par certains neurones, en effectuant une suppression aléatoire de certains neurones dans différentes couches de notre modèle.

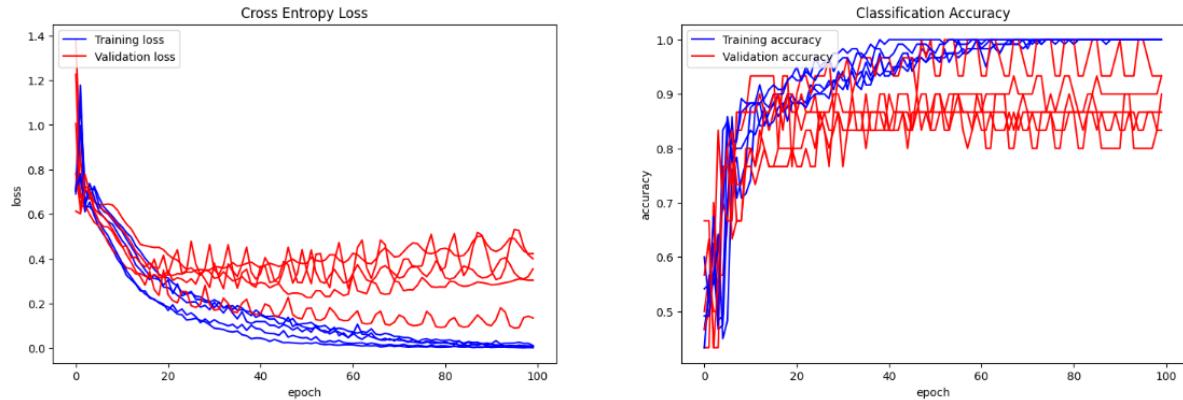
La figure suivante illustre le schéma du réseau :

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(124, 124, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(2, activation='sigmoid'))
```

Nous avons essayé la combinaison de plusieurs paramètres et les paramètres déjà définis pour le modèle de base semblent donner les meilleurs résultats.

Ci-dessous, les résultats obtenus :

Jeu de données : K-fold sur les données de base — Modèle : elephant_model — value-Accuracy : 89.333 % , écart-type : 5.578

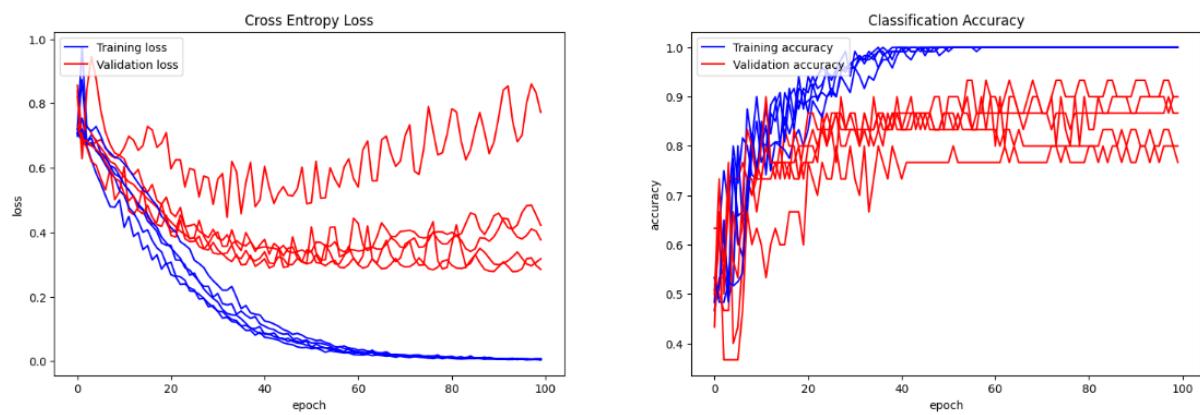


3.2 Fox vs autres animaux

Tout comme le modèle Eléphant, nous avons procédé étape par étape pour tenter de réduire au mieux le surapprentissage tout en maximisant l'accuracy. Cependant, nous avons fini par utiliser les mêmes couches et paramètres que le modèle Éléphant.

Ci-dessous, les résultats obtenus :

Jeu de données : Kfold sur les données de base — Modèle : fox_model — value-Accuracy : 90.667% , ecart-type: 6.325



3.3 Tigre vs autres animaux

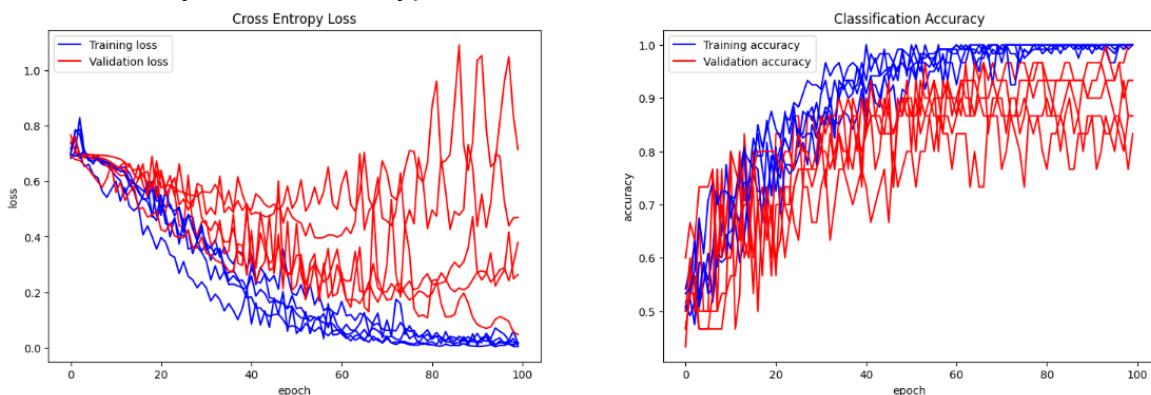
Tout d'abord, nous avons essayé la même approche que celle utilisée pour le modèle Fox et Elephant, en pensant que les mêmes couches fonctionnaient pour nos trois jeux de données, mais sans succès. Nous avons donc dû mener des expérimentations et avons remarqué une légère amélioration en utilisant trois couches de convolution, deux couches de dropout et deux couches denses et en utilisant early stopping qui permet d'arrêter le processus d'entraînement dès qu'il est détecté que les performances de notre modèle sur l'ensemble de validation commencent à se détériorer.

La figure suivante illustre le schéma du réseau :

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(124, 124, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.7))
model.add(Dense(1, activation='sigmoid'))
```

Ci-dessous, les résultats obtenus :

Jeu de données : Kfold sur les données de base — Modèle : tiger_model — value-Accuracy: 85% , écart-type : 5.735



3.4 Analyse des résultats

Nous observons une légère amélioration des performances par rapport aux modèles de base.

Cependant malgré plusieurs tests et en raison du manque de données, nous n'avons pas réussi à éliminer le phénomène de surapprentissage, l'écart entre la perte de validation et d'entraînement est toujours plus grand malgré l'utilisation des techniques de régularisation notamment le dropout et le early stopping.

On a constaté que le dropout permet simplement de ralentir le surapprentissage, mais pas de l'éliminer.

Dans la suite de notre analyse, nous présentons l'utilisation du générateur de données dans le but d'améliorer le modèle et de résoudre le problème d'overfitting.

4 Image Data Generator

Afin de remédier au manque de données, nous avons mis en place un générateur de données basé sur les données existantes. Il fait partie de la bibliothèque Keras et s'intègre facilement dans notre code. Avec ce générateur, nous n'utilisons plus la technique de k-fold puisque les données d'entraînement sont toujours différentes.

Le jeu d'entraînement est généré automatiquement lors de l'appel de la méthode 'fit' pour entraîner le modèle.

Chaque image du jeu d'entraînement est modifiée selon les paramètres suivants :

```
datagen_DG = ImageDataGenerator( width_shift_range=0.2,  
                                height_shift_range=0.2, rotation_range=20,  
                                horizontal_flip=True)
```

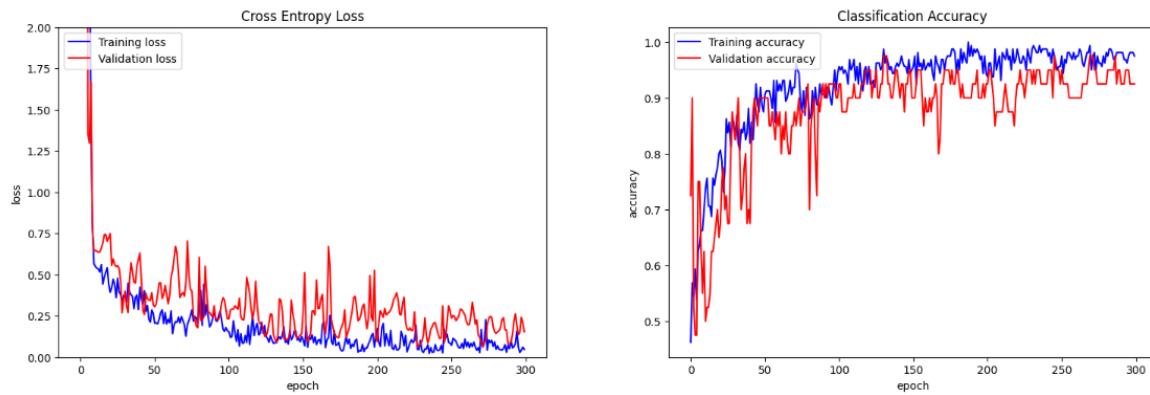
Les résultats n'étaient pas très satisfaisants avec le modèle baseline, mais avec les modèles améliorés de suite, le surapprentissage est grandement réduit.

En effectuant quelques modifications du taux d'apprentissage, nous obtenons de bons résultats sans modifier les modèles précédents.

4.1 Elephant vs autres animaux

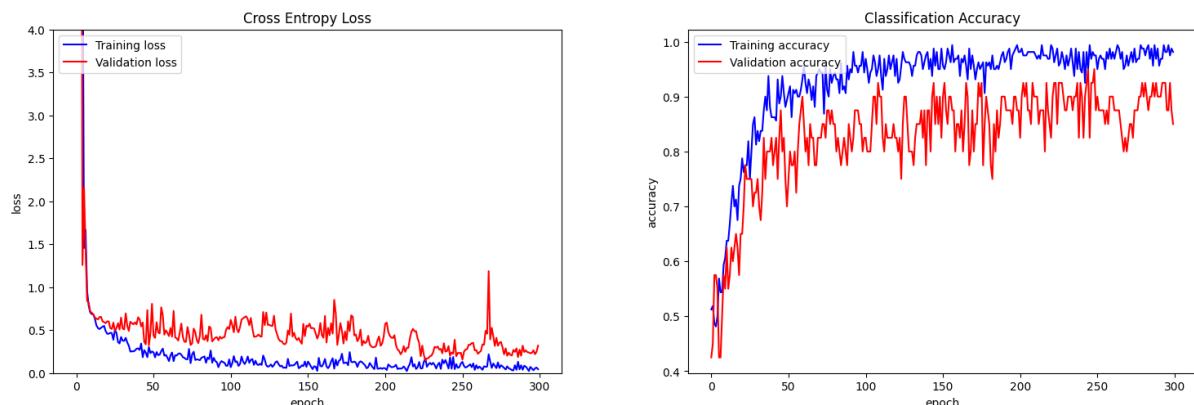
Jeu de données : Data generator — Modèle : elephant_model —

value-Accuracy : 92.5%



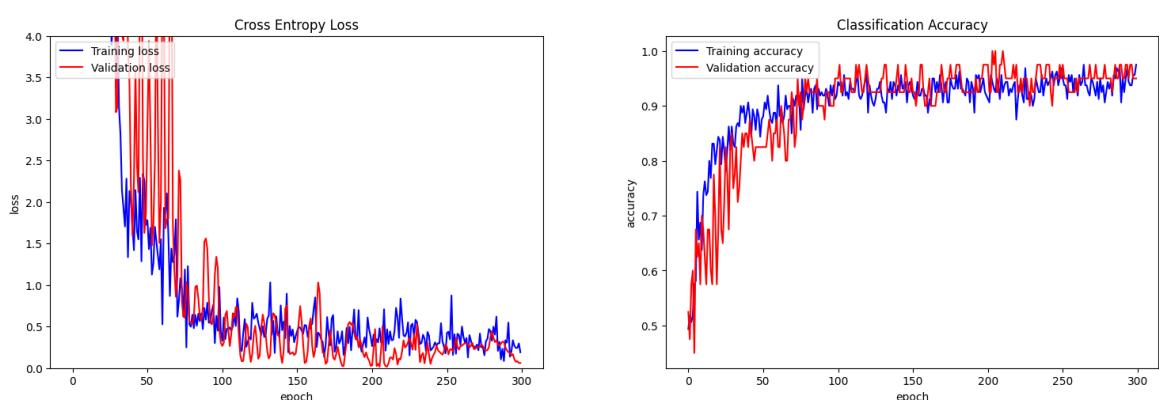
4.2 Fox vs autres animaux

Jeu de données : Data generator — Modèle : fox_model — value-Accuracy : 90%



4.3 Tiger vs autres animaux

Jeu de données : Data generator — Modèle : tiger_model — value-Accuracy : 95%



4.4 Analyse des résultats

L'augmentation des données générées par le Data Generator a aidé à réduire le surapprentissage pour les différents jeux de données , on voit très bien que l'écart entre la perte de validation et d'entraînement diminue pour les 3 modèles ,et cela en exposant les modèles à une plus grande variété de données et en les empêchant de mémoriser simplement les exemples d'entraînement.

Cependant, on constate que les courbes présentent des pics importants et que l'apprentissage est très lent. Pour améliorer les performances de cette classification, nous pourrions opter pour le transfert learning.

5 Approche Transfer Learning

Dans cette section, nous présentons notre approche du transfert learning appliquée à nos données. Nous avons utilisé le modèle pré-entraîné ResNet50.

Étant donné que nous avions identifié un problème de manque de données précédemment, dans cette section, nous avons également généré des images à l'aide de notre générateur de données sur lesquelles le transfert learning sera appliqué.

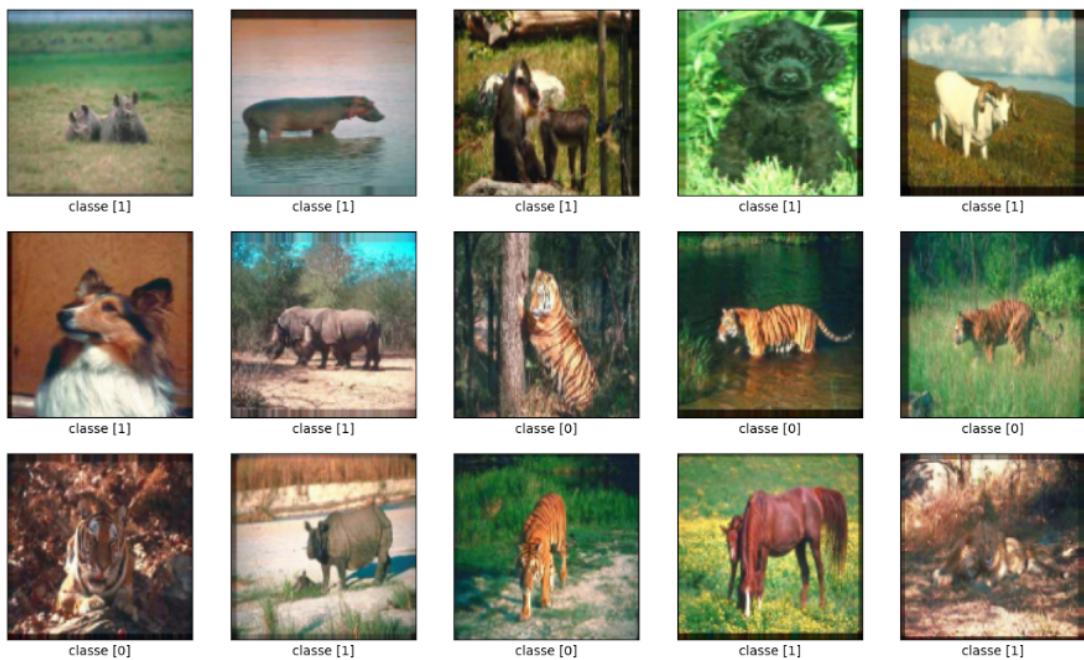
Initialement, nous avons pris le modèle ResNet50 en gelant les poids de toutes ses couches. De plus, plusieurs couches ont été ajoutées dans la partie classification dans l'espoir qu'elles s'adaptent mieux à nos images. Cependant, cela a conduit à des valeurs d'accuracy peu satisfaisantes.

Le premier problème majeur provient du fait que nous avions conservé les paramètres du générateur de données précédent. Comme ResNet50 a été entraîné sur un large ensemble de données réelles, les images déformées par le générateur ne semblaient pas toujours correctes, ce qui a introduit de nombreux défauts dans l'apprentissage. De plus, le fait d'ajouter plusieurs couches de convolution a complexifié notre modèle.

Pour remédier à ces deux problèmes, nous avons d'abord testé plusieurs paramètres pour construire un nouveau générateur de données qui renvoie des images similaires à celles du jeu de données d'origine. Après plusieurs tests, nous avons stabilisé au générateur de données suivant:

```
datagen_TL = ImageDataGenerator(zoom_range=0.1, fill_mode="nearest",
horizontal_flip=True)
```

Voici un exemple de ce que cela donne :



Ensuite, nous avons simplifié notre modèle en ne gardant qu'une simple couche dense pour la classification parce que nous avons constaté que les couches de transfert learning sont entraînées sur des objets plus généraux, ajouter des couches pour les rendre plus spécifiques à nos images diminue les performances du modèle. Les résultats étaient déjà bien meilleurs, avec une accuracy pour la validation dépassant 0.95 pour les trois jeux de données et beaucoup moins de pics dans les courbes.

Voici le modèle final, ainsi que les courbes obtenues sur les trois jeux de données avec ce modèle :

```
input_t = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
res_model = ResNet50(include_top=False, weights='imagenet', input_tensor=input_t)

for layer in res_model.layers[:143]:
    layer.trainable = False

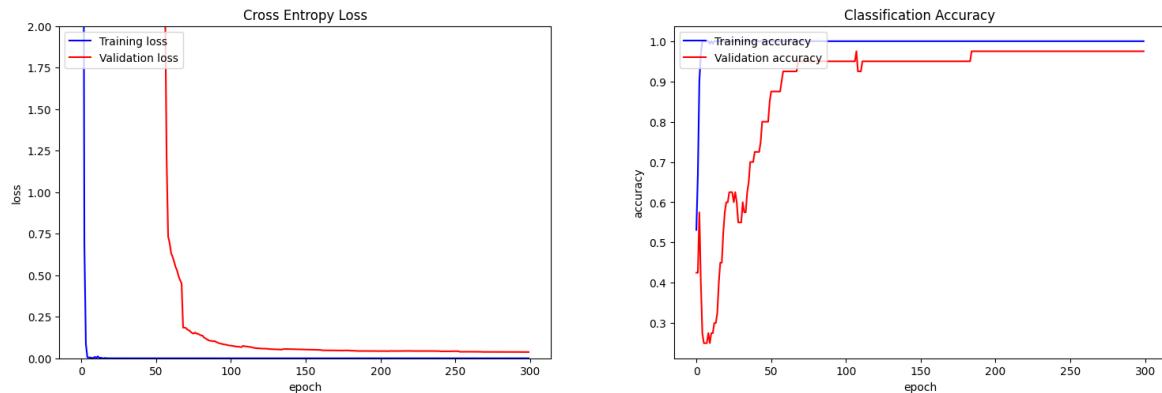
model = tf.keras.Sequential()
model.add(res_model)
model.add(Flatten())

model.add(Dense(1, activation='sigmoid'))
```

5.1 Elephant vs autres animaux

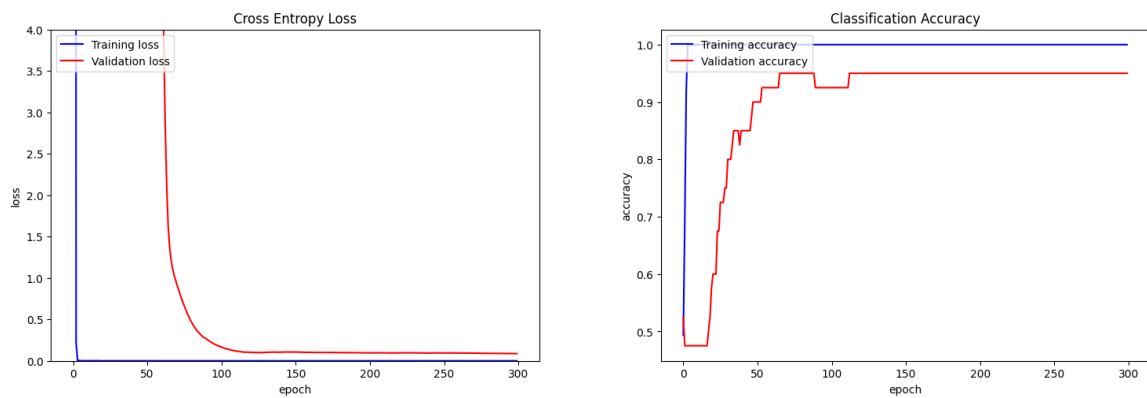
Jeu de données :Data_generator_TL — Modèle : model_TL —

value-Accuracy : 97.5%



5.2 Fox vs autres animaux

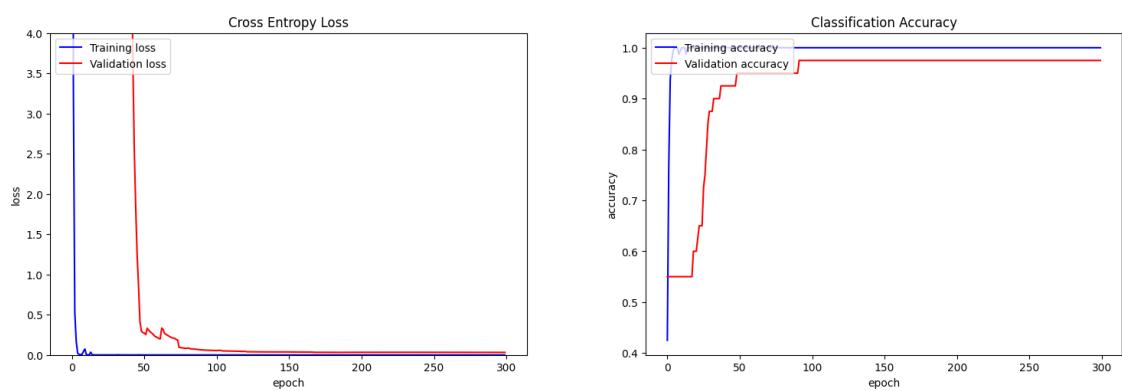
Jeu de données :Data_generator_TL — Modèle : model_TL — value-Accuracy : 95%



5.3 Tiger vs autres animaux

Jeu de données :Data_generator_TL — Modèle : model_TL —

value-Accuracy : 97.5%



5.4 Analyse des résultats

On peut remarquer que la perte pour la validation est à zéro pour les 3 jeux de données à partir de nombre de poches = 100 on peut bien faire un early stopping pour accélérer le processus.

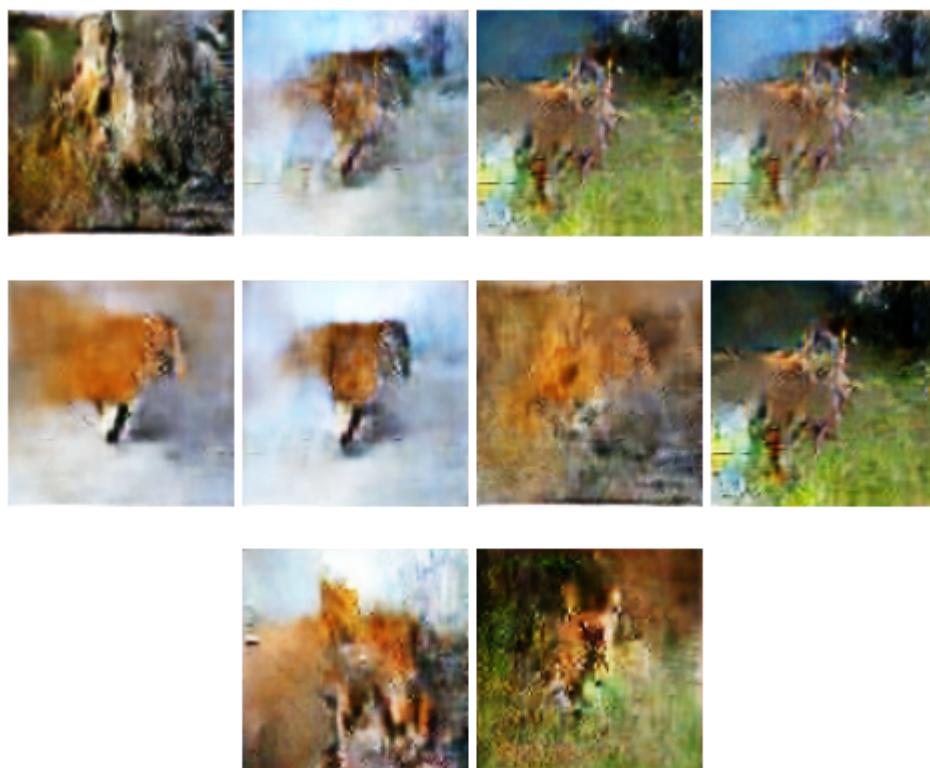
Cette approche du transfert d'apprentissage s'est avérée hautement efficace en améliorant considérablement les performances, en accélérant significativement le processus d'entraînement, en garantissant l'utilisation de plusieurs couches sans dégradation de performance tout en réglant efficacement le problème de surapprentissage.

6 GAN

6.1 Mise en place

Nous avons suivi le notebook sur le GANet et utilisé notre ensemble de données Fox. Cependant, les premiers résultats n'étaient pas très satisfaisants. Après une analyse approfondie, nous avons constaté qu'il faudrait un jeu de données beaucoup plus vaste, cela permettrait de rendre les caractéristiques reconnues par le discriminateur moins facilement reproduisibles par le générateur. Par conséquent, nous avons utilisé un data generator pour augmenter la taille du jeu de données.

Les résultats sont les suivants :



6.2 Analyse des résultats

On constate que les images générées contiennent des formes de renard, cependant, elles sont floues. Cela est dû à la diversité des images utilisées, on observe des images de renards sur la neige, d'autres dans l'herbe, et cela impacte l'entraînement en le perturbant. De plus, on remarque que cela a également un impact sur le temps d'exécution, qui est devenu très lent.

6.3 Prédiction

Nous avons d'abord choisi 10 images arbitrairement parmi les dernières sauvegardées par le GAN. Nous les avons regroupées dans un dossier, ajouté à son tour dans le dossier du jeu de données initial. Cela nous permet de les réutiliser dans notre notebook avec l'un des modèles sauvegardés après l'entraînement.

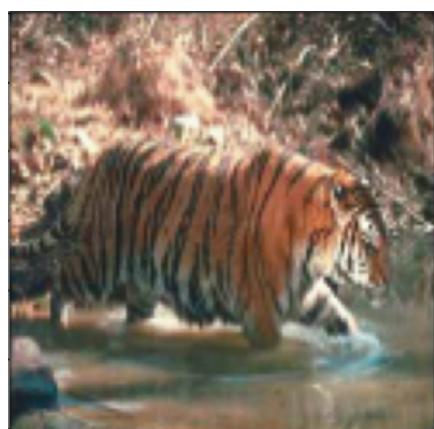
Pour le modèle, nous avons choisi le modèle qui a donné les meilleurs résultats sur le jeu de données de fox, à savoir le modèle model_TL entraîné avec le générateur de données.

Le modèle a réussi à prédire correctement toutes les images, ce qui démontre la performance de notre modèle et du GAN mis en place.

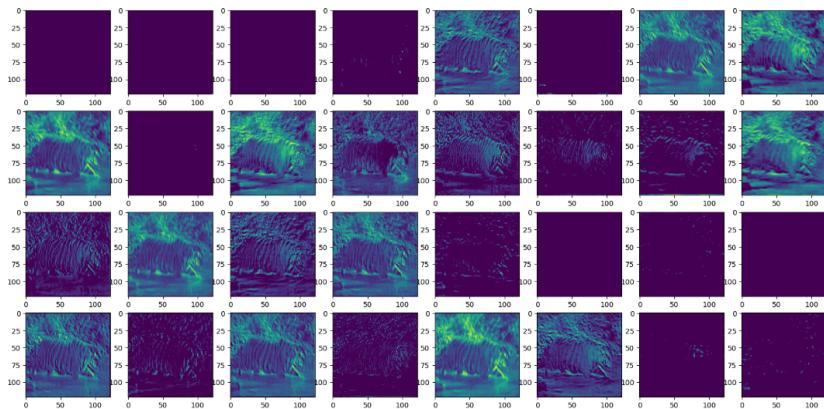
7 Visualisation des features d'un modèle

Notre modèle le plus complexe est le tiger_model, qui se compose de trois couches de convolutions.

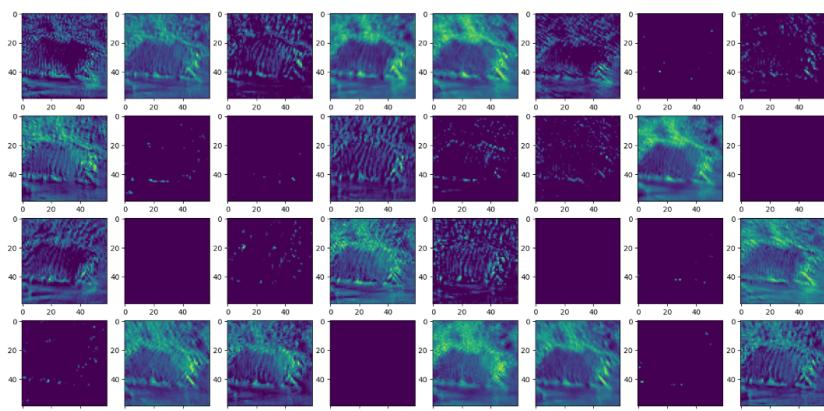
l'image à prédire est la suivante :



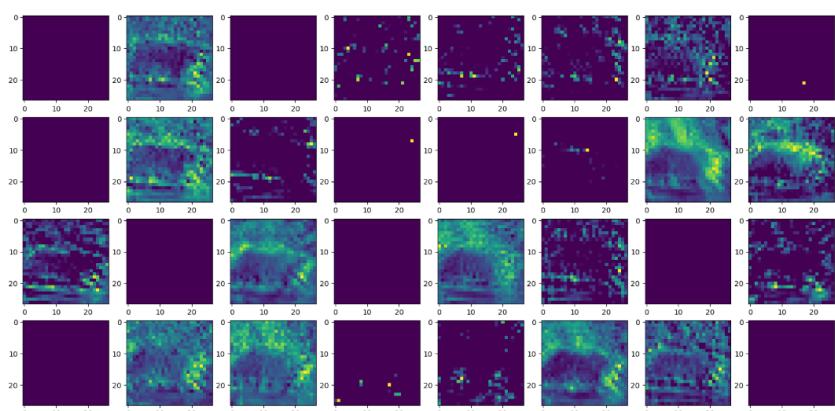
Sortie de la première couche de convolution :



Sortie de la deuxième couche de convolution :



Sortie de la troisième couche de convolution :



On constate que les sorties des trois couches de convolutions sont différentes et on voit que lors de la dernière couche, le modèle réussit à détecter des motifs qui ressemblent au tigre.

8 Coloration d'images

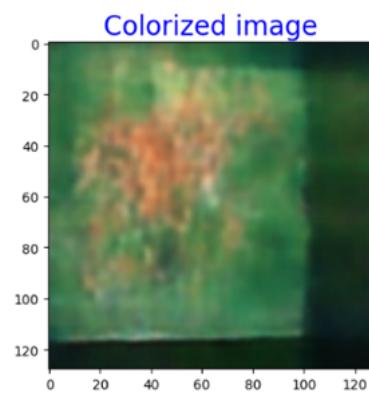
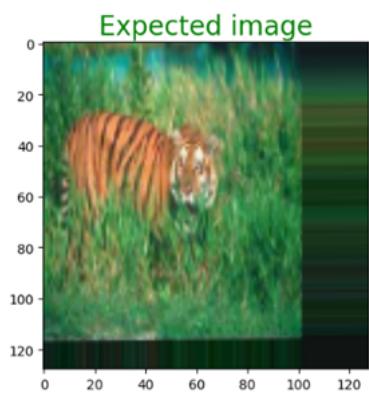
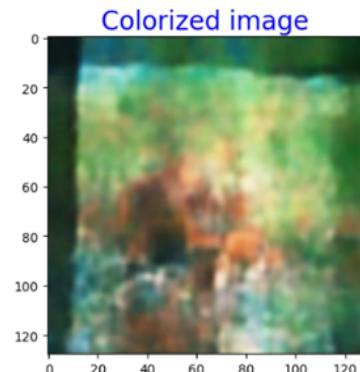
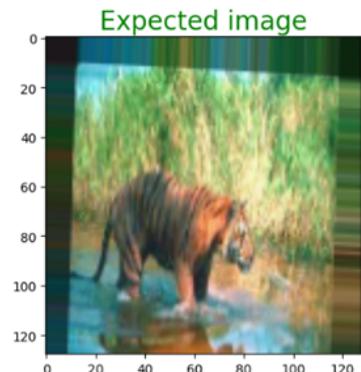
8.1 Mise en place

Au début nous avons implémenté l'encodeur avec des couches Conv2D successives empilées pour réduire la dimension de l'entrée tout en extrayant des caractéristiques importantes. Le vecteur latent est ensuite généré à partir de ces caractéristiques. Pour le décodeur, le processus est inversé avec des couches Conv2DTtranspose, qui effectuent l'opération inverse .

Cependant, les résultats obtenus étaient peu concluants. Ayant constaté les avantages du data generator dans les étapes précédentes, notamment en termes de réduction de la perte du modèle, nous avons décidé de nous pencher dans cette direction.

nous avons généré des images avec datagenerator, les transformer en gris pour les utiliser pour l'entraînement avec les images RGB et nous avons utilisé notre jeu de données tigre pour la validation (en gris).

Voici un exemple des résultats obtenus



8.2 Analyse des résultats

On constate que l'autoencodeur parvient à détecter les couleurs principales, mais on voit que les images sont floues. Cela est justifié par la diversité des images utilisées pour l'entraînement, appartenant aux différentes classes parmi les trois, et même des images n'appartenant à aucune classe. Cela a perturbé le modèle, qui tente de corriger les couleurs à chaque fois qu'il voit une image très différente de celles qu'il a déjà rencontrées.

9. Conclusion

Le modèle baseline et les modèles améliorés sur le dataset de 200 ont affiché une bonne accuracy mais présentaient un problème d'overfitting. Après de nombreux tests et ajustements de paramètres, nous avons conclu que le overfitting était attribuable à la petite taille du dataset, insuffisante pour des résultats plus pertinents. Cela nous a conduits à réaliser de nouveaux tests avec le générateur d'images, où nous avons observé une différence remarquable dans les résultats. Plus il y a de données, plus le modèle a la possibilité de devenir performant.

Ensuite, nous avons effectué des tests avec le transfert learning en utilisant le modèle ResNet50, où nous avons constaté une meilleure performance pour cette classification après plusieurs ajustements de paramètres.

Le visualiseur des features que nous avons mis en place nous a également permis de mieux comprendre ce que le modèle voyait.

En ce qui concerne la coloration d'images avec les auto-encodeurs, nous avons pu utiliser le générateur de données comme en témoignent les résultats satisfaisants obtenus.

Le problème majeur était d'avoir réalisé le projet sur une seule machine, avec les limitations en termes de ressources rencontrées dans Google Colab.

À part cela, à travers ce projet nous constatons que le machine learning n'est pas une science exacte et qu'il n'existe pas de recette magique pour créer un modèle performant. C'est une discipline qui repose sur l'expérimentation, mais où il est essentiel de savoir lire et interpréter les résultats fournis par un modèle afin de pouvoir l'améliorer au maximum. Nous retenons comme leçon l'importance de la taille du jeu de données, qui représente la problématique majeure de l'état actuel du machine learning.