1. Introduction

Au cours des dernières années, les progrès dans le domaine de l'apprentissage se sont avérés fulgurants de telle sorte que les modèles génératifs ont suscité beaucoup d'intérêt.Ainsi les réseaux génératifs ont se sont dotés d'une grande capacité de production d'éléments de contenus telles que les images, les textes pour ne citer que ceux et ceci en s'appuyant sur des architectures bien déterminées, de grosses quantités de données et des systèmes tutoriels intelligents.Un de ses réseaux génératif est connu sous le nom d'auto encoder.Les auto-encodeurs sont des réseaux de neurones qui tentent d'apprendre un mappage compressé à partir de l'entrée. Pour ce faire, il force d'abord l'entrée vers un goulot d'étranglement d'informations (encodeur), puis tente de recréer l'entrée d'origine à partir de la représentation compressée (décodeur).S'inscrivant dans la même logique le but de notre travail et de concevoir une architecture d'auto-encodeur capable comme son nom l'indique d'encoder et de reconstruire les images de deux espèces d'animaux à savoir les éléphants et les vaches et ensuite d'appliquer un SVM sur les résultats issues de l'encodeur c'est-à-dire le ***embedding*** dans le but de classifier les images des vaches et des éléphants.A cet effet un template a été fourni pour nous diriger dans notre travail.Il s'agit d'un code qui conçoit une architecture d' auto-encodeur sur les images des chiffres 2 et 7 sur les ***données de MNIST.***Le code en question est constitué d'un fichier ***Modele.py*** chargé de la conception de l'architecture et de l'entraînement de l'architecture sur les données d'entraînement prévues à cet effet et d'un second fichier ***Evaluation.py*** dans lequel l'architecture est d'abord soumis a des données de test sur lesquelles les images doivent être reconstruites.Le but de notre travail est d'abord de partir sur la base de ce qui a deja ete fait dans les deux fichier et concevoir notre architecture puis le tester et ensuite récupérer les données issues du passage des images dans l'encodeur (***embedding***) pour y appliquer un SVM après l'application du SVM sur les images originales, après visualiser le embedding en deux dimensions dans un scatter plot et finalement conclure.

2. Montage de l'architecture et entraînement du modèle

2.1. Ensemble de données

L'ensemble de données faisant l'objet de notre travail est un ensemble de 2800 images provenant de deux espèces d'animaux différents contenues chacun dans un dossier contenant le nom de l'espèce et proportionnellement répartis en données d'entraînement contenant 2400 images pour toutes espèces confondus et de test contenant 400 images.

Il est important de préciser que la partie entraînement de notre modèle se fait à deux niveau: une partie sur les données d'entraînement et une autre partie sur les données de validation.Il convient donc de séparer nos données en données d'entraînement qui prennent 2160 images soit 90% des données d'entraînement proprement dit et en données de validation qui prennent 240 images soit 10% des données d'entraînement.

	Vaches	Eléphants
Données d'entraînement	1200	1200
Données de test	200	200

2.2. Traitement des données

Dans le cadre du traitement,étant donné que les données de notre TP sont des images, il convenait alors de faire une configuration des images notamment de la mise à l'échelle(scaling) de hauteur et de largeur à 120, fixer le nombre de canaux a 3 pour obtenir les composantes rouge vert et bleu et le mode de couleur en rgb afin que toutes les images puissent avoir les mêmes caractéristiques au cours de leur passage dans l'autoencodeur et ce pour faciliter l'apprentissage.Aussi les couleurs sont tres important pour la reconstruction des images parce que comme nous pouvons le constater par exemple les vaches n'ont pas la meme couleur que les elephants. Nous avons aussi essayé l'augmentation de données notamment l'horizontal flip, shear_range, zoom_range, width shift_range, height shift range qui a réussi à augmenter le temps d'entraînement du réseau de neurone compte tenu de la transformation sur toutes les images qui constitue les données d'entraînement de notre jeu de données avant de passer dans notre architecture pour la phase d'entraînement.

2.3. Paramètres et hyperparamètres

2.3.1. L'optimiseur

Pour notre modèle l'optimiseur finalement retenu a été l'optimiseur ADAM compte tenu de sa vitesse ainsi que de sa qualité de convergence vers une solution optimale.Aussi avec l'hyper-paramètre du taux d'apprentissage fixe a 0.001 nous remarquons que le modèle met moins de temps à converger vers le minimum global.

2.3.2. La taille du lot (batch_size) d'entraînement

Dans l'optique de gagner en terme de temps d'apprentissage et de précision du modèle nous avons eu à essayer plusieurs paramètres en ce qui concerne la taille du lot.Nous avons essayé avec des tailles de lot (batch_size) de 20, 32, 64 et nous en sommes arrivés à la conclusion que plus la taille du lot est petite plus le modèle gagne en précision et en temps d'apprentissage.Nous avons donc jugé bon de choisir une taille de lot de 32 ce qui s'avère intuitivement être un bon compromis entre précision et temps.A noter qu'avec une taille de lot de 32 nous avons obtenu la meilleure précision vis-à-vis de notre modèle.

2.3.3. Le nombre d'époques

En ce qui concerne le nombre d'époques nous avons eu à le fixer à 100 parce que cela permet au modèle de s'améliorer sur 100 passages en entier sur les données d'entraînement.

2.4. Architecture

Comme un auto encodeur normal notre auto encodeur convolutif est subdivisé en deux parties notamment une première partie constituant l'encodeur définie dans le fichier modèle par la fonction ***encoder*** et la partie décodeur définie dans le même fichier par la fonction ***decoder.***

2.4.1. L'encodeur (encoder)

Cette première partie contient différentes couches ayant pour but d'extraire les caractéristiques propres à chaque image en les compressant de manière à réduire leur taille initiale.En résumé l'image fournie en entrée passe par une succession de filtres et il en résulte des descripteurs appelés ***embedding***.

La couche de convolution: Cette couche a pour rôle d'analyser toutes les images fournies en entrée et d'en relever un ensemble de caractéristiques appelle feature map qui nous indique exactement ou est situe notre caractéristique dans l'image.Pour cette partie nous avons eu à utiliser des filtres de taille respectivement de 32 , 64 et 128.

La couche de correction ou couche Leaky Relu: cette couche fait intervenir une fonction d'activation , la fonction Leaky ReLu dans notre cas dans le but d'améliorer l'efficacité du traitement tout en intercalant entre les couches de traitement.Il est aussi tres important d'utiliser la fonction ***Leaky Relu*** parce qu'elle est populaire pour les tâches génératives et contrairement à une fonction Relu simple il résout le problème de Relu mort***(dying Relu)***.

La couche de Pooling ou Downsampling: cette couche se base sur les résultats obtenus de la couche de convolution.Son but est de recevoir en entrée les features maps et de réduire la taille tout en gardant les caractéristiques importantes de l'image.Cette couche converse la valeur moyenne de la fenêtre de filtre.

La couche dropout: Cette technique consiste à désactiver ou éteindre aléatoirement un neurone pendant la phase d'apprentissage et ceci dans le but de permettre à chaque neurone de bien apprendre évitant ainsi la co-adaptation.

L'architecture telle que décrite a été utilisée **trois fois** dans cette partie de l'architecture.Le résultat qui en résulte a donc été passé en paramètre à la deuxième partie de notre architecture d'auto encodeur convolutif.

2.4.2. Le décodeur (decoder)

Cette partie se charge de reconstruire l'image originale en recevant en paramètre les caractéristiques extraites et encodées au niveau de l'encodeur.

La couche de convolution: Cette couche a pour rôle d'analyser toutes les images fournies en entrée et d'en relever un ensemble de caractéristiques appelle feature map qui nous indique exactement ou est situe notre caractéristique dans l'image.Pour cette partie contrairement à la partie encodeur nous avons eu à utiliser des filtres de taille respectivement de 128, 64 et 32.

La couche de correction ou couche Leaky Relu: cette couche fait intervenir une fonction d'activation , la fonction ReLu dans notre cas dans le but d'améliorer l'efficacité du traitement tout en intercalant entre les couches de traitement.

La couche de suréchantillonnage(Upsampling): cette couche est à l'opposé de la couche de ***Pooling*** au niveau de ***l'encodeur*** en ce sens que au lieu de réduire la taille de l'image tout en gardant les caractéristiques importantes cette couche au contraire augmente la taille de l'encodeur l'image par une répétition des pixels avec un filtre 2 par 2 et ce dans le but d'augmenter la taille de l'image.

La couche dropout: Cette technique consiste à désactiver ou éteindre aléatoirement un neurone pendant la phase d'apprentissage et ceci dans le but de permettre à chaque neurone de bien apprendre évitant ainsi la co-adaptation.

2.5. Affichage des résultats d'entraînement

L'entraînement de notre auto encodeur convolutif s'est avéré assez concluant.Rappelons que l'entraînement des auto encoder rentre dans le cas de l'apprentissage non supervisé.Les résultats dans ce cas sont axés vers l'image elle-même le but étant de réduire autant que possible l'erreur de reconstruction c'est-à-dire faire en sorte que l'image reconstruite ressemble le plus possible à l'image originale.Ceci étant les résultats de notre modèle d'auto encodeur convolutif s'évaluent à travers l'erreur minimale commise uniquement.

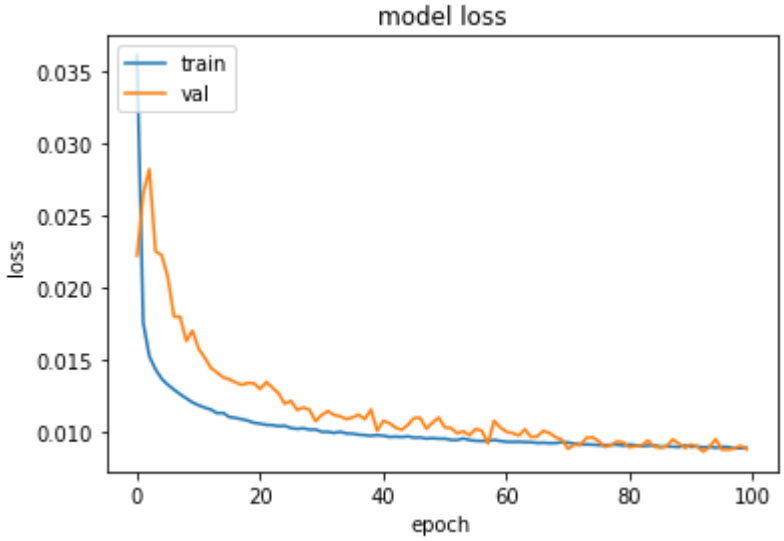
2.5.1. Le temps total d'entraînement en minutes

Notons que l'entraînement de notre modèle a été effectué avec une taille de lot de(batch_size) 32 et ce sur 100 époques(epocs) ceci pour permettre au modèle de réduire autant que possible l'erreur minimale commise.Le temps mis a cet effet pour entraîner notre modèle est de 7 minutes en raison de 4 secondes et 60 millisecondes en moyenne d'entraînement par époque.Cependant il convient de notifier que ces résultats ont été obtenus à partir de la ***version Pro de Google Colab***.

2.5.2. L'erreur minimale commise lors de l'entraînement sur les données d'entraînement et de validation

L'erreur minimale commise pour les données d'entraînement sur notre modèle est de 0.0089 et sur les données de validation elle est de 0.0086.Ci-dessous

le comportement de l'erreur minimale sur les données d'entraînement et de validation.



2.6. Justification du choix de l'architecture

2.6.1. Processus suivi pour l'aboutissement des meilleurs résultats

Le processus suivi pour réaliser notre architecture est assez simple.Nous sommes partis sur la base des exemples fournis a cet effet à savoir l'implémentation des autoencoder sur les jeux de données de MNIST.

Aussi s'agissant d'un auto encodeur convolutif les couches intervenant dans réseaux de neurones convolutifs ont beaucoup été mis en évidence.Pour ce faire nous nous sommes inspirés pour la partie encodeur en plus de l'implémentation sur le jeu de données de MNIST d'un travail pratique effectué sur les réseaux de neurones convolutifs.Il est important que la partie encodeur est utilisée pour l'extraction des caractéristiques. Ainsi les couches se présentent donc comme suit:

La première couche étant la couche de convolution nous permet comme nous l'avons dit tout haut d'extraire les caractéristiques importantes en appliquant des filtres de dimension d'abord de 32, ensuite 64 et finalement de 128 pour chaque image qui passe dans le réseau.

La deuxième couche de la partie encodeur est la couche de correction qui nous permet d'améliorer l'efficacité du traitement en appliquant une fonction d'activation sous les caractéristiques issues de la couche de convolution.Nous avons eu à choisir la fonction d'activation ***Leaky Relu*** auquel l'hyper paramètre alpha a été fixé à 0.3 pour cette architecture parce qu'elle améliore très nettement l'efficacité de l'apprentissage.Cependant est important de souligner que nous avons essayé la fonction d'activation Relu mais elle ne s'est pas avérée très efficace pour ce cas de figure ce qui a motivé notre choix de la fonction d'activation ***Leaky Relu.***Aussi la

fonction Leaky Relu est généralement meilleure pour la reconstruction ou la génération et résout le problème de ***Relu mort(dying Relu)***.

Notre troisième couche est la couche de pooling qui sous-échantillonne l' input reçu de la couche de correction.Elle nous permet de réduire la taille tout en conservant les informations importantes.De l'input reçu est cree une fenetre 2 par 2 dans notre cas de laquelle est choisi le maximum qui sera placé en sortie pour la prochaine sous-couche.En observant attentivement notre architecture nous pouvons remarquer que a chaque couche la taille de l'input est divisé par 2.La fonction de cette couche en plus de condenser les données pour obtenir une meilleure caractéristique est d'aboutir à une reduction considerable de la quantité d'opération et aussi de la mémoire.À l'aide du paramètre padding fixe a "same" permet aux entrées de la couche d'avoir une uniformité dans le remplissage en tout point de l'entrée.

La quatrième couche étant la couche de Dropout permet nous l'avons dit plus haut une indépendance des neurones et l'hyperspécialisation d'un neurone dans la détection d'une seule catégorie d'animaux par exemple mais au contraire aide le neurone a avoir un bon pouvoir de généralisation.Nous avons fixé dans notre cas l'hyper paramètre du taux d'extinction à 0.2 ce qui signifie le fermeture aléatoire de 20% des neurones dans toutes les couches.Ces couches ont été répétés trois fois.

Au niveau de décodeur nous avons eu à appliquer les mêmes couches à l'exception de la couche de upsampling en substitution à la couche de Pooling qui est appropriée pour la reconstruction de l'image.

En plus de tout cela pour la sortie nous appliquons une fonction ***sigmoid*** parce que nos sorties doivent être comprises entre 0 et 1 et que l'auto encodeur essaie de prédire les valeurs des pixels entre ces valeurs.

2.6.2. Justification des choix(paramètres, d'hyper paramètres, d'architecture et de traitement de données)

2.6.2.1. Choix des paramètres et hyperparamètres

Pour ce qui est du choix de l'optimiseur nous avons eu à choisir l'optimiseur ***ADAM*** compte tenu de compte tenu de sa vitesse ainsi que de sa qualité de convergence vers une solution optimale.Aussi avec l'hyper-paramètre du taux d'apprentissage fixe a 0.001.

En ce qui concerne la taille du lot d'entraînement nous avons eu à choisir une taille de lot de 32 parce qu'elle s'avère être intuitivement un bon compromis pour accélérer la reconstruction de l'image en prenant en compte le facteur temps.Et aussi nous avons remarqué que plus la taille de lot est importante plus le modèle met le temps à reconstruire les images.

Pour ce qui est du choix du nombre d'époques, nous avons eu à fixer le nombre d'époques à 100 et ce dans le but d'améliorer les résultats de reconstruction à travers chaque époque et ainsi minimiser l'erreur de reconstruction.

2.6.2.2. Choix de l’architecture

Pour notre modèle, nous utilisons une architecture semi-métrique de l'encodeur-décodeur.

Nous avons expérimenté de nombreuses versions différentes de cette architecture et nous sommes arrivés à la conclusion que cette architecture est la meilleure pour capturer les caractéristiques des deux classes.

Nous avons finalement retenu trois blocs de "convolution-sampling".Nous avons essayé d'utiliser deux blocs au lieu de trois au début, le résultat obtenu n'était pas assez satisfaisant, nous avons donc augmenté à trois blocs dans le but de réduire beaucoup plus l'information spatiale (de 120x120 à 15x15) et de faire en sorte que l'algorithme se base et se concentre sur les caractéristiques calculées pour représenter les classes.

Concernant l’**ordre suivi par les poids** ([128,64,32] pour l'encodeur et [32,64,128] pour le décodeur.Nous avons expérimenté avec beaucoup de valeurs différentes, et nous avons conclu que la dimensionnalité totale et les paramètres à la fin de la phase d'encodage sont très importants pour la généralisation, c'est pourquoi le nombre de filtres diminue avec la dimensionnalité de l'image (pour la partie encodeur). Notre objectif est que le modèle ne garde que les quelques caractéristiques qui représentent les classes dans la couche d'intégration, ce choix rend l'image reconstruite plus floue car les paramètres de la couche d'intégration sont peu nombreux. Cependant, ce choix a permis d'augmenter le score de l'évaluation de l'encodeur.

La couche de dropout : nous avons utilisé le dropout pour empêcher le modèle de mémoriser les données d'entraînement afin de lutter contre l'overfitting.

Nous n’avons **pas ajouté de couche de normalisation par lot** : l'utilisation de la normalisation par lot peut aider à accélérer le taux d'apprentissage, mais dans notre cas, nous n'avons pas vu la nécessité de l'utiliser car ***les pixels sont déjà normalisés entre les valeurs 0 et 255.***

Le leaky relu : les avantages du leaky relu sont qu'il augmente le temps d'apprentissage et qu'il résout le problème du dying relu.Il est également populaire pour les tâches d'auto encodage et de génération.

Si nous utilisions plus ou moins de filtres, les résultats étaient affectés négativement, dans notre dernière tentative, nous sommes arrivés à la conclusion d'augmenter le nombre de filtres à chaque fois que nous diminuons la taille de l'image, notre hypothèse est que nous n'avons pas besoin de beaucoup d'extraction

de caractéristiques lorsque l'image était grande, mais nous devons augmenter le nombre de filtres tout en diminuant la taille pour essayer de condenser les caractéristiques petit à petit jusqu'à ce que nous atteignons un nombre suffisant et faible de paramètres à la fin de l'étape d'encodage.

2.6.2.3. Choix du traitement des données

En ce qui concerne le traitement des données comme dit tout haut nous avons essayé de fixer toutes les images avec les mêmes caractéristiques à savoir une largeur de 120 pixels, une longueur de 120 pixels les canaux fixes à 3 parce qu'il s'agit d'images avec des couleurs et ce dans le but de permettre leur passage dans l'auto encodeur sans difficulté.Nous n'avons pas effectuer l'augmentation des données dans ce travail.

3. Evaluation du modèle

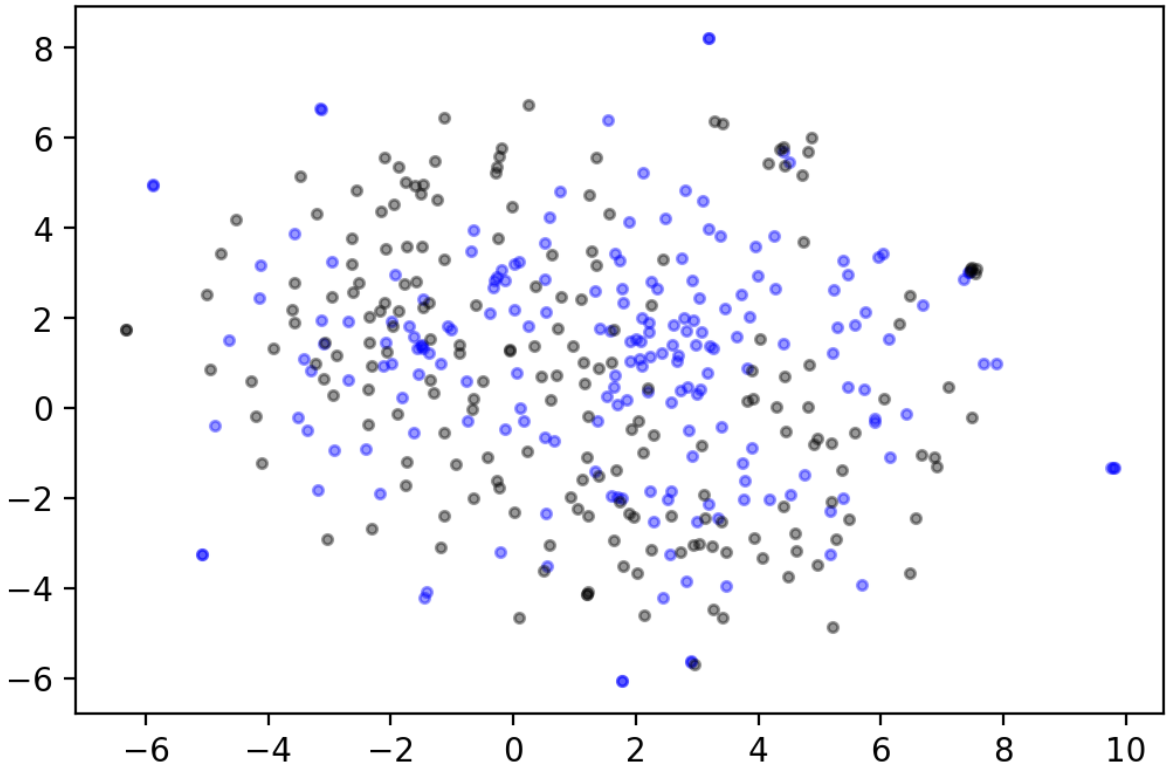
Après la phase d'entraînement notre modèle est soumis à une phase d'évaluation ou il reçoit en entrée des images issues des données de test prévues à cet effet.Le modèle reçoit donc des input de 400 images en entrées et ce pour chaque catégorie d'animal(200 images par classe) et doit être à même de les reconstruire le plus fidèlement possible en réduisant autant que possible l'erreur minimale commise.

3.1. Affichage des résultats

Ci-dessous une image de chaque classe avec en première ligne les images originales et ensuite en seconde ligne les images reconstruites

L'entraînement du SVM-Linéaire sur le embedding a donné une exactitude(Accuracy) de 68.75%.

L'entraînement du SVM-Linéaire directement sur les données de test a donné une exactitude de 79.75%.



Affichage en deux dimensions du embedding dans un scatter plot

3.2. Analyse et discussion des résultats de l'évaluation du modèle

3.2.1. La fidélité de reconstruction

Dans les premières versions de notre modèle, la sortie de l'encodeur était plus grande en termes de dimensionnalité spatiale (largeur X hauteur), ce qui conduisait à une meilleure et moins bruyante reconstruction des images originales. Cependant, cela signifie que l'encodeur garde beaucoup d'informations sur la valeur initiale des pixels et pas les caractéristiques essentielles dont nous avons besoin pour différencier les vaches des éléphants, c'est pourquoi nous avons ajouté plus de blocs de convolution-d'échantillonnage à notre modèle, dans le but de faire en sorte que l'encodeur ne garde que les caractéristiques importantes.

3.2.2. La séparation des classes dans le scatter plot

Le nuage de points nous montre que l'intégration résultante après avoir passé les données de test n'est pas parfaitement séparée pour les deux classes que nous avons, ce qui est attendu puisque l'évaluation actuelle est de 79%, ce qui signifie qu'il y a plus de 40 mauvaises classifications pour chaque classe, et plus de 80 au total. De plus, on peut voir que les notes bleues qui représentent la classe des éléphants sont plus regroupées au centre et en haut à droite du graphique, alors que les notes

noires qui représentent la classe des vaches sont plus regroupées à gauche et en bas du graphique.

3.2.3. Les résultats de classification du SVM (appliqué sur le embedding et sur les données de test originaux).

L'application sur les données originales : Puisque nous travaillons avec des images sans représentation caractéristique, l'algorithme SVM va essayer de séparer les classes les unes des autres en dépendant uniquement des valeurs des 120*120*3 = 43200 pixels qui composent chaque image. Après l'application de la validation croisée, le modèle a réussi à atteindre une précision de 68,7% sur les images brutes.

Application sur l'embedding: ici, nous utilisons l'encodeur développé dans la tâche 1 pour obtenir une représentation caractéristique des images de test. Après l'application de la validation croisée, nous avons obtenu une précision moyenne de 79,75% en battant le SVM sur les données brutes de 11%, ce qui indique que notre modèle auto-encodeur a participé positivement à la tâche d'extraction de caractéristiques et de classification des deux animaux, les vaches et les éléphants.

4. Conclusion

Somme toute, nous pouvons affirmer que toutes les tâches réalisées ont été assez concluantes et que les objectifs dans le cadre de ce travail ont bien été atteints.Cependant il est important de notifier que nous avons eu à rencontrer différents problèmes.

Un des problèmes rencontrés est **le surajustement** : l'absence de méthodes de régularisation a fait que le modèle a surajusté les données, nous avons résolu ce problème en ajoutant une couche de dropout pour tous les blocs de convolution que nous avons.

Un autre problème rencontré est **la simplicité du modèle avec des images décodées presque parfaites** : la simplicité de la première architecture que nous avons essayée a conduit à ce que la couche d'intégration ait une dimension spatiale élevée (hauteur X largeur), ce qui a conduit à ce que la partie décodeur ait beaucoup d'informations sur l'image originale, ce qui, à terme, a conduit à une reconstruction presque parfaite des images. Nous avons résolu ce problème en ajoutant plus de couches à nos parties encodeur et décodeur, ce qui nous a permis de réduire la dimensionnalité spatiale de (120 X 120) à (15 X15) et a conduit à une reconstruction floue des images, mais globalement à une meilleure représentation caractéristique des images.

Les valeurs du nombre de filtres étaient également le principal paramètre que nous avons essayé d'ajuster pour obtenir une meilleure précision. Nous avons donc

essayé de nombreuses combinaisons sans succès, jusqu'à ce que nous parvenions à une bonne combinaison permettant de dépasser la précision de 77% pour l'évaluation du codeur.

Pour améliorer notre modèle nous pourrions essayer d'implémenter une couche dense à la fin de l'encodeur pour voir son effet sur l'apprentissage. Nous pourrons aussi expérimenter avec des architectures et des tailles d'images originales plus complexes et aussi ajouter plus de données.