INF7370

Apprentissage automatique

ADIL LABIAD

LABA06058003

ISSA BABAN CHAWAI, ABDOULAYE

ISSA23038902

Devoir 3

Université de Montréal à Québec

Hiver 2024

Table des matières

1 Introduction

Ce projet vise à développer un auto-encodeur convolutionnel capable d'encoder et de reconstruire des images de deux espèces d'animaux : vaches et éléphants. L'objectif est de créer un modèle d'auto-encodeur qui puisse générer des *embeddings* efficaces à partir des images d'entrée. Ces *embeddings* sont ensuite utilisés pour classer les images à l'aide d'une machine à vecteurs de support *SVM*. Une partie essentielle du projet consiste à comparer les performances du *SVM* appliqué directement sur les données originales et du *SVM* appliqué aux *embeddings* générés par l'auto-encodeur. Une exigence clé de ce travail est que le SVM appliqué sur les embeddings associés aux données de test doit atteindre une exactitude minimale de 77%.

1.1 Mise en contexte

Les auto-encodeurs convolutionnels sont des réseaux de neurones composés d'une phase d'encodage et d'une phase de décodage. Dans ce projet, l'auto-encodeur extrait des caractéristiques pertinentes des images de vaches et d'éléphants, les encode dans une représentation compacte, puis les reconstruit à partir de ces *embeddings*. Cette capacité d'extraction de caractéristiques permet d'utiliser les *embeddings* comme entrée pour un classificateur, tel qu'un *SVM*, pour établir une distinction entre les deux espèces.

L'objectif de ce projet est double. Premièrement, nous voulons évaluer la capacité de l'auto-encodeur à encoder et reconstruire des images avec une précision élevée. Deuxièmement, nous souhaitons examiner dans quelle mesure ces *embeddings* peuvent améliorer les performances d'un *SVM* par rapport à un *SVM* appliqué directement sur les données originales.

1.2 Description générale du code

Le code du projet est structuré comme suit :

1. Chargement des Données : Les images sont chargées à partir de dossiers contenant des ensembles d'entraînement et de test. Des techniques d'augmentation des données sont utilisées pour accroître la diversité des images d'entraînement.

2. Configuration de l'auto-encodeur : Le modèle est conçu pour encoder et reconstruire des images tout en générant des *embeddings* informatifs. Des couches de convolution, de *pooling* et de suréchantillonnage sont utilisées pour extraire les caractéristiques clés des images. La régularisation est appliquée pour éviter l'*overfitting*.

3. Entraînement du modèle : L'auto-encodeur est entraîné avec des *callbacks* pour sauvegarder le modèle avec le meilleur score de validation et pour stopper l'entraînement en cas de stagnation des performances. Les données de validation sont utilisées pour évaluer le modèle.

4. Application du *SVM* : Après l'entraînement, le *SVM* est appliqué aux *embeddings* générés par l'encodeur. L'exactitude (accuracy) minimale cible pour le *SVM* appliqué sur les embeddings associés aux données de test est de 77%. Les résultats obtenus par le *SVM* sont comparés avec ceux obtenus en appliquant le *SVM* directement sur les données originales.

1.3 Objectif

L'objectif de ce projet est de créer un auto-encodeur capable de produire des *embeddings* utiles pour la classification des images de vaches et d'éléphants, tout en atteignant une exactitude minimale de 77% avec un *SVM* appliqué sur ces embeddings pour les données de test. Cette exigence représente un critère de succès pour le projet.

Le projet vise également à comparer les performances du *SVM* appliqué directement sur les données originales avec celles du *SVM* appliqué sur les *embeddings*. L'objectif est de déterminer si l'utilisation d'auto-encodeurs pour l'extraction de caractéristiques peut améliorer la performance des modèles de classification tout en réduisant la dimensionnalité des données. Ces résultats peuvent avoir des implications importantes pour des applications telles que la reconnaissance d'images, la compression de données et la classification d'images à grande échelle.

2 Montage de l'architecture et entraînement du modèle

2.1 Ensemble de données

L'ensemble de données utilisé dans ce projet se compose d'images de deux classes principales : vaches et éléphants. Ces données sont divisées en ensembles d'entraînement, de validation et de test, chaque ensemble jouant un rôle spécifique dans le processus d'apprentissage et d'évaluation du modèle. Voici une description détaillée des proportions de chaque catégorie :

L'ensemble d'entraînement constitue la majorité des données disponibles pour l'apprentissage du modèle. Il contient un total de 1920 images, avec 960 images pour chaque classe. Ces données d'entraînement servent à ajuster les poids du modèle lors des étapes d'entraînement.

L'ensemble de validation est un sous-ensemble des données d'entraînement utilisé pour évaluer les performances du modèle après chaque époque. Il représente 20% des données d'entraînement initiales, soit 240 images par classe, pour un total de 480 images. Le but de cet ensemble est de

s'assurer que le modèle n'est pas en sur-apprentissage *overfitting* et qu'il généralise bien sur des données nouvelles.

Enfin, l'ensemble de test est utilisé pour évaluer la performance finale du modèle après l'entraînement. Il contient 400 images, avec 200 images pour chaque classe. Cet ensemble permet d'estimer la précision du modèle dans un contexte de données complètement nouvelles, simulant son utilisation dans des situations réelles.

La table 1 montre la répartition des données pour chaque classe et chaque catégorie, indiquant le nombre total d'images disponibles pour l'entraînement, la validation et le test :

Catégorie	**Vaches**	**Éléphants**	**Total**
Données d'entraînement (après validation)	960	960	1920
Données de validation	240	240	480
Données de test	200	200	400

TABLE 1 – Distribution des données par classe

2.2 Traitement de données

Le traitement des données est une étape essentielle dans le développement d'un modèle de *Deep learning*, car il contribue à la qualité et à la robustesse des données utilisées pour l'entraînement. Dans cette section, nous décrivons le traitement appliqué à nos ensembles de données pour garantir que le modèle reçoit des données de qualité et peut généraliser efficacement.

2.2.1 Normalisation des données

Une des étapes fondamentales du traitement des données consiste à normaliser les images. Cela signifie que les valeurs des pixels des images sont mises à l'échelle entre 0 et 1, plutôt que d'être dans leur échelle d'origine (0 à 255). Cette normalisation facilite l'entraînement du modèle en assurant que les données d'entrée ont des valeurs cohérentes, ce qui aide à stabiliser l'optimisation pendant l'entraînement. Dans notre cas, cette normalisation est réalisée via le paramètre **rescale=1. / 255** dans **ImageDataGenerator**.

2.2.2 Augmentation des données

L'augmentation des données est une technique utilisée pour augmenter la diversité des données d'entraînement sans collecter plus d'images. Cela permet d'éviter le sur-apprentissage (*overfitting*)

en exposant le modèle à différentes variantes des images d'entraînement. Les techniques d'augmentation appliquées dans notre projet incluent :

- Décalage horizontal et vertical : Les images sont légèrement décalées horizontalement et verticalement, ce qui ajoute de la variabilité au modèle. Dans notre cas, ces décalages sont définis avec **width_shift_range=0.1** et **height_shift_range=0.1**, signifiant que les images peuvent être déplacées de 10% dans chaque direction.

- Retournement horizontal : Les images peuvent être retournées horizontalement, ce qui simule des variations réalistes dans les données. Cela est réalisé avec le paramètre **horizontal_flip=True**.

Ces augmentations aident le modèle à être plus robuste et à apprendre des représentations plus générales, augmentant ainsi ses capacités de généralisation.

2.2.3 Création de l'ensemble de validation

Pour garantir que les résultats du modèle sont fiables, un ensemble de validation est créé à partir de l'ensemble d'entraînement. Dans notre projet, 20% des données d'entraînement sont réservés pour la validation, ce qui permet de surveiller les performances du modèle au fur et à mesure de l'entraînement. Cette division des données est effectuée avec le paramètre **validation_split=0.2** de **ImageDataGenerator**.

Ces techniques de traitement des données garantissent que le modèle d'auto-encodeur reçoit des données diversifiées, bien normalisées et structurées de manière à permettre une évaluation précise des performances, contribuant ainsi à un entraînement efficace et une meilleure capacité de généralisation.

2.3 Paramètres et Hyperparamètres

Les paramètres et hyperparamètres définissent le comportement et l'entraînement du modèle de *Deep learning*. Ils influencent la manière dont le modèle apprend à partir des données et affectent directement ses performances. Dans cette section, nous décrivons les principaux paramètres et hyperparamètres utilisés dans le projet, notamment l'optimiseur, la taille des lots d'entraînement, le nombre d'époques, et l'arrêt précoce.

2.3.1 Optimiseur

L'optimiseur est un composant clé de tout modèle de *Deep learning*, car il détermine la méthode utilisée pour mettre à jour les poids du modèle pendant l'entraînement. Dans ce projet, l'optimiseur utilisé est *Adam*, connu pour sa stabilité et sa capacité à converger rapidement. *Adam* combine les avantages des optimisateurs de type gradient avec ceux de type *momentum*, et il utilise des paramètres internes pour ajuster dynamiquement le taux d'apprentissage. L'optimiseur *Adam* est utilisé avec un taux d'apprentissage par défaut de **0.001**. D'autres paramètres associés à *Adam*, tels que *beta_1*, *beta_2*, et *epsilon*, utilisent également leurs valeurs par défaut :

Hyperparamètre	**Valeur**
beta_1	0.9
beta_2	0.999
epsilon	1e-7

TABLE 2 – Hyperparamètres de l'optimiseur *Adam*

2.3.2 Taille du lot

La taille du lot (*batch size*) détermine le nombre d'exemples utilisés dans chaque étape d'entraînement. Pour ce projet, la taille du lot utilisée est de **32**. Cela signifie que 32 images sont traitées ensemble avant que le modèle ne mette à jour ses poids. Cette taille de lot est un compromis entre la vitesse d'entraînement et l'efficacité de la convergence. Une taille de lot trop grande pourrait rendre l'entraînement inefficace, tandis qu'une taille de lot trop petite pourrait introduire de l'instabilité.

2.3.3 Nombre d'époques

Le nombre d'époques (*Epochs*) correspond au nombre de fois que le modèle parcourt l'ensemble d'entraînement. Pour ce projet, le nombre d'époques est fixé à **100**, ce qui signifie que le modèle passe 100 fois sur l'ensemble d'entraînement.

2.3.4 Arrêt précoce

Le mécanisme d'arrêt précoce (*early stopping*) est également mis en place pour éviter l'*overfitting*. L'arrêt précoce utilise le paramètre patience, qui indique combien d'époques le modèle peut tolérer sans amélioration du critère de surveillance (ici, la perte de validation). Si le modèle n'améliore pas sa perte de validation pendant 10 époques consécutives, l'entraînement s'arrête, et les meilleurs poids sont restaurés. Cela permet de prévenir l'*overfitting* et de conserver les meilleurs résultats obtenus pendant l'entraînement.

Ces paramètres et hyperparamètres jouent un rôle important dans l'efficacité de l'entraînement du modèle, la prévention de l'*overfitting*, et la stabilité générale du processus d'apprentissage. En les réglant correctement, vous pouvez optimiser les performances du modèle et assurer une meilleure généralisation sur les données de test.

2.4 Architecture

L'architecture de l'auto-encodeur dans ce projet se compose de plusieurs couches de convolution, de *Pooling* et de sur-échantillonnage. Elle comprend également des fonctions d'activation spécifiques pour chaque couche et des techniques d'initialisation des poids pour favoriser la convergence.

2.4.1 Couches et paramètres

L'auto-encodeur comporte deux parties principales : l'encodeur et le décodeur. L'encodeur extrait les caractéristiques des images, tandis que le décodeur reconstruit les images à partir des *embeddings*.

- L'encodeur :
  - La première couche est une couche de convolution avec **32** filtres de taille **3x3**. Elle utilise une activation **relu**, un **kernel_initializer= "he_uniform"**, et un **padding="same"**.
  - Une couche de *Pooling* suit, utilisant *MaxPooling2D* avec une taille de fenêtre **2x2**, et un **padding="same"**.
  - Ensuite, il y a une deuxième couche de convolution avec **64** filtres de taille **3x3**, suivie d'une couche de *Pooling* avec les mêmes paramètres que la première.
  - La dernière couche de convolution de l'encodeur utilise **128** filtres de taille **3x3**, avec une activation **relu**, suivie d'une dernière couche de *Pooling*.
- Le décodeur :
  - Le décodeur commence par une couche de convolution avec **128** filtres de taille **3x3** et une activation **relu**, suivie d'une couche *UpSampling2D* avec une taille de fenêtre de **2x2**.
  - Ensuite, il y a une couche de convolution avec **64** filtres de taille **3x3**, puis une autre couche *UpSampling2D*.
  - La dernière couche de convolution du décodeur utilise **32** filtres de taille **3x3**, suivie d'une couche *UpSampling2D*.

- La couche de sortie est une couche de convolution avec le même nombre de canaux que l'image d'entrée (3 pour les images en couleur), suivie d'une activation **sigmoid** pour garantir que les valeurs des pixels sont dans la plage [0, 1].

2.4.2 Fonctions d'activation

Les fonctions d'activation jouent un rôle clé dans le comportement des couches de l'auto-encodeur. Pour cet architecture, plusieurs fonctions d'activation sont utilisées :

- Les couches de convolution intermédiaires utilisent l'activation **relu**, qui est couramment utilisée dans les réseaux de neurones convolutifs pour sa capacité à accélérer l'entraînement.

- La dernière couche du décodeur utilise l'activation **sigmoid**, qui convient aux auto-encodeurs car elle contraint les sorties entre 0 et 1, ce qui correspond à la plage des valeurs des pixels des images normalisées.

Ces détails de l'architecture décrivent le nombre de couches utilisées, les paramètres de chaque couche, l'absence de *Dropout*, et les types de fonctions d'activation utilisés dans le modèle d'auto-encodeur de ce projet.

2.5 Affichage des résultats d'entraînement

Le temps total d'entraînement, les erreurs minimales commises, et la courbe de perte sont des indicateurs importants de la performance de l'entraînement et de la convergence du modèle.

2.5.1 Temps d'entraînement

Le temps total d'entraînement mesure le temps écoulé pendant l'entraînement du modèle. Cela peut varier en fonction du nombre d'époques, de la taille du lot, et de la complexité de l'architecture. Pour ce projet, l'entraînement s'est déroulé sur **100 époques** avec une **taille de lot de 32**, et a pris environ **32 minutes**.

2.5.2 Erreur minimale

La perte minimale (*Minimum Loss*) est un indicateur de la précision du modèle. Dans ce projet, nous utilisons le *Mean Squared Error* (MSE) comme fonction de perte. Pendant l'entraînement, le modèle ajuste ses paramètres pour minimiser cette perte. Pour l'ensemble d'entraînement la perte minimale atteinte est de **0.0033** tandis que pour l'ensemble de validation la perte minimale est de **0.0031**.

2.5.3 Courbe de perte

La courbe de perte (*Loss curve*) 1 illustre l'évolution de la perte pendant l'entraînement, ce qui permet de visualiser la convergence du modèle et de détecter tout signe de sur-apprentissage.

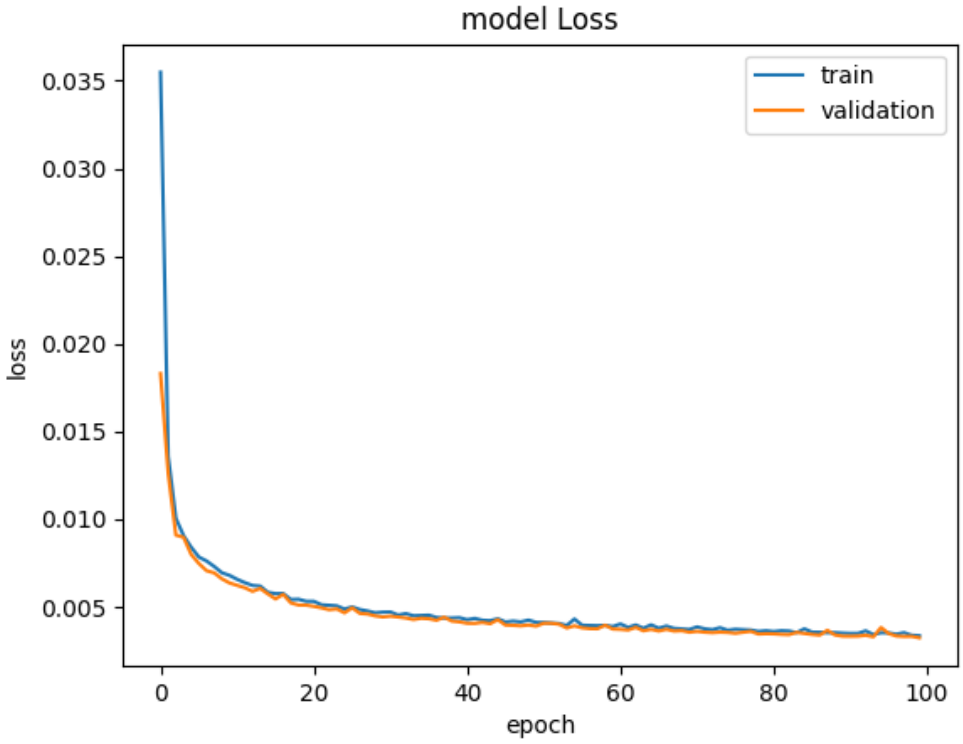

FIGURE 1 – Perte du modèle par epoch.

2.6 Justification du choix de l'architecture

Le choix de l'architecture pour un projet de *Deep learning* est crucial pour obtenir des résultats optimaux. Cette section décrit le processus de décision suivi pour choisir l'architecture de l'auto-encodeur et justifie les paramètres, les hyperparamètres, l'architecture, et le traitement de données utilisés.

2.6.1 Justification du choix de l'architecture

Le processus d'atteinte des meilleurs résultats possibles pour ce projet a consisté en une série d'ajustements soigneusement planifiés, basés sur des principes de *Deep learning* et des résultats

expérimentaux. Cette section décrit en détail les choix de paramètres, d'hyperparamètres, d'architecture, et de traitement des données, en mettant en évidence les modifications qui ont conduit à une précision de 78%.

Pour atteindre une précision de 78%, plusieurs aspects de l'architecture et du traitement des données ont été ajustés. Voici les étapes clés qui ont été suivies :

- Choix de la fonction d'activation : L'utilisation de la fonction d'activation **relu** (*Rectified Linear Unit*) a contribué à une meilleure convergence du modèle. Contrairement à d'autres fonctions d'activation comme *leaky_relu* ou *tanh*, **relu** permet d'éviter le problème de disparition de gradients, ce qui facilite l'entraînement des réseaux profonds. Ce changement a permis une meilleure performance globale du modèle.

- Initialisation des poids : L'utilisation de **kernel_initializer="he_uniform"** a été essentielle pour garantir une bonne distribution des poids au début de l'entraînement. Cette méthode d'initialisation, conçue spécifiquement pour les fonctions *relu*, aide à prévenir l'explosion et la disparition des gradients, améliorant ainsi la stabilité de l'entraînement.

- Optimiseur : Le choix de l'optimiseur **Adam** a été déterminant pour la convergence rapide et efficace du modèle. **Adam** combine les avantages du *momentum* et de l'adaptive *learning rate*, ce qui a permis une meilleure adaptation au fur et à mesure de l'entraînement, améliorant ainsi la précision.

- Augmentation des données : L'ajout de techniques d'augmentation des données, telles que le décalage horizontal et vertical et le retournement horizontal, a contribué à augmenter la diversité des données d'entraînement. Cela a réduit le risque de sur-apprentissage (*overfitting*) et a aidé le modèle à généraliser plus efficacement sur l'ensemble de validation.

2.6.2 Justification des choix de paramètres

Les paramètres et hyperparamètres jouent un rôle crucial dans le comportement du modèle et sa capacité à atteindre les meilleurs résultats possibles. Voici les justifications des choix effectués :

- Batch size : Une taille de lot de **32** a été choisie comme compromis entre la vitesse d'entraînement et la stabilité. Des lots trop petits peuvent entraîner une instabilité, tandis que des lots trop grands peuvent ralentir la convergence.

- Epochs : Un nombre initial de **100** époques a été utilisé pour garantir que le modèle ait suffisamment de temps pour apprendre. Cependant, l'ajout de l'arrêt précoce (avec patience=10)

a permis d'éviter l'*overfitting* en interrompant l'entraînement lorsque la perte de validation ne s'améliorait plus.

- Traitement des Données : La normalisation des images a été effectuée en les mettant à l'échelle entre 0 et 1, ce qui a contribué à stabiliser l'entraînement. L'utilisation de la division de validation (20% des données d'entraînement) a permis de surveiller les performances du modèle pendant l'entraînement.

Ces ajustements spécifiques à l'architecture, aux paramètres, et au traitement des données ont contribué à atteindre une précision de 78%. Les modifications de la fonction d'activation, de l'initialisation des poids, et de l'optimiseur ont permis d'améliorer la stabilité et la convergence du modèle, tandis que l'utilisation de techniques d'augmentation des données a aidé à augmenter la diversité de l'ensemble d'entraînement, ce qui a conduit à de meilleurs résultats.

3 Évaluation du Modèle

Après l'entraînement du modèle d'auto-encodeur et l'application du *SVM*, nous avons obtenu des résultats intéressants qui démontrent la performance du modèle. Dans cette section, nous présentons les résultats obtenus, y compris des images originales et reconstruites, les précisions du *SVM* sur les *embeddings* et sur les données de test originales, ainsi qu'une visualisation des *embeddings* en deux dimensions.

- Images originales et reconstruites : La capacité de l'auto-encodeur à reconstruire des images à partir de leurs *embeddings* est un indicateur clé de son efficacité. Pour évaluer la qualité de la reconstruction, nous avons préparé une matrice 4x4 2 illustrant les résultats obtenus.

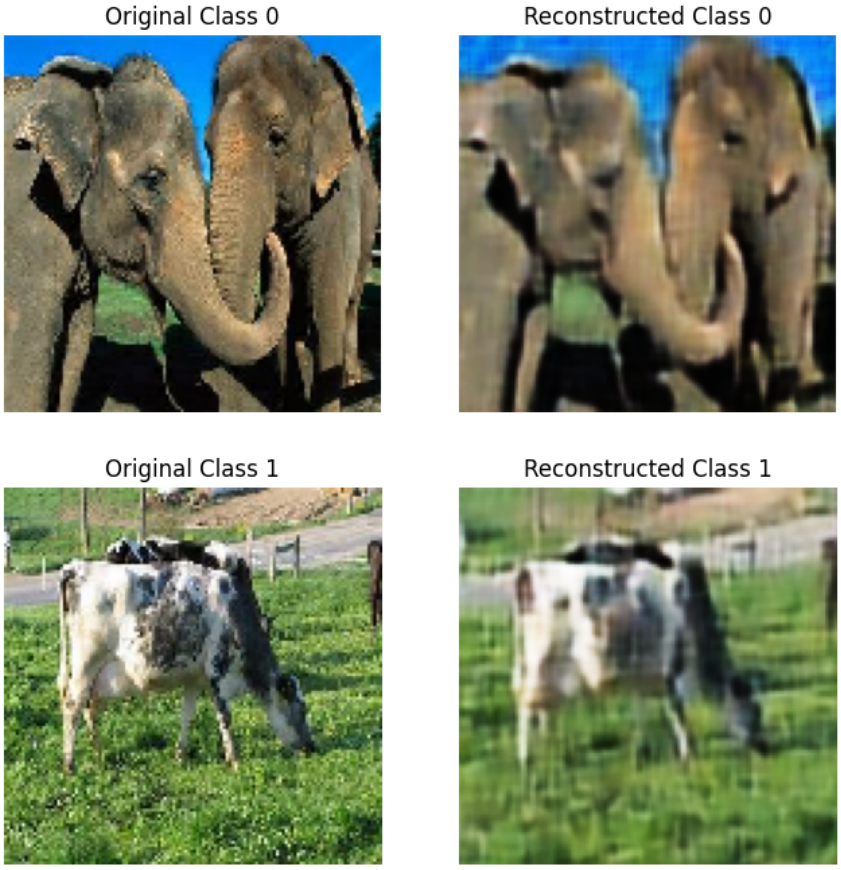

FIGURE 2 – Illustration de la reconstruction d'image par l'auto-encodeur.

La première ligne de la matrice représente une reconstruction d'une image d'éléphant. Elle commence par l'image originale, suivie de trois reconstructions générées par l'auto-encodeur. Ces reconstructions montrent dans quelle mesure le modèle a capturé les détails de l'image originale. En comparant les reconstructions, on peut observer la progression du modèle dans sa capacité à reconstruire des éléments comme les contours, les formes et les textures caractéristiques des éléphants.

La seconde ligne de la matrice illustre une reconstruction d'une image de vache. Comme pour la première ligne, elle commence par l'image originale, suivie de trois reconstructions. Ces

reconstructions offrent un aperçu de la capacité du modèle à reproduire les caractéristiques visuelles des vaches, y compris les motifs distinctifs et les formes générales.

En analysant ces images, on peut évaluer la qualité de la reconstruction et déterminer si l'auto-encodeur capture les détails nécessaires pour une reconstruction fidèle. Ces résultats fournissent des indications sur les forces du modèle et les domaines où des améliorations peuvent être apportées.

- Précision du *SVM* sur le *embedding* : L'un des résultats les plus intéressants concerne la précision du *SVM* lorsqu'il est appliqué sur les *embeddings* générés par l'auto-encodeur. Cette précision est de **78%**, ce qui démontre l'efficacité de l'auto-encodeur pour extraire des caractéristiques pertinentes à partir des images d'entrée. Cela suggère que les *embeddings* peuvent être utilisés pour des tâches de classification avec une précision élevée.

  Cette amélioration de la précision par rapport à l'utilisation du *SVM* directement sur les données originales indique que l'auto-encodeur joue un rôle crucial dans la transformation des données d'entrée en représentations compactes et informatives. Ce résultat est essentiel pour évaluer la qualité des embeddings et leur utilité pour les tâches de classification.

- Précision du *SVM* sur les données de test : La précision du *SVM* lorsqu'il est appliqué directement sur les données de test originales est de **68%**. Ce résultat sert de référence pour mesurer l'amélioration obtenue grâce à l'utilisation des *embeddings*. La différence de précision entre les deux approches montre que le *SVM* appliqué sur les *embeddings* a une capacité de généralisation plus élevée, ce qui peut être attribué à la réduction de la dimensionnalité et à la capture des caractéristiques clés par l'auto-encodeur.

- Visualisation du *embedding* en deux dimensions : Pour comprendre comment les *embeddings* se répartissent dans l'espace, nous avons utilisé des techniques *t-SNE* de réduction de dimensionnalité pour visualiser les *embeddings* en deux dimensions. Cela nous permet de voir comment les différentes classes se regroupent et à quel point elles sont distinctes.

  La Figure 3 montre un *scatter plot* des *embeddings* réduits en deux dimensions. Les couleurs représentent les différentes classes, ce qui permet de visualiser la séparation entre les classes de vaches et d'éléphants. Cette visualisation fournit des informations précieuses sur la structure des *embeddings* et peut servir de guide pour évaluer l'efficacité de l'auto-encodeur.

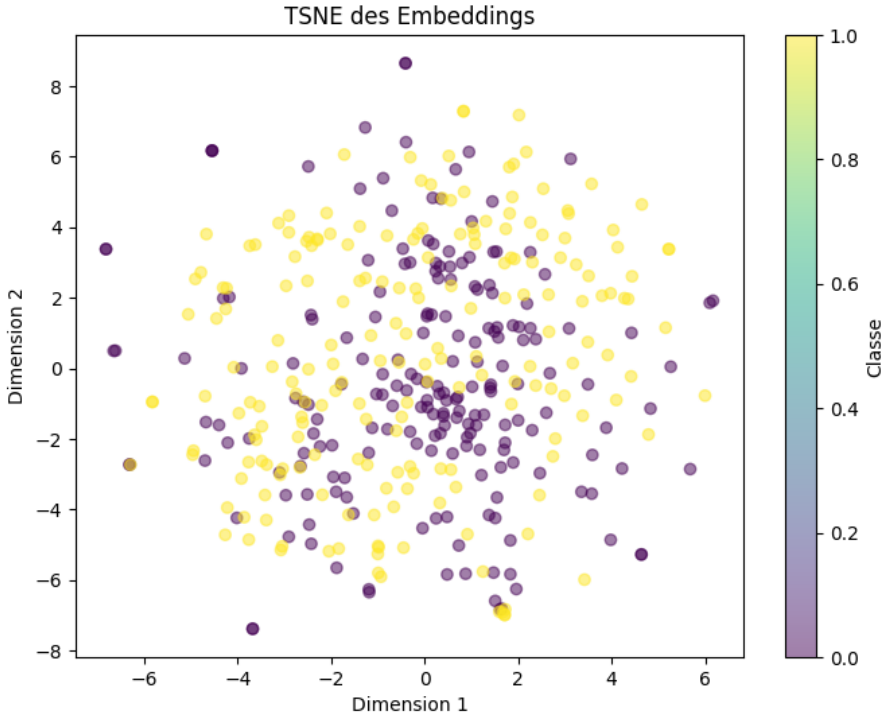

FIGURE 3 – Illustration de la reconstruction d'image par l'auto-encodeur.

Ces résultats et visualisations fournissent une compréhension approfondie des performances de l'auto-encodeur et du *SVM*, mettant en évidence l'importance des *embeddings* dans l'amélioration de la précision de la classification. Ils démontrent également la valeur de l'auto-encodeur pour des tâches de réduction de dimensionnalité et d'extraction de caractéristiques pertinentes.

4 Conclusion

Au terme de ce travail, nous avons approfondi notre compréhension des auto-encodeurs et des machines à vecteurs de support (*SVM*), en explorant comment utiliser leurs atouts respectifs pour améliorer la classification. Cette étude a permis de mettre en évidence plusieurs aspects cruciaux de ces techniques, tout en fournissant des perspectives sur la manière d'optimiser le modèle.

Les images reconstruites par notre auto-encodeur présentent un certain niveau de flou, indiquant que le modèle peut ne pas reproduire tous les détails de l'image originale. Cependant, ce constat ne remet pas en question la qualité de l'auto-encodeur. Au contraire, cela suggère que le modèle a

réussi à capter les caractéristiques les plus importantes des images, tout en générant des embeddings efficaces pour la reconstruction.

La visualisation des *embeddings* dans le *scatter plot* montre que les classes de vaches et d'éléphants ne sont pas toujours linéairement séparables, ce qui indique un certain chevauchement entre les deux classes. Cette observation souligne l'importance de techniques d'extraction de caractéristiques puissantes pour faciliter la classification.

Un des défis rencontrés lors de l'entraînement était de trouver des hyperparamètres optimaux pour assurer un temps d'exécution raisonnable, tout en tenant compte des limitations de la mémoire RAM. Nous avons effectué plusieurs essais avec différents hyperparamètres, tels que la taille du lot, le nombre d'époques, et les fonctions d'activation, pour trouver un équilibre entre la qualité de la reconstruction et la précision du modèle.

Nous avons également remarqué que l'utilisation d'images de taille plus grande pourrait potentiellement améliorer les performances du modèle, car des images de plus haute résolution contiennent plus d'informations. Cependant, cela nécessite des ressources informatiques supplémentaires, ce qui pourrait nécessiter une mémoire plus importante et des temps d'entraînement plus longs.

Pour l'amélioration future du modèle, plusieurs pistes peuvent être envisagées :

- L'augmentation des ressources matérielles pour permettre un traitement plus rapide et la manipulation de données plus volumineuses.

- L'optimisation du nombre de couches et des architectures pour trouver la configuration la plus efficace.

- L'application de techniques supplémentaires d'augmentation des données pour augmenter la diversité des données d'entraînement.

- La variation des fonctions d'activation et l'expérimentation avec d'autres optimisateurs pour trouver des combinaisons qui améliorent la convergence et la précision du modèle.

En conclusion, malgré les défis rencontrés, ce projet a mis en lumière l'importance des auto-encodeurs et des *SVM* dans la classification des images. Avec des ajustements appropriés et des ressources adéquates, nous pensons que la précision du modèle peut être encore améliorée. Les solutions envisagées permettront de tirer le meilleur parti de ces techniques et de continuer à optimiser le modèle pour des performances supérieures.