Projet : Deep Colorization

Introduction

Le but de ce projet est d'entraîner un réseau de neurones capable de coloriser des images en noir et blanc. La méthode retenue s'inspire largement de celle décrite par Zhang et al. dans l'article Colorful Image Colorization.

Rendus attendus

- Votre code, sous forme de fichier(s) aux formats `.py` ou `.ipynb`.
- La version finale de votre modèle entraîné, au format `.keras`.
- Vos réponses aux questions posées dans ce sujet, à l'intérieur d'un fichier dans l'un des formats suivants : `.ipynb`, `.doc`, `.txt`, `.pdf` ou `.md`.

Conseils

- Pour l'entraînement de vos réseaux, n'hésitez pas à travailler avec les environnements en ligne mis à disposition par Kaggle en activant l'option GPU. Cela réduira grandement le temps d'apprentissage. Attention cependant : le nombre d'heures d'utilisation du GPU est limité ! Les 30 heures offertes par Kaggle devraient être largement suffisantes pour compléter ce projet, mais pensez quand même à désactiver votre session quand vous ne l'utilisez pas, au cas où.

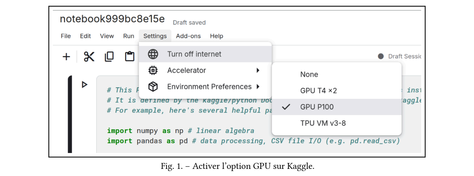

Fig. 1. – Activer l'option GPU sur Kaggle.

- Tant que vous n'avez pas à entraîner votre modèle, travailler en local est sûrement plus agréable. Vous pourrez toujours mettre en ligne votre code le moment venu.
- N'hésitez pas à faire des `print` pour déboguer. La plupart des problèmes viennent du fait qu'un tableau n'a pas la bonne dimension, attention à sa `shape` !

Analyse du problème

À faire : Lire les deux premiers paragraphes de l'introduction de l'article Colorful Image Colorization par Zhang et al.

Question 1 – De quelles données va-t-on avoir besoin pour entraîner un modèle à coloriser des images en noir et blanc ? Si un tel jeu de données n'existe pas, peut-on le créer aisément ?

À faire : Se renseigner sur l'espace de couleur `CIELab`.

Question 2 – À quoi correspondent les trois canaux `L`, `a` et `b` de l'espace de couleur `CIELab` ?

Question 3 – Quel va être l'intérêt de l'espace de couleur `CIELab` par rapport au `RGB` pour notre problème ?

Question 4 – Notre modèle sera une fonction $f(x) = y$. À quoi correspondront x et y ? En d'autres termes, quel sera l'entrée de notre modèle et quelle sera sa sortie ? (Indice : x sera de dimension $128 \times 128 \times 1$, et y sera de dimension $128 \times 128 \times 2$.)

Question 5 – Le problème que nous cherchons à résoudre est-il un problème de classification ou de régression ? Justifiez.

Question 6 – Comment évaluer la performance de notre modèle ?

Préparation du jeu de données

Nous allons utiliser une partie du jeu de données CelebA.

Question 7 – Que contient précisément le jeu de données CelebA ?

À faire :

- Dézipper l'archive `celebs.zip`.
- Créer deux objets `Dataset`, pour l'entraînement et la validation respectivement, à l'aide de la fonction `keras.utils.image_dataset_from_directory`.
- Afficher une image du jeu d'entraînement à l'aide de la fonction `plt.imshow`. (Pour récupérer un élément d'un `Dataset`, vous aurez certainement besoin d'un coup de `next(iter(dataset))`)
- À l'aide de la fonction `Dataset.map`, préparer les jeux de données de sorte que :
  - Les images soient de dimensions 128×128.
  - Les images soient au format CIELab.

  Pour ce faire, vous pouvez vous aider des fonctions suivantes : `keras.layers.Resizing`, `skimage.color.rgb2lab`. Si `Dataset.map` vous pose des soucis, ne vous découragez pas, ces deux fonctions vont vous aider : `tf.ensure_shape`, `tf.py_function`.
- Toujours à l'aide de `Dataset.map`, transformer les deux jeux de données afin que chaque élément soit une paire `(x, y)` (où `x` est l'entrée et `y` la sortie attendue, cf. question 4)
- Écrire une fonction `reconstituer_image(x, y)` qui prend en entrée une paire entrée-sortie `x,y` et renvoie l'image correspondante au format `RGB`. Vous aurez besoin de `np.concatenate`, `np.expand_dims` et `skimage.color.lab2rgb`.

Conception du réseau de neurones

Question 8 – Quelles seront les dimensions de la couche d'entrée de notre réseau ? De sa couche de sortie ?

Question 9 – Quel est l'intérêt d'une couche convolutive par rapport à une couche complètement connectée ?

Question 10 – Comment peut-on interpréter l' « image » en sortie d'une convolution ?

Question 11 – Si un filtre convolutif de taille 3×3 est appliqué sans *padding* et avec un pas de 1 à une image de dimensions 128×128, quelle est la dimension de l'image de sortie ? Et avec un filtre de taille 5×5 ?

Question 12 – Un réseau convolutif vous paraît-il approprié pour notre tâche ? Pourquoi ?

Question 13 – Quelles sont les plages de valeurs possibles pour les canaux a et b d'une image au format Lab ? En prenant en compte cette plage, quelle fonction d'activation choisiriez-vous pour la sortie de votre réseau de neurone ?

À faire : À l'aide de `keras.Sequential`, créez un modèle dont l'architecture est celle de la figure 2.

La première couche `Rescaling` est paramétrée de sorte à ce que les valeurs du canal L en entrée, qui sont dans l'intervalle $[0, 100]$, soit comprises dans l'intervalle $[-1, 1]$ en sortie.

Toutes les couches `Conv2D` et `Conv2DTranspose` ont des filtres de taille 3×3, des pas (`strides`) de 2, `padding="same"`, et une activation `RELU` (à l'exception de la dernière, qui a l'activation de votre choix, cf. question 13).

La dernière couche `Rescaling` est paramétrée de sorte à ce que la sortie du réseau couvre toute la bande des valeurs possibles pour les canaux a et b, c'est-à-dire $[-128, 128]$.

Layer (type)	Output Shape	Param #
rescaling_22 (Rescaling)	(None, 128, 128, 1)	0
conv2d_55 (Conv2D)	(None, 64, 64, 64)	640
batch_normalization_55 (BatchNormalization)	(None, 64, 64, 64)	256
conv2d_56 (Conv2D)	(None, 32, 32, 128)	73,856
batch_normalization_56 (BatchNormalization)	(None, 32, 32, 128)	512
conv2d_57 (Conv2D)	(None, 16, 16, 256)	295,168
batch_normalization_57 (BatchNormalization)	(None, 16, 16, 256)	1,024
conv2d_58 (Conv2D)	(None, 8, 8, 512)	1,180,160
batch_normalization_58 (BatchNormalization)	(None, 8, 8, 512)	2,048
conv2d_59 (Conv2D)	(None, 4, 4, 1024)	4,719,616
batch_normalization_59 (BatchNormalization)	(None, 4, 4, 1024)	4,096
conv2d_transpose_55 (Conv2DTranspose)	(None, 8, 8, 512)	4,719,104
conv2d_transpose_56 (Conv2DTranspose)	(None, 16, 16, 256)	1,179,904
conv2d_transpose_57 (Conv2DTranspose)	(None, 32, 32, 128)	295,040
conv2d_transpose_58 (Conv2DTranspose)	(None, 64, 64, 64)	73,792
conv2d_transpose_59 (Conv2DTranspose)	(None, 128, 128, 2)	1,154
rescaling_23 (Rescaling)	(None, 128, 128, 2)	0

Fig. 2. – L'architecture du modèle, obtenue avec `model.summary`. Il s'agit d'un réseau *auto-encodeur* : la première moitié du modèle compresse l'image spatialement à l'aide de convolutions (jusqu'à obtenir une « image » de $4 \times 4 = 64$ « pixels » à 1024 canaux), tout en augmentant sa profondeur. La deuxième moitié du modèle performe l'opération inverse.

À faire : Compilez votre modèle à l'aide de `model.compile`, avec l'optimiseur `adam` et l'erreur quadratique moyenne comme perte. Vérifiez que votre architecture est semblable à celle de la figure 2 à l'aide de `model.summary`.

Entraînement

À faire : Entraînez le modèle pendant 10 époques sur les données d'entraînement. Afin d'avoir un retour en direct sur l'évolution de votre modèle, passez aussi votre jeu de validation en argument à la fonction `model.fit`. En cas d'erreur cryptique à base de « `unknown shape` », c'est probablement parce que vous devez appliquer des `tf.ensure_shape` à votre jeu de données (à l'aide de `Dataset.map`, comme d'habitude). Si tout fonctionne, c'est le moment idéal pour prendre une pause café.

Question 14 – Qu'observe-t-on lorsqu'on compare la perte sur le jeu d'entraînement avec la perte sur le jeu de validation au cours de l'entraînement ? Comment s'appelle ce phénomène ? Que pourrait-on faire pour l'éviter ?

Évaluation et itération

À faire :

- Écrivez une fonction `coloriser(im)` qui prend en entrée une photo en noire et blanc, et renvoie en sortie la version colorisée par votre modèle. N'hésitez pas à réutiliser votre fonction `reconstituer_image`. Prenez garde aux dimensions !
- Affichez (sur une seule figure) cinq images tirées aléatoirement du jeu de validation, leurs versions en noir et blanc, et leurs versions colorisées par le modèle.
- Testez votre modèle sur cinq portraits tirés d'internet et affichez les résultats sous la même forme qu'à la question précédente. Vous aurez besoin de la fonction `skimage.io.imread`.

Question 15 – Le modèle est beaucoup moins bon avec des photos tirées d'internet. Pourquoi d'après vous ?

À faire :

- Modifiez les jeu de données d'entraînement avec `Dataset.map` afin de lui appliquer des translations aléatoires dans l'intervalle $\pm 20\%$ (avec `RandomTranslation`).
- Entraînez votre modèle sur le nouveau jeu d'entraînement puis évaluez-le à nouveau sur des images tirées du jeu de validation ainsi que sur des images tirées d'internet.

Question 16 – Le modèle est-il meilleur qu'auparavant sur les images tirées d'internet ? Si oui, pourquoi ?

À faire : Enregistrez votre modèle entraîné au format `.keras` avec `model.save`.

Question 17 – (Bonus) Les colorisations effectuées par le modèle sont systématiquement plus ternes que les images réelles. L'article Colorful Image Colorization propose une explication de ce phénomène dans la section 2.1. Quelle est cette explication ? Donnez-en une illustration.

Pour aller plus loin

S'il vous reste du temps, voici quelques pistes que vous pouvez explorer :

- Déployer le modèle dans une application web.
- Entraîner un modèle capable de fonctionner avec des images plus grandes.
- Augmenter davantage le jeu de données avec des transformations supplémentaires, ou tester d'autres jeux de données.
- Implémenter l'architecture complète décrite dans l'article Colorful Image Colorization.