Deep Image Prior

Projet Modélisation 3MIC

Membres

PHAM Tuan Kiet

VO Van Nghia

2 mai 2021

Table des matières

Table des figures

1 Introduction

Réseau neuronal convolutif (en anglais ***CNN***) est actuellement l'une des techniques les plus connues dans les problèmes de reconstruction d'image inverse. Ils se sont avérés efficace dans un grand nombre de tâches, y compris le débruitage d'image, la super-résolution d'image et la reconstruction d'images. Elle est un type de réseau de neurones artificiels acycliques (feed-forward), dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux.

La puissance de cette architecture est attribuée à sa capacité d'apprendre à partir de nombreux grands ensembles d'images. Cependant, [**2007.02471** ; **1711.10925**] et de nombreux autres travaux ont démontré que l'architecture d'un CNN peut agir comme un préalable suffisamment fort pour résoudre des problèmes inverses comme la reconstruction d'image, même sans étape d'apprentissage. Plus précisément, les réseaux non entrans fonctionnent bien pour le débruitage d'image [**2007.02471**], l'acquisition comprimée [**1806.06438**] et même pour la reconstruction de vidéos [**1910.01684**].

Dans ce travail, nous nous concentrerons sur la technique ***Deep Image Prior***. Dans la partie suivante, nous allons examiner quelques exemples. En suite, nous approfondissons le principe sous-jacent de celui-ci à travers l'exemple précédent. Enfin, nous expérimenterons ce que nous venons de présenter ci-dessus et ferons quelques études complémentaires.

Tout d'abord, nous clarifierons certaines terminologies.

1.1 Réseau neuronal convolutif

Un réseau neuronal est l'association, en un graphe plus ou moins complexe, d'objets élémentaires, les neurones formels. Les principaux réseaux se distinguent par l'organisation du graphe (en couches, complets, *etc*,), c'est-à-dire leur architecture, son niveau de complexité (le nombre de neurones, présence ou non de boucles de rétroaction dans le réseau) par le type des neurones (leurs fonctions de transition ou d'activation) et enfin par l'objectif visé : apprentissage supervisé ou non, optimisation, systèmes dynamiques, *etc*.

Le neurone formel est un modèle qui se caractérise par un état interne s , des signaux d'entrée x_1, ..., x_p et une fonction d'activation

$$s = h(x_1, \dots, x_p) = g(\alpha_0 + \sum_{j=1}^{p} \alpha_j x_j) = g(\alpha_0 + \alpha' x)$$

La fonction d'activation opère une transformation d'une combinaison affine des signaux d'entrée, α_0, terme constant, étant appelé le biais du neurone. Cette combinaison affine est déterminée par un vecteur de poids $[\alpha_0, \dots, \alpha_p]$ associé à chaque neurone et dont les valeurs sont estimées dans la phase d'apprentissage. Ils constituent la mémoire ou connaissance répartie du réseau.

Un réseau de neurones convolutifs ou réseau de neurones à convolution est un type de réseau de neurones artificiels, dont architecture est formée par un empilement de couches (les combinaisons des neurones formels) de traitement :

— La couche de convolution (CONV)
— La couche de pooling (POOL)
— La couche d'activation
— La couche "entièrement connectée" (FC)
— La couche de perte (LOSS)

1.2 Les problèmes inverses

Les problèmes inverses peuvent être formulés comme la tâche d'optimisation avec la formule :

$$x^* = min_x E(x; x_0) + R(x)$$

Le réseau neuronal convolutionnel va décoder et générer une fonction de type :

$$x = f_\theta(z)$$

qui correspond bien aux données. La fonction va lier un vecteur qui est donné au hasard avec une image X. La méthode est sélectionnée spécifiquement pour chaque application.

2 Les aspects techniques

Dans cette partie, un résumé des différents aspects techniques est une introduction. Cela nous aidera à établir une connexion entre les mathématiques et l'informatique plus facilement et fournira également des informations supplémentaires sur "Deep Image Prior".

2.1 Le code

Tout d'abord, nous entrons dans le code. Comme indiqué ci-dessus, nous avons essayé de réécrire le code dans [**1711.10925**]. Étant donné que chaque problème nécessite un modèle différent, nous nous concentrons uniquement sur le modèle de débruitage (bien que notre code devrait fonctionner avec la plupart des modèles de cet article).

Outre la motivation de mieux comprendre le code, il y a quelques raisons techniques à nos motivations pour un réimplémentation :

— Le code est obsolète (il a été écrit pour la première fois en 2017).

— Mélange entre Pytorch Sequential et Module API, ce qui rend le code difficile à suivre (en particulier la fonction de création de modèle).
— Modèle irritant par boucle brute, ce qui rend plus difficile l'intégration avec des API et des outils modernes (par exemple TensorBoard).

Nous le réécrivons à l'aide de TensorFlow, et nous ajoutons également des métriques supplémentaires pour mesurer le PSNR entre l'image débruitée et l'image bruité et le PSNR entre l'image débruitée et l'image originale.

2.2 Le modèle

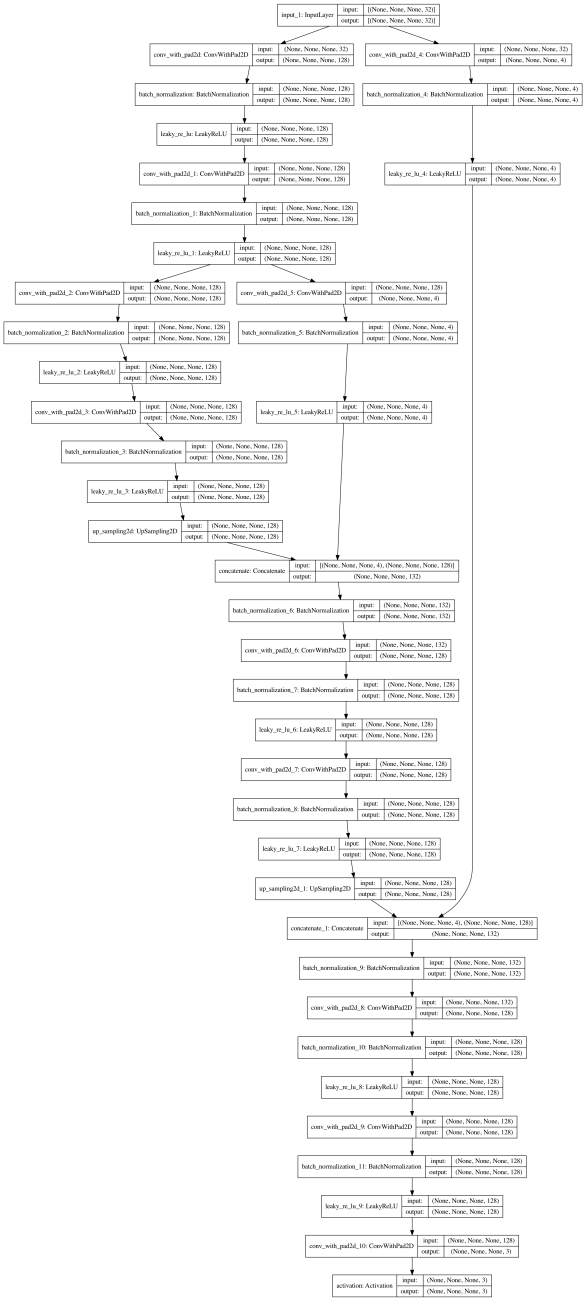


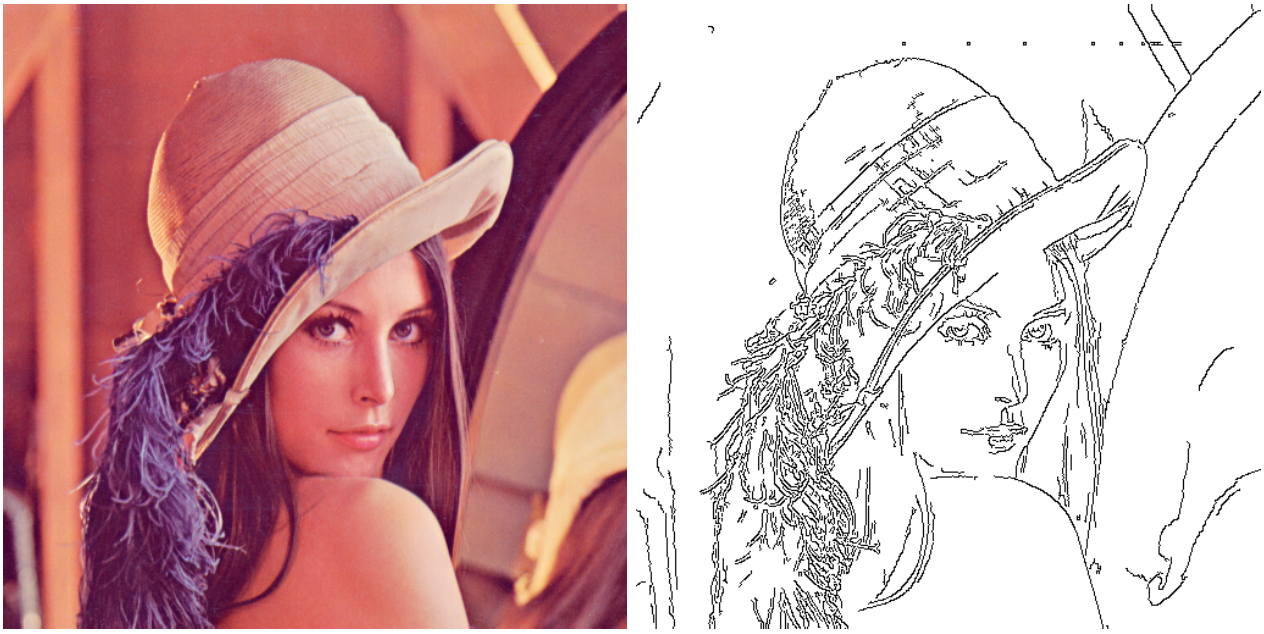
FIGURE 1 – Exemple de modèle de débruitage

Ici, nous voyons une architecture CNN (**??**) très typique. Bien que le modèle dans [**1711.10925**] soit plus profond et ait beaucoup plus de couches, il a toujours le même groupe de couches qu'un auto-encodeur (décodage et encodage) avec sauts de connexions.

L'entrée du modèle est un tenseur de forme `[batch_size, image_height, image_width, 32]` où `batch_size` vaut 1 dans notre cas. 32 est le nombre de canaux (caractéristiques) qui seront expliqués plus tard.

Sa sortie est un tenseur avec une forme de `[batch_size, image_height, image_width, 3]` qui pourrait être affiché comme une image (3 canaux car nous avons une image RGB (Red-Green-Blue)).

2.3 Décodage et encodage



(a) original (b) bord

FIGURE 2 – La détection de bord

Le décodage est un processus d'extraction de caractéristiques d'une image. Sur le côté opposé, l'encodeur collectera les caractéristiques des décodeurs et reconstruira l'image. Les caractéristiques peuvent être n'importe quoi, varient des bords (**??**), de la couleur à la résolution.

Dans notre cas, le nombre de 32 de la section précédente signifie que nous capturons 32 caractéristiques différentes à partir d'une image bruité pour reconstruire l'image d'origine. Cela explique pourquoi si nous répétons le processus trop souvent, l'image de sortie tendra vers l'image bruité. Parce qu'à une certaine étape, les décodeurs considéreront le bruit comme une caractéristique et le captureront.

Sous le capot, le décodeur est construit à partir d'opérations convolutives. En parcourant chaque pixel et en appliquant le noyau convolutif, nous pourrions extraire les fonctionnalités souhaitées.

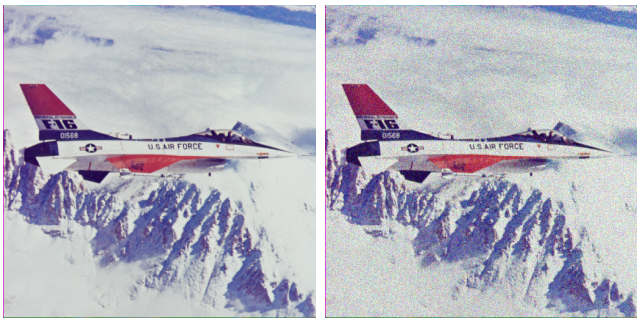
L'illustration d'une opération convolutive en image peut être vue comme suit :

$$\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \right) [1,1] = 0 \cdot a + 0 \cdot b + 0 \cdot c + 0 \cdot d + 1 \cdot e + 2 \cdot f + 0 \cdot g + 4 \cdot h + 5 \cdot i$$

2.4 La sortie

Pour bien comprendre, nous avons bouclé le processus 6000 fois.

Voici notre image de test :



(a) original (b) bruyant

FIGURE 3 – L'image originale et l'image bruité

Et le résultat que nous obtenons :

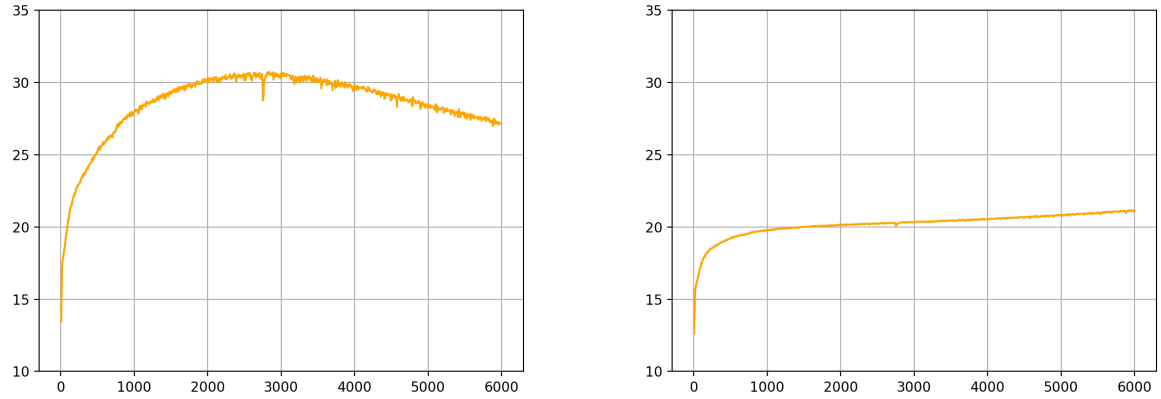
(a) 1 (b) 1000 (c) 3000 (d) 6000

FIGURE 4 – La sortie de l'epoch

Nous pouvions voir clairement les différences entre chaque epoche. Cela pourrait s'expliquer comme suit :

— A la première époque, l'entrée est un bruit aléatoire et le modèle est initialisé avec des poids aléatoires. Nous voyons donc une image aléatoire, qui n'est pas liée à la sortie souhaitée.
— Vers la 1000e époque, les décodeurs à l'intérieur du modèle commencent à s'ajuster pour extraire les caractéristiques de l'image bruité. Premièrement, ils extraient les plus importants comme le bord, la couleur, la résolution, *etc.*
— A la 3000ème époque, les décodeurs et encodeurs sont désormais capables d'extraire toutes les caractéristiques souhaitées et de reconstruire une image de sortie comme on le voit dans **??**.
— Plus nous bouclons le processus, plus il y a de parties extraites et des caractéristiques indésirables comme le bruit sont également incluses. Par conséquent, nous voyons une image assez proche de l'image bruyante (**??**).

Pour une vue plus scientifique, ce sont les métriques que nous avons utilisées lors de l'itération de ce modèle :



(a) originale (b) bruité

FIGURE 5 – Le PSNR entre l'image débruitée et l'image

D'après la figure **??**, il y a une tendance croissante dans le PSNR entre l'image débruitée et l'image bruyante. D'autre part, le PSNR entre celui-ci et l'image d'origine atteint son apogée vers 3000 epoches et commence à diminuer par la suite. Cela correspond à ce que nous avons vu auparavant en comparant différentes sorties avec nos yeux.

C'est la fin des aspects techniques. Dans la partie suivante, nous approfondirons les principes mathématiques sous-jacents, en particulier pourquoi cette méthode fonctionne ?