Deep Image Prior

Projet Modélisation 3MIC

Membres

PHAM Tuan Kiet

VO Van Nghia

24 mai 2021

Table des matières

Table des figures

1 Introduction

Le 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 compression d'images avec perte. 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 entrainés 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. Puis, 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 d'objets élémentaires, les neurones formels, interconnectés permettant la résolution de problèmes complexes tels que la reconnaissance des formes ou le traitement du langage naturel, grâce à l'ajustement des coefficients de pondération dans une phase d'apprentissage. 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 premier modèle mathématique et informatique du neurone biologique est proposé par Warren McCulloch et Walter Pitts Dans le modèle de McCulloch et Pitts, à chaque entrée est associé une valeur numérique appelé le poids synaptique. Donc, nous pouvons écrire le neurone formel comme un modèle qui se caractérise par un état interne s , des signaux d'entrée x_1, ..., x_p avec les poids synaptique associé α_1, ..., α_p et une fonction d'activation

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

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 $[\alpha_0, ..., \alpha_p]$ 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 Deep Image Prior

2.1 Principles

La technique ***Deep Image Prior*** a fait une percée dans la dans le contexte de problèmes inverses mal posés. Le réseau est connu pour le reconstruction d'image sans étape d'apprentissage. Nous allons étudier les principles dedans et comment cette technique fonctionne.

Normalement, le ***Deep learning*** aborde un problème par deux étapes. Premièrement, c'est l'étape d'apprentissage. Dans cette étape, le paramètre de réseau est optimisé par minimiser un approprié fonction de perte à partir de un grande ensemble de données. Puis, les nouvelles données est traité dans le réseau pour résoudre le problème. Néanmoins, la technique ***Deep Image Prior*** est basé

seulement sur une seul point de données y^γ. C'est-à-dire la technique est d'entrainer un réseau avec son paramètre par minimiser la fonction de perte : $\min_\xi \mid\mid A\varphi_\xi(z) - y^\gamma \mid\mid^2$.

Le resultat sur le réseau de ***Deep Prior*** utilise le feed-forward architecture qui commence par :

$$x^0 = z \text{ et } x^{k+1} = \phi(W_k x^k + b_k) \text{ for } k = 0, ..., L-1$$

Donc, les résultats obtenues sont $\varphi_\xi(z) = x^L$ avec les paramètres $\xi = \{W_0, ..., W_{L-1}, b_0, ..., b_{L-1}\}$ Pour la technique ***Deep Image Prior***, il n'y a pas d'étape d'apprentissage. Donc, ξ n'est pas fixé. La solution pour le problème inverse est $x = \varphi_\xi(z)$ avec z fixé. La technique vise à paramétriser la solution avec ξ.

2.2 Méthode

Dans ce projet, nous cherchons à comprendre le fonctionnement de cette technique en utilisant le prieur implicitement capturé par le choix d'une structure de réseau de générateurs particulière. Nous avons mis un paramétrage $x = f_\theta(z)$ avec x est un image de $R^{3xHeightxWidth}$ (une image colorée de taille Height x Width pixels a chaque pixel un combinaison de 3 couleurs rouge, vert et bleu). Ici, θ sont des paramètres du réseau.

3 Les aspects techniques

Dans cette partie, nous proposons un résumé des différents aspects techniques de la techniques de DIP(ou Deep Image Prior). 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".

3.1 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. 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 etl'image originale.

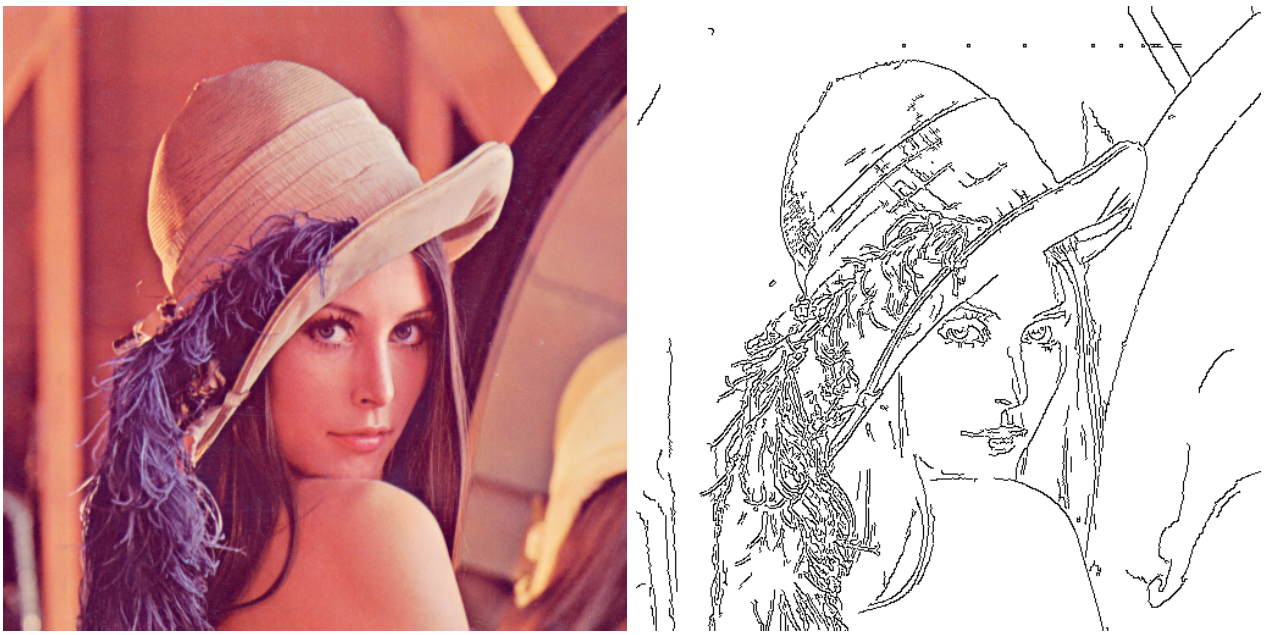
3.2 Le modèles

Nous avons fait 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. Notons qu'un décodeur soit suffit, nous avons aussi essayé d'écrire seulement un décodeur pur.

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)).

3.3 Décodage et encodage

Dans cette partie, nous travaillons sur la première architecture de CNN que nous avons fait. Elle est l'architecture ayant toujours un étape d'auto-encodeur.



(a) original (b) bord

FIGURE 1 – 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 (1), de la couleur à la résolution.

Dans notre cas, le nombre de 32 de l'architecture signifie que nous capturons 32 caractéristiques différentes à partir d'une image bruité pour reconstruire l'image d'origine. En fait, on pourra re-

construire l'image seulement si le nombre de paramètre du réseau est supérieur au nombre de pixels dans l'image. La technique ***Deep Image Prior*** ne comprend pas d'étape d'apprentissage. Donc, les décodeurs considéreront le bruit comme une caractéristique et le captureront. Cela explique pourquoi si nous répétons le processus trop souvent, l'image de sortie tendra vers l'image bruité ou vers un résultat que nous n'attendons pas.

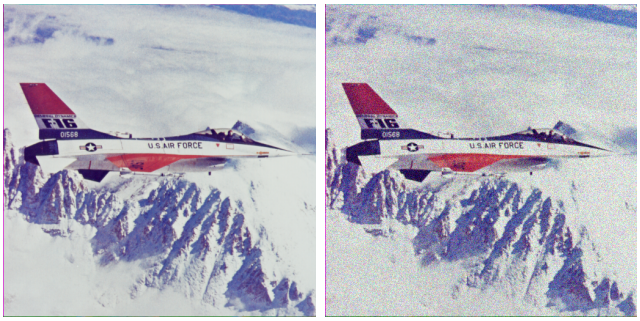
Sous le capot, le décodeur est construit à partir d'opérations convolutives. En parcourant chaque pixel et en appliquant le noyau convolutif, nous pourrions extraire les fonctionnalités souhaitées. L'illustration d'une opération convolutive en image avec un $2x2$ kernel peut être vue comme suit :

$$\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & d & e & f & 0 \\ 0 & g & h & j & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right) = \begin{bmatrix} a & a+b & b+c & c \\ a+d & a+b+d+e & b+c+e+f & c+f \\ d+g & d+e+g+h & e+f+h+j & f+j \\ g & g+h & h+j & j \end{bmatrix}$$

3.4 La sortie

Pour mieux comprendre, nous avons bouclé le processus 6000 fois. Nous étudions les différences entre chaque étape de boucle pour avoir un mieux vue de notre algorithm.

Voici notre image de test :



(a) original (b) bruyant

FIGURE 2 – L'image originale et l'image bruité

Et le résultat que nous obtenons :

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

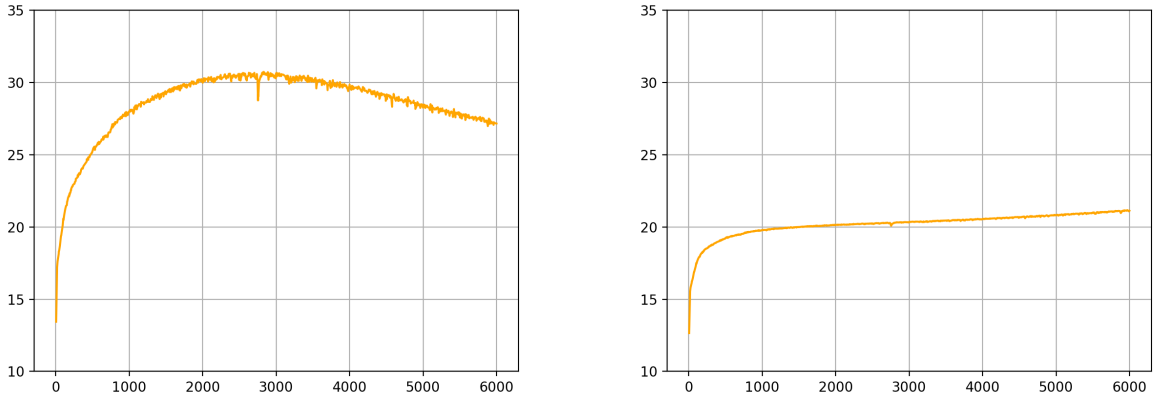
FIGURE 3 – La sortie de l'epoch

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 3.
— 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 (2).

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

D'après la figure 4, 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.



(a) originale (b) bruité

FIGURE 4 – Le PSNR entre l'image débruitée et l'image

4 Conclusion

Dans ce projet, nous avons étudié d'un nouvelle technique de traitement d'image. Cette technique n'a pas besoin quantité massive de données (un seul point de données y^{γ}) ni un modèle pre-entrainé (sans étape d'apprentissage). Nous avons testé différents modèles et données pour avoir un meilleur vue de fonctionnement de nos architecture et de la technique DIP. Ce que nous avons obtenu nous montre des résultats prometteurs.