

Deep Image Prior

Projet Modélisation 3MIC

Membres

PHAM Tuan Kiet

VO Van Nghia

24 mai 2021

Table des matières

Table des matières	i
1 Introduction	1
2 Les aspects techniques	2
2.1 Le code	2
2.2 Le modèle	3
2.3 Décodage et encodage	4
2.4 La sortie	5
3 Mathématiques et Deep Learning	8
3.1 Les problèmes inverses	8
3.2 Réseaux profonds	8
3.3 Réseau neuronal convolutif	9
3.3.1 Définition	9
3.3.2 Principes mathématiques	10
3.4 Optimisation globale	11
4 Deep Image Prior	12
4.1 Principes	12
4.2 Méthode appliquée	13
4.3 Application sur le débruitage et reconstruction d'image	14
5 Conclusion	15

Table des figures

1 Exemple de modèle de débruitage	3
2 La détection de bord	4
3 L'image originale et l'image bruité	5
4 La sortie de l'epoch	6
5 Le PSNR entre l'image débruitée et l'image	7
6 Illustration d'un réseau neuronal convolutif	9

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 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, [DH20 ; UVL17] 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 entrants fonctionnent bien pour le débruitage d'image [DH20], l'acquisition compressée [Vee+18] et même pour la reconstruction de vidéos [Yoo+19].

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.

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 [UVL17]. 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

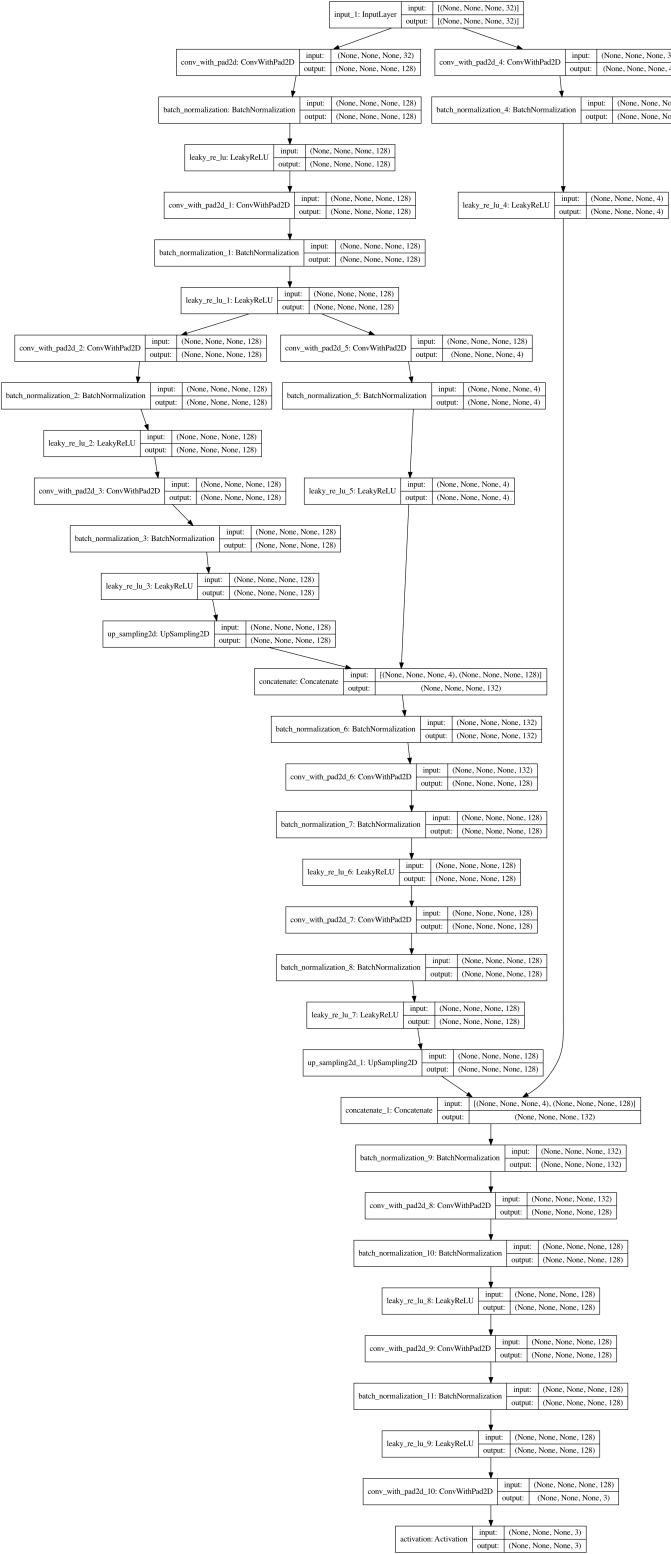


FIGURE 1 – Exemple de modèle de débruitage

Ici, nous voyons une architecture CNN (1) très typique. Bien que le modèle dans [UVL17] 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

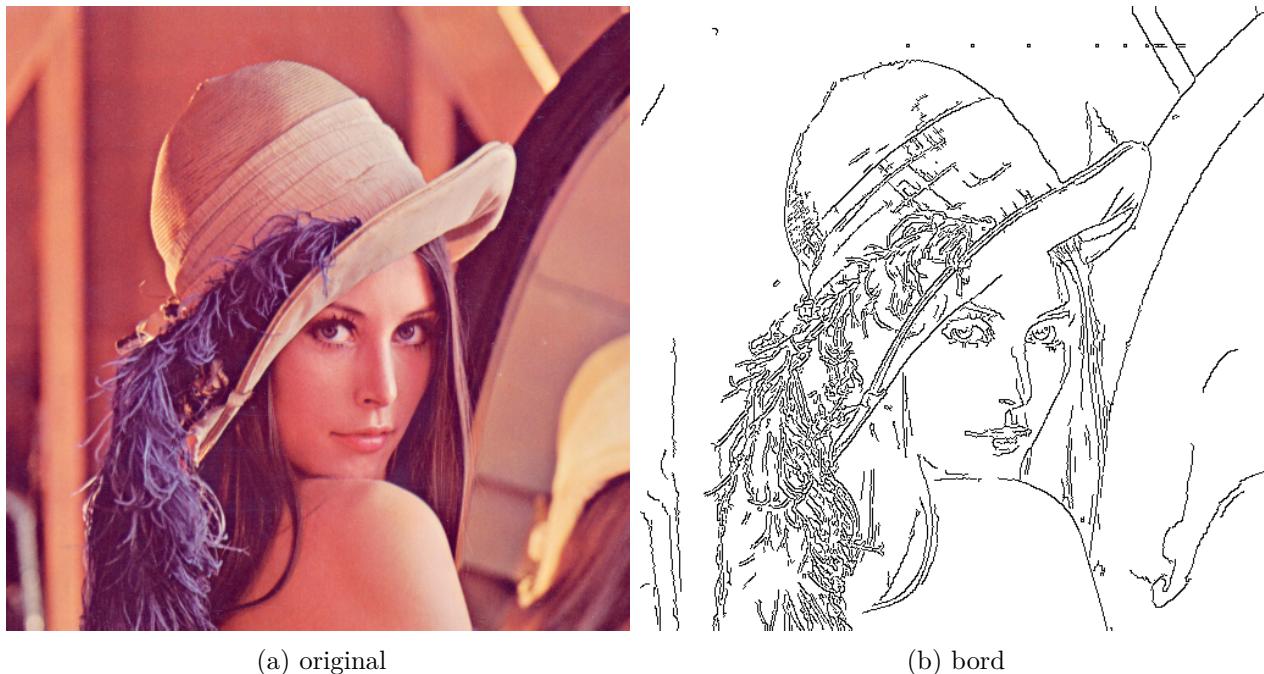


FIGURE 2 – La détection de bord

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 (2), 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 convulsive en image peut être vue comme suit :

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

2.4 La sortie

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

Voici notre image de test :

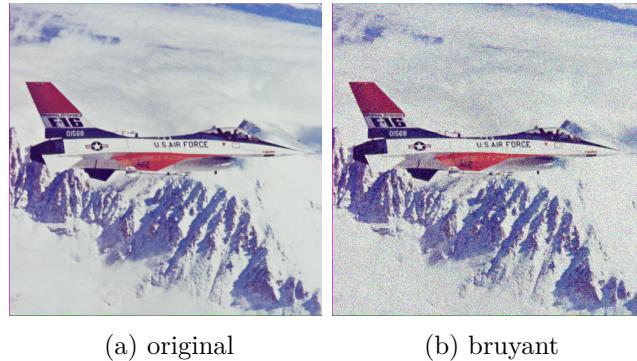


FIGURE 3 – L'image originale et l'image bruité

Et le résultat que nous obtenons :

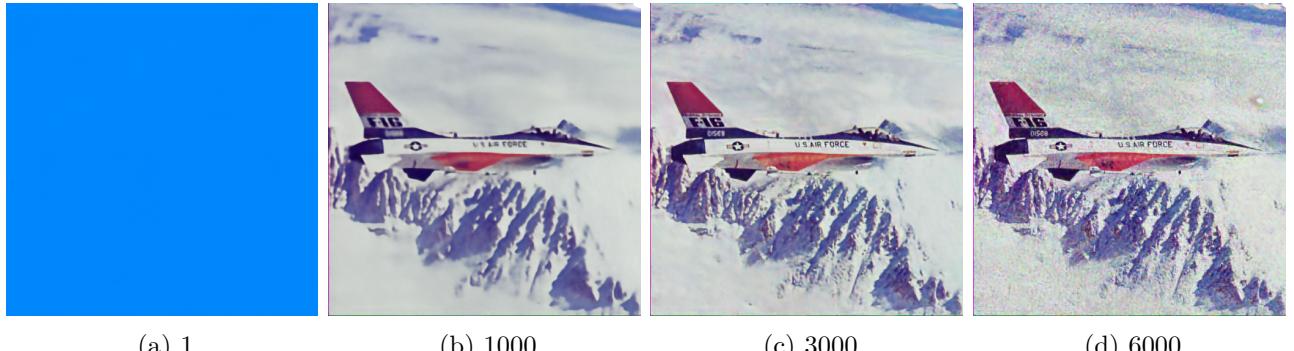


FIGURE 4 – La sortie de l'epoch

Nous pouvions voir clairement les différences entre chaque époque. 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 4.
- 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 (3).

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

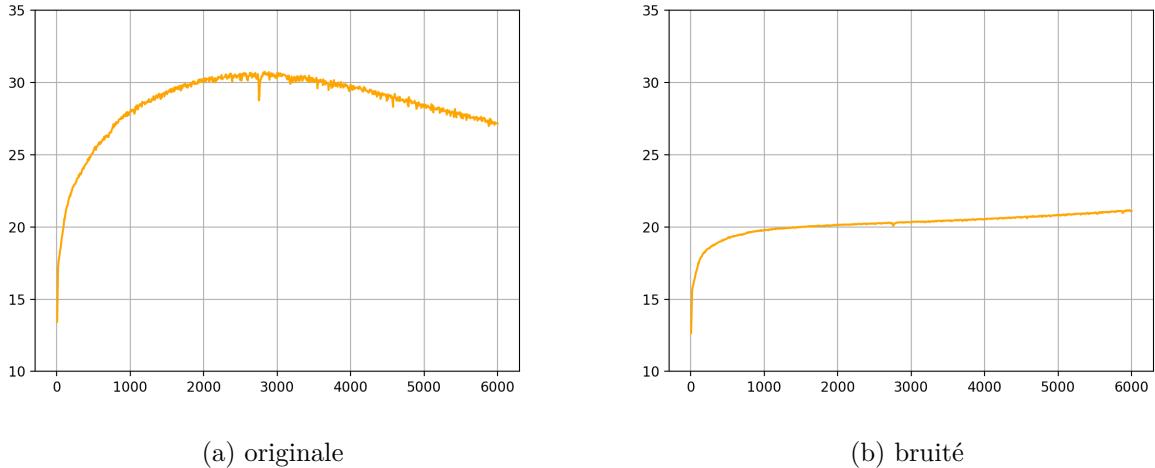


FIGURE 5 – Le PSNR entre l'image débruitée et l'image

D'après la figure 5, 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 époques 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 ?

3 Mathématiques et Deep Learning

Tout d'abord, nous clarifierons certaines terminologies.

3.1 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.

3.2 Réseaux profonds

Les réseaux profonds sont des modèles paramétriques qui effectuent des opérations séquentielles sur leurs données d'entrée. Chacune de ces opérations, communément appelée «couche», consiste en une transformation linéaire, disons, une convolution de son entrée, suivie d'une «fonction d'activation» non linéaire ponctuelle, par exemple, un sigmoïde. Les réseaux profonds ont récemment conduit à des améliorations spectaculaires des performances de classification dans diverses applications de traitement de la parole et du langage naturel et de la vision par ordinateur. La propriété cruciale des réseaux profonds que l'on pense être la racine de leurs performances est qu'ils ont un grand nombre de couches par rapport aux réseaux de neurones classiques ; mais il existe d'autres modifications architecturales telles que les activations linéaires rectifiées (ReLUs) et les connexions «raccourcies» résiduelles. D'autres facteurs majeurs de leur succès sont la disponibilité d'ensembles de données massifs, par exemple des millions d'images dans des ensembles de données comme ImageNet, et du matériel informatique GPU efficace pour résoudre le problème d'optimisation de haute dimension qui en résulte, qui peut avoir jusqu'à 100 millions de paramètres. Le succès empirique de l'apprentissage profond, en particulier des réseaux de neurones convolutionnels (CNN) pour les tâches basées sur l'image, présente de nombreuses énigmes aux théoriciens. En particulier, il existe trois facteurs clés dans l'apprentissage profond, à savoir les architectures, les techniques de régularisation et les algorithmes d'optimisation, qui sont

essentiels pour former des réseaux profonds performants et comprendre leur nécessité et leur interaction est essentiel si nous voulons percer les secrets de leur succès

3.3 Réseau neuronal convolutif

3.3.1 Definition

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. Le réseau exécute quelques opération avec les données. Pour chaque opération, familièrement appelée **couche**, consiste une transformation linéaire des données avec une fonction d'activation.

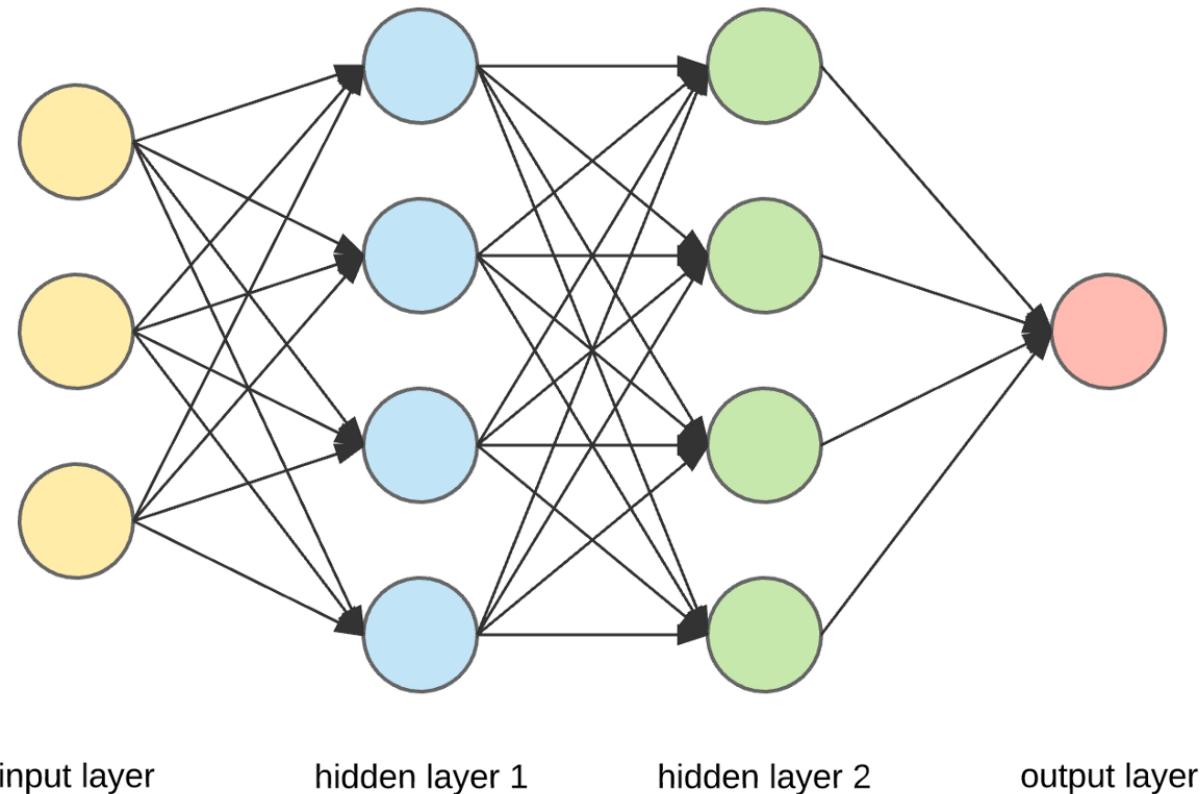


FIGURE 6 – Illustration d'un réseau neuronal convolutif

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, \dots, x_p avec les poids synaptique associé $\alpha_1, \dots, \alpha_p$ et une fonction d'activation

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

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.*

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)

3.3.2 Principes mathématiques

Soit $x \in R^{N \cdot D}$ un point de données tel que chaque colonne de x est un D -dimensions point de données et N le nombre de exemples à entraîner. Soit $W^k \in R^{d_{k-1} \cdot d_k}$ la matrice représentant une transformation linéaire de la sortie de **couche** $k - 1$ (soit $x_{k-1} \in R^{Nd_{k-1}}$) pour obtenir une d_k -dimension $x_{k-1}W_k \in R^{N \cdot d_k}$

Soit $\xi(x)$ une fonction d'activation non-linéaire (par exemple : une fonction sigmoïde $\xi(x) = (1 + e^{-x})^{-1}$, un ReLU $\xi(x) = \max(0, x)$). Donc, la sortie x_k sera :

$$\phi(x, W^1, \dots, W^K) = \xi_K(\xi_{K-1}(\dots \xi_1(xW^1) \dots W^{K-1})W^K)$$

3.4 Optimisation globale

Nous cherchons à aborder le problème d'apprendre les paramètres d'un réseau neuronal. Soit \$(X, Y)\$ \$N\$ exemples à entraîner. Sur le réglage de classification, chaque ligne de \$X \in \mathbb{R}^{N \times D}\$ dénote un point de données dans \$\mathbb{R}^D\$ et chaque ligne de \$Y \in \{1,0\}^{N \times C}\$ dénote la cotisation de chaque point de données. Donc, nous cherchons à optimiser le problème d'apprendre le réseaux :

$$\min_{W^k}_{k=1} l(Y, \phi(X, W^1, \dots, W^K)) + \lambda \xi(W^1, \dots, W^K)$$

avec \$l\$ est la fonction de perte qui donne la conformité entre la sortie et le résultat prévu, \$\xi\$ la fonction de régularisation désigné à éviter le surapprentissage et \$\lambda\$ équilibre paramètres.

4 Deep Image Prior

4.1 Principes

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 Principes 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.

Nous considérons un problème de reconstruction d'une image dégradée avec :

$$x \rightarrow \text{bonne image } x_0 \rightarrow \text{image dégradée } x^* \rightarrow \text{image sortie}$$

Dans **Deep Image Prior**, l'auteur tente de combler le fossé entre deux méthodes populaires en construisant un nouveau préalable explicite en utilisant un réseau de neurones convolutif. Nous pouvons utiliser la distribution a posteriori maximale pour estimer la valeur non observée à partir des données empiriques :

$$x^* = \operatorname{argmax}(x_0 \mid x)$$

En utilisant le Bayesian rule nous obtenons :

$$p(x \mid x_0) = \frac{p(x_0 \mid x)p(x)}{p(x_0)} \propto p(x_0 \mid x)p(x)$$

Au lieu de travailler séparément avec les distributions, nous pouvons formuler l'équation sous forme de problème d'optimisation :

$$x^* = \operatorname{argmax} p(x_0 \mid x) = \operatorname{argmin} -\log p(x_0 \mid x) - \log p(x) = \operatorname{argmin} E(x; x_0) + R(x)$$

avec $E(x; x_0)$ est le terme de données et $R(x)$ est l'image de $p(x)$ Donc, le problème est de minimiser :

$$\min_{\xi} \| A\varphi_{\xi}(z) - y^\gamma \|^2$$

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

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

Donc, les résultats obtenus sont

$$\varphi_\xi(z) = x^L$$

avec les paramètres $\xi = \{W_0, \dots, W_{L-1}, b_0, \dots, b_{L-1}\}$. 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 sera

$$x = \varphi_\xi(z)$$

avec z fixé. La technique vise à paramétriser la solution avec ξ .

4.2 Méthode appliquée

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 une image de $R^{3 \times \text{Height} \times \text{Width}}$ (une image colorée de taille Height x Width pixels à chaque pixel une combinaison de 3 couleurs rouge, vert et bleu). Ici, θ sont des paramètres du réseau.

Suite le travail, nous essayons d'aborder le problème de reconstruire une image x d'une image endommagée x_0 suite une tâche de minimiser :

$$\min_x E(x; x_0) + R(x)$$

où $E(x; x_0)$ est une fonction dépendant des données et $R(x)$ est une régularisation. Dans ce travail, nous essayons de prendre la régularisation $R(x)$ comme le prieur capturé par le réseau convolutif. L'idée est de prendre une appropriée application surjective

$$g : \theta \mapsto x$$

pour avoir :

$$\min_{g(\theta)} E(g(\theta); x_0) + R(g(\theta))$$

pour éliminer le terme $R(x)$. Enfin, nous avons la formule :

$$\min_{\theta} E(f_{\theta}(z); x_0)$$

avec f_{θ} est initialisé au hazard.

En utilisant un méthode d'optimisation (par exemple : descente de gradient), nous cherchons à minimiser le terme $\theta \rightarrow \theta^*$:

$$\theta^* = \operatorname{argmin}_{\theta} E(f_{\theta}(z); x) \text{ et } x^* = f_{\theta^*}(z)$$

4.3 Application sur le débruitage et reconstruction d'image

Le but du débruitage est de reconstruire une image x d'une image bruité x_0 . Le modèle dégradé est parfois connu sur la forme : $x_0 = x + \epsilon$. En prenant

$$E(x; x_0) = \|x - x_0\|^2$$

nous avons un problème d'optimisation :

$$E(f_{\theta}(z); x_0) = \|f_{\theta}(z) - x_0\|^2$$

5 Conclusion

Dans ce projet, nous avons eu le chance de faire la connaissance avec le réseau neuronal convolutif, sa fonctionnement et sa application. Nous avons aussi étudié d'un nouvelle technique de traitement d'image. Cette technique n'a pas besoin quantité massive de données (un seul point de donnée) ni un modèle pre-entraîné (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. Grâce à de nombreuses études inspirées par cette technique, nous espérons que le traitement de l'image à l'avenir sera plus rapide et plus efficace.

Références

- [DH20] Mohammad Zalbagi DARESTANI et Reinhard HECKEL. *Accelerated MRI with Un-trained Neural Networks*. 2020. arXiv : [2007.02471](#).
- [UVL17] Dmitry ULYANOV, Andrea VEDALDI et Victor LEMPITSKY. « Deep Image Prior ». In : (2017). DOI : [10.1007/s11263-020-01303-4](#). arXiv : [1711.10925](#).
- [Vee+18] Dave Van VEEN et al. *Compressed Sensing with Deep Image Prior and Learned Regularization*. 2018. arXiv : [1806.06438](#).
- [Yoo+19] Jaejun YOO et al. *Time-Dependent Deep Image Prior for Dynamic MRI*. 2019. arXiv : [1910.01684](#).