

Une image dans une fonction

Représentations implicites, biais spectral et fréquences

représentation implicite

biais spectral

Fourier

SIREN

NeRF

JPEG

Remplacer une grille de pixels par une formule à 985 paramètres, et comprendre pourquoi elle rate d'abord les détails.

Un cadeau de fin de cours, à lire par curiosité : rien n'est à rendre. Le projet « Une image cachée dans un réseau » reste décrit par sa fiche sur le site et par le README de son squelette ; ce fascicule raconte les idées qui l'entourent et ce qu'elles deviennent au-delà du cours.

1 Pourquoi ce fascicule

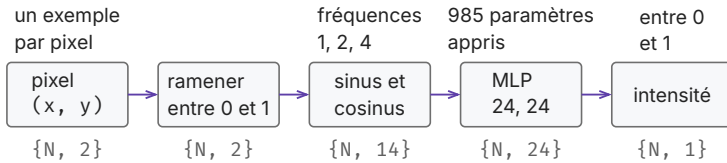
Dans le projet « Une image cachée dans un réseau », vous entraînez un petit réseau à prédire l'intensité d'un pixel à partir de ses coordonnées. Vous écrivez la normalisation, le MLP à deux couches ReLU, la MSE, puis des caractéristiques sinus et cosinus à plusieurs fréquences ; vous comparez les deux représentations sur les mêmes pixels, avec le même budget, et vous expliquez ce que l'expérience démontre.

Ce fascicule prend du recul. Pourquoi représenter une image par une fonction plutôt que par une grille, et qu'est-ce que cela change ? Pourquoi un MLP apprend-il d'abord les variations lentes, et qu'est-ce qui, dans la descente de gradient, produit cette préférence ? D'où viennent les sinus et cosinus, quel rapport ont-ils avec les séries de Fourier de la prépa et avec le format JPEG de vos photos ? Enfin, que devient cette idée quand on passe de deux coordonnées à trois, pour reconstruire une scène entière à partir de photographies, ou quand on l'utilise pour compresser ?

Vous n'y trouverez aucune des quatre fonctions du projet. Les exemples de code sont de petits calculs voisins, en Elixir sans dépendance, dont la sortie a été vérifiée. Rien n'est à rendre ; c'est une lecture du soir.

2 Une image est une fonction

Une image numérique est une grille : W colonnes, H lignes, un nombre par case. Nous avons tellement l'habitude de cette grille que nous oublions qu'elle est un choix. Derrière elle, il y a une fonction $I(x, y)$ qui associe une intensité à chaque position du plan, et la grille n'en est qu'un échantillonnage : la valeur de I aux points (k, l) entiers, et rien entre deux. Le projet propose de représenter l'image directement par une fonction, calculée par un réseau de paramètres θ : $I_\theta(x, y)$ reçoit deux coordonnées et rend une intensité. On parle de *représentation implicite*, par opposition à la grille, explicite, où chaque valeur est écrite noir sur blanc. Dans la littérature récente, on dit aussi *champ neuronal*.



La chaîne du projet, de gauche à droite. Chaque pixel est un exemple ; les coordonnées normalisées, puis leurs sinus et cosinus aux fréquences 1, 2 et 4, entrent dans un MLP qui produit une intensité. Les formes des tenseurs suivent la convention d'AP4, un exemple par ligne.

Qu'est-ce qui change, concrètement ? Trois choses. La *résolution* d'abord : une fonction est définie partout, et l'on peut l'interroger entre deux pixels, sur une grille de 64 par 64 après l'avoir apprise sur 16 par 16. Le dessin obtenu est lisse et agréable, mais retenez bien que la fonction n'a vu que 153 pixels : ce qu'elle dessine entre eux est une interpolation choisie par ses poids, pas une information retrouvée. La *dérivabilité* ensuite : un réseau est une formule, et N_x sait la dériver par rapport à ses entrées aussi bien que par rapport à ses paramètres. La dérivée de I_θ en x donne la pente de l'intensité, c'est-à-dire les contours, sans différence finie ni filtre. La *taille* enfin : la grille coûte $W \times H$ nombres, la fonction coûte ses paramètres. Pour notre image de 256 pixels, le réseau à fréquences en a 985 : ce n'est pas une compression, et le README vous le dit. Pour une photographie de plusieurs millions de pixels, le rapport s'inverse, et c'est une piste que nous retrouverons.

PROPRIÉTÉ

Interroger la fonction sur une grille plus fine n'ajoute aucune information : les seules valeurs connues sont les pixels d'apprentissage, et tout le reste est décidé par la famille de fonctions du réseau et par le chemin suivi par l'optimiseur. Pourquoi c'est vrai : deux réseaux qui coïncident sur les 153 pixels appris peuvent différer arbitrairement entre eux, et la perte ne les distingue pas. Une image agrandie « plus nette » que l'original est donc une invention, parfois plausible, jamais une mesure.

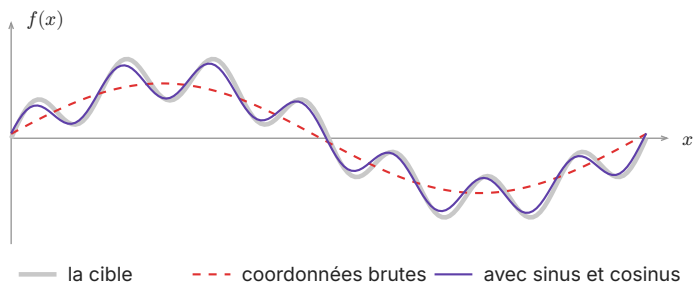
Le lien avec le cours est direct, et c'est sans doute le projet d'AP9 le plus proche d'AP4 et d'AP5. En AP4, les coordonnées d'une grille étaient déjà les données, rangées dans un tenseur $\{n, 2\}$, et une règle connue donnait la cible. Ici, la règle est l'image elle-même. En AP5, un MLP, une perte et des lots suffisaient à apprendre une frontière ; ici, les mêmes briques apprennent une surface d'intensités. Ce qui est nouveau est la question posée : non plus « le modèle généralise-t-il à de nouveaux points ? », mais « quelle représentation de l'entrée permet au même réseau de décrire les détails ? ». Les pixels de validation et de test sont des pixels laissés de côté dans la même image ;

ils mesurent une interpolation à l'intérieur d'une image connue, pas une capacité à décrire une autre image.

3 Le biais spectral : les détails viennent en dernier

Lancez l'entraînement avec les coordonnées brutes et regardez `comparaison.svg` : les grandes zones claires et sombres sont en place, mais les ondulations fines de l'image sont lissées, comme si le réseau avait dessiné l'image à travers un verre dépoli. Prolonger l'entraînement aide peu. Ce n'est pas une particularité de votre code : c'est une propriété des réseaux ReLU entraînés par descente de gradient, décrite et nommée en 2019 par Rahaman et ses coauteurs à la conférence ICML, le *biais spectral*. En décomposant la sortie d'un réseau en fréquences au fil de l'entraînement, ils observent que les basses fréquences, les variations lentes, sont apprises en premier et vite, tandis que les hautes fréquences, les variations rapides, arrivent tard, ou jamais.

Prenons une fonction d'une seule variable pour le voir : une onde lente sur laquelle se superpose une ondulation sept fois plus rapide. Avec le même budget de pas, un MLP nourri de la coordonnée brute suit l'onde lente et ignore l'ondulation ; le même MLP nourri de sinus et de cosinus suit les deux.



Tracé schématisé, pas une mesure : la cible en gris contient une onde lente et une ondulation rapide. Un réseau sur coordonnées brutes retrouve d'abord l'onde lente ; avec des caractéristiques de Fourier, il peut suivre aussi l'ondulation.

Deux raisons, l'une de représentation, l'autre d'optimisation, expliquent ce comportement. Commençons par la représentation. Un réseau à activations ReLU calcule une fonction *affine par morceaux* : entre deux « cassures », la sortie est une droite, et une cassure apparaît là où l'entrée d'une unité ReLU change de signe. Avec une seule coordonnée en entrée et K unités cachées sur une couche, il y a au plus K

cassures, donc au plus $K + 1$ morceaux. Or suivre une sinusoïde qui fait f périodes sur l'intervalle demande au moins deux cassures par période. Avec 24 unités, on ne peut pas dépasser une douzaine de périodes, et chaque période coûteuse en cassures est une période de moins pour le reste. Une seconde couche multiplie le nombre de morceaux possibles, mais ne change pas l'ordre dans lequel l'optimiseur les place.

Vérifions l'affirmation sur un réseau minuscule, écrit à la main : une entrée, trois unités ReLU, une sortie. Nous évaluons la fonction sur cent intervalles, calculons la pente de chacun et comptons les changements de pente.

```

1  relu = fn z -> max(z, 0.0) end
2  # une entrée, trois unités cachées, une sortie
3  reseau = fn x ->
4    0.5 * relu.(4 * x - 1) + 0.5 * relu.(3 - 4 * x)
5    - 0.3 * relu.(5 * x - 3) + 0.1
6  end
7
8  xs = Enum.map(0..100, &(81 / 100))
9  pentes =
10    xs
11    |> Enum.map(reseau)
12    |> Enum.chunk_every(2, 1, :discard)
13    |> Enum.map(fn [a, b] -> Float.round((b - a) * 100, 6) end)
14  cassures =
15    pentes
16    |> Enum.chunk_every(2, 1, :discard)
17    |> Enum.count(fn [a, b] -> a != b end)
18
19 IO.inspect(Enum.uniq(pentes), label: "pentes rencontrées")
20 IO.inspect(cassures, label: "cassures")

```

```

CONSOLE  pentes rencontrées: [-2.0, 0.0, -1.5, 0.5]
cassures: 3

```

Quatre pentes, trois cassures, en $x = 0,25, 0,6$ et $0,75$: une par unité, là où son argument s'annule. Pouvez-vous retrouver ces trois positions à partir des coefficients ? Un réseau de ce genre, quels que soient ses poids, ne peut pas osciller plus d'une fois et demie.

La seconde raison tient à la descente de gradient elle-même. Chaque pas modifie tous les paramètres d'après la somme des erreurs sur le lot. Une erreur lente, du même signe sur une large région, tire chaque poids dans une direction cohérente d'un pixel à l'autre : les contributions s'additionnent et le pas est grand. Une erreur rapide, qui

change de signe d'un pixel au suivant, tire tantôt dans un sens, tantôt dans l'autre : les contributions s'annulent presque, et le pas est minuscule. Les composantes lentes de l'erreur sont donc corrigées vite, les composantes rapides lentement. Rahaman et ses coauteurs, comme d'autres équipes la même année, donnent à cet argument une forme précise, mais l'intuition suffit pour lire vos courbes : une MSE qui stagne n'est pas forcément un taux mal réglé, c'est peut-être un réseau qui a fini d'apprendre ce qu'il savait apprendre.

À RETENIR

Un réseau ReLU est affine par morceaux et la descente de gradient corrige d'abord les erreurs lentes. Les deux effets vont dans le même sens : les détails fins d'une image sont ce qu'un MLP sur coordonnées brutes apprend en dernier, et souvent jamais dans le budget.

4 Donner des fréquences au réseau

Puisque le réseau peine à fabriquer des oscillations, fournissons-les-lui. C'est l'idée des *caractéristiques de Fourier* : au lieu de la coordonnée x , on donne au réseau $\sin(2\pi f x)$ et $\cos(2\pi f x)$ pour plusieurs fréquences f , et de même pour y . Ces valeurs sont calculées une fois pour toutes, elles ne s'apprennent pas ; le réseau apprend ensuite à les combiner, et une combinaison affine de sinusôides oscille sans qu'aucune ReLU n'ait à le faire. Avec les fréquences 1, 2 et 4 du projet, l'entrée passe de 2 à 14 nombres par pixel.

L'article de référence est celui de Tancik et ses coauteurs, en 2020, à la conférence NeurIPS. Leur version tire les fréquences au hasard selon une loi normale d'écart-type σ , et c'est σ qui règle tout : trop petit, l'image reste floue ; trop grand, le réseau suit le bruit et ajoute des oscillations entre les pixels appris. Ils montrent que ces caractéristiques reviennent à donner au réseau un noyau de similarité entre points dont la largeur se règle, un résultat qui s'appuie sur la théorie du noyau tangent. La même année, Sitzmann et ses coauteurs proposent une autre route, SIREN : remplacer la ReLU par un sinus dans toutes les couches, avec une initialisation soignée et un facteur 30 sur la première couche. La dérivée d'un sinus est un cosinus, c'est-à-dire un sinus décalé : la dérivée d'un SIREN est encore un SIREN, et l'on peut entraîner le réseau sur ses dérivées, par ex. pour qu'il satisfasse une équation aux dérivées partielles.

Le projet, lui, reprend la variante de NeRF que nous retrouverons plus loin : des fréquences fixes, en puissances de deux, 1, 2, 4. C'est la plus simple à lire et à tester, et elle suffit pour voir l'effet.

4.1 La fréquence la plus fine qu'une grille peut porter

Pourquoi ne pas ajouter des fréquences 8, 16, 32, pour être sûr ? Posons la question à la grille. Nos seize pixels d'une ligne ont pour coordonnées normalisées $x = k/15$, k de 0 à 15. La caractéristique $\cos(2\pi f x)$ ne prend donc que seize valeurs, et pour un entier f , on a $\cos(2\pi(15 - f)k/15) = \cos(2\pi k - 2\pi f k/15) = \cos(2\pi f k/15)$, parce que k est entier. Les fréquences f et $15 - f$ donnent exactement les mêmes cosinus, et des sinus opposés ! Au-delà de 7, une fréquence entière ne porte rien de neuf : $f = 8$ répète $f = 7$, et $f = 15$ répète $f = 0$, c'est-à-dire une constante. C'est le *repliement*, en anglais *aliasing*, et c'est le contenu du théorème d'échantillonnage que Shannon a énoncé en 1949 : une grille de pas $1/15$ ne peut représenter que les fréquences inférieures à 7,5 périodes par unité.

```

1 pixels = Enum.to_list(0..15)
2 cosinus = fn f ->
3   Enum.map(pixels, fn k ->
4     Float.round(:math.cos(2 * :math.pi() * f * k / 15), 3)
5   end)
6 end
7
8 IO.inspect(Enum.take(cosinus.(1), 5), label: "f = 1, cinq premiers
  pixels")
9 IO.inspect(Enum.take(cosinus.(7), 5), label: "f = 7, cinq premiers
  pixels")
10 IO.inspect(cosinus.(7) == cosinus.(8), label: "f = 7 et f = 8
    identiques")
11 IO.inspect(Enum.uniq(cosinus.(15)), label: "f = 15, valeurs
    distinctes")

```

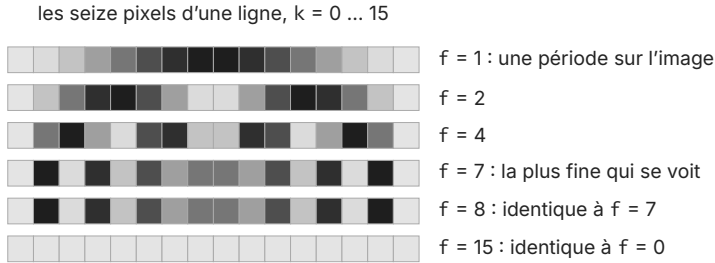
```

CONSOLE  f = 1, cinq premiers pixels: [1.0, 0.914, 0.669, 0.309, -0.105]
f = 7, cinq premiers pixels: [1.0, -0.978, 0.914, -0.809, 0.669]
f = 7 et f = 8 identiques: true
f = 15, valeurs distinctes: [1.0]

```

Vous comprenez maintenant deux remarques du README. D'abord, les coordonnées brutes sont conservées à côté des sinusoides : des fréquences entières prennent les mêmes valeurs en $x = 0$ et en $x = 1$, et seule la coordonnée distingue les deux bords. Ensuite, un réglage de fréquences n'est pas transportable d'une taille d'image

à une autre : sur une grille de 64 pixels, $f = 8$ est une fréquence utile ; sur 16, c'est un doublon. La figure ci-dessous montre ce repliement sur les seize pixels.



Le cosinus de fréquence f évalué sur les seize pixels d'une ligne, en niveaux de gris. Les lignes $f = 7$ et $f = 8$ sont identiques ; la ligne $f = 15$ est uniforme. Au-delà de la moitié du nombre de pixels, une fréquence entière se replie sur une fréquence plus basse.

PIÈGE

Une fréquence plus haute que la grille ne peut porter n'est pas inoffensive. Elle ajoute des colonnes à l'entrée, donc des paramètres, et ces colonnes répètent une information déjà présente ou valent une constante : le réseau grossit sans rien gagner, et la comparaison « brut contre Fourier » devient plus difficile à interpréter. Choisissez vos fréquences en regardant la grille.

5 Une image est déjà une somme de fréquences

Si le réseau a besoin de sinusôides pour décrire une image, c'est que l'image en est faite. Vous connaissez l'idée depuis la prépa : Fourier, dans sa *Théorie analytique de la chaleur* de 1822, montre qu'une fonction périodique raisonnable est une somme de sinus et de cosinus de fréquences entières, chacun affecté d'un coefficient. Un signal carré, qui vaut 1 sur une demi-période et -1 sur l'autre, s'écrit ainsi $4/\pi$ fois la somme des $\sin(2\pi kx)/k$ pour k impair. Les coefficients décroissent comme $1/k$: les hautes fréquences sont là, mais de plus en plus faibles. Vérifions au sommet du créneau, en $x = \frac{1}{4}$, où la fonction vaut 1 :

```

1 somme = fn x, n ->
2   1..n
3   |> Enum.filter(&(rem(&1, 2) == 1))
4   |> Enum.map(fn k -> :math.sin(2 * :math.pi() * k * x) / k end)
5   |> Enum.sum()
6   |> Kernel.*(4 / :math.pi())
7 end
8
9 for n <- [1, 3, 7, 15, 63, 255] do
10   IO.puts("#{n} harmoniques : #{Float.round(somme.(0.25, n), 4)}")
11 end

```

```

CONSOLE  1 harmoniques : 1.2732
3 harmoniques : 0.8488
7 harmoniques : 0.9216
15 harmoniques : 0.9604
63 harmoniques : 0.9901
255 harmoniques : 0.9975

```

La somme approche 1 en oscillant, et la convergence est lente parce que le créneau a des sauts, le pire cas pour une série de Fourier. Et pourtant, sept harmoniques suffisent à porter 95 % de l'énergie du signal : la part qui reste au-delà de l'harmonique n est, par le théorème de Parseval, la somme des carrés des coefficients abandonnés, soit environ $0,4/n$. C'est la même structure que le goulot d'un autoencodeur : tronquer une somme de fréquences coûte exactement ce que les termes supprimés contenaient, ni plus ni moins.

Les images naturelles, photographies de paysages ou de visages, ont une propriété statistique qui rend cette troncature rentable : leur énergie décroît à peu près comme l'inverse du carré de la fréquence, une observation que Field a établie en 1987 sur des photographies de scènes naturelles. L'essentiel d'une image est dans ses variations lentes, et les détails fins, bien que visibles, pèsent peu dans l'erreur quadratique. Le format JPEG, normalisé en 1992 sous le nom de recommandation T.81, exploite exactement cela. Il découpe l'image en blocs de huit pixels par huit, applique à chaque bloc une transformée en cosinus discrète, la DCT proposée par Ahmed, Natarajan et Rao en 1974, qui exprime le bloc comme une somme de 64 cosinus à deux dimensions, puis *quantifie* les coefficients : les basses fréquences sont conservées avec précision, les hautes sont arrondies grossièrement, souvent à zéro. Un fichier JPEG est, au fond, une image décrite par ses fréquences, dont on a jeté les plus fines.

Relisons alors le biais spectral à cette lumière. Un MLP qui apprend d'abord les basses fréquences apprend d'abord ce qui compte le plus dans la MSE ; sur une

photographie, ce n'est pas un défaut mais un bon réflexe. Le souci n'apparaît que lorsque les détails comptent, comme sur `ondes`, et c'est là que les caractéristiques de Fourier tendent la main au réseau : elles lui donnent les fréquences qu'il mettrait trop longtemps à construire. Votre comparaison brut contre Fourier mesure cette différence, sur une image choisie pour la rendre visible. Sur `degrade`, elle pourrait ne rien montrer du tout, et ce serait aussi un résultat.

6 Au-delà du projet : trois dimensions, et la compression

Si une fonction de (x, y) peut décrire une image, une fonction de (x, y, z) peut décrire un volume. C'est le pas franchi par NeRF, pour *Neural Radiance Fields*, présenté par Mildenhall, Srinivasan, Tancik, Barron, Ramamoorthi et Ng à la conférence ECCV en 2020. Leur réseau reçoit un point de l'espace et une direction de vue, et rend une couleur et une densité, c'est-à-dire la probabilité que la lumière s'arrête là. Pour produire un pixel, on lance un rayon depuis la caméra, on interroge le réseau en une ou deux centaines de points le long du rayon, et l'on accumule couleurs et densités comme le ferait un rendu volumique. Et pour l'entraîner ? Exactement comme le vôtre : on dispose de quelques dizaines de photographies d'une scène, avec la position de la caméra pour chacune ; chaque pixel de chaque photo est un exemple, et la perte est la MSE entre la couleur rendue et la couleur photographiée. Les coordonnées passent par un encodage en sinus et cosinus de dix fréquences en puissances de deux, celui que votre projet reprend à trois fréquences, et l'article montre que sans lui, les scènes sont floues. La figure du projet, avec ses deux coordonnées, est la coupe à deux dimensions de cette idée.

La *compression* est l'autre direction. Nous avons noté que 985 paramètres pour 256 pixels n'est pas une bonne affaire ; mais Dupont, Goliński, Alizadeh, Teh et Doucet, en 2021, retournent l'argument avec COIN. Ils entraînent un petit réseau à sinus, un SIREN, sur une photographie de 768 par 512 pixels, puis stockent ses poids arrondis sur 16 bits au lieu des pixels. Pour décoder, on évalue le réseau en chaque position. À très bas débit, quelques dixièmes de bit par pixel, le résultat est meilleur que JPEG, sans même recourir au codage entropique dont JPEG se sert. Le prix est le temps d'encodage, qui est un entraînement complet par image. Ce n'est pas un format que vous utiliserez demain, mais c'est la preuve qu'une fonction peut être un fichier.

Depuis, le domaine a bougé vite. Müller, Evans, Schied et Keller, en 2022, remplacent l'essentiel du MLP par une table de hachage de caractéristiques apprises, et

l'entraînement d'un NeRF passe d'heures à secondes. En 2023, Kerbl, Kopanas, Leimkühler et Drettakis proposent le *Gaussian splatting*, qui abandonne la fonction au profit de millions de petites gaussiennes explicites placées dans l'espace, chacune avec sa couleur et son opacité, ajustées par la même boucle rendu, comparaison, gradient ; le rendu devient temps réel, et la question de savoir ce que les champs neuronaux gardent d'irremplaçable est, au moment où j'écris, ouverte.

EN PRATIQUE

Toutes ces méthodes partagent le protocole de votre projet : des exemples qui sont des positions, une cible mesurée, un rendu dérivable, une MSE et un optimiseur. Quand vous lirez un de ces articles, cherchez d'abord ces cinq éléments ; le reste est de l'ingénierie autour.

POUR EN PARLER

Trois affirmations à pouvoir défendre : une image agrandie par la fonction n'est pas une information retrouvée ; un MLP ReLU apprend d'abord les basses fréquences, pour une raison de représentation et une raison d'optimisation ; sur une grille de seize pixels, les fréquences f et $15 - f$ sont indiscernables.

Pour aller plus loin

Fourier features let networks learn high frequency functions in low dimensional domains, M. Tancik et al. (2020).

<https://arxiv.org/abs/2006.10739>

L'article dont le projet s'inspire ; la page du projet, sur bmild.github.io/fourfeat, montre l'effet de l'écart-type des fréquences sur une image en quelques secondes.

Implicit neural representations with periodic activation functions, V. Sitzmann, J. Martel, A. Bergman, D. Lindell et G. Wetzstein (2020).

<https://arxiv.org/abs/2006.09661>

SIREN : le sinus comme activation, et ce que cela permet sur les dérivées ; les exemples sur images, sons et formes sont parlants.

On the spectral bias of neural networks, N. Rahaman et al. (2019).

<https://arxiv.org/abs/1806.08734>

La mise en évidence expérimentale et théorique du biais spectral ; les figures sur des fonctions d'une variable prolongent la section 3.

NeRF: representing scenes as neural radiance fields for view synthesis, B. Mildenhall et al. (2020).

<https://arxiv.org/abs/2003.08934>

La même idée en trois dimensions, avec l'encodage en fréquences que vous avez programmé ; lisez d'abord la section sur l'encodage positionnel.

COIN: compression with implicit neural representations, E. Dupont, A. Golinski, M. Alizadeh, Y. W. Teh et A. Doucet (2021).

<https://arxiv.org/abs/2103.03123>

Quatre pages : un réseau par image, des poids sur 16 bits, et une comparaison honnête avec JPEG qui dit aussi ce qui ne marche pas.

Neural fields in visual computing and beyond, Y. Xie et al. (2022).

<https://arxiv.org/abs/2111.11426>

Un état de l'art qui range des centaines d'articles selon le protocole de la section 6 ; utile pour voir le paysage d'un coup.

3D Gaussian splatting for real-time radiance field rendering, B. Kerbl, G. Kopanas, T. Leimkühler et G. Drettakis (2023).

<https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/>

La page du projet, avec vidéos et code : ce que devient le rendu de scènes quand on remplace la fonction par des primitives explicites.

Recommandation T.81 : le format JPEG, ITU-T (1992).

<https://www.w3.org/Graphics/JPEG/itu-t81.pdf>

La norme elle-même, librement accessible ; les tables de quantification de l'annexe K montrent noir sur blanc quelles fréquences sont sacrifiées.

Une question pour la route

Entraînez deux réseaux de même architecture, avec la même graine, sur deux images différentes, et faites la moyenne de leurs poids, paramètre par paramètre. Obtenez-vous l'image moyenne ? Avant d'essayer, demandez-vous ce que la ReLU fait d'une moyenne, et si deux réseaux qui calculent la même fonction ont forcément les mêmes poids. Pour toute question, vous pouvez me contacter par email avec la commande, l'erreur et le plus petit exemple qui reproduit le souci.