Elaboration d'une stratégie de pruning par régularité

Analyse par transformée de Fourier des filtres de convolutions

Alexandre CHALIN, Josselin TESTE

2 novembre 2025

Résumé

Cette étude présente la conception d'une méthode de pruning structurel basée sur la mesure de la régularité des poids au sein des filtres de convolution. En utilisant la transformée de Fourier, cette méthode permet de détecter la présence de régularités à l'échelle du filtre, mais aussi localement en son sein. Appliquée à un modèle conçu pour son évaluation, elle permet de réduire la taille de ce dernier tout en limitant la perte de précision par rapport aux méthodes de pruning classiques.

Table des matières

1 Conception d'un nouveau modèle **3**
1.1 Introduction . . . 3
1.2 Objectif . . . 5
1.3 Approche . . . 6
1.4 Conception du nouveau modèle . . . 7
1.4.1 Augmentation du kernel . . . 7
1.4.2 Amélioration du nouveau modèle . . . 9
Réduction du kernel size de 5 à 4 . . . 9
Bottleneck 1*1 . . . 10
AdaptativeAvgPool(1,1) . . . 10
1.5 Entrainement du modèle . . . 12

2 Définition de la méthode pruning **13**
2.1 Projection dans un espace continue : l'espace de Schwartz $S(\mathbb{R}^2)$ . . . 14
2.1.1 Définition . . . 14
2.1.2 Usage . . . 15
2.1.3 Construction . . . 15
2.1.4 Simulation . . . 16
2.1.5 Zero-padding . . . 16
2.2 Mesure de régularité globale . . . 17
2.2.1 Transformée de Fourier . . . 17
2.2.2 Théorème de Plancherel . . . 17
2.2.3 Espace de Sobolev . . . 17
Définition . . . 17
Usage . . . 18
Application . . . 18
Simulation . . . 19
Limite . . . 20
2.3 Mesure de régularité locale . . . 21
2.3.1 Transformer de fourier fenêtré . . . 21
2.3.2 Choix de la fenêtre . . . 21
2.3.3 Choix des paramètres de la mesure de régularité : τ . . . 23

3 Test et comparaison **25**
3.1 Protocole de test . . . 25
3.2 Paramètres de la simulation . . . 26
3.3 Résultats . . . 26
3.3.1 Accuracy avant fine-tuning . . . 26
3.3.2 Accuracy après fine-tuning . . . 31
3.4 Conclusion et perspectives . . . 35

1 Conception d'un nouveau modèle

L'étude de l'embarquement du modèle a révélé que celui-ci ne pouvait pas être déployé sur la cible en raison de sa taille mémoire et de sa complexité computationnelle. L'objectif est donc de concevoir une version plus compacte du modèle, tout en conservant les principes architecturaux de la version initiale.

1.1 Introduction

Une image est composée de deux zones d'intérêt : le domaine physique Ω et l'espace du signal $X(\Omega)$. Le domaine physique Ω est défini comme le produit cartésien de deux ensembles discrets, $\Omega = X \times Y$, où $H = |X|$ et $W = |Y|$ représentent respectivement la hauteur et la largeur de l'image. Dans notre cas, la grille régulière est de dimension 32×32. L'espace du signal $X(\Omega)$ associe à chaque élément de Ω, c'est-à-dire à chaque pixel, un triplet (R, G, B) avec $R, G, B \in [0, 255]$, décrivant la couleur du pixel.

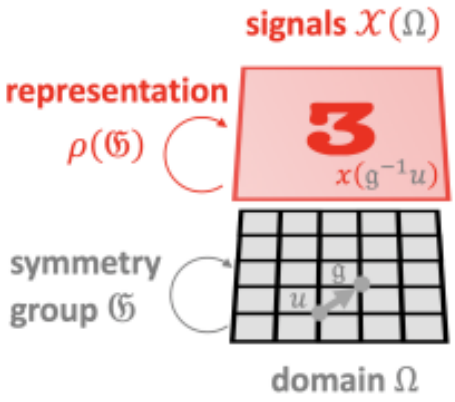


FIGURE 1 – Représentation structuré d'une image

Soit $x \in \Omega$ tel que $x = (i, j)$ avec $i \in X$ et $j \in Y$. Le pixel correspondant possède quatre voisins : $(i+1, j)$, $(i-1, j)$, $(i, j-1)$ et $(i, j+1)$. Notre postulat principal est que les valeurs de ces points voisins — c'est-à-dire leur représentation dans $X(\Omega)$ — ne sont pas totalement décorrélées de la valeur du signal au point x. En effet, si tel était le cas, toutes les images présenteraient une structure aléatoire, semblable à celle-ci :

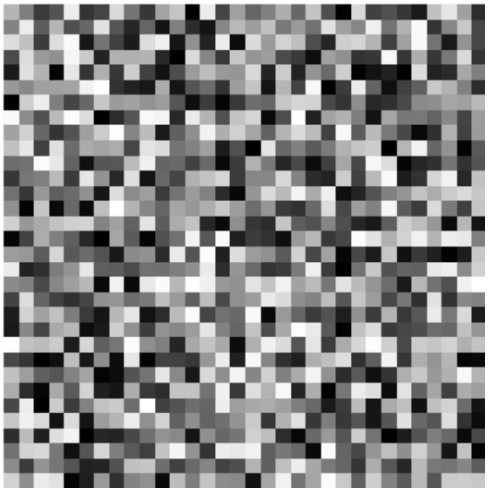


FIGURE 2 – Image aléatoire

La performance d'un réseau de neurones peut être évaluée non seulement par la précision de ses prédictions, mais aussi — dans le cas d'un déploiement sur un système embarqué — par le temps d'inférence et la taille du modèle. Dans cette étude, nous nous concentrons sur ce dernier critère.

Étant donné un réseau de neurones convolutionnel (CNN), comment réduire sa taille sans dégrader sa précision ? La taille d'une couche linéaire dépend du nombre de neurones qu'elle contient, tandis que celle d'une couche de convolution est déterminée par le nombre de filtres et leurs dimensions. Réduire la taille d'une couche de convolution revient donc à diminuer le nombre ou la taille des filtres. En pratique, c'est généralement le nombre de filtres que l'on ajuste, car les réseaux de neurones modernes peuvent comporter des dizaines, voire des centaines de couches convolutionnelles, alors que la taille des noyaux (kernels) est souvent fixée à 3×3.

Ainsi, si l'on considère que la compression d'un réseau de convolution consiste avant tout à réduire le nombre de ses filtres, la question essentielle devient : comment sélectionner les filtres à conserver et ceux à supprimer ? Plusieurs approches ont été proposées dans la littérature, les plus courantes étant les suivantes :

Soit Ω_h le domaine d'un filtre $h = (h_{i,j})_{i,j \in \Omega_h}$, avec $h_{i,j} = X_j(i,j)$, où X_j est un signal défini sur Ω_h, représentant la valeur du signal au point (i,j) ou le "poids" en (i,j). On définit alors :

La norme L1 du filtre h :

$$\|h\|_1 = \sum_{i=1}^{M} \sum_{j=1}^{N} |h_{i,j}|$$

La norme L2 du filtre h :

$$\|h\|_2 = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{N} h_{i,j}^2}$$

La norme infini du filtre h :

$$\|h\|_\infty = \max_{i,j} |h_{i,j}|$$

D'autres normes existent, mais celles présentées ci-dessus demeurent les principales. Il s'agit alors d'appliquer la norme choisie à chacun des filtres des couches de convolution, puis de définir un seuil sur les valeurs de ces normes : un filtre dont la norme est supérieure au seuil est conservé, sinon il est supprimé. Ces normes sont des mesures de magnitude, quantifiant l'importance numérique des poids. Elles supposent ainsi qu'un filtre de faible magnitude contribue peu à la sortie du modèle et peut donc, selon un seuil donné, être supprimé. Cette approche est naturelle, puisque l'opération de convolution repose sur la simple valeur numérique des poids. De plus, en s'appuyant sur un calcul de somme, ces normes sont invariantes par permutation des valeurs du signal sur le filtre.

FIGURE 3 – 5 filtres au hasard extrait du modèle initial entrainé

Les deux filtres ci-dessus, bien que présentant des motifs radicalement différents, possèdent des normes L_1, L_2 et L_∞ identiques. L'objectif de cet exemple volontairement simpliste n'est pas de pointer les limites des normes L_1, L_2 ou L_∞, mais plutôt d'introduire une approche conceptuelle nouvelle : la régularité.

Comme mentionné en introduction, les images présentent, par la présence de motifs, formes ou bords, certaines régularités. Le rôle d'un filtre de convolution est de capturer de tels motifs en agissant comme un masque glissant sur le domaine de l'image. Ainsi, ces filtres doivent eux-mêmes témoigner de régularités.

Cependant, si ces régularités existent, comment les mesurer ? Un filtre peut être globalement régulier, localement régulier, ou régulier à la fois localement et globalement, mais selon des motifs différents.

Malgré ces nuances, nous faisons l'hypothèse que la magnitude d'un filtre n'est plus une condition suffisante pour sa conservation dans une méthode de pruning : un filtre, au-delà de sa magnitude, devrait également être régulier. Par ailleurs, nous n'affirmons pas que la régularité soit suffisante, mais que ces deux approches — magnitude et régularité — doivent être complémentaires.

1.2 Objectif

Notre objectif est de construire un modèle de classification pour le dataset CIFAR-10. Nous souhaitons en particulier minimiser sa taille tout en conservant une précision d'au moins 80%. En effet, comme nous l'avons constaté précédemment, le modèle VGG proposé est trop volumineux pour être embarqué sur un STM32.

1.3 Approche

Afin de définir un modèle plus compact, nous partirons d'une architecture similaire à celle initialement proposée. Ensuite, nous construirons une mesure de régularité. Cette mesure nous permettra de définir une méthode de pruning structurel, que nous utiliserons ensuite pour réduire la taille de notre modèle.

Une fois la mesure créée, les étapes de pruning structurel seront les suivantes :

1. **Entraînement du modèle** : nous récupérons le modèle initial puis l'entraînons afin de pouvoir extraire les poids des filtres pour chaque couche.
2. **Classification des filtres** : nous classifions, selon une méthode définie, les filtres en fonction de leur régularité après le premier entraînement.
3. **Suppression des filtres et reconstruction du modèle** : nous retirons un pourcentage p de filtres selon la classification précédente, puis reconstruisons un modèle avec ces nouveaux filtres.
4. **Fine tuning du nouveau modèle** : une fois $p\%$ des filtres supprimés dans le modèle initial, nous réentraînons le modèle réduit obtenu.
5. **Évaluation du nouveau modèle** : nous évaluons le modèle réduit et comparons ses performances en termes de précision et de taille mémoire avec le modèle initial non pruné.

Le postulat de départ repose sur la présence de régularités dans les poids des filtres. Pour cela, commençons par visualiser les poids du modèle entraîné afin d'intuiter cette hypothèse initiale.

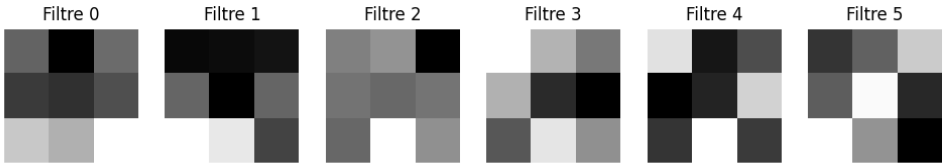


FIGURE 4 – 5 filtres extrait au hasard du modèle initial entrainé

Ici, on constate un problème d'échelle. Pour des kernels de dimension 3, il est difficile de distinguer un signal ou une régularité d'un simple bruit. Les données des filtres ne sont pas suffisamment riches pour identifier des motifs dans leurs distributions ; avec si peu de données, la probabilité d'interpréter un signal comme du bruit est élevée. Ainsi, nous ne pouvons pas travailler avec le modèle tel quel et privilégierons plutôt des kernels de taille 5.

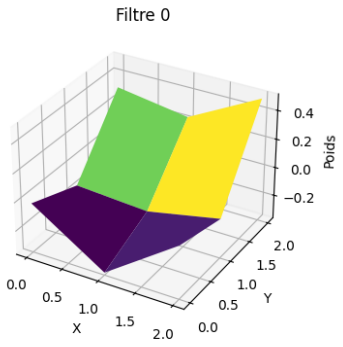


FIGURE 5 – Filtre de convolution affiché en 3D pour kernel size 3x3 : difficile d'identifier quel que pattern que ce soit.

En augmentant la taille des kernels, les filtres font apparaitre une topologie plus riche, proprice à l'analyse.

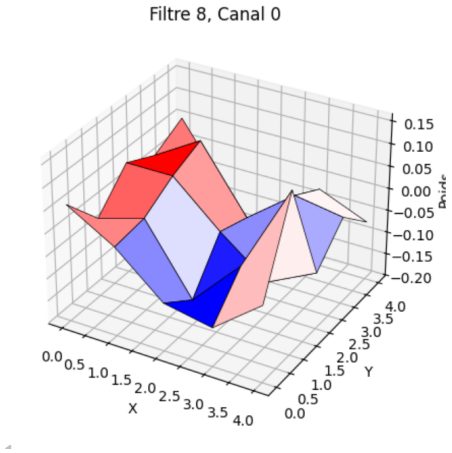


FIGURE 6 – Filtre de convolution avec un kernel de taille 5x5

Sur l'image ci-dessus, les géométries rouges correspondent aux valeurs de poids positives, tandis que les géométries bleues représentent les valeurs de poids négatives. À première vue, il semble plus facile d'identifier des points critiques.

1.4 Conception du nouveau modèle

1.4.1 Augmentation du kernel

Les filtres de convolution utilisés dans le modèle initial ont un kernel de dimension 3×3. Dans notre cas, afin de pouvoir établir une mesure suffisamment pertinente sur les filtres, nous augmentons la taille des kernels de 3 à 5. Cette augmentation accroît considérablement le nombre de paramètres du réseau : chaque filtre passe de 9 à 25 paramètres. Ainsi, pour une couche avec c_{in} canaux et c_{out} filtres, la couche contient $\Delta = c_{\text{in}} \times c_{\text{out}} \times 25$ nouveaux paramètres.

$$\Delta param = c_{in} * c_{out} * (25 - 9)$$

Ainsi pour chaque couche on obtient :

1. Conv2D(32, 3×3, input_shape=(32,32,3))

$$\Delta = 16 \times 3 \times 32 = 1,536$$

2. Conv2D(32, 3×3) (entrée = 32 canaux)

$$\Delta = 16 \times 32 \times 32 = 16,384$$

3. Conv2D(64, 3×3) (entrée = 32 canaux)

$$\Delta = 16 \times 32 \times 64 = 32,768$$

4. Conv2D(64, 3×3) (entrée = 64 canaux)

$$\Delta = 16 \times 64 \times 64 = 65,536$$

5. Conv2D(128, 3×3) (entrée = 64 canaux)

$$\Delta = 16 \times 64 \times 128 = 131,072$$

6. Conv2D(128, 3×3) (entrée = 128 canaux)

$$\Delta = 16 \times 128 \times 128 = 262,144$$

Soit en tout 509,440 nouveaux paramètres. Le modèle devient alors celui-ci :

```
model = nn.Sequential(
 # Bloc 1
 nn.Conv2d(3, 32, kernel_size=5, padding=2),
 nn.BatchNorm2d(32),
 nn.ReLU(),
 nn.Conv2d(32, 32, kernel_size=5, padding=2),
 nn.ReLU(),
 nn.BatchNorm2d(32),
 nn.SpatialDropout2d(0.25) if hasattr(nn, 'SpatialDropout2d')
    else nn.Dropout2d(0.25),
 nn.MaxPool2d(2),

 # Bloc 2
 nn.Conv2d(32, 64, kernel_size=5, padding=2),
 nn.BatchNorm2d(64),
 nn.ReLU(),
 nn.Conv2d(64, 64, kernel_size=5, padding=2),
 nn.ReLU(),
 nn.BatchNorm2d(64),
 nn.SpatialDropout2d(0.25) if hasattr(nn, 'SpatialDropout2d')
    else nn.Dropout2d(0.25),
 nn.MaxPool2d(2),

 # Bloc 3
 nn.Conv2d(64, 128, kernel_size=5, padding=2),
```

```
        nn.BatchNorm2d(128),
        nn.ReLU(),
        nn.SpatialDropout2d(0.25) if hasattr(nn, 'SpatialDropout2d')
            else nn.Dropout2d(0.25),
        nn.MaxPool2d(2),

        # Bloc 4
        nn.Conv2d(128, 128, kernel_size=5, padding=2),
        nn.BatchNorm2d(128),
        nn.ReLU(),
        nn.SpatialDropout2d(0.25) if hasattr(nn, 'SpatialDropout2d')
            else nn.Dropout2d(0.25),
        nn.MaxPool2d(2),

        # INNOVATION: Adaptive pooling pour taille fixe
        nn.AdaptiveAvgPool2d((4, 4)),

        # Classificateur compact progressif
        nn.Flatten(),  # 128 * 4 * 4 = 2048
        nn.Linear(2048, 1024),
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(1024, 512),
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(512, 10)
    ).to(device)
```

1.4.2 Amélioration du nouveau modèle

Le modèle ainsi créé possède 3 425 450 paramètres et une taille estimée de 16,20 Mo. Toutefois, au-delà des convolutions, l'augmentation de la taille des kernels accroît également de manière significative le nombre de paramètres des couches linéaires. En effet, dans ce nouveau modèle, 77% des paramètres appartiennent aux couches linéaires, lesquelles, ne permettant que des transformations linéaires, ne jouent pas un rôle déterminant dans l'identification des motifs pour justifier une telle répartition de poids.

Afin de réduire la taille initiale du modèle tout en conservant une taille de kernel supérieure à 3, plusieurs changements seront mis en place :

Réduction du kernel size de 5 à 4 Rappelons que dans la cadre du projet, nous voulons utiliser notre modèle sur des images de taille 32*32, un kernel size de 4 sera alors suffisant et permettrait d'abord de diminuer la taille du modèle. En effet, par exemple sur Conv(32→32)

- Avec kernel size égale 5 : 32*32*5*5 = 25.600 paramètres
- Avec kernel size égale 4 : 32*32*4*4 = 16.384 paramètres

Soit une réduction effective de 36 % des paramètres pour cette couche. De plus, si on calcule le réceptive field (la région de l'image d'entrée qui influence un pixel de sortie), on constate que diminuer le kernel size à 4 est toujours suffisant. En effet :

- Après Conv1 : RF = 4

— Après Conv2 : RF = 4 + (4-1) = 7
— Après Pool1 : RF = 7 * 2 = 14
— Après Conv3 : RF = 14 + (4-1) = 17
— Après Conv4 : RF = 17 + (4-1) = 20
— Après Pool2 : RF = 20 * 2 = 40

Ainsi le RF du modèle pour kernel size de 4 est suffisant pour des images de taille 32*32.

Bottleneck 1*1 Un bottleneck est l'équivalent d'une transformation linéaire appliquée spatialement. Plus précisément, la convolution 1×1 apprend une transformation linéaire du vecteur de taille égale au nombre de canaux pour chaque pixel sur lequel elle s'applique.

En pratique, il s'agit simplement d'une couche de convolution avec un kernel de taille 1×1. L'avantage du bottleneck réside également dans la réduction du nombre de paramètres. Par exemple, pour réduire le nombre de canaux de 128 à 64, une convolution 1×1 nécessite :

— 1×1 : 128 * 64 * 1 * 1 = 8,192 paramètres
— 5×5 : 128 * 64 * 5 * 5 = 204,800 paramètres

Soit une réduction de 96 % des paramètres dans cet exemple.

AdaptativeAvgPool(1,1) En utilisant un AdaptiveAvgPool de taille (1,1) au lieu de (4,4), on divise par 16 la taille de sortie. Auparavant, une taille de 4×4 revenait à diviser la carte en une grille 8×8 puis à conserver la moyenne de chaque région 2×2. Avec cette modification, pour chaque canal, on conserve la moyenne de l'ensemble de la carte spatiale.

Ce changement introduit une invariance naturelle par translation : quelle que soit la position d'un objet à détecter dans l'image, l'adaptive pooling produira des valeurs similaires. Cela permet de réduire l'overfitting lié à la position des objets sur l'image.

Nous proposons ainsi un nouveau modèle incorporant ces trois changements :

```
model = nn.Sequential(
    # Bloc 1
    nn.Conv2d(3, 32, kernel_size=4, padding=1, bias=True),
    nn.ReLU(),
    nn.BatchNorm2d(32),

    nn.Conv2d(32, 32, kernel_size=4, padding=1, bias=True),
    nn.ReLU(),
    nn.BatchNorm2d(32),
    nn.MaxPool2d(2, 2),
    nn.Dropout2d(0.2),

    # Bloc 2
    nn.Conv2d(32, 64, kernel_size=4, padding=1, bias=True),
    nn.ReLU(),
    nn.BatchNorm2d(64),

    nn.Conv2d(64, 64, kernel_size=4, padding=1, bias=True),
    nn.ReLU(),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(2, 2),
```

```
    nn.Dropout2d(0.3),

    # Bloc 3
    nn.Conv2d(64, 128, kernel_size=4, padding=1, bias=True),
    nn.ReLU(),
    nn.BatchNorm2d(128),

    nn.Conv2d(128, 128, kernel_size=1, padding=0, bias=True),  #
       bottleneck
    nn.ReLU(),
    nn.BatchNorm2d(128),
    nn.AdaptiveAvgPool2d((1,1)),

    nn.Flatten(),

    # Classifier compact
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(128, 10)
    ).to(device)
```

Concrètement, nous observons l'effet directement sur la taille de nos couches linéaires, lesquelles sont non seulement moins nombreuses (3 dans l'ancien modèle contre 2 dans le nouveau), mais contiennent également moins de paramètres. En effet, dans l'ancien modèle, ces couches comptaient 2 626 560 paramètres, contre seulement 17 664 dans la nouvelle architecture, soit une réduction de 99,3%.

Ci-dessus, l'analyse final de notre modèle :

```
    ----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 31, 31]           1,568
              ReLU-2           [-1, 32, 31, 31]               0
       BatchNorm2d-3           [-1, 32, 31, 31]              64
            Conv2d-4           [-1, 32, 30, 30]          16,416
              ReLU-5           [-1, 32, 30, 30]               0
       BatchNorm2d-6           [-1, 32, 30, 30]              64
         MaxPool2d-7           [-1, 32, 15, 15]               0
         Dropout2d-8           [-1, 32, 15, 15]               0
            Conv2d-9           [-1, 64, 14, 14]          32,832
             ReLU-10           [-1, 64, 14, 14]               0
      BatchNorm2d-11           [-1, 64, 14, 14]             128
           Conv2d-12           [-1, 64, 13, 13]          65,600
             ReLU-13           [-1, 64, 13, 13]               0
      BatchNorm2d-14           [-1, 64, 13, 13]             128
        MaxPool2d-15             [-1, 64, 6, 6]               0
        Dropout2d-16             [-1, 64, 6, 6]               0
           Conv2d-17           [-1, 128, 5, 5]         131,200
             ReLU-18           [-1, 128, 5, 5]               0
      BatchNorm2d-19           [-1, 128, 5, 5]             256
           Conv2d-20           [-1, 128, 5, 5]          16,512
             ReLU-21           [-1, 128, 5, 5]               0
      BatchNorm2d-22           [-1, 128, 5, 5]             256
AdaptiveAvgPool2d-23           [-1, 128, 1, 1]               0
```

```
           Flatten-24                  [-1, 128]                0
            Linear-25                  [-1, 128]           16,512
              ReLU-26                  [-1, 128]                0
           Dropout-27                  [-1, 128]                0
            Linear-28                   [-1, 10]            1,290
================================================================
Total params: 282,826
Trainable params: 282,826
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.19
Params size (MB): 1.08
Estimated Total Size (MB): 3.28
----------------------------------------------------------------
```

En particulier, nous constatons que ce nouveau modèle pèse 3,28 Mo contre 16,2 Mo pour l'ancien, lequel contenait 3,4 millions de paramètres, contre 282 000 paramètres pour la nouvelle architecture.

1.5 Entrainement du modèle

Maintenant que nous avons augmenté la taille des kernels tout en conservant une dimension raisonnable, nous entraînons le modèle afin d'examiner ses poids.

L'entraînement est réalisé avec une fonction de perte `CrossEntropyLoss`, l'optimiseur `Adam`, sur 30 epochs. Les données d'entraînement sont chargées avec un *DataLoader* de batch size 32, et les valeurs des pixels des images sont normalisées en les divisant par 255.

Les résultats de l'entraînement sont les suivants :

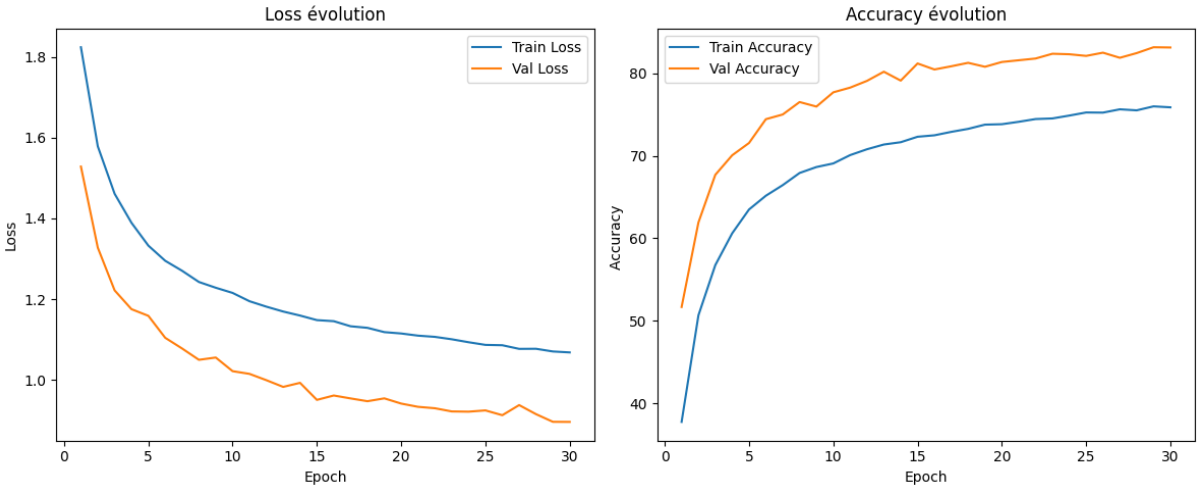


FIGURE 7 – Résultat d'entrainement du modèle sur 30 epochs

A la fin de l'entrainement, cette version augmentée du modèle initial est aussi précise que le modèle VGG16 et déjà inférieure à 5Mo. L'absence d'oscillation confirme le choix du batch size, du taux d'apprentissage, des paramètres dont la mauvaise calibration tend à créer du bruit statistique sur les courbes de loss et d'accuracy.

2 Définition de la méthode pruning

Cette augmentation de la taille des kernels nous permet désormais de travailler dans un sous-ensemble de $\mathbb{R}^{16}$. Cela offre non seulement la possibilité d'observer des géométries plus fines dans la distribution des poids, mais également d'avoir une vue plus globale, ce qui, à terme, facilite l'identification de régularités à différentes échelles dans les filtres.

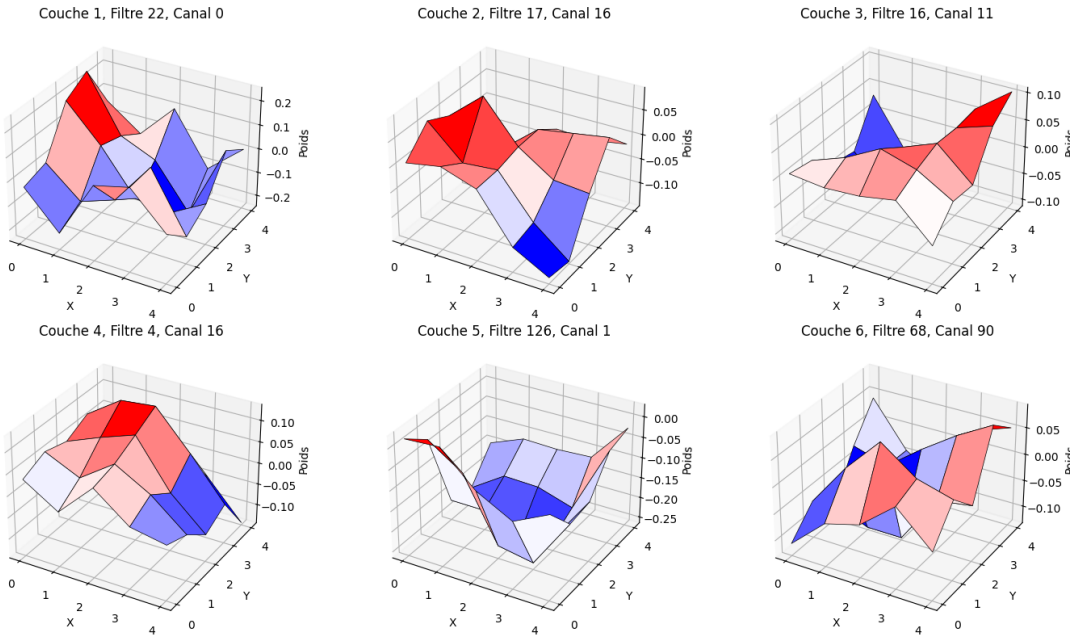


FIGURE 8 – Poids de 6 filtres tirés au hasard dans 6 couches de convolutions différentes

En effet, la suppression d'un filtre doit se baser à la fois sur l'absence de signal sur l'ensemble de ses poids et sur l'absence de signal dans des sous-ensembles de ces poids. Ci-dessous, une figure présente une première ligne de filtres dont les valeurs des poids sont aléatoires, tandis que dans la seconde ligne, une partie des filtres contient des valeurs aléatoires et l'autre suit une distribution gaussienne.

L'objectif est ainsi de développer une méthode permettant de distinguer les filtres ne présentant pas de signal globalement, ceux ne présentant pas de signal localement, et ceux ne présentant aucun signal.

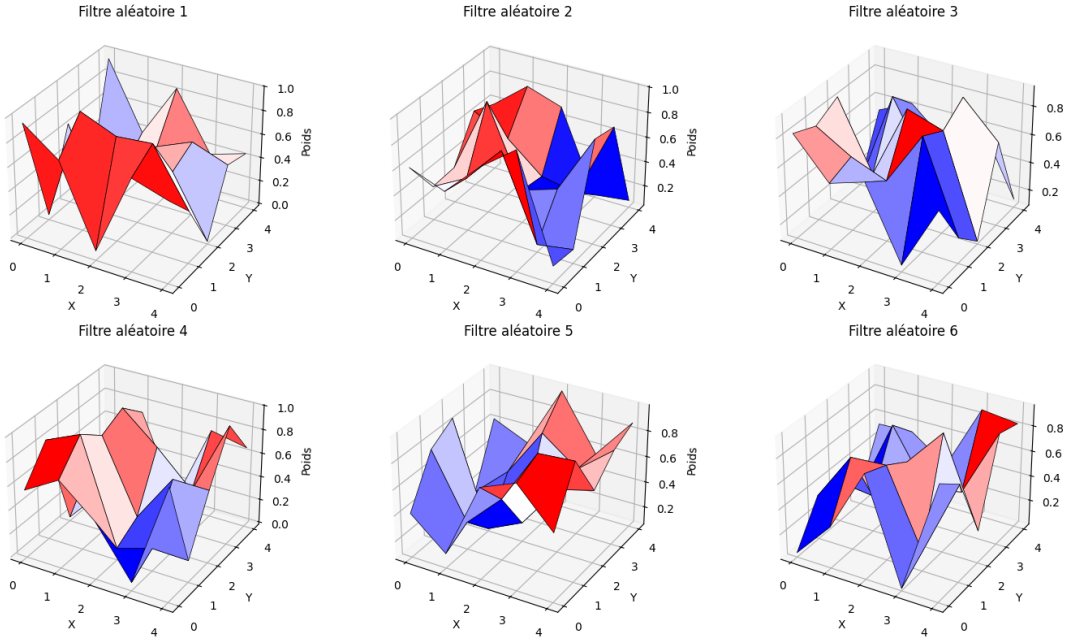


FIGURE 9 – Poids de 6 filtres tirés aléatoires

L'aspect multi-échelle de notre analyse est au cœur de notre approche, car ce sont bien des régularités de différentes échelles (c'est-à-dire tailles) que nos filtres doivent détecter. Un filtre dont les valeurs semblent irrégulières peut ainsi être interprété comme un filtre ayant échoué à détecter un motif à une certaine échelle.

Bien qu'une taille de kernel de 4×4 facilite notre analyse, elle reste insuffisante pour nos outils d'évaluation. Ne pouvant pas augmenter davantage la taille du kernel, nous projetons donc artificiellement notre espace de dimension 16 dans un espace de dimension plus grande et continue : l'espace de Schwartz $S(\mathbb{R}^2)$. Projeter revient à interpoler les poids de notre filtre à l'aide d'une fonction définie dans cet espace, ici l'espace de Schwartz.

2.1 Projection dans un espace continue : l'espace de Schwartz $S(\mathbb{R}^2)$

2.1.1 Définition

Une fonction $f : \mathbb{R}^2 \to \mathbb{R}$ est dite *à décroissance rapide* si elle est infiniment dérivable et si, pour tout couple d'entiers $m, n \geq 0$, il existe une constante $C_{m,n} > 0$ telle que

$$\forall (x, y) \in \mathbb{R}^2. \quad |x^m y^n f(x, y)| \leq C_{m,n},$$

Autrement dit, f et toutes ses dérivées décroissent plus vite que n'importe quelle puissance négative de $|x|$ et $|y|$ lorsque (x, y) tend vers l'infini.

On appelle alors $S(\mathbb{R}^2)$ (espace de Schwartz) l'espace vectoriel des fonctions de $\mathbb{R}^2$ dans $\mathbb{C}$ qui vérifient les deux conditions suivantes :

— f est $\mathbb{C}^\infty$

— f et toutes ses dérivées sont à décroissantes rapide.

2.1.2 Usage

Dans notre cas, nous travaillons en faite pas directement dans l'espace de Schwartz qui est à valeur dans $\mathbb{C}$ mais dans un sous espace de Schwartz à valeur réelle. Travaillé dans $S(\mathbb{R}^2)$ permet plusieurs choses :

1. C'est un espace de fonction très régulière et de convergence rapide, permettant d'éviter d'éventuels effets de bord sur les abords du filtres ou singularité dans la distribution.
2. Pour notre analyse, nous souhaitons utiliser la transformée de Fourier laquelle se comporte bien dans cette espace. En effet, l'espace est stable par transformée de Fourier, donc pas de divergence possible ou création d'artefacts lors de l'application de la transformée de Fourier, le filtre restera lisse.

2.1.3 Construction

Pour simuler un tel espace, nous allons procéder en trois temps. D'abord, il s'agit de définir le filtre comme une fonction continue :

$$F : [0, H-1] \times [0, W-1] \to \mathbb{R}$$

telle qu'elle coïncide avec les valeurs discrètes sur la grille :

$$\forall (n, m) \in \Omega_h \quad F(n, m) = h[n, m]$$

Entre les points de la grille, F est lisse, ie $F \in C^2([0, H-1] \times [0, W-1])$. On utilise ensuite l'interpolateur cubique bilinéaire lequel minimise une énergie de lissage quadratique locale afin de déterminer F en dehors de ces points. F réalise le min suivant :

$$\min_{F \in C^2} \Big(\sum_{n,m} |F(n, m) - h[n, m]|^2 + \lambda \iint \left(\left(\frac{\partial^2 F}{\partial x^2} \right)^2 + 2 \left(\frac{\partial^2 F}{\partial x \partial y} \right)^2 + \left(\frac{\partial^2 F}{\partial y^2} \right)^2 \right) dx\, dy\Big)$$

On redéfinit ensuite une grille fine (y_i, x_j) avec $i, j = 0, \ldots, S-1$ pour un S un "smooth operator" :

$$y_i = \frac{i(H-1)}{S-1}, \quad x_j = \frac{j(W-1)}{S-1}, \quad i, j = 0, \ldots, S-1$$

et on construit la matrice de valeurs interpolées :

$$Z_{\text{smooth}}[i, j] = F(y_i, x_j), \quad i, j = 0, \ldots, S-1$$

Toutefois, nous n'avons pas encore complètement généré notre espace de Schwartz : il faut pour cela multiplier notre fonction par un facteur garantissant à la fois la régularité et la stabilité de l'ensemble. Pour cela, nous utilisons naturellement une fenêtre gaussienne :

On peut multiplier par une fenêtre gaussienne centrée sur le filtre.

$$F_S(y_i, x_j) = Z_{smooth}[i, j] \cdot \exp\left(- \frac{(y_i - H/2)^2 + (x_i - W/2)^2}{2\sigma^2} \right)$$

où σ contrôle la décroissance de la gaussienne, ce qui permet de réguler la continuité et la rapidité de décroissance.

2.1.4 Simulation

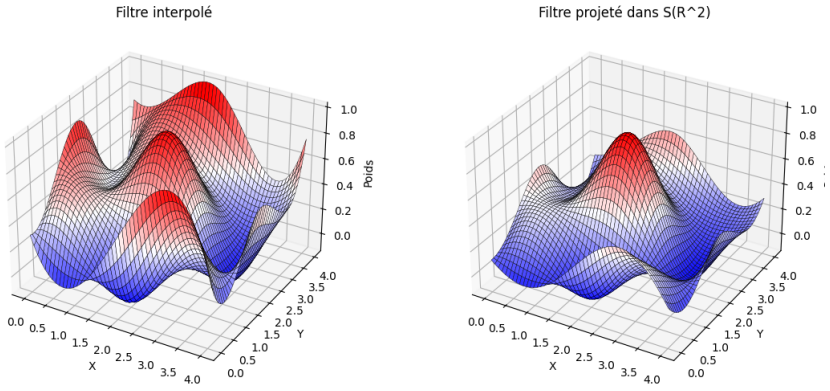


FIGURE 10 – Projection dans l'espace de Schwartz d'un filtre aléatoire

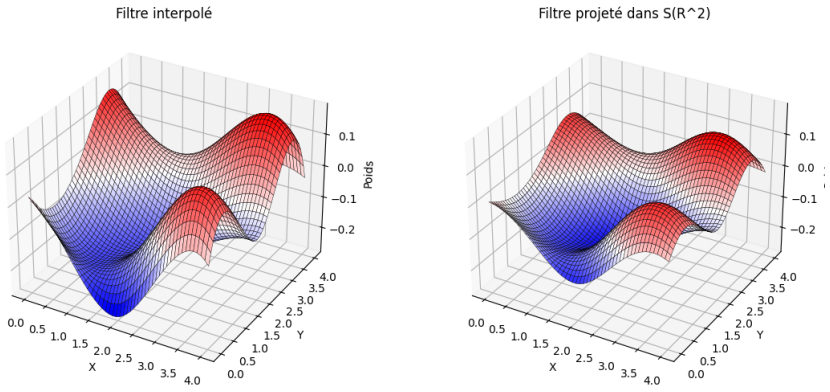


FIGURE 11 – Projection dans l'espace de Schwartz d'un filtre du modèle entrainé

Sur les deux exemples ci-dessus, on constate l'effet de « lissage » induit par la projection, en particulier pour le filtre aléatoire, qui apparaît moins régulier. -t Le code en annexe reprend successivement les deux étapes de constructions précédentes de la projection des filtres ci-dessus.

2.1.5 Zero-padding

Afin d'affiner notre analyse fréquentielle avec la transformée de Fourier, il est nécessaire de zéro-padder le filtre obtenu. Cela permet d'éviter les effets de bord. Aucune nouvelle information fréquentielle n'est créée (le support fréquentiel effectif reste inchangé), mais la grille d'échantillonnage fréquentielle devient plus fine. Soit un filtre h de taille $H \times W$. On définit une version zéro-paddée $\tilde{h}$ sur une grille plus grande $H' \times W'$ (avec $H' \geq H,\ W' \geq W$) :

$$\forall (n,m) \in H' \times W' \quad \tilde{h}[n,m] = \begin{cases} h[n,m], & 0 \leq n < H,\ 0 \leq m < W \\ 0, & \text{sinon.} \end{cases}$$

2.2 Mesure de régularité globale

Maintenant que le cadre d'analyse est posé, nous allons définir notre mesure de régularité. Nous cherchons à analyser la présence de régularité dans nos filtres. En particulier la présence de régularités dans tous sous ensemble d'un même filtre. La transformée de Fourier permet de passer dans domaine spatial à un domaine fréquentiel permettant de considérer une fonction donnée en une somme d'ondes pures, fournissant ainsi une analyse spectrale de la régularité. Cette analyse spectrale va nous permettre de distinguer signaux forts de signaux faibles autrement dit des régularités de ce qui n'en est pas. C'est donc naturellement qu'elle constitue l'outil principale de notre étude. En particulier, elle présente plusieurs propriété utile à notre démarche que nous allons détailler.

2.2.1 Transformée de Fourier

Soit $F : \mathbb{R}^2 \to \mathbb{R}$ une fonction de Schwartz, $F \in \mathcal{S}(\mathbb{R}^2)$. Sa transformée de Fourier (unités 2π comme ci-dessous) est définie par

$$\widehat{F}(\xi) = \iint_{\mathbb{R}^2} F(x)\, e^{-2\pi i \langle x,\xi\rangle}\, dx, \quad \xi = (\xi_1, \xi_2).$$

2.2.2 Théorème de Plancherel

Si $F \in L^2(\mathbb{R}^2)$, alors sa transformée de Fourier $\widehat{f}$ est aussi dans $L^2(\mathbb{R}^d)$, et on a l'égalité d'énergie :

$$\|f\|^2_{L^2(\mathbb{R}^2)} = \int_{\mathbb{R}^2} |f(x)|^2\, dx = \int_{\mathbb{R}^2} |\widehat{f}(\xi)|^2\, d\xi = \|\widehat{f}\|^2_{L^2(\mathbb{R}^2)}.$$

Le théorème de Plancherel est crucial dans notre cas, car il garantit que travailler dans le domaine spatial ou dans le domaine fréquentiel est strictement équivalent en norme L^2. Autrement dit, passer au domaine fréquentiel via la transformée de Fourier conserve la magnitude des signaux. Cela est important ici, car, comme mentionné précédemment, les approches basées sur la régularité et sur la magnitude doivent être complémentaires. L'application de la transformée de Fourier préserve donc l'intégrité des poids selon la norme L^2. Notons aussi que $S(\mathbb{R}^2) \subset L^2(\mathbb{R}^2)$ le résultat s'appliquer donc à notre fonction.

2.2.3 Espace de Sobolev

Si la transformée de Fourier est utile pour notre analyse, elle ne fait pas tout ! En particulier, ce n'est pas une norme. Cependant, on peut en définir une par extension sur un espace particulier : l'espace de Sobolev.

Définition Soit $\Omega_h \subset \mathbb{R}^2$ le domaine du filtre.

Pour un entier $k \in \mathbb{N}$, l'espace de Sobolev $H^k(\Omega_h)$ est défini comme l'ensemble des fonctions $f \in L^2(\Omega_h)$ dont toutes les dérivées jusqu'à l'ordre k appartiennent aussi à $L^2(\Omega)$:

$$H^k(\Omega_h) = \big\{ f \in L^2(\Omega_h) : \partial^\alpha f \in L^2(\Omega_h), \ \ |\alpha| \le k \big\},$$

où $\alpha = (\alpha_1, \ldots, \alpha_d)$ est un multi-indice.

La norme associée est alors :

$$\|f\|^2_{H^k(\Omega_h)} = \sum_{|\alpha|\leq k} \|\partial^\alpha f\|^2_{L^2(\Omega_h)}.$$

Pour $s \in \mathbb{R}$, pas nécessairement entier, on définit ainsi $H^s(\mathbb{R}^2)$ par transformée de Fourier :

$$H^s(\mathbb{R}^2) = \left\{ f \in L^2(\Omega_h) : \int_{\mathbb{R}^2} (|\xi|^2)^k |\hat{f}(\xi)|^2 \, d\xi < \infty \right\}.$$

La norme correspondante étant alors :

$$\|f\|^2_{H^s} = \int_{\mathbb{R}^d} (|\xi|^2)^s |\hat{f}(\xi)|^2 \, d\xi.$$

Ce résultat est une application directe du théorème de Plancherel et que pour tout n $\in \mathbb{N}$

$$\widehat{f^{(n)}}(\xi) = (2i\pi\xi)^n \, \hat{f}(\xi)$$

Usage Le terme $(|\xi|^2)^s$ pondère le spectre en donnant plus d'importance aux hautes fréquences lorsque s est grand.

— Si $\hat{f}(\xi)$ décroît rapidement, l'intégrale reste finie même pour s grand, ce qui correspond à une fonction très régulière.

— Si $\hat{f}(\xi)$ décroît lentement, seuls les petits s sont permis, ce qui correspond à une fonction rugueuse.

En fait, plus s est grand, plus f est régulière ; un s élevé impliquant un controle fort des hautes fréquences. Aussi, en ce qui concerne le choix de la valeur s, on sollicite le théorème d'injection de Sobolev lequel implique :

$$f \in H^s(\mathbb{R}^d), \quad s > \frac{d}{2} \implies f \text{ est continue.}$$

On prendra donc le s le plus petit possible (pour limiter la perte d'information dans les hautes fréquénces) vérifiant cette condition autrement dit s = 2.

Application Ainsi on définit la mesure de localité :

$$\|f\|^2_{H^2} = \int_{\mathbb{R}^d} (s^2 + l^2)^2 |\hat{f}(\xi)|^2 \, d\xi.$$

On approxime cette norme sur notre filtre discret par :

$$\|\widetilde{h}\|^2_{H^2} \approx \sum_{k,\ell} \left(k^2 + \ell^2\right)^2 \left|\widetilde{H}[k,\ell]\right|^2$$

Avec l'approximation discrète de la transformée de Fourier sur notre filtre zero-padded :

$$\widetilde{H}[k,\ell] = \sum_{n=0}^{N'-1} \sum_{m=0}^{M'-1} \widetilde{h}[n,m] \, e^{-2\pi i \left(\frac{kn}{H'} + \frac{\ell m}{W'}\right)}.$$

Cette norme va pouvoir nous donner une estimation de la "régularité" ou rugosité de la fonction de $\mathbb{R}^2$ que nous avons définit. Plus la valeur de la norme est éleve, plus le filtre est irrégulier. Regardons sur l'exemple suivant :

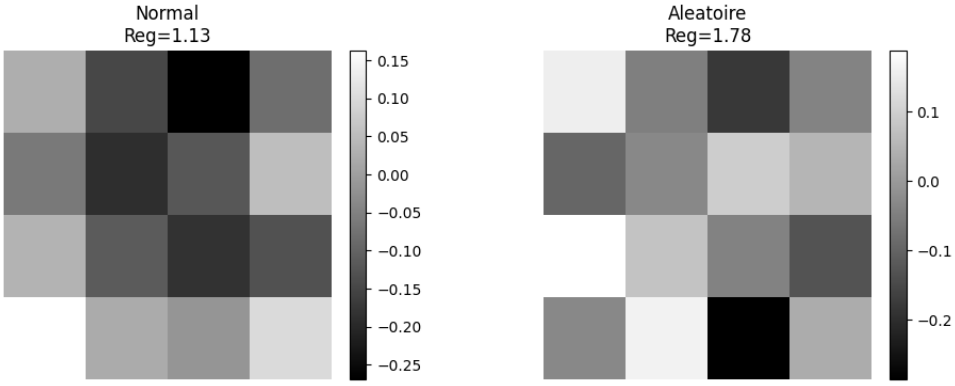


FIGURE 12 – Application de la norme de Sobolev à un filtre extrait du modèle et un filtre aléatoire

Sur l'exemple ci-dessus, nous observons que la mesure de Sobolev est plus faible pour le filtre issu de notre modèle.

Simulation À présent, observons l'effet de la projection. Rappelons que celle-ci dépend d'un critère σ. Nous allons donc tirer 100 filtres aléatoires et comparer la valeur de leur régularité avec celle d'un filtre extrait de notre modèle, de même amplitude, pour deux valeurs de σ : 50 et σ : 1.

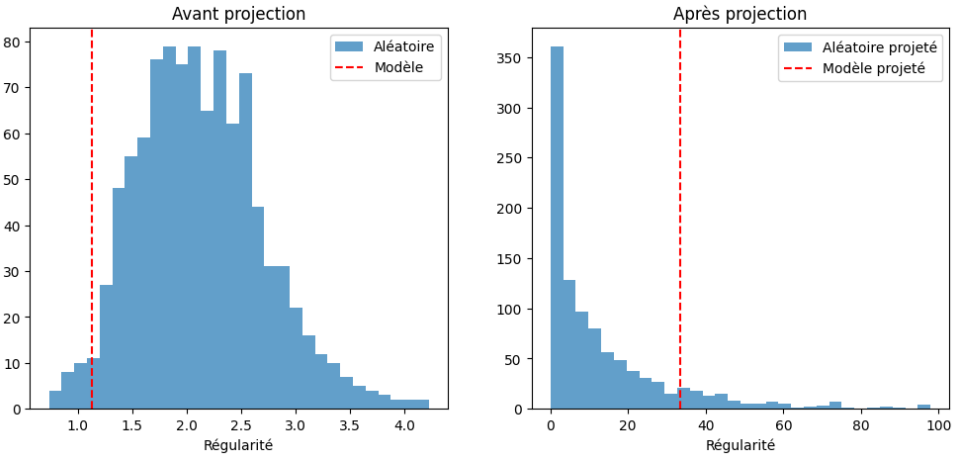


FIGURE 13 – Test de régularité pour une projection de valeur sigma égale à 1

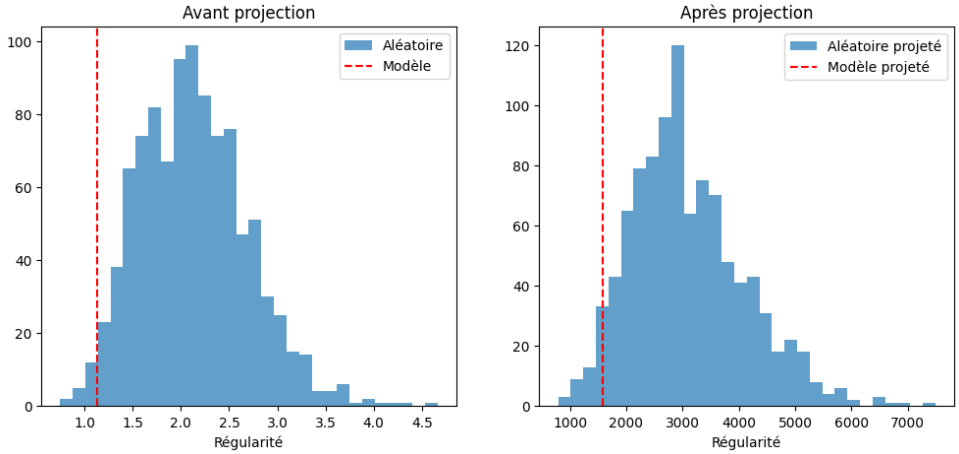


FIGURE 14 – Test de régularité pour une projection de valeur sigma égale 50

Plus la valeur de σ est petite, plus la distribution des poids n'est plus uniforme et se concentrate autour de 0. Cette concentration brouille la mesure, qui ne permet plus de distinguer la régularité des filtres. Au contraire, pour $\sigma = 50$, la distribution uniforme des poids semble conservée. La projection ne semble pas améliorer la distinction entre les poids réguliers — du moins pour l'instant.

Limite Cette mesure ne permet que de quantifier la régularité globale du filtre, et non sa régularité locale. Prenons l'exemple suivant :

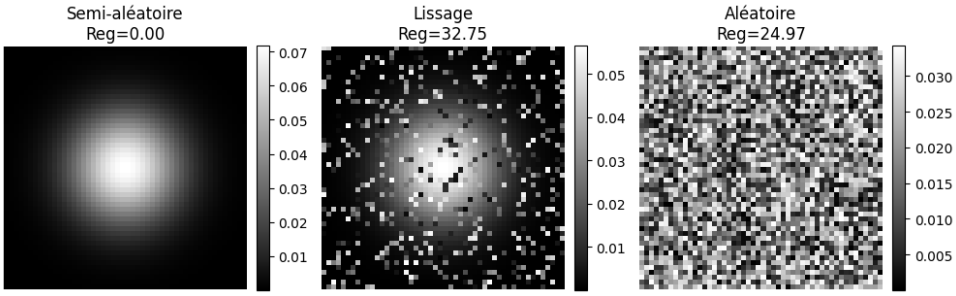


FIGURE 15 – Application de la norme de Sobelev

Ci-dessus, trois filtres : un premier suivant une gaussienne, un second une gaussienne avec 20% de valeurs aléatoires, et un troisième totalement aléatoire. Nous pouvons constater que, selon la norme établie, le second filtre — bien que suivant visiblement une gaussienne — serait moins régulier qu'un filtre totalement aléatoire ! En effet, la norme telle que définie ne mesure que la régularité globale, en moyennant les variations sur tout l'espace. Il nous faut donc définir une mesure locale permettant de refléter la présence de régularités au sein de sous-ensembles du filtre.

2.3 Mesure de régularité locale

2.3.1 Transformer de fourier fenêtré

Afin de parvenir à capturer l'information locale de régularité, nous utilisons une transformée de fourier à fenêtre glissante :

La transformée de Fourier à fenêtre (STFT) est définie de la manière suivante :

$$V_g F(x,\xi) = \int_{\mathbb{R}^2} f(\lambda)\,\overline{g(\lambda - x)}\,e^{-2\pi i\,\xi\cdot\lambda}\,d\lambda,$$

Avec une fenêtre $g \in S(\mathbb{R}^2)$. Cela nous permet de définir ensuite la norme de Sobolev fenetré :

$$\|h\|^2_{H^2_{\text{loc}}} = \int_{\mathbb{R}^2}\int_{\mathbb{R}^2} (|\xi|^2)^s\,|V_g h(x,\xi)|^2\,d\xi\,dx$$

Concrètement, cela nous donne une mesure de la régularité autour de l'abscisse x pour la fréquence λ Le résultat est une mesure multi-résolution : on peut voir la régularité zone par zone. Toutefois, demeure la question du choix de la fenêtre.

2.3.2 Choix de la fenêtre

Nous recherchons une fonction régulière, centrée sur l'origine. Une fenêtre rectangulaire n'est pas un bon choix en raison de la discontinuité aux bords et de l'effet de Gibbs qu'elle entraîne (déformation du signal à l'approche d'une discontinuité).

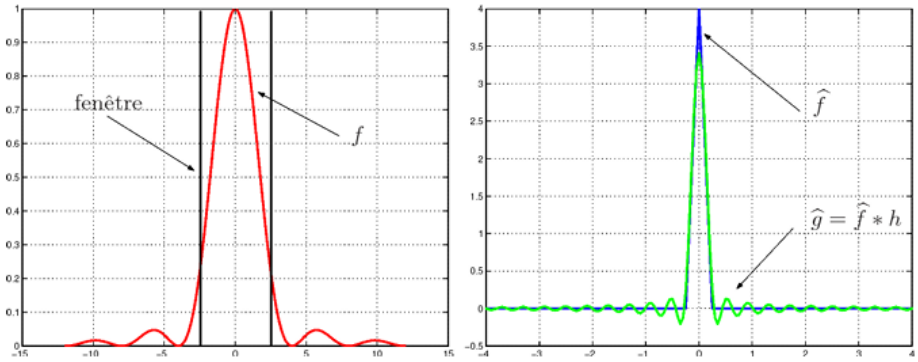


FIGURE 16 – Illustration de l'effet de Gibbs sur la fonction : $f : x \longrightarrow \text{sinc}^2(\frac{1}{4}x\pi)$, au bord du support de la fonction f apparition d'oscillations

Ci-dessus, aux abords du support, la transformée de Fourier de la fonction f est multipliée par une seconde fonction : le sinus cardinal. Cet effet est commun à toute tentative de fenêtrage par une fenêtre rectangulaire et découle de la propriété suivante : pour a,b quelconques,

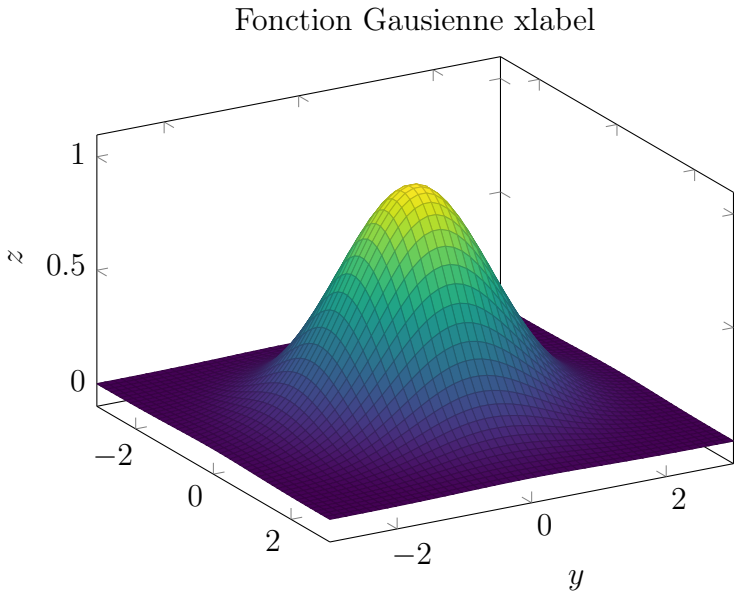
$$\mathcal{F}\Big(f\,\hat{\mathbf{1}}_{[a,b]}\Big) = \hat{f} * \Big(\hat{\mathbf{1}}_{[a,b]}\Big)$$

Or $\hat{\mathbf{1}}_{[a,b]}$ est la fonction sinus cardinal, d'où le phénomène. Concernant le choix de notre fonction de fenêtrage, plusieurs critères orientent la sélection de candidats possibles :

— Symétrie spectrale : la pondération spectrale ne favorise pas un sens de rotation dans le spectre. En effet, nous travaillons sur le cercle trigonométrique complexe donc le sens importe.

— Conservation de l'énergie : en utilisant une fenêtre l'énergie du spectre ie la norme L^2 totale doit être inchangé.

Ces deux hypothèses nous amènent à chercher, d'une part, une fonction paire et, d'autre part, une fonction de norme $L^2 = 1$. Ainsi, nous choisissons la fonction suivante :



On définit ainsi notre fenêtre gausienne centrée avec :

$$g[n,m] = \exp\Big(-2\tau^2\big((n-n_0)^2+(m-m_0)^2\big)\Big), \quad n,m \in \mathbb{Z}, \quad \sum_{n,m} g[n,m]^2 = 1$$

À présent, appliquons cette mesure locale sur deux exemples : un filtre extrait de notre modèle entraîné et un filtre aléatoire.

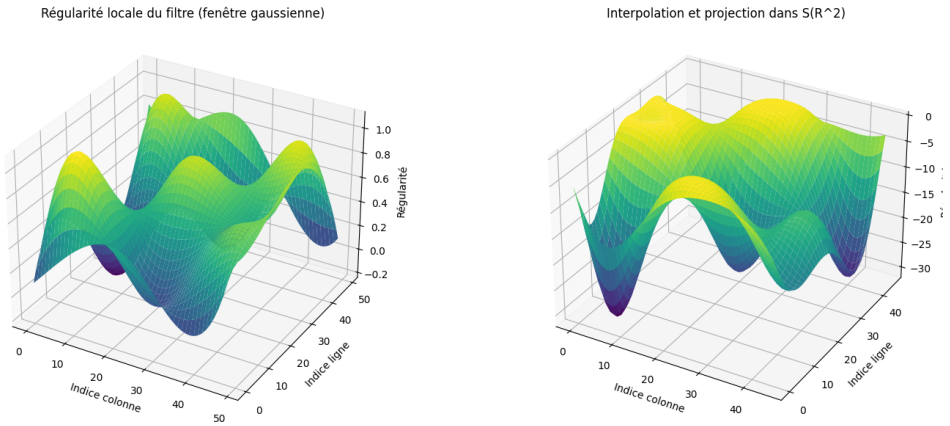


FIGURE 17 – Application de la norme de Sobolev fenêtrée sur un filtre aléatoire.

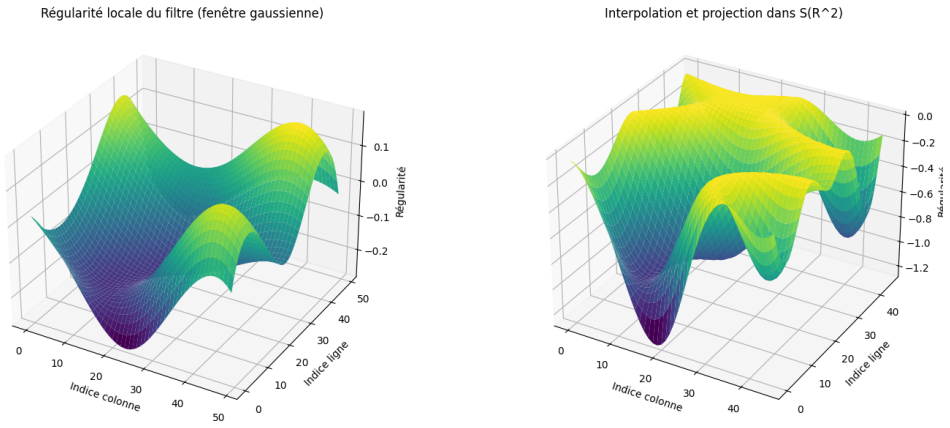


FIGURE 18 – Application de la norme de Sobolev fenêtrée sur un filtre extrait du modèle entraîné

Nous pouvons constater, sur les deux exemples ci-dessus, que notre mesure permet de distinguer — ici visuellement — un filtre aléatoire d'un filtre régulier. En effet, la map de régularité du filtre extrait du modèle est globalement plus proche de zéro que celle du filtre aléatoire. Observons bien la différence d'échelle entre nos deux représentations de régularité ! Afin d'unifier les représentations de régularité des filtres, nous les comparons par la somme des valeurs absolues des valeurs de régularité. Nous pourrions définir des méthodes moins naïves de comparaison, en utilisant par exemple la médiane, la variance, etc., mais cela sort du cadre de ce rapport. Pour le filtre régulier, cette somme vaut 714, et pour le filtre non régulier, elle vaut 23 374. Si l'on reprend l'exemple précédent illustrant la nécessité d'une mesure locale, la régularité de la gaussienne « brouillée » est de 8 264, bien inférieure à 23 374 obtenue pour le filtre totalement aléatoire, tandis que la mesure globale donnait une valeur supérieure à celle de la matrice aléatoire, ce qui aurait indiqué à tort qu'elle était moins régulière.

2.3.3 Choix des paramètres de la mesure de régularité : τ

La définition de notre norme nous permet également de revenir sur le choix de τ utilisé pour la fenêtre gaussienne. Le reste des résultats présentés a été obtenu pour $\sigma = 50$. Nous cherchons une valeur de τ permettant de maximiser le gap de régularité (selon notre norme) entre un filtre extrait de notre modèle, supposé régulier, et un filtre irrégulier. Nous allons procéder de manière expérimentale. Pour ce faire, nous allons extraire un nombre constant de filtres du modèle entraîné ainsi qu'un nombre constant de filtres aléatoires, et pour chaque filtre, calculer la valeur de la norme pour différentes valeurs de τ.

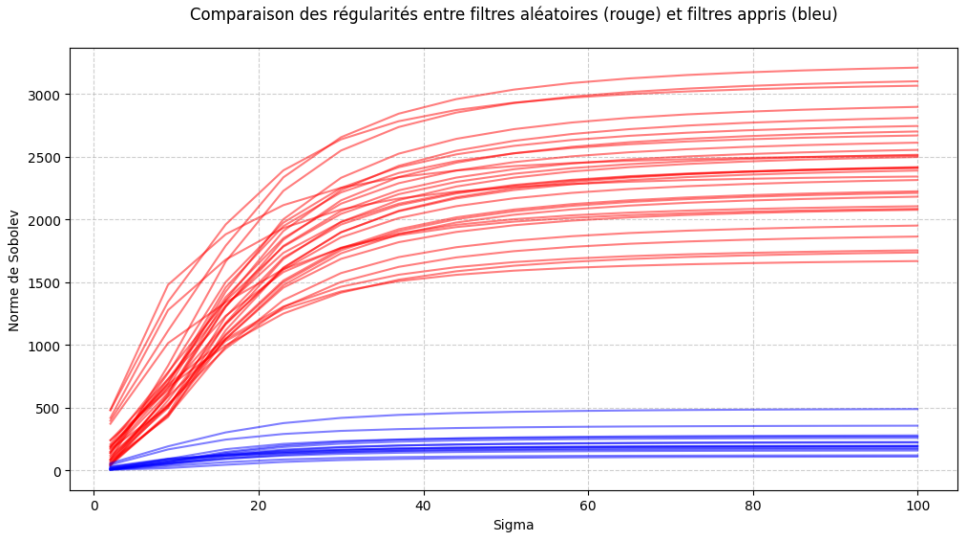


FIGURE 19 – Application de la somme de la norme de Sobolev fenêtrée sur 10 filtres du modèles (en bleu) et 10 filtres instanciés aléatoirement (en rouge) pour différentes valeurs de τ appelé sigma ici

Tout d'abord, constatons sur ce premier graphique que les filtres utilisés dans le modèle semblent, au sens de cette norme, être significativement plus réguliers que les filtres aléatoires, du moins à partir de $\tau = 10$. En effet, cette différence diminue à mesure que la valeur de τ est petite. Cet effet est similaire à celui observé lors du choix de σ pour la projection dans l'espace de Schwartz. Enfin, nous constatons que la valeur de régularité plafonne à partir de $\tau = 60$. Plus précisément, la valeur est maximale à plus ou moins 5% à partir de $\tau = 58$.

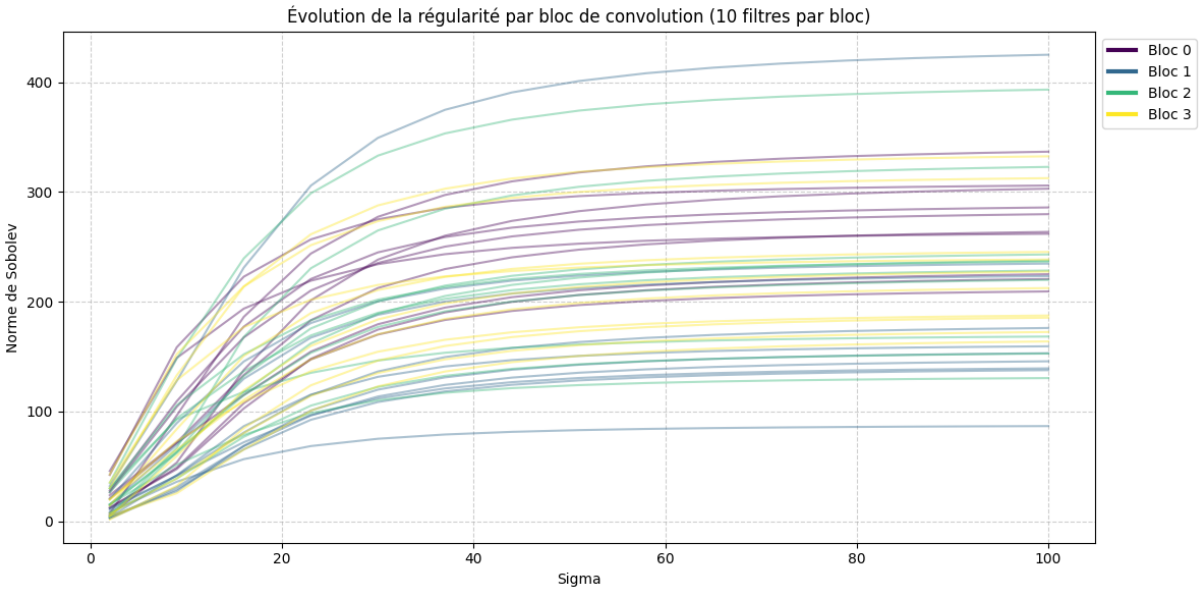


FIGURE 20 – Application de la somme de la norme de Sobolev fenêtrée sur 10 filtres choisies aleatoirement dans chaque bloc de convolution pour différentes valeurs de τ appelé sigma ici

Plus particulièrement, nous retrouvons cette évolution caractéristique au sein des filtres du modèle entraîné. On constate également que les valeurs de régularité sont très étendues au sein du modèle, allant de 80 à plus de 400 selon les différentes couches. Cette observation souligne le cadre plus souple d'analyse apporté par l'approche fenêtrée.

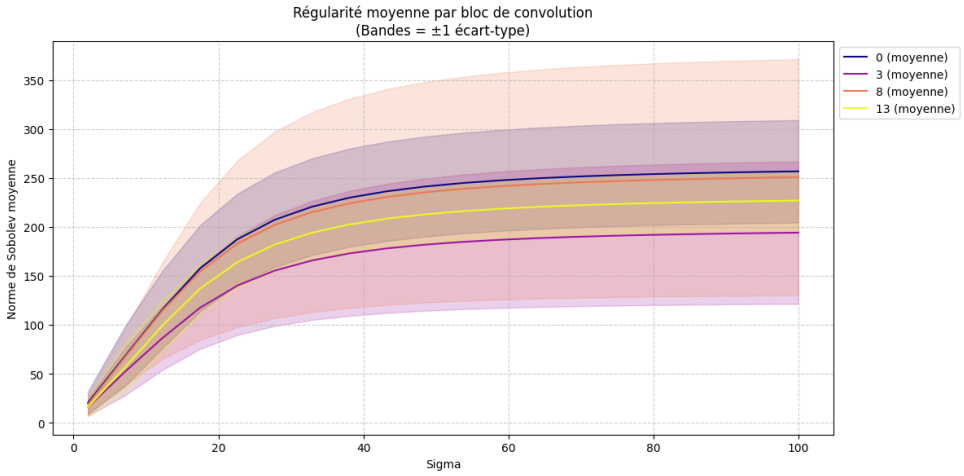


FIGURE 21 – Moyenne de la somme de la norme de Sobolev par bloc

Plus globalement, en moyennant la valeur des filtres par bloc de convolution, nous observons que le niveau de profondeur du bloc de convolution ne semble pas, à première vue en tout cas, être déterminant pour la régularité des filtres. De même, la valeur de l'écart type par filtre ne nous permet pas de conclure quant à l'existence d'une corrélation à ce sujet. Toutefois, au vu de la similarité des courbes, la valeur $\tau = 58$ semble être un bon compromis pour les raisons évoquées précédemment.

3 Test et comparaison

L'objectif est à présent de tester l'efficacité de notre norme de régularité, au sens de sa capacité à distinguer les filtres utiles au modèle de ceux qui ne le sont pas.

Un filtre est considéré comme plus ou moins utile en fonction de l'impact que sa suppression a sur la valeur de l'accuracy du modèle sur le dataset d'évaluation.

3.1 Protocole de test

Notre protocole de test est le suivant :

— Évaluer la régularité de l'ensemble des filtres de notre modèle.

— Conserver un nombre constant de filtres en fonction de leurs régularités.

— Modifier l'architecture du modèle en supprimant les filtres sélectionnés.

— Évaluer notre modèle pruné sur le dataset d'évaluation.

Afin de comparer les résultats de ce protocole, nous allons également implémenter des méthodes de pruning structuré standards (impliquant une reconstruction du modèle) selon la norme L1, L2, ou la variance des filtres.

Enfin, nous comparerons ces résultats à une méthode témoin : le pruning par suppression aléatoire de filtres, ce qui nous permettra de valider l'hypothèse nulle selon laquelle notre méthode de pruning serait plus efficace que le pruning aléatoire.

3.2 Paramètres de la simulation

Notre protocole repose sur la définition de plusieurs paramètres :

— La méthode de pruning.

— Dans le cas de la méthode de pruning par régularité, le percentile des filtres les plus réguliers que l'on souhaite utiliser pour la mesure. Nous ferons varier cette proportion de 20% à 100% par pas de 20% (notée P).

— Pour le modèle entraîné, le percentile de filtres les plus réguliers à conserver. Nous ferons varier cette proportion de 10% à 90% par pas de 10%, correspondant à un niveau de compression de 1,2 à 12,8.

3.3 Résultats

Nous allons observer les résultats du protocole en deux temps : d'abord le pruning avant le fine-tuning, puis le pruning après le fine-tuning. En effet, bien que le fine-tuning permette de récupérer une partie de l'accuracy perdue par le pruning, il demeure qu'il lisse également l'impact du pruning. Rappelons d'ailleurs que l'accuracy du modèle original est de : 84,05%.

3.3.1 Accuracy avant fine-tuning

Observer les résultats avant le fine-tuning, nous permet de voir l'impact brut de notre pruning. Selon, la méthode décrite ci-dessus, nous obtenons les résultats suivants :

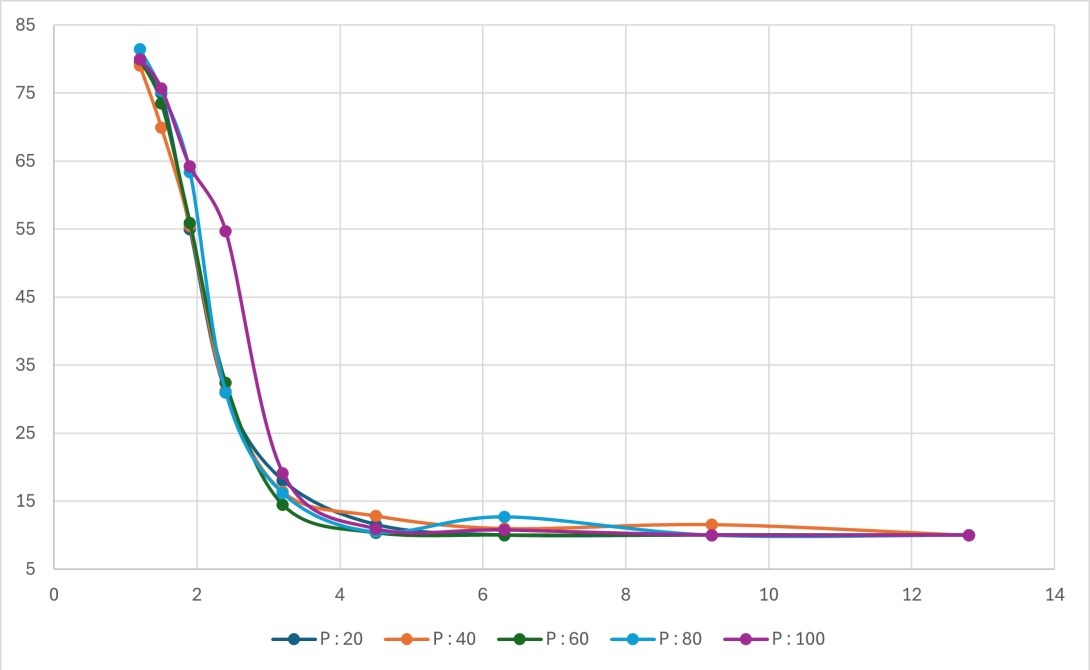


FIGURE 22 – Accuracy du modèle entrainé pruné avant fine-tuning pour différentes valeurs de P et de compression

Tout d'abord, nous constatons que, pour toutes les valeurs de P, la courbe possède une variation et une pente similaires. En particulier, un calcul d'écart quadratique à la moyenne montre une variation d'accuracy moyenne par niveau de compression de 3,89. L'accuracy est maximale à chaque niveau de compression pour des valeurs de P différentes. Aussi, l'accuracy chute à partir d'une compression égale à 2. Plus précisément, la pente est maximale pour le niveau de compression 2,4, à l'exception de $P = 100$. À partir du niveau de compression 4, l'accuracy pour toute valeur de P est proche de 10%, le modèle étant alors aléatoire. La valeur maximale d'accuracy, obtenue pour la plus faible compression (1,2), avoisine les 80% (81.42%) pour $P = 80$. Comparons à présent avec les méthodes benchmarks de pruning :

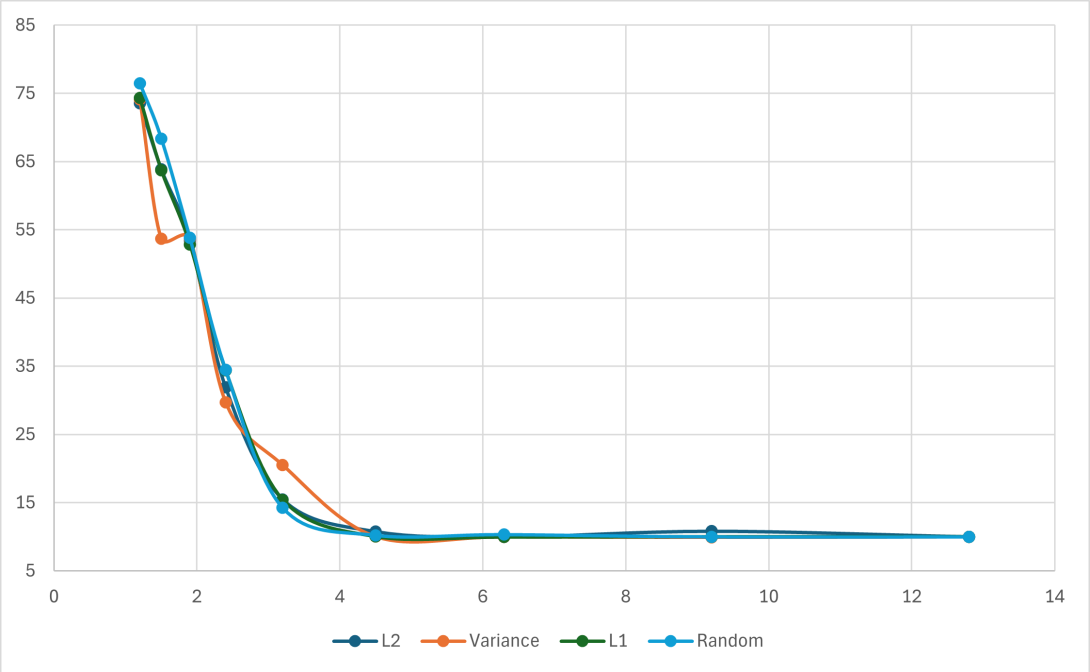


FIGURE 23 – Accuracy du modèle entrainé pruné par méthodes standards avant fine-tuning pour différentes valeurs de compression

Nous constatons que les méthodes standards de pruning suivent une tendance similaire à notre méthode de pruning. En effet, en moyennant les résultats d'accuracy pour les différentes valeurs de P et pour les différentes méthodes de pruning standards, nous calculons une corrélation de Pearson de 0,9950. Au-delà de cette corrélation, il est étonnant de constater que des mesures fondamentalement différentes des poids des filtres fournissent une évaluation similaire de l'accuracy du modèle. En effet, un calcul de l'écart quadratique moyen pour les 4 méthodes nous donne un écart moyen de 2,09 points d'accuracy par mesure. Filtrer de manière aléatoire est en ce sens aussi performant, voire localement plus performant, que de mesurer par les normes L1, L2 ou par variance. Ce constat nous permet d'affirmer, dans le cas de notre modèle, que le pruning L1, L2 et par variance ne sont pas des mesures fiables de l'importance des filtres du modèle. Globalement, les résultats avant fine-tuning des méthodes standards atteignent le plateau des 10% d'accuracy plus rapidement que pour notre méthode de pruning et présentent une accuracy maximale plus faible à 76,5% contre 81,42% pour notre méthode pour $P = 80$.

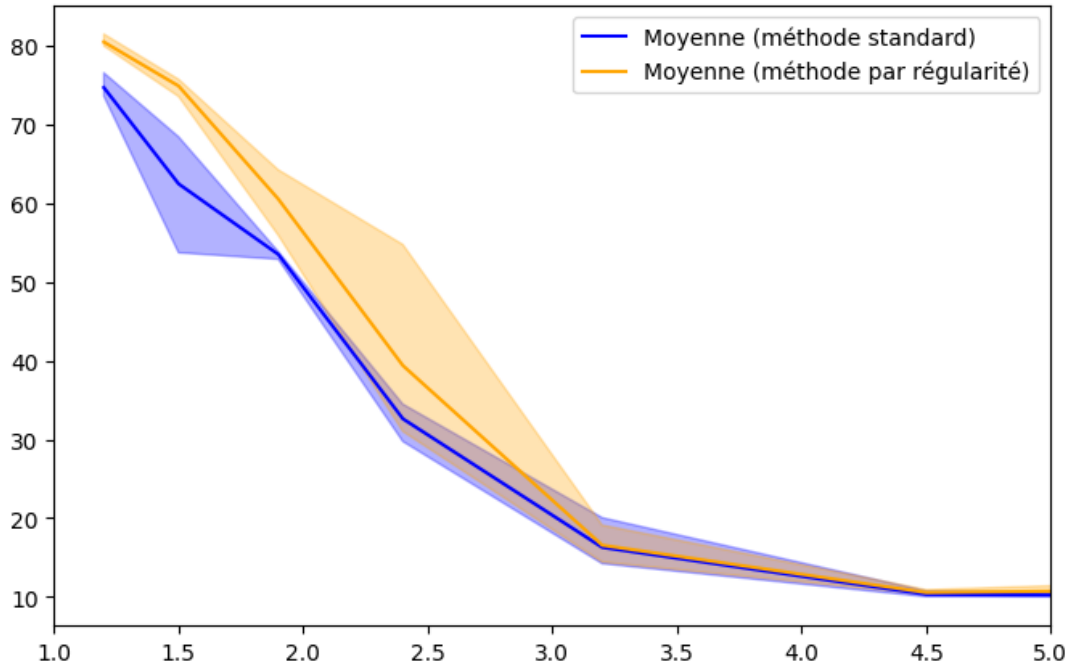


FIGURE 24 – Comparaison des modèles de pruning avant fine-tuning pour niveau de compression inférieur à 4

Ci-dessus, le graphique représente l'accuracy moyenne des méthodes standard et des méthodes par régularité pour les différentes valeurs de P. Le canal représente les valeurs minimale et maximale pour ces deux méthodes. Ainsi, nous constatons qu'avant fine-tuning, notre méthode d'analyse des filtres par régularité permet d'obtenir en moyenne une meilleure accuracy jusqu'au niveau de compression 3, puis une accuracy similaire pour les niveaux de compression supérieurs.

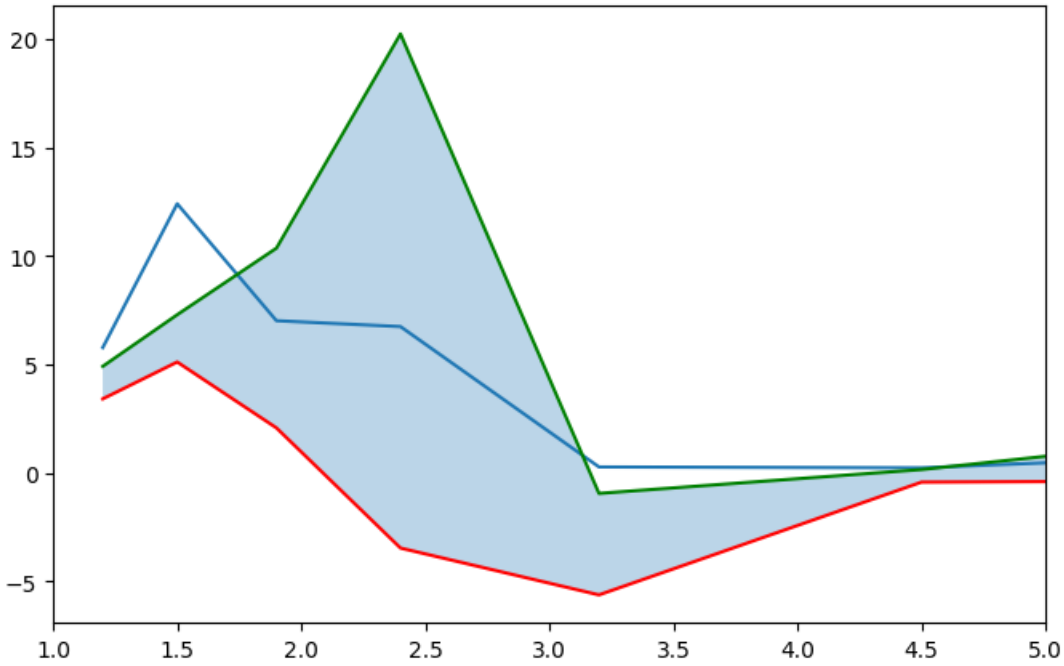


FIGURE 25 – Difference d'accuracy entre les moyennes des méthodes standards et de régularités

Le graphique ci-dessus représente l'écart d'accuracy entre notre modèle et les modèles standards pour les différentes valeurs de compression. La courbe bleue représente l'écart moyen entre l'ensemble des modèles de régularité et des modèles standards. Sur l'ensemble de la plage de compression, notre modèle surpasse en moyenne de 3,7 points d'accuracy les modèles standards, et la version la plus précise de notre modèle surpasse de 4,84 cette moyenne. La courbe verte représente la différence d'accuracy entre le modèle de régularité le plus précis et le modèle standard le plus précis. La courbe rouge représente la différence d'accuracy entre le modèle de régularité le moins précis et le modèle standard le plus précis. Ainsi, nous constatons que toutes les versions des modèles de régularité ne surpassent pas en accuracy les modèles standards. Le modèle de régularité le moins précis correspond à la valeur de $P = 20$, lorsque seulement 20% des valeurs les plus régulières sont utilisées pour le classement des filtres.

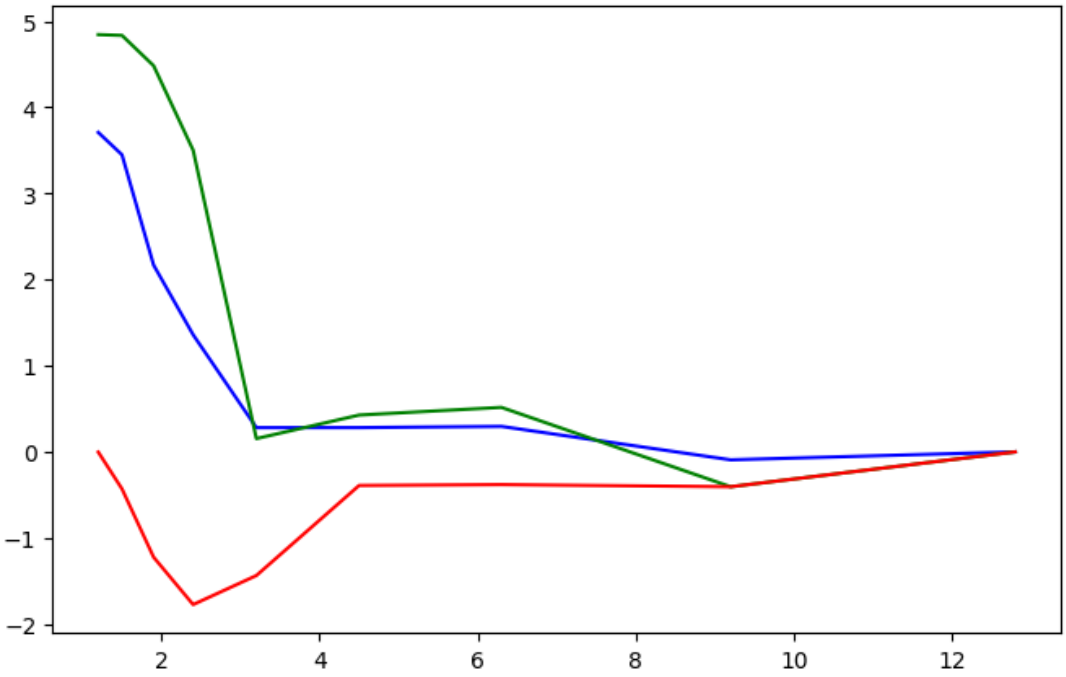


FIGURE 26 – Moyenne glissante de la différence d'accuracy entre les modèles

Le graphique ci-dessus représente la moyenne glissante de différence d'accuracy sur les 12 niveaux de compression. Ainsi, nous constatons que la différence d'accuracy est surtout notable sur les 4 premiers niveaux de compression.

3.3.2 Accuracy après fine-tuning

Une fois le modèle pruné, nous le fine-tunons sur 5 epochs. Cela permet d'augmenter l'accuracy après la suppression des filtres. Le choix du nombre d'epochs est crucial. En effet, nous avons pu observer lors d'un essai précédent non présenté ici que le fine-tuning sur un nombre d'epochs supérieur ou égal à la moitié du nombre d'epochs utilisé pour l'entraînement lisse considérablement les valeurs d'accuracy pour les différentes approches, rendant imperceptible le bénéfice de notre méthode ou de toute autre méthode. Nous sommes arrivés à la conclusion que, parce qu'il s'agit d'étudier le pruning de notre modèle, ce nombre d'epochs devait rester très inférieur au nombre d'epochs d'entraînement. Ici, nous réalisons ainsi le fine-tuning sur 5 epochs. Cette fois-ci, nous ne mesurons plus l'accuracy du modèle mais la rétention d'accuracy, c'est-à-dire le pourcentage d'accuracy conservé du modèle entraîné. Les résultats sont les suivants :

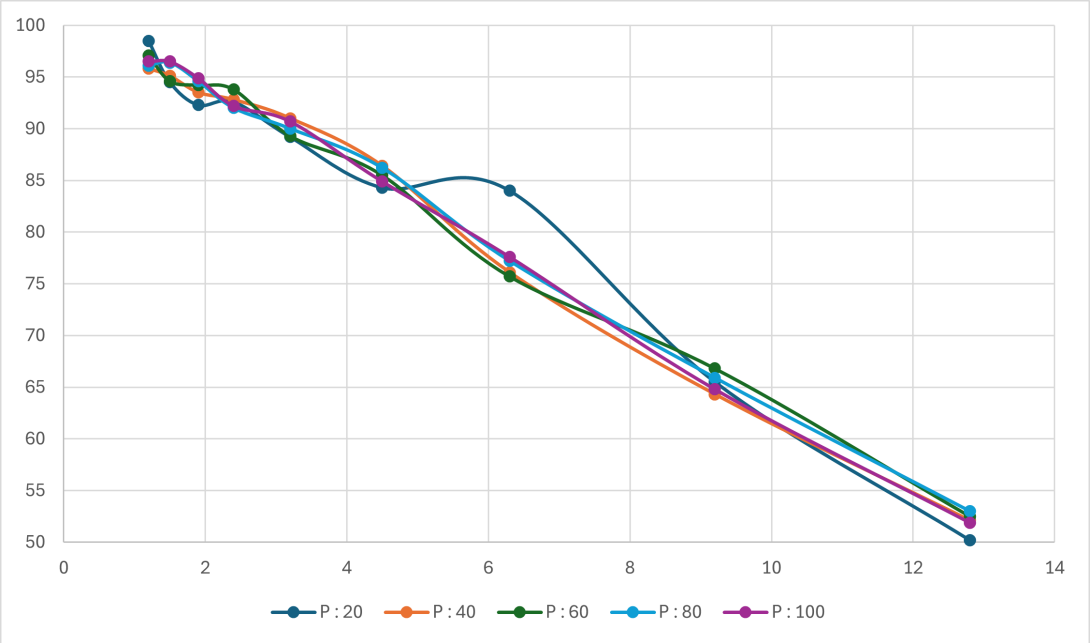


FIGURE 27 – Retention d'accuracy (en %) du modèle entrainé pruné après fine-tuning pour différentes valeurs de compression

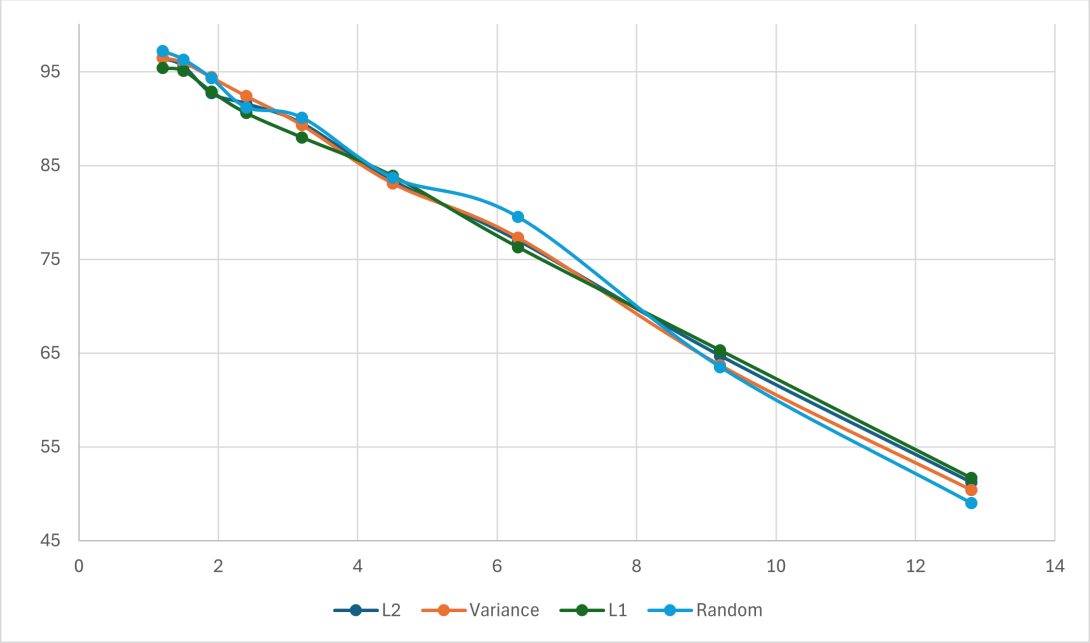


FIGURE 28 – Retention d'accuracy du modèle entrainé (en %) pruné après fine-tuning pour différentes valeurs de compression des méthodes standards

Nous constatons d'abord que les courbes sont linéaires. Ainsi, les courbes d'accuracy post-pruning ont le même profil que la courbe d'accuracy initiale, mais avec une pente plus élevée (la courbe linéaire étant décroissante). Une régression linéaire sur la moyenne des résultats pour les méthodes standard et par régularité (pour les différentes valeurs de P) nous donne la pente de ces courbes, soit $-3,87\%$ pour le pruning par régularité

et $-3,98\%$ pour les méthodes standards. Ainsi, l'accuracy finale par méthode standard aurait une pente plus élevée et un minimum plus faible que l'accuracy finale par la méthode de régularité. Une comparaison en moyenne des deux approches permet de préciser leurs différences :

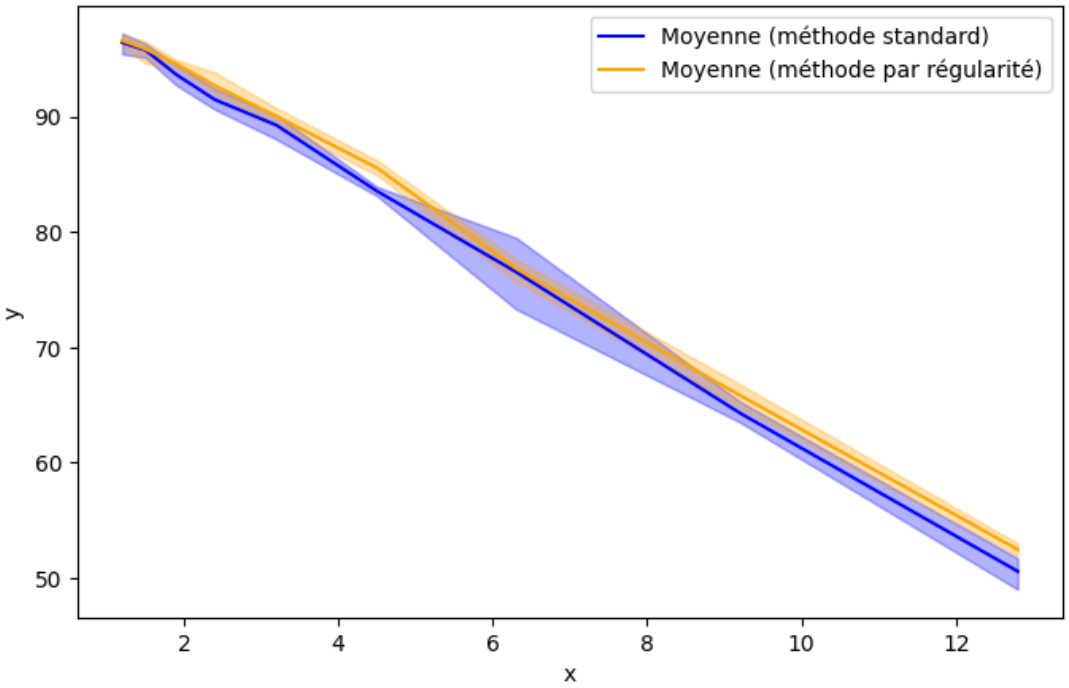


FIGURE 29 – Comparaison, en moyenne, des modèles de pruning après fine-tuning

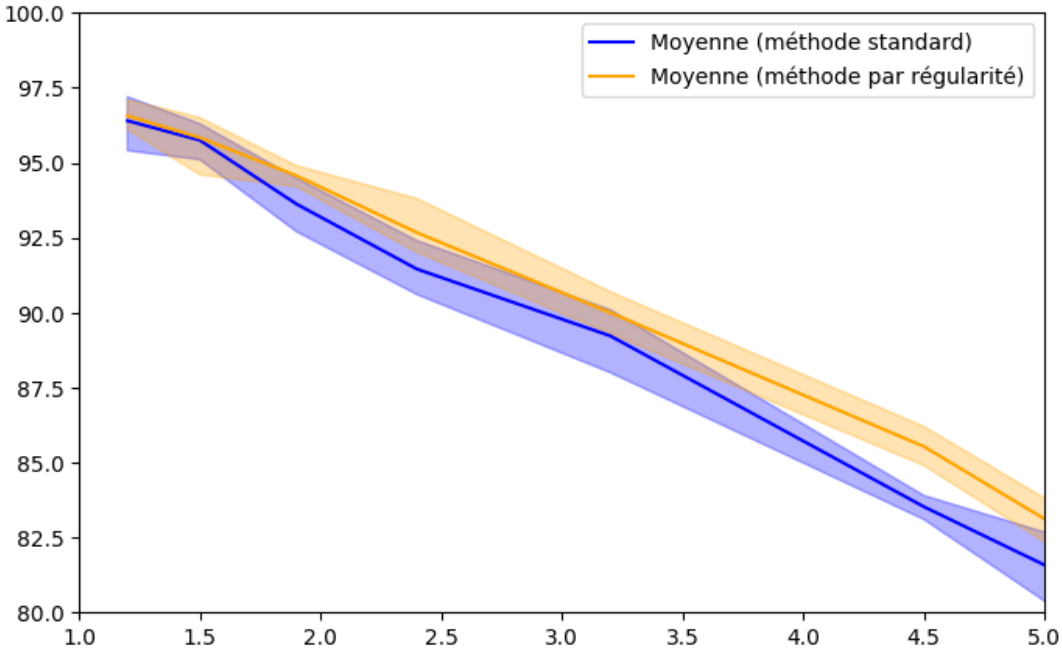


FIGURE 30 – Comparaison des modèles de pruning après fine-tuning pour niveau d'accuracy inférieur à 4

Ainsi, nous constatons qu'en moyenne, après fine-tuning, notre approche par régularité sur-performe les approches classiques jusqu'au niveau de compression 5. Les courbes ci-dessus représentent, en effet, non seulement la moyenne d'accuracy des approches standards et par régularité pour différentes valeurs de compression, mais aussi respectivement la méthode la plus et la moins performante au sein de l'accuracy pour chaque méthode.

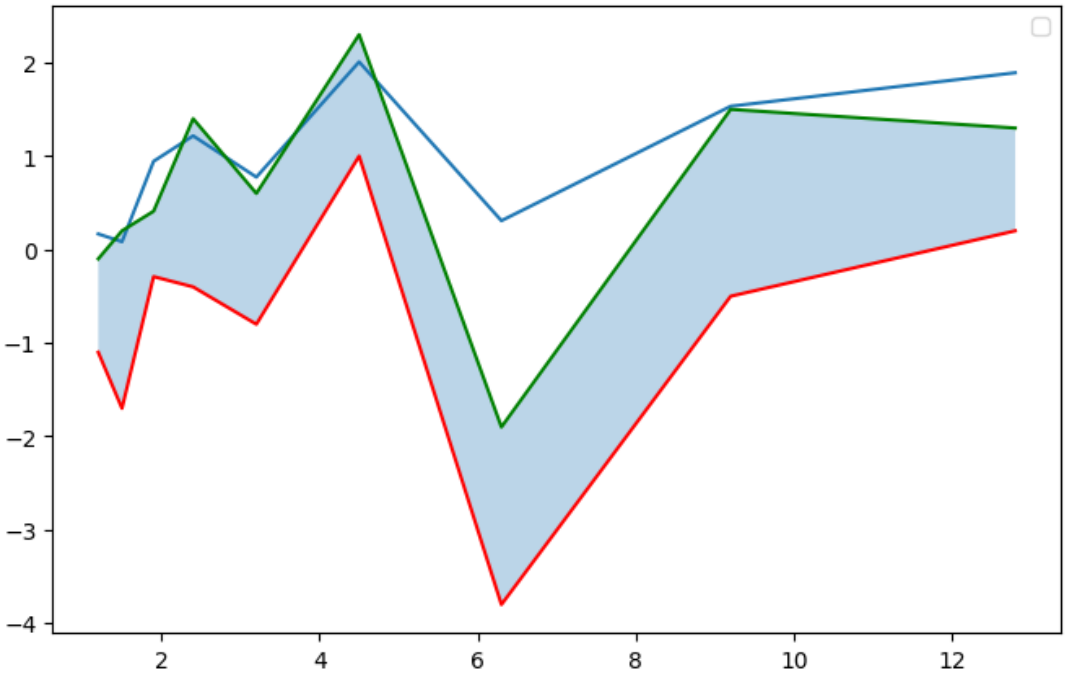


FIGURE 31 – Difference d'accuracy entre les moyennes des méthodes standards et de régularités

Ce graphique nous permet d'observer l'évolution de la performance de notre modèle en comparaison avec les méthodes standards. En particulier, notre modèle est pour toutes les valeurs de compression plus performant que la moyenne des modèles standards. Plus précisément, la courbe verte représente la différence d'accuracy entre le meilleur modèle de régularité (rappelons que nous effectuons plusieurs mesures en fonction de la quantité d'information de régularité conservée par filtre) et le meilleur modèle standard. Sauf pour le niveau de compression 6,3, notre meilleur modèle (au sens de l'accuracy) sur-performe le meilleur modèle des méthodes standards. Toutefois, la différence de performance reste faible. La courbe rouge représente la différence d'accuracy entre le modèle de régularité le moins précis et le modèle standard le plus précis. Similairement au résultat avant fine-tuning, la courbe rouge montre que tous les modèles de régularité ne sont pas plus précis que le meilleur modèle standard. Ainsi, nous constatons l'importance d'avoir filtré l'information par filtre. En effet, si la méthode par régularité a pu surperformer les méthodes standards, elle peut également les sous-performer selon la quantité d'information de régularité sélectionnée par filtre.

3.4 Conclusion et perspectives

En créant une norme basée sur la transformée de Fourier, nous avons pu fournir une mesure de régularité sur les filtres des couches de convolution. La méthode de pruning structurel basée sur cette mesure a finalement confirmé l'hypothèse initiale. En effet, au cours de cette étude, en comparant différentes approches, nous avons constaté que la régularité des filtres d'un CNN semblait être au moins aussi importante que la magnitude de ces derniers. Les performances, dans le pruning de notre modèle, de la méthode de régularité suggèrent en fait qu'elle le sera même davantage sur certains paliers de compression. Toutefois, nous avons effectué notre étude sur une architecture CNN donnée ; conclure plus globalement nécessiterait d'effectuer des tests complémentaires sur des couches de convolution différentes. En particulier, notre mesure de régularité repose sur un kernel size supérieur à 3. Il pourrait ainsi être envisageable que notre conclusion ne s'applique qu'à ce type de filtre et ne tienne plus pour un kernel size de 3 ou pour un agencement différent des couches de convolution. Aussi, le classement des filtres pourrait être amélioré. En effet, une fois la mesure de régularité construite, nous avons appliqué cette dernière à l'ensemble des filtres et sélectionné les filtres les plus réguliers en conséquence. Dès lors, nous individualisons complètement le traitement des filtres. Cependant, lors de l'entraînement, c'est bien la complémentarité des poids des filtres qui permet d'optimiser ces derniers, les filtres étant successivement appliqués couche après couche aux résultats des convolutions précédentes. Il s'agirait ainsi de trouver un moyen de conserver la relation des filtres inter-couches, afin que la suppression d'un filtre non régulier ne soit pas préjudiciable à un filtre régulier dans une couche suivante. Bien que le fine-tuning permette de compenser cet effet, il demeure que le traiter dès la base de notre méthode permettrait peut-être de mieux bénéficier de l'entraînement initial du modèle. Enfin, nous comparions la mesure locale en sommant les valeurs de régularité sur l'ensemble du filtre ; des approches statistiques comparant moyenne, médiane ou variance de régularité pourraient aussi être envisagées afin de comparer les régularités.

Enfin de compte appliquer notre méthode d'étude par régularité, nous permet de diminuer la taille de notre modèle de 25% en conservant une accuracy de 80.2%. Soit en choisissant $P = 80$ et compression 1.9 (afin de conserver une accuracy supérieur à 80%). Une prochaine étape non traitée ici serait de, comme annoncer initialement, compléter notre approche par régularité par une approche par magnitude. Il s'agirait alors de filtre les filtres par magnitude mais seuleument pour les filtres les plus réguliers, ou pour les sous-ensembles de filtre les plus réguliers (selon à seuil variable comme fait précédemment), ci-dessus la description du modèle pruné :

```
    ----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 22, 31, 31]           1,078
              ReLU-2          [-1, 22, 31, 31]               0
       BatchNorm2d-3          [-1, 22, 31, 31]              44
            Conv2d-4          [-1, 22, 30, 30]           7,766
              ReLU-5          [-1, 22, 30, 30]               0
       BatchNorm2d-6          [-1, 22, 30, 30]              44
         MaxPool2d-7          [-1, 22, 15, 15]               0
         Dropout2d-8          [-1, 22, 15, 15]               0
```

```
            Conv2d-9           [-1, 44, 14, 14]          15,532
             ReLU-10           [-1, 44, 14, 14]               0
      BatchNorm2d-11           [-1, 44, 14, 14]              88
           Conv2d-12           [-1, 44, 13, 13]          31,020
             ReLU-13           [-1, 44, 13, 13]               0
      BatchNorm2d-14           [-1, 44, 13, 13]              88
        MaxPool2d-15             [-1, 44, 6, 6]               0
        Dropout2d-16             [-1, 44, 6, 6]               0
           Conv2d-17             [-1, 89, 5, 5]          62,745
             ReLU-18             [-1, 89, 5, 5]               0
      BatchNorm2d-19             [-1, 89, 5, 5]             178
           Conv2d-20            [-1, 128, 5, 5]          11,520
             ReLU-21            [-1, 128, 5, 5]               0
      BatchNorm2d-22            [-1, 128, 5, 5]             256
AdaptiveAvgPool2d-23            [-1, 128, 1, 1]               0
          Flatten-24                  [-1, 128]               0
           Linear-25                  [-1, 128]          16,512
             ReLU-26                  [-1, 128]               0
          Dropout-27                  [-1, 128]               0
           Linear-28                   [-1, 10]           1,290
================================================================
Total params: 148,161
Trainable params: 148,161
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.53
Params size (MB): 0.57
Estimated Total Size (MB): 2.11
----------------------------------------------------------------
```