Deepström: Émulsion de noyaux et d'apprentissage profond

Luc Giffon, Stéphane Ayache, Hachem Kadri, et Thierry Artières

Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France

Résumé

Les modèles à base de méthodes à noyaux et d'apprentissage profond ont essentiellement été étudiés séparemment jusqu'à aujourd'hui. Des travaux récents se sont focalisés sur la combinaison de ces deux approches afin de tirer parti du meilleur de chacune d'elles. Dans cette optique, nous introduisons une nouvelle architecture de réseaux de neurones qui bénéficie du faible coût en espace et en temps de l'approximation de Nyström. Nous montrons que cette architecture atteint une performance du niveau de l'état de l'art sur la classification d'images des jeux de données MNIST et CIFAR10 tout en ne nécessitant qu'un nombre réduit de paramètres.

Mots-clef : appentissage profond, méthodes à noyaux, approximation Nyström.

1 Introduction

Les méthodes à noyaux et les méthodes d'apprentissage profond, les réseaux de neurones profonds, sont deux familles de modèles qui ont essentiellement évolué en parallèle. Chacune a ses forces et ses faiblesses et ces méthodes apparaissent complémentaires en termes de contextes dans lesquels elles sont pertinentes. Les réseaux de neurones peuvent être utilisés sur des quantités gigantesques de données et sont capables d'apprendre à extraire des caractéristiques pertinentes à partir des données. Les méthodes à noyaux sont puissantes de part leur capacité à étendre les méthodes d'apprentissage linéaire au cas non-linéaire et ont des fondements théoriques robustes. Cependant, leur complexité calculatoire les rend difficiles à mettre en œuvre quand le nombre de données à traiter devient très grand. Par ailleurs, elles reposent sur un choix a priori de noyau, donc des caractéristiques prises en compte pour l'apprentissage.

Des travaux récents tentent de tirer parti du meilleur de chacune de ces deux familles d'algorithmes pour concevoir de nouvelles méthodes de classification. Certaines visent à la construction de noyaux profonds qui permettent d'apprendre des représentations profondes comme les couches profondes d'un réseau de neurones [CS09, MBM11, HLE+16]. D'autres visent plutôt à « brancher » des méthodes à noyaux à des réseaux de neurones [MKHS14, YMD+15]. Notre travail se situe dans ce dernier axe.

Récemment, les *Deep Fried Convnets* ont été proposés afin de réduire le nombre de paramètres des réseaux de neurones en utilisant en lieu et place des couches denses d'un réseau de neurones convolutionnel une variante adaptative d'une approximation de fonction noyaux appelée *Fastfood* [LSS13]. Cette stratégie permet de conserver la capacité d'apprentissage de représentation des couches de convolution d'un réseau profond tout en utilisant un nombre de paramètres réduits inhérent à cette méthode d'approximation de noyaux. Si cette approche a montré des résultats à l'état de l'art et un gain significatif en terme de complexité en espace et en inférence, elle reste limitée par le choix de noyaux car les noyaux Gaussiens sont les seuls compatibles avec l'approximation *Fastfood*.

Ce travail présente les réseaux *Deepström* comme une alternative aux *Deep Fried Convnets*. Nous revisitons l'idée de ces derniers en exploitant l'approximation Nyström plutôt que *Fastfood* pour intégrer un noyau après les couches de convolution. L'avantage majeur de cette méthode est qu'elle ne se cantonne pas à l'approximation de noyaux Gaussiens mais est capable d'approximer tout type de noyaux. Par ailleurs, elle permettrait d'utiliser simultanément voire en cascade plusieurs fonctions noyaux. Nos expérimentations sur les bases de données MNIST et CIFAR10 montrent que notre approche atteint des résultats de l'ordre de ceux de l'état de l'art avec un nombre de paramètres réduit et du même ordre de grandeur que pour les *Deep Fried Convnets* en pratique.

Cet article est organisé comme suit. Nous présentons tout d'abord dans la section 2 le cadre dans lequel nous travaillons : nous introduisons une vue modulaire des

réseaux de neurones puis nous présentons les travaux qui ont inspiré cet article : les *DeepFried Convnets*. Ensuite, nous décrivons en section 3 l'approche que nous proposons, les réseaux *Deepström*. Enfin nous produisons des résultats expérimentaux sur les bases de données MNIST et CIFAR10 en comparant différentes instances de notre approche avec les méthodes à l'état de l'art.

2 Cadre de travail

Nous nous intéressons à la tâche de la classification supervisée d'images à partir d'un jeu de données $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ avec n la taille de l'ensemble d'apprentissage, $\mathbf{x}_i \in \mathbb{R}^d$, d le nombre d'attributs des données et $y_i \in \{1, \ldots, c\}$. Dans cette tâche, nous voulons apprendre une fonction de classification f qui minimise $\sum_{i=1}^n l(f(\mathbf{x}_i), \mathbf{y}_i)$ où l est une fonction de perte sur la classification pour un exemple donné. En particulier, nous nous intéressons à la fonction apprise pour cette tâche par des réseaux de neurones convolutifs car ils sont extrêmement efficaces en vision par ordinateur. Cette efficacité est principalement due à leur capacité à apprendre conjointement des filtres de convolutions et un Perceptron Multi-couche (MLP) par descente de gradient. Les filtres de convolution permettent d'obtenir une représentation haut-niveau de la donnée brute initiale qui simplifie le travail de classification du MLP. Nous notons la fonction de filtrage convolutif *conv* et la fonction de classification calculée par le MLP *mlp*. Ainsi, nous pouvons définir la fonction de classification apprise par un réseau de neurone convolutif telle que la composition de *conv* et *mlp* soit :

$$f(\mathbf{x}) \;=\; (mlp \circ conv)(\mathbf{x}). \tag{1}$$

Nous définissons à présent un cadre de travail dans lequel nous précisons le fonctionnement de la fonction de classification *mlp* qui est composée de deux parties qui sont (I) une fonction de représentation non-linéaire ϕ et (II) un classifieur linéaire noté *lc*. La fonction ϕ plonge d'abord les exemples dans un espace de dimension q où les données de classes différentes sont linéairement séparables puis la fonction *lc* identifie leur classe. Cette vision modulaire des réseaux de neurones convolutifs est représentée schématiquement en Figure 1.

Dans ce papier, nous nous intéressons tout particulièrement à la fonction de représentation non-linéaire ϕ. Nous verrons en Section 2.1 et 2.2 deux approches pour apprendre cette fonction de représentation non-linéaire.

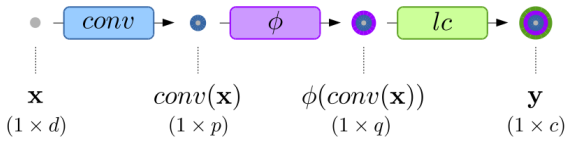

FIGURE 1 – Vision modulaire des réseaux de neurones convolutifs. La fonction ϕ (représentée par le rectangle central) peut être obtenue suivant différentes méthodes.

2.1 Les couches denses

Les réseaux de neurones convolutifs peuvent être vus comme composés d'une série de couches convolutionnelles qui projettent les données dans un nouvel espace de représentation puis d'un Perceptron Multi-couche (MLP) qui réalise la classification des exemples. La dernière couche de ce MLP assure le rôle de la fonction *lc* et toutes les couches cachées avant elle sont chargées de calculer la représentation non-linéaire des exemples. Nous notons ϕ_{nn} la fonction de représentation non-linéaire issue de ces couches cachées. Pour garder l'explication simple mais sans perdre en généralité, nous considérons que le MLP ne possède qu'une seule couche cachée. Ainsi, à partir d'un exemple $\mathbf{x}$, le calcul de la fonction de représentation non-linéaire est :

$$\phi_{nn}(\mathbf{x}) = a(\mathbf{x}\mathbf{W}), \tag{2}$$

où $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{W} \in \mathbb{R}^{d \times q}$, a est une fonction d'activation non-linéaire et q est la dimension de l'espace de plongement de ϕ_{nn}.

Enfin, la fonction implémentée par un tel réseau convolutif (Figure 2) est $f(\cdot) = (lc \circ \phi_{nn} \circ conv)(\cdot)$ où *lc*, ϕ_{nn} et *conv* sont appris conjointement par descente de gradient.

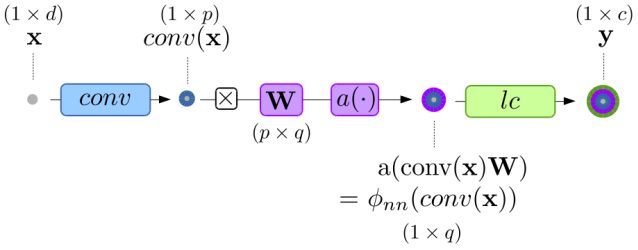

FIGURE 2 – Représentation d'un réseau de neurones convolutif avec une transformation non-linéaire ϕ_{nn}.

2.2 Réseaux *Deep Fried Convnets*

Les fonctions noyaux $k(\cdot,\cdot) = \langle\phi(\cdot),\ \phi(\cdot)\rangle$ permettent de calculer le résultat du produit scalaire entre une représentations non-linéaire ϕ de deux exemples sans connaître explicitement cette représentation non-linéaire. Pour réduire le coût calculatoire des méthodes à noyau, il existe des méthodes d'approximation qui permettent de calculer des $\tilde{\phi}$ tels que $\langle\tilde{\phi}(\cdot),\tilde{\phi}(\cdot)\rangle = \tilde{k}(\cdot,\cdot) \simeq k(\cdot,\cdot)$.

Deep Fried Convnets [YMD+15] est une architecture de réseaux de neurones convolutifs (Figure 3) où la fonction de représentation non-linéaire ϕ_{nn} est remplacée pas une fonction alternative ϕ_{ff} obtenue grâce à la méthode d'approximation de noyaux *Fastfood* décrite ci-dessous. Ainsi, la fonction apprise par les réseaux *Deep Fried Convnets* est $f(\mathbf{x}) = (lc \circ \phi_{ff} \circ conv)(\mathbf{x})$.

Random Kitchen Sinks* et *Fastfood La méthode *Random Kitchen Sinks* (RKS) [RR09] permet le calcul d'une fonction de représentation non-linéaire ϕ_{rks} sous-jacente à l'approximation $\tilde{k}$ d'une fonction noyau k invariante par translation appelée *Radial Basis Function* (RBF) :

$$\phi_{rks}(\mathbf{x}) = \frac{1}{\sqrt{q}}[\cos(\mathbf{Q}\mathbf{x})\sin(\mathbf{Q}\mathbf{x})]^T, \tag{3}$$

où $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{Q} \in \mathbb{R}^{q\times p}$ et les $\mathbf{Q}_{i,j}$ sont tirés aléatoirement suivant une certaine distribution de probabilités. En particulier, si les $\mathbf{Q}_{i,j}$ sont tirés suivant une loi normale alors cette méthode permet d'approximer le noyau RBF particulier qu'est le noyau Gaussien $k(\mathbf{x}_1,\mathbf{x}_2) = \exp(-\gamma * ||\mathbf{x}_1 - \mathbf{x}_2||)$ dont γ est l'hyper-paramètre appelé *largeur de bande* du noyau.

La méthode *Fastfood* [LSS13] est une amélioration de la méthode RKS qui se restreint à l'approximation de fonctions noyaux Gaussien mais à coût réduit. Dans l'équation 3, nous voyons que le calcul de ϕ_{rks} nécessite le produit matriciel $\mathbf{Qx}$ qui a une complexité en temps et en stockage de l'ordre de $O(pq)$. La méthode de *Fastfood* (Équation 5) permet de réduire ce coût en remplaçant la matrice $\mathbf{Q}$ par un produit de matrices diagonales :

$$\mathbf{V} = \frac{1}{\sigma\sqrt{d}}\mathbf{SHG\Pi HB}, \tag{4}$$

où :

- $\mathbf{S}$,$\mathbf{G}$ et $\mathbf{B}$ sont des matrices diagonales de dimension p ;
- $\mathbf{\Pi}$ est une matrice de permutation aléatoire qui peut être implémentée comme une table de recherche de taille p
- $\mathbf{H}$ est une matrice de Hadamard qui n'est pas stockée en mémoire
- σ est un hyper-paramètre.

L'utilisation de l'équation 4 permet de tirer parti de l'algorithme de la transformation rapide de *Hadamard* pour atteindre une complexité en temps de l'ordre de $O(q\log(p))$ et une complexité en stockage de l'ordre de $O(q)$.

La matrice $\mathbf{V}$ ainsi obtenue est utilisée à la place de $\mathbf{Q}$ dans l'équation 3 pour obtenir la fonction de représentation non-linéaire suivante :

$$\phi_{ff}(\mathbf{x}) = \frac{1}{\sqrt{q}}[\cos(\mathbf{Vx})\sin(\mathbf{Vx})]^T. \tag{5}$$

Notons que cette fonction nécessite que d soit une puissance de 2 du fait de la présence de la matrice de Hadamard et que pour obtenir une dimension de représentation $q > p$, il suffit de construire plusieurs $\mathbf{V}$ et de concaténer les résultats des ϕ_{ff} obtenus pour chacune d'elles. Comme, par construction de ϕ_{ff}, $q \propto p$, nous écrirons alors que sa complexité en temps est $O(p\log(p))$ et celle en espace est $O(p)$.

Architecture des *Deep Fried Convnets* Ce type de réseau ce présente sous deux formes : l'une utilise la méthode d'approximation de noyau *Fastfood* telle quelle et l'autre utilise une variante de *Fastfood*, appelée *Adaptative-Fastfood*, qui consiste à apprendre les poids des matrices $\mathbf{S}$,$\mathbf{G}$ et $\mathbf{B}$ par descente de gradient plutôt que de les déterminer aléatoirement. Les matrices $\mathbf{\Pi}$ et $\mathbf{H}$ restent constantes. L'utilisation de cette méthode permet de tirer parti de l'algorithme de transformation rapide de Hadamard pour remplacer la complexité en espace de ϕ_{nn} qui est de $O(pq)$ par une complexité de $O(p)$. Cependant, comme ces réseaux utilisent *Fastfood*, la fonction noyau approximée grâce à la fonction ϕ_{ff} doit être de type RBF.

Comme, la majeure partie des calculs d'un réseau convolutif sont réalisés dans les couches de convolutions, nous ne relevons pas la réduction de la complexité en temps apportée par cette méthode.

Afin de remédier au manque de flexibilité des *Deep Fried Convnets* dans le choix du noyau approximé nous présenterons notre nouvelle architecture de réseau de neurones convolutifs qui s'inspire des réseaux *Deep Fried convnets* mais utilise l'approximation Nyström des fonctions à noyaux en lieu et place de l'approximation *Fastfood*. Nous appelons cette architecture : *Deepström*.

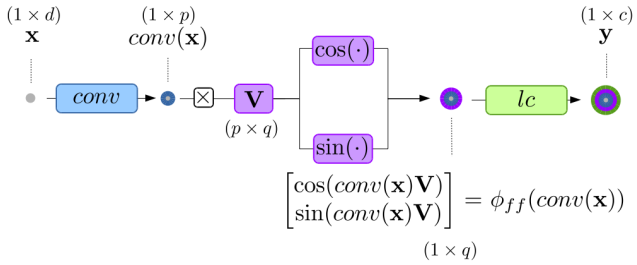

FIGURE 3 – Représentation d'un réseau de neurones convolutif suivant l'architecture des *Deep Fried Convnets* : transformation non-linéaire ϕ_{ff}

3 Réseaux *Deepström*

Nous introduisons l'architecture *Deepström* (Figure 4) qui est un type de réseau de neurones faisant intervenir une fonction de représentation non-linéaire obtenue à partir de l'approximation de Nyström. À partir de *Deep Fried Convnets*, nous remplaçons ϕ_{ff} par ϕ_{nys} de façon à ce que notre modèle apprenne une fonction $f(\mathbf{x}) = (lc \circ \phi_{nys} \circ conv)(\mathbf{x})$.

Approximation de Nyström La méthode de Nyström [WS01] permet de calculer une approximation de rang faible d'une matrice de Gram $\boldsymbol{K}$ telle que :

$$\boldsymbol{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

où $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{x}_j) \rangle \ \forall \ i, j \in [1, \dots, n]$.

Dans cette méthode, un sous ensemble $\mathbf{L} = \{\mathbf{x}_i\}_{i=1}^{m}$ d'exemples est sélectionné à partir de l'ensemble d'apprentissage (par exemple aléatoirement) puis la matrice $\boldsymbol{K}$ est redéfinie en tenant compte de cette sélection :

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{21}^T \\ \boldsymbol{K}_{21} & \boldsymbol{K}_{22} \end{bmatrix}$$

où $\boldsymbol{K}_{11}$ est la matrice de Gram obtenue à partir de $\mathbf{L}$. En utilisant cette sous-matrice $\boldsymbol{K}_{11}$, l'approximation $\tilde{\boldsymbol{K}}$ peut être obtenue telle que :

$$\boldsymbol{K} \simeq \tilde{\boldsymbol{K}} = \begin{bmatrix} \boldsymbol{K}_{11} \\ \boldsymbol{K}_{21} \end{bmatrix} \boldsymbol{K}_{11}^{-1} \begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{21}^T \end{bmatrix}$$

De cette expression, nous pouvons déduire la représentation non-linéaire de Nyström pour un seul exemple $\mathbf{x}$:

$$\phi_{nys}(\mathbf{x}) = \mathbf{k}_{\mathbf{x},\mathbf{L}} \mathbf{W}, \tag{6}$$

où $\mathbf{k}_{\mathbf{x},\mathbf{L}} = [k(\mathbf{x}, \mathbf{x}_1), \dots, k(\mathbf{x}, \mathbf{x}_m)]^T$ (avec $\forall i, \mathbf{x}_i \in \mathbf{L}$) est appelé vecteur noyau et $\mathbf{W}$ est fixé à $\boldsymbol{K}_{11}^{-\frac{1}{2}}$.

Architecture des réseaux *Deepström* Pour pouvoir calculer l'approximation de Nyström sur un exemple $\mathbf{x}$, les réseaux *Deepström* doivent avoir à disposition un sous-ensemble $\mathbf{L}$ d'exemples d'apprentissage. Cependant, comme ϕ_{nys} intervient sur les représentations des exemples calculées par les filtres de convolution $conv(\mathbf{x})$, les exemples de $\mathbf{L}$ doivent eux aussi avoir été pré-traités par les même filtres de convolution. Une fois que les convolutions ont été calculées, la fonction noyau peut-être appliquée à l'exemple et au sous ensemble pour obtenir le vecteur noyau $\mathbf{k}_{\mathbf{x},\mathbf{L}}$ qui subit ensuite une transformation linéaire par $\mathbf{W}$ avant d'être soumis au classifieur linéaire.

Nous notons toutefois deux différences majeures entre *Deep Fried convnets* et *Deepström* qui sont inhérentes aux méthodes de transformation utilisées par les deux types de réseaux : (I) Nyström est compatible avec n'importe quelle fonction noyau là où *Fastfood* ne peut qu'approximer des noyaux Gaussiens et (II) contrairement à *Fastfood* l'approximation de Nyström est dépendante des données d'apprentissage.

Toutefois, pour calculer $\mathbf{K}_{11}^{-\frac{1}{2}}$, l'approximation de Nyström fait intervenir le calcul de la Décomposition en Valeurs Singulières (SVD) de $\mathbf{K}_{11}$ ce qui pourrait ne pas être envisageable dans un réseau convolutif dans le cas où m, la taille de $\mathbf{L}$, est grand car la complexité de cette SVD est en $O(m^3)$.

Pour répondre à ce problème, nous avons également utilisé une architecture *Adaptative-Deepström* dans laquelle, plutôt que de fixer les poids de $\mathbf{W}$, nous les apprenons avec les autres paramètres par gradient.

4 Résultats expérimentaux

Dans cette section, nous présenterons les résultats obtenus lors de la mise en œuvre de *Deepström*. Nous commencerons par présenter les données ainsi que les détails des architectures de réseaux de neurones utilisées. Enfin, nous présenterons les résultats avec des graphiques et une analyse.

4.1 Matériel et méthodes

4.1.1 Données

Les expériences ont été menées sur les jeux de données de classification d'images MNIST [LC] et CIFAR10 [Kri09]. Les détails sur ces jeux de données sont

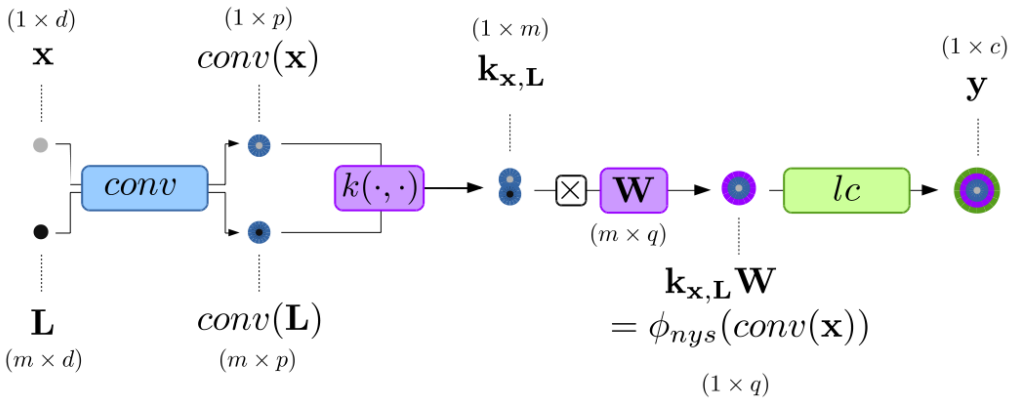

FIGURE 4 – Représentation d'un réseau de neurones convolutifs suivant l'architecture *Deepström* : transformation non-linéaire ϕ_{nys}

présentés dans le tableau 1. Pour l'apprentissage des filtres de convolution, nous réalisons une augmentation des données par retournement et translation. Les données sont aussi normalisées avant leur traitement.

4.1.2 Architecture

Convolution Les architectures de convolution utilisées sont l'architecture Lenet [LBBH98] pour le jeu de données MNIST et VGG19 [SZ14] pour CIFAR10. Notons que les filtres de convolution de Lenet ont été légèrement modifiés de façon à ce que la dimension des exemples après convolution soit une puissance de 2. Les poids des filtres de convolution ont été pré-entraînés [BIG18] puis fixés dans les expériences. Dans ce papier, nous ne montrons pas de résultats sur l'apprentissage de bout en bout du réseau Deepström.

Fonctions de représentation Nous comparons les architectures de réseaux de neurones convolutifs qui utilisent les fonctions de représentation ϕ_{nn} (Équation 2), ϕ_{ff} (Équation 5) et ϕ_{nys} (Équation 6).

Couches Denses Nous considérons uniquement le cas où ϕ_{nn} contient une seule couche de représentation avec activation *relu*. Nous utilisons $\{2, 4, 8, 16, 32, 64, 128, 1024\}$ comme dimensions de représentation de ϕ_{nn}.

Fastfood Nous considérons le cas où ϕ_{ff} est obtenue à partir d'une seule matrice $\mathbf{V}$. Ainsi, la dimension de représentation est égale à la dimension des données après leur traitement par les couches de convolution. Par ailleurs, nous n'utilisons que la version *Adaptative* de cet algorithme.

Notons que n'ayant pas d'implémentation disponible pour la transformation rapide de Hadamard, nous avons opté pour une implémentation naïve de *Fastfood* dans laquelle nous stockons la matrice de Hadamard en mémoire et calculons simplement le produit matrice-vecteur. Cette stratégie ne nous permet pas de comparer expérimentalement les différences de temps d'exécution entre *Deep Friedconvnet* et *Deepström* mais elle est facilement implémentable.

Nyström La fonction de représentation ϕ_{nys} est paramétrée par les exemples du sous-ensemble $\mathbf{L}$, par la fonction noyau utilisée et la dimension de représentation souhaitée. Dans nos expériences, nous utilisons $\{2, 4, 8, 16, 32, 64, 128\}$ exemples dans le sous-ensemble, nous utilisons les noyaux Gaussien ($k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\gamma * ||\mathbf{x}_1 - \mathbf{x}_2||^2)$), Linéaire ($k(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$) et Chi2 ($k(\mathbf{x}_1, \mathbf{x}_2) = ||\mathbf{x}_1 - \mathbf{x}_2||^2/(\mathbf{x}_1 + \mathbf{x}_2)$) et nous choisissons une dimension de représentation égale à la taille du sous-ensemble utilisé. Par ailleurs, dans la section suivante, nous évaluons l'impact d'une fonction de non-linéarité *relu* supplémentaire après le calcul de ϕ_{nys}. Enfin, la fonction ϕ_{nys} est testée dans sa version basique et sa version *Adaptative*.

Optimisation La fonction de perte utilisée est l'entropie croisée après activation *softmax* et l'algorithme d'optimisation utilisé est Adam. Par ailleurs un *dropout* avec une probabilité de 0.5 est appliqué sur les résultats des calculs de représentation ϕ.

Nom	Dimension	Nombre de classes	Taille entraînement	Taille validation	Taille test
MNIST	$(28 \times 28 \times 1)$	10	40 000	10 000	10 000
CIFAR10	$(32 \times 32 \times 3)$	10	50 000	10 000	10 000

TABLE 1 – Détail des jeux de données utilisés

Hyper-paramètres Les hyper-paramètres des modèles qui sont considérés sont le pas de gradient qui est fixé à $1e^{-4}$ et le γ du noyau Gaussien qui est choisi comme étant l'inverse de la moyenne des normes des différences entre les exemples après la convolution : $\gamma = \frac{1}{n^2} \sum_{i,j=1}^{n} ||\mathbf{x}_i - \mathbf{x}_j||^2$.

Implémentation D'une part, l'implémentation et l'entraînement des couches convolutives a été faite grâce à la librairie Keras [C+15]. D'autre part, l'implémentation et l'évaluation des différentes méthodes d'apprentissage des fonctions de représentation a été faite grâce à la librairie Tensorflow [ABC+16].

4.2 Résultats obtenus

Dans cette section nous verrons comment *Deepström* se comportent face à *DeepfriedConvnet* ou des réseaux convolutifs conventionnels. Nous faisons varier le nombre de paramètres dans chaque architecture pour mettre en évidence la performance de chacune en fonction de l'espace mémoire nécessaire pour les stocker.

En Figure 5, les graphiques présentent les résultats de qualité de classification par rapport au nombre de paramètres apprenables dans la fonction de classification *lc*. Chaque expérience a été répliquée 10 fois et les valeurs moyennes et écarts types sont représentés sur la figure. Les résultats numériques précis sont notés dans le tableau 2.

Qualité d'approximation En analysant la Figure 5 et le Tableau 2, nous voyons que l'architecture *Deepström* permet d'atteindre une qualité de classification de l'ordre de celle de l'architecture Dense et de *Deepfried Convnet*. Pour le jeu de données MNIST, l'architecture Dense obtient des résultats légèrement meilleurs que *Deepström*. Cependant, pour le jeu de données CIFAR10, *Deepström* bat l'architecture Dense et ce, avec nettement moins de paramètres.

***Adaptative-*ϕ_{nys} et ϕ_{nys} simple** Dans la Figure 5, nous constatons que la version de non-*Adaptative* de ϕ_{nys} est généralement meilleure que la version *Adaptative*. Nous pouvons justifier ce résultat par le fait que la version *Adaptative* apprend une représentation sans aucune contrainte sur $\mathbf{W}$ et par là s'éloigne de la méthode de l'approximation de Nyström. Dans des travaux futurs, nous pourrions essayer d'améliorer ces résultats en contraignant la matrice $\mathbf{W}$ à partager des propriétés de $\mathbf{K}^{-\frac{1}{2}}$.

Non-linéarité La fonction de décision finale de l'architecture *Deepström* est un classifieur linéaire. Ainsi la fonction de classification est de la forme $lc(\mathbf{x}) = \mathbf{k}_{\mathbf{x},\mathbf{L}}\mathbf{WD}$ avec $\mathbf{D} \in \mathbb{R}^{m \times c}$ et donc, en l'absence de non-linéarité après ϕ_{nys}, cette fonction est équivalente à $lc(\mathbf{x}) = \mathbf{k}_{\mathbf{x},\mathbf{L}}\mathbf{D}$. Nous nous attendions donc à obtenir un gain de performance en appliquant une non-linéarité sur le résultat de $\phi_{nys}(\mathbf{x})$ mais nous voyons en Figure 5 que ce n'est pas le cas avec la non-linéarité *relu*.

5 Conclusion

Dans cet article, nous présentons une architecture hybride d'apprentissage profond, nommé Deepström, qui associe à la fois des réseaux de neurones et des méthodes à noyaux. L'architecture proposée est basée sur l'approximation Nyström permettant d'intégrer des représentations de différents fonctions noyaux dans l'apprentissage de réseaux de neurones profonds, tout en réduisant les coûts en temps de calcul et en espace mémoire. Des expériences sur les jeux de données MNIST et CIFAR10 montrent que les réseaux Deepström peuvent atteindre les performances de l'état de l'art avec un nombre de paramètres significativement réduit comparé aux réseaux denses et aux réseaux *Deep Fried Convnets*. Ce nouveau type d'architecture laisse par ailleurs entrevoir la possibilité de travailler sur des données multi-vues via l'utilisation combinée de plusieurs fonctions noyaux.

Références

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga,

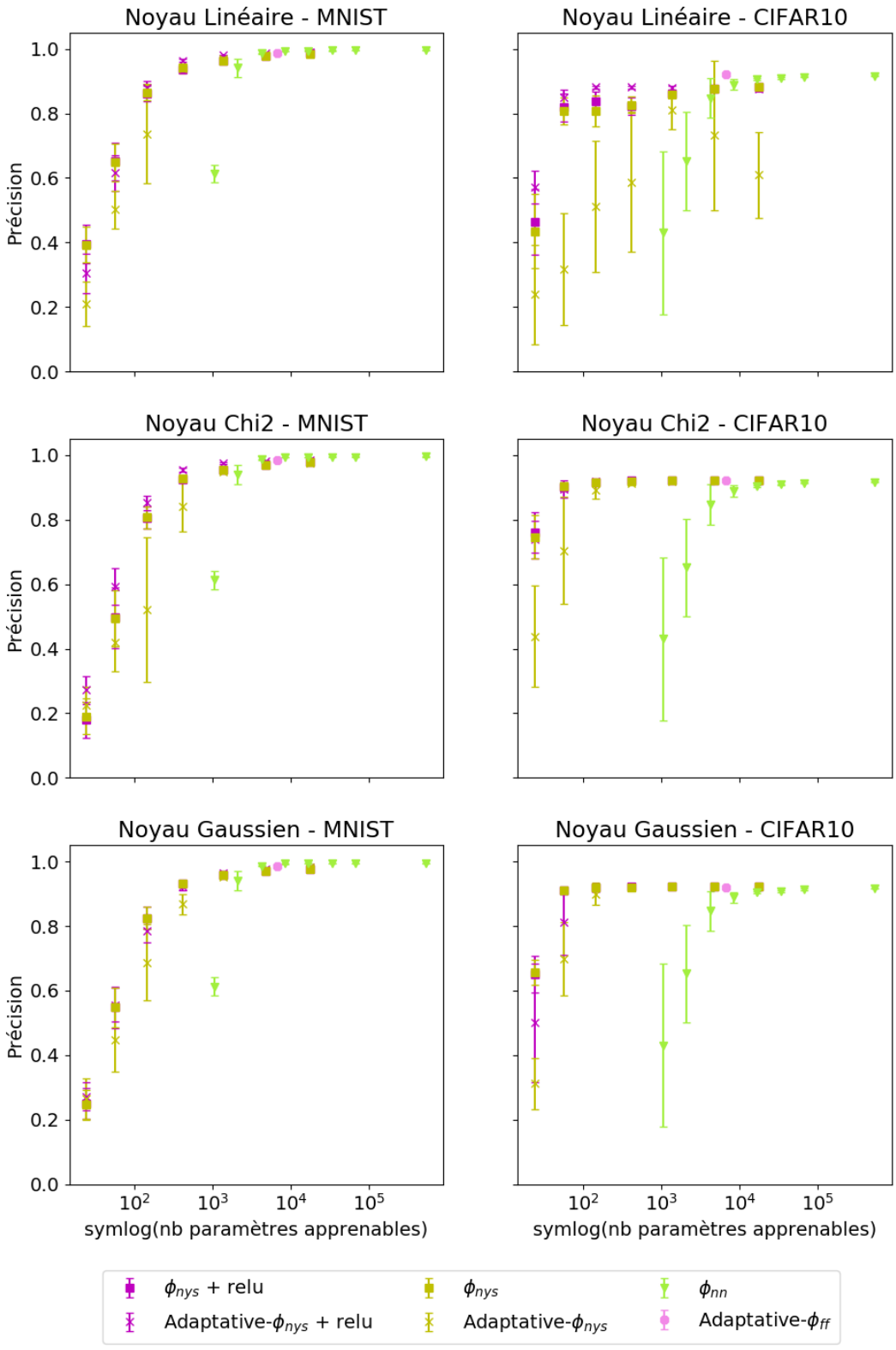

FIGURE 5 – Résultat de la précision des modèles en fonction de leur nombre de paramètres

Jeu de données	$\phi_{nn} - 2$	$\phi_{nn} - 4$	$\phi_{nn} - 8$	$\phi_{nn} - 16$	$\phi_{nn} - 32$	$\phi_{nn} - 64$	$\phi_{nn} - 128$	$\phi_{nn} - 1024$
MNIST	61.32	94.09	98.66	99.32	99.46	99.50	99.53	**99.56**
CIFAR10	43.03	65.23	84.76	88.93	90.52	90.89	91.37	91.66
	ϕ_{ff}	$\phi_{nys} - 2$	$\phi_{nys} - 4$	$\phi_{nys} - 8$	$\phi_{nys} - 16$	$\phi_{nys} - 32$	$\phi_{nys} - 64$	$\phi_{nys} - 128$
MNIST	98.63	30.30	58.04	84.12	93.79	96.90	97.87	98.36
CIFAR10	92.06	74.34	90.37	**91.74**	**92.15**	**92.16**	**92.19**	**92.20**

TABLE 2 – Tableau de résultats numériques. Nous notons $\phi_{nn} - q$ les meilleurs résultats obtenus avec l'architecture Dense avec un espace de représentation égal à q ; ϕ_{ff} les résultats obtenus avec l'architecture *Deep Fried Convnet* ; et $\phi_{nys} - m$ les meilleurs résultats obtenus avec l'architecture *Deepström* sans non-linéarité pour un nombre m d'exemples dans l'échantillon $\mathbf{L}$, indépendamment du noyau utilisé

Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow : A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association.

[BIG18] BIGBALLON. cifar-10-cnn. `https://github.com/BIGBALLON/cifar-10-cnn`, 2018.

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015.

[CS09] Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 342–350. Curran Associates, Inc., 2009.

[HLE+16] Uri Heinemann, Roi Livni, Elad Eban, Gal Elidan, and Amir Globerson. Improper deep kernels. In *Artificial Intelligence and Statistics*, pages 1159–1167, 2016.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, November 1998.

[LC] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits.

[LSS13] Quoc V. Le, Tamas Sarlos, and Alex Smola. Fastfood-computing hilbert space expansions in loglinear time. In *International Conference on Machine Learning*, 2013.

[MBM11] GrÃŠgoire Montavon, Mikio L Braun, and Klaus-Robert MÃtller. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 12(Sep) :2563–2581, 2011.

[MKHS14] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *Advances in neural information processing systems*, pages 2627–2635, 2014.

[RR09] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks : Replacing minimization with randomization in learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1313–1320. Curran Associates, Inc., 2009.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[WS01] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[YMD+15] Z. Yang, M. Moczulski, M. Denil, N. d. Freitas, A. Smola, L. Song, and Z. Wang. Deep fried convnets. In *2015 IEEE International Conference on Computer Vision (ICCV)*, volume 00, pages 1476–1483, Dec. 2015.