Learning Deep Architectures

Attia - Constantini - Hayat

Mai 2016

Contents

1 Introduction

Les réseaux de neurones sont devenus très populaires par leur capacité à reproduire des tâches intelligentes comme la reconnaissance d'objets dans une image. La popularisation de cette technique est notamment due aux avancées dans le deep learning ainsi qu'à l'augmentation de la puissance de calcul des GPU (Graphics Processing Unit). On distingue les réseaux normaux des réseaux profonds, dont le nombre de couches est supérieure à 3 convolutions. Les meilleurs résultats de classification sont donnés par des réseaux profonds (AlexNet: 100 couches), grâce au nombre important de non-linéarités. Le but de ce projet est donc d'étudier un réseau profond pré-entrainé soumis à différentes troncatures. On ajoute à ces réseaux tronqués différents classifieurs dont on étudiera la performance. L'étude porte donc à la fois sur l'impact de la profondeur et sur le type de classifieur à ajouter. Tout d'abord, nous avons entrainé un réseaux de neurones convolutionnels simple sur la base CIFAR 10 pour comprendre l'intérêt de la profondeur des réseaux. Ce réseau rassemble convolutions, non-linéarités, max-pooling et dropout (pour diminuer l'overfitting). Nous avons utilisé des Rectified Linear Unit (ReLU) comme non-linéarités ($max(0, x)$). On remarque une bonne performance du réseau qui n'est néanmoins pas comparable à celle des réseaux profonds. Notre étude consistera à un travail similaire sur un réseau de neurones convolutionnels plus complexe : Network in Network. Nous étudierons ce réseau, l'impact des troncatures à différents endroits du réseau puis l'influence de l'ajout d'un classifieur.

2 Base de données

Nous avons utilisé trois bases de données pour notre étude.

- CIFAR 10. Cette base de données contient 10 classes basiques : avion, voiture, oiseau, chat, cerf, chien, chat, grenouille, cheval, bateau, camion, de taille 32x32, avec 6000 données (divisées entre une partie d'entrainement et une partie test) par classes.Le preprocessing commun est de passer des canaux RGB aux canaus YUV puis de normaliser par une gaussienne et enfin de découper le dataset en un training set et un testing set.
- ImageNet. Cette base de données est plus complexe et contient environ 1000 image par classe avec presque 80000 classes. ImageNet est constituée d'environs 2 millions d'images étiquetées en mille classes. Ce dataset étant très volumineux et exhaustive
- Notre base de données. Nous avons construit une base de données à partir d'ImageNet, nous avons choisi 10 classes assez différentes, avec 10000 données d'entraînement et 3000 données test séparées aléatoirement.

3 Nos réseaux de neurones

Nous avons utilisé des réseaux de neurones convolutionnels, i.e. des reseaux de neurones contenant une à plusieurs convolutions. Dans notre étude, il a fallu ajouter des couches convolutionnels et donc savoir adapter ces couches à l'ensemble du réseau et aux données d'entrées, de sortie, aux marges et à la taille de la convolution souhaitée. Pour ceci, nous avons utilisé la formule suivante :

$$\text{dimensions des données de sortie} = \frac{\text{dimensions des données d'entrée} - \text{taille du kernel} + 2 \times \text{zero-padding}}{\text{pas}} + 1$$

Grâce à cette formule, nous avons pu déterminé les paramètres de nos couches du réseau convolutionnel. Dans notre projet, nous avons utilisé deux modèles de reseaux de neurones convolutionnels : ConvNet simple et Network in Network.

3.1 Réseau simple à convolution

Le réseau simple à convolution est un réseau composé de trois blocks identiques puis d'un perceptron multi-couche et enfin d'un classifieur. Chaque block est construit de la manière suivante: une couche convolutionnelle, une non-linéarité (Rectified Linear Unit), un couche de max pooling qui diminue la taille de la donnée et enfin une couche de Dropout pour remédier à le sur-apprentissage que nous avions. En effet, le Dropout désactive, au hasard avec une probabilité 0.5, les neurones de telle manière que les poids ne soient pas modifiés lors de la rétropropagation. Cette technique permet ainsi de généraliser et d'éviter le sur-apprentissage.

Nous avons entraîner notre réseau sur Cifar 10 avec 20000 données d'entraînement et 5000 de test.

3.2 Network in Network

Le réseaux Network in Network est un réseau convolutionel constitué de blocks de petits réseaux de neurones. Nous avons d'abord utilisé ce réseau de neurones sur la base de données Imagenet dont nous avons extrait 5 classes d'oiseaux. Le résultat fut de 40 % en ré-entrainant tout le réseau. A cause du temps de calcul trop important, nous nous sommes tournés vers la base de données Cifar10 qui est constitué d'images plus petites (32x32). Néanmoins adapté un réseau de neurones et le ré-entraîner peut s'avérer compliqué et couteux en temps. Nous avons alors décidé de garder le réseau NiN et la base de données Imagenet.

3.3 Network in Network tronqué

Notre base données est donc extraite d'Imagenet et comporte 10 classes. On va alors tronqué le réseau à différents endroit et nous avons décidé d'entraîner seulement les couches rajoutées. L'entraînement se fait sur 5000 données d'entraînement et 1000 données de test.

4 Entraînement

L'entraînement du réseau de neurones est effectué par l'algorithme de rétropropagation du gradient. Cet algorithme repose sur la règle de la chaîne pour calculer le gradient à chaque couche.

Détaillons l'algorithme:

- **Forward Pass** Les données passent dans le réseaux dont chaque couche a pour fonction F_i : $X_i = F_i(X_{i-1}, W_i)$, X_i la donnée en sortie de la couche i.
- **Rétropropagation** Notons $E = C(X_n, Y)$ l'erreur de prédiction, c'est-à-dire le coût entre la prédiction et l'étiquette. Pour mettre à jour les poids W_i, on utilise la règle de la chaîne :

$$\begin{aligned} \frac{\partial E}{\partial W_i} &= \frac{\partial E}{\partial X_i}\frac{\partial X_i}{\partial W_i} = \frac{\partial E}{\partial X_i}\frac{\partial F_i(X_{i-1}, W_i)}{\partial W_i} \quad \text{et en calculant} \frac{\partial E}{\partial X_i} \text{ avec:} \\ \frac{\partial E}{\partial X_{i-1}} &= \frac{\partial E}{\partial X_i}\frac{\partial F_i(X_{i-1}, W_i)}{\partial X_{i-1}} \end{aligned} \tag{1}$$

  Ainsi, l'algorithme propage en arrière l'erreur de prédiction et met à jour les poids.

On remarque ainsi que l'algorithme de rétropropagation a des meilleurs chances de marcher lorsque la dérivée se calcule simplement. Ainsi, pour des architectures profondes, l'erreur étant propagée à travers plusieurs couches, il est nécessaire que ce calcul se fasse simple. C'est le cas des réseaux à convolution.

Nous noterons que la rétropropagation est l'étape la plus coûteuse en temps et en calcul et une des raisons pour lesquelles nous avons choisi prendre un réseau pré-entraîné.

4.1 Nos classifieurs

A la sortie du réseau tronqué nous avons transformé notre entrée x en un élément $\phi(x)$ où ϕ est la fonction qui renvoie l'image d'une image par notre réseau de neurones. Après une transformation linéaire (et parfois des convolutions) qu'on appellera ψ, nous récupérons ainsi $f = \psi \circ \phi(x)$ de dimension variable selon notre réseau. Dans la suite on posera $\psi \circ \phi$.
Notre objectif est donc de déterminer des classifieurs qui prennent en entrée le vecteur $f(x)$ et doivent prédire y. Nous avons choisis deux types de classifieurs qui utilisent des fonctions de coûts différentes:

- Lineaire: Negative Log Likelihood.
- SVM: Multi Margin Criterion.

4.1.1 Negative Log Likelihood

Ce critère est très utile pour les régressions multi-classes. On a pour une régression logistique classique:

- les labels $y \in \{0, 1\}$
- la fonction de régression était $h_\theta = \frac{1}{1+exp(-\theta^T x)}$
- la fonction de coût était alors: $J(\theta) = -\frac{1}{N}[\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i)]$

L'idée est donc d'étendre cette classification multi-classe (ici 10). Pour cela on va poser la fonction h_θ:

$$h_\theta(x) = \begin{pmatrix} \mathbb{P}(y = 1|x; \theta) \\ \mathbb{P}(y = 2|x; \theta) \\ \dots \\ \mathbb{P}(y = 10|x_i; \theta) \end{pmatrix} = \frac{1}{\sum_{i=0}^{10} exp(\theta_i^T x)} \begin{pmatrix} exp(\theta_1^T x) \\ exp(\theta_2^T x) \\ \dots \\ exp(\theta_{10}^T x) \end{pmatrix}$$

On peut remarquer que le terme en $\frac{1}{\sum_{i=0}^{10} exp(\theta_i^T x)}$ est une renormalisation en vecteur de probabilités et par rapport à la régression binaire, le vecteur θ des paramètres d'apprentissage est:

$$\theta = \begin{pmatrix} -\theta_1 \\ -\theta_2 \\ \dots \\ -\theta_{10} \end{pmatrix}$$

Donc la fonction de coût est:

$$J(\theta) = -\frac{1}{N}[\sum_{i=1}^{N}\sum_{j=1}^{10} \mathbb{1}y_i = j \log(\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{k} e^{\theta_l^T x_i}}$$

4.1.2 Multi Margin Criterion

La fonction de coût pour ce critère est la fonction hinge loss multi-classe:

$$\forall 1 \leq i \leq 10, L(x) = \sum_{j=1, j\neq y_i}^{N} \max(0, f(x_j) - f(x_{y_i}) + \Delta)$$

où $\Delta = 1$ et pas de régularisation ce qui nous une fonction de coût pour N le nombre de données

$$L(x) = \frac{1}{N}\sum_{i=1}^{N} L(x_i)$$

5 Résultats

Précisons que les résultats suivant portent sur les données de test. Les graphiques en annexes présentent les résultats sur les données d'entraînement et de test.

5.1 Réseau simple à convolution

Pour notre étude, nous avons donc réalisé de nombreuses expériences pour comparer les classifieurs et comparer les profondeurs du réseau. Notre but étant de toujours comprendre l'influence de la profondeur et de vérifier l'impact positif de la profondeur sur la precision de classification d'un réseau de neurones convolutionnels.
Nous avons donc au préalable réalisé deux expériences comparatives sur ce réseau initial que nous avons construit pour constater l'influence de la profondeur.

	Tronqué	Non-Tronqué
Convnet	65%	62%

Figure 1: Précision de classification pour le réseau simple à convolution

En effet, plus le réseau est profond, plus la précision de classification est bonne.

5.2 Network in Network

Nous avons réalisé un plus gros nombre d'experiences pour notre réseau Network in Network et ses différentes troncatures.
Durant nos experiences, nous avons également essayé de trouver un classifieur optimal pour ce réseau Network in Network. Nous avons obtenu des courbes retracant l'evolution de la precision et de l'érreur de classification au cours des itérations pour pouvoir au mieux comparer nos résultats (cf Annexe). Voici nos résultats synthétisés sous forme d'un tableau avec la précision asymptotique de classification :

Classifieur	Tronqué 3	Tronqué 2	Tronqué 1	Non-Tronqué
Régression Logistique	74,9%	84,3%	90%	92,9%
SVM	71,6%	90,9%	90,7%	92,9%
Convolution et Régression Logistique	76,7%	88,8%	90,9%	/

Figure 2: Précision de classification de nos mesures pour différents classifieurs et différentes profondeurs

Comme nous l'avions présagé, les résultats pour tous les classifieurs sont meilleurs lorsque l'on augmente la profondeur du réseau (ie on rajoute un micro réseau à notre réseau de neurones). On constate que l'on obtient sur notre base de données, un score de precision de classification de $92,9\%$ soit un score semblable à celui obtenir sur la base de données CIFAR 10. Cette valeur nous a servi de référence par rapport aux resultats pour nos troncatures. Nous avons également constaté que plus la profondeur du réseau est importante, plus la converge est rapide.
Par contre, nous n'avons pu conclure sur les classifieurs. En effet, le classifieur, avec la meilleure precision, varie selon la profondeur.

6 Sur-apprentissage

Nous avons utilisé differents outils pour diminuer le sur-apprentissage dans notre étude. En effet, initialement, nous avons pu constater un fort sur-apprentissage avec notre premier réseau de neurones convolutionnels Convnet (plus de 10% d'écart de precision de classification). Pour lutter contre ce sur-apprentissage. Nous avons utilisé plusieurs couches de Drop-Out disposés après les couches de max-pooling. Durant l'entrainement du réseau, ces couches de Drop-Out permettent de masquer une partie des données d'entrées. On entre en paramètres la probabilité p pour chaque donnée d'entrée d'etre non prise en compte i.e que la valeur de sortie associée à cette donnée d'entrée est nulle (BIBLI see Hinton et al. 2012). Lors du reste de notre étude, nous avons eu un peu de problèmes de sur-apprentissage. Dans l'entrainement des couches complémentaires (après troncature ou non) du réseau Network in Network, nous avons ajusté différents paramètres pour n'avoir que très peu de sur-apprentissage. Nous avons essayé differents paramètres de régularisation ($\alpha = 5 * 10^-4$, $5*10^-3$ et $5*10^-4$, appelé WeightDecay sur Torch7). Pour cela, nous avons utilisé un algorithme de gradient stochastique (pour un mini-lot de 64 données) pour lequel nous avons essayé différents pas d'apprentissage avant d'éviter les minimum locaux et réduire le sur-apprentissage. Nous avons finalement décidé de nous arreter sur la valeur de 0.01 de pas d'apprentissage.

7 Conclusion

Durant notre étude, nous avons pu constater l'influence de la profondeur sur un réseau de neurones. En effet, quelque soit le classifieur utilisé (SVM ou Regression Logistique), moins notre réseau Network in Network est profond et contient de micro-réseaux, plus les performances et la précision de classification des images sont meilleures. Nous n'avons pu par contre conclure sur le meilleur classifieur pour ce réseau. Les différences de performances entre classifieurs sont peu perceptibles. Le classifieur n'est donc pas pertinent, contrairement au noyau determiné par le le réseau NiN.

Nous avons fait face à plusieurs difficultés classiques de l'utilisation du Deep Learning :

- Temps d'entrainement. Les temps d'entrainement de nos réseaux tronqués étaient assez longs et variaient selon la profondeur. Nous avons également souhaiter entrainer le réseau entier mais le temps d'entrainement était beaucoup trop long et donc impossible avec nos ordinateurs.
- Capacités techniques de nos ordinateurs. Dans notre cas, un GPU embarqué de capacités moyennes est moins performant qu'un CPU multi-coeurs.
- Données. Une question importante est survenue, celle du nombre de données à utiliser pour avoir des performances raisonnables. Nous avons donc du arbitrer entre le temps d'entrainement et le nombre de données à utiliser. Il a également fallu choisir quelles données et quels paramètres à adapter en conséquence.
- Permutations des données. Une de nos difficultés a été de bien mélanger nos données sans quoi nous obtenions les performances d'un classifieur aléatoire.

Dans un second temps, nous aurions souhaité nous attarder à d'autres classifieurs, par-exemple ceux liés aux arbres de décisions (Random Forest, Gradient Boosting, etc.) et les comparer à nos précédents classifieurs. D'après le texte Y. Bengio, nous aurions pu initialisé les couches de nos différents réseaux de neurones avec un pré-apprentissage avec un apprentissage non supervisée couche par couche plutôt qu'une initialisation aléatoire.

8 Remerciements

Nous souhaitons remercier : Sergey Zagoruyko pour son aide précieuse pour l'usage de ses fonctions et Guillaume Obozinski pour nous avoir indiqué le cadre et les limites de ce projet.

Annexes

Simple Convnet

Nous avons entraîner notre réseau sur Cifar 10 avec 20000 données d'entraînement et 5000 de test.

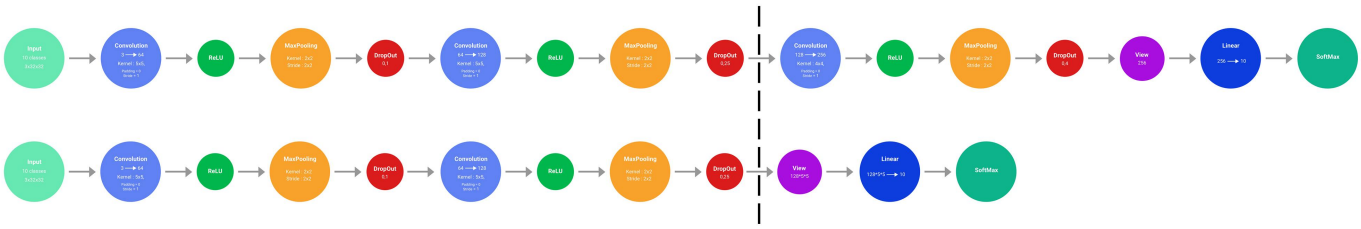

Figure 3: Our convnet neural networks

CNN

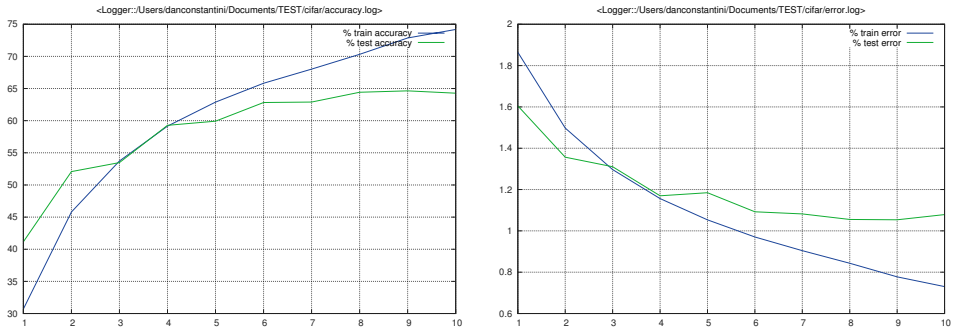

Figure 4: Précision & Erreur

Truncate CNN

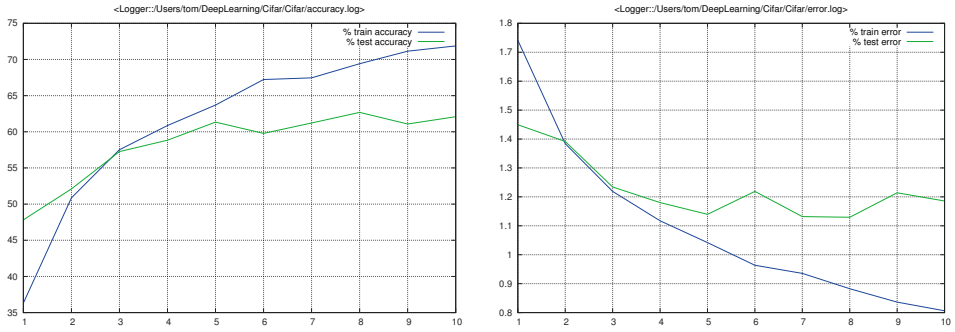

Figure 5: Précision & Erreur

Base de données

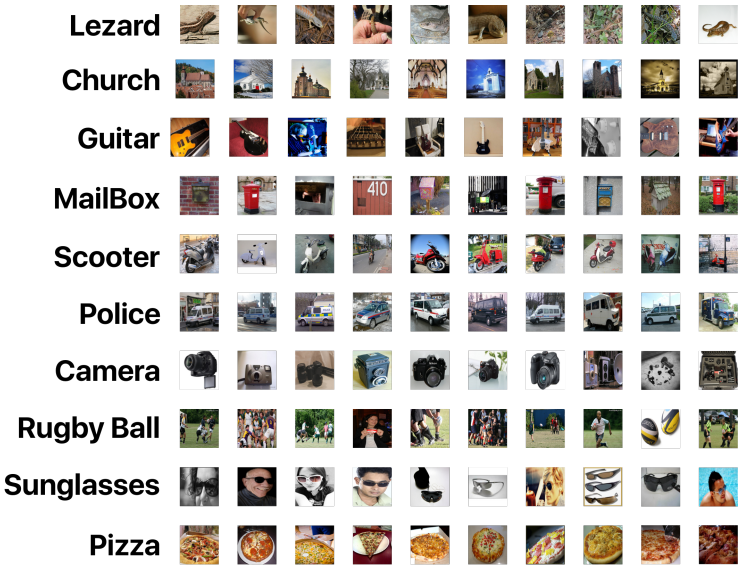

Figure 6: Notre base de données

Network in Network

```
(1): nn.SpatialConvolution(3 -> 96, 11x11, 4,4, 5,5)
(2): nn.ReLU
(3): nn.SpatialConvolution(96 -> 96, 1x1)
(4): nn.ReLU
(5): nn.SpatialConvolution(96 -> 96, 1x1)
(6): nn.ReLU
(7): nn.SpatialMaxPooling(3x3, 2,2, 1,1)
(8): nn.SpatialConvolution(96 -> 256, 5x5, 1,1, 2,2)
(9): nn.ReLU
(10): nn.SpatialConvolution(256 -> 256, 1x1)
(11): nn.ReLU
(12): nn.SpatialConvolution(256 -> 256, 1x1)
(13): nn.ReLU
(14): nn.SpatialMaxPooling(3x3, 2,2, 1,1)
(15): nn.SpatialConvolution(256 -> 384, 3x3, 1,1, 1,1)
(16): nn.ReLU
(17): nn.SpatialConvolution(384 -> 384, 1x1)
(18): nn.ReLU
(19): nn.SpatialConvolution(384 -> 384, 1x1)
(20): nn.ReLU
(21): nn.SpatialMaxPooling(3x3, 2,2, 1,1)
(22): nn.SpatialConvolution(384 -> 1024, 3x3, 1,1, 1,1)
(23): nn.ReLU
(24): nn.SpatialConvolution(1024 -> 1024, 1x1)
(25): nn.ReLU
(26): nn.SpatialConvolution(1024 -> 1024, 1x1)
(27): nn.ReLU
(28): nn.SpatialAveragePooling(7x7, 1,1)
(29): nn.View(-1)
```

```
(30): nn.Linear(1024 -> 1000)
```

Résultats Graphiques du NiN

Sans troncatures

Dans cette partie, nous étudions le réseau Network in Network non tronqué. Nous utilisons la version non normalisée de ce réseau et nous allons utilisé différents classifieurs et calculer l'erreur et la précision.

Linéaire

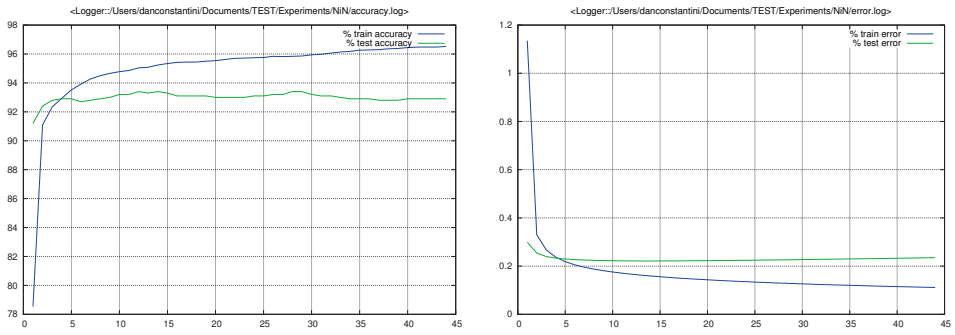

Figure 7: Linéaire - Précision & Erreur

Dans le cas de la régression logistique (classification linéaire), nous obtenons ainsi 92.5% de précision, comparable au résultat du réseau NiN sur Cifar-10. Les courbes de précision et d'erreurs sont assez lisses. Il faut environ 5 itérations pour atteindre la valeur asymptotique.

Support Vector Machine

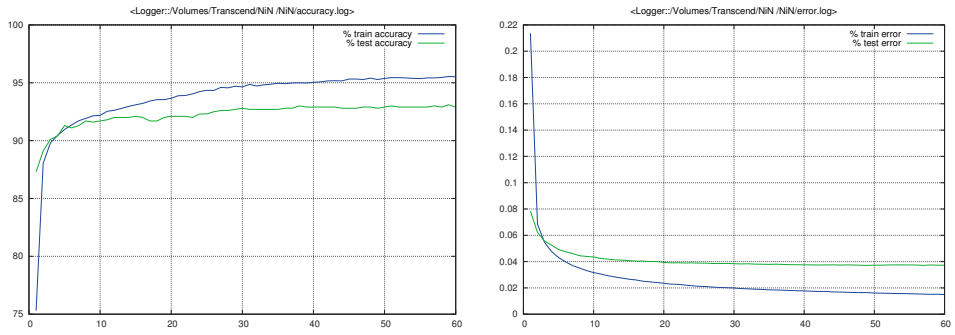

Figure 8: SVM - Précision & Erreur

Dans le cas du Support Vector Machine, nous obtenons un score assez similaire.

Troncatures

Nous avons décidé de tronquer notre réseau à 3 endroits différents:

- Après la couche (7).
- Après la couche (14).
- Après la couche (21).

Couche 7

Linear Nous avons rajouté ici un classifieur multi-classes qui prend en entrée des images de taille $96*28*28$.

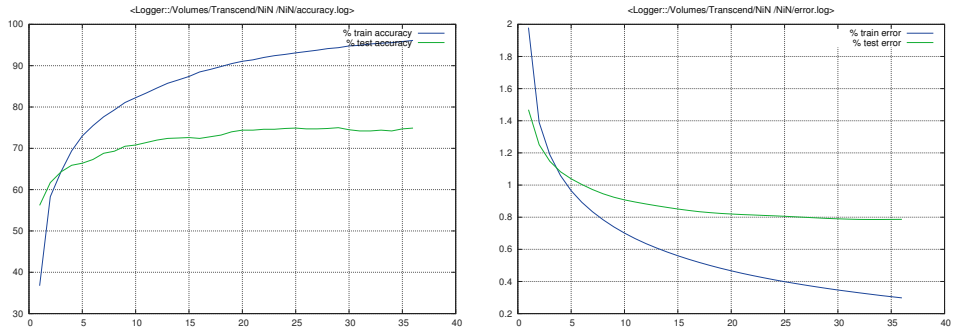

Figure 9: Linéaire - Précision & Erreur

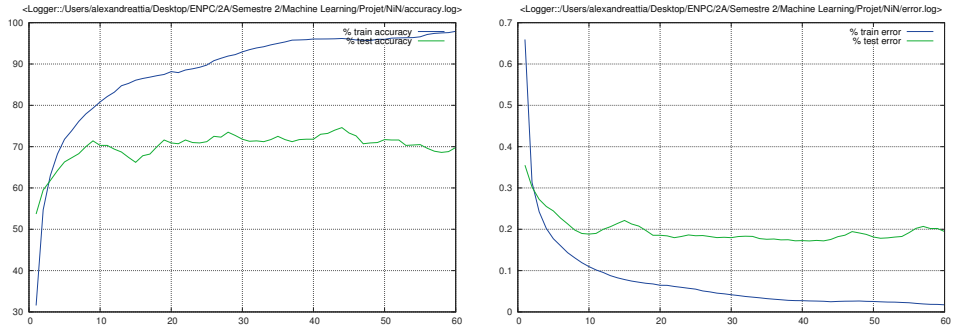

Figure 10: SVM - Précision & Erreur

Support Vector Machine

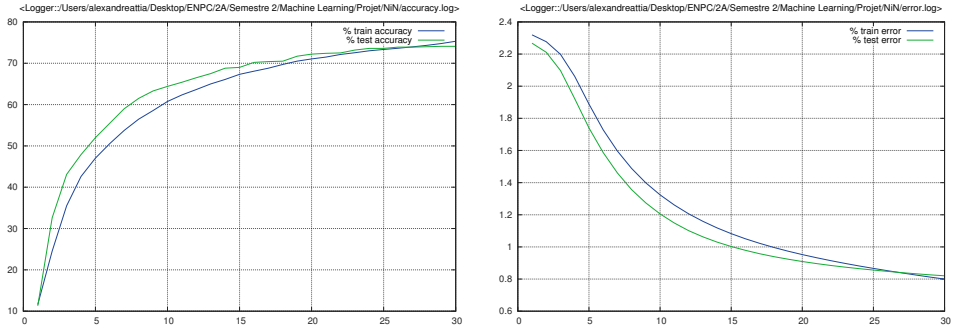

Figure 11: Conv - Précision & Erreur

Convolution

Couche 14

Linear Nous avons rajouté ici un classifieur multi-classes qui prend en entrée des images de taille $256*14*14$.

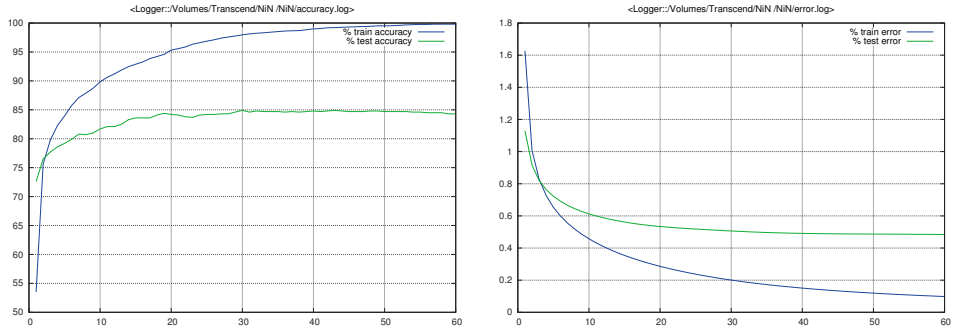

Figure 12: Précision & Erreur

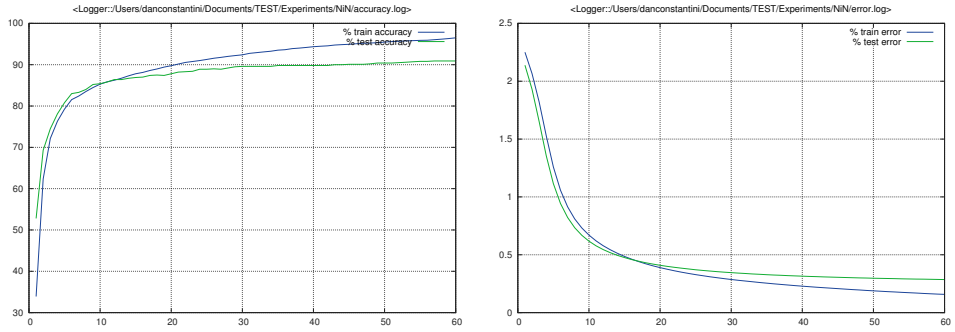

Figure 13: Précision & Erreur

Support Vector Machine

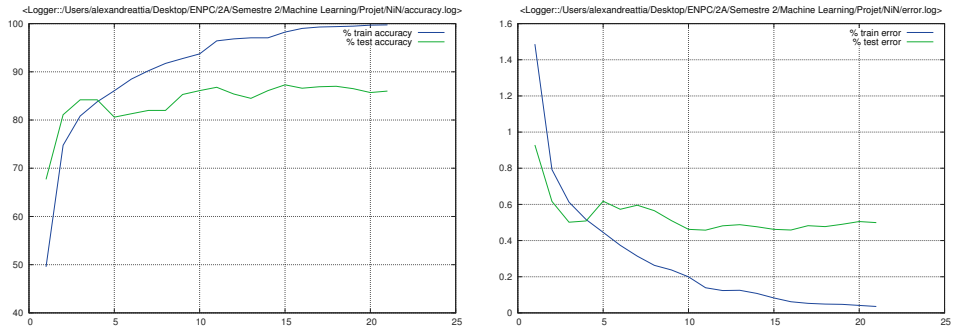

Figure 14: Précision & Erreur

Convolution

Couche 21

Linear Nous avons rajouté ici un classifieur multi-classes qui prend en entrée des images de taille $384 * 7 * 7$.

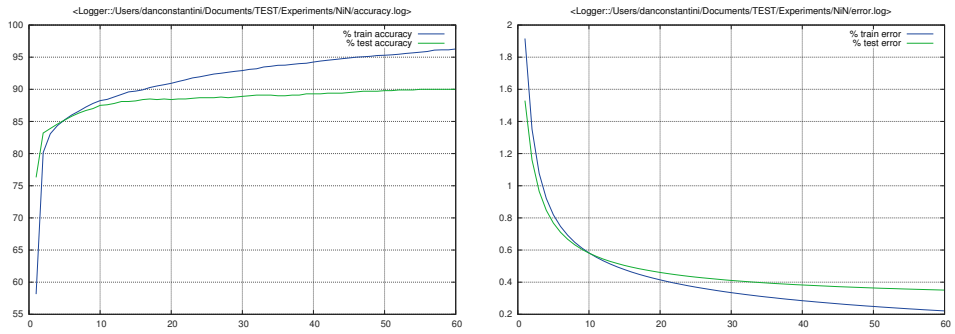

Figure 15: Précision & Erreur

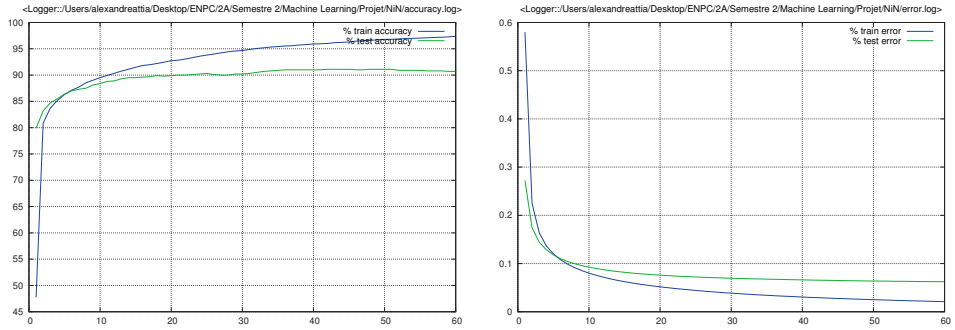

Figure 16: Précision & Erreur

Support Vector Machine

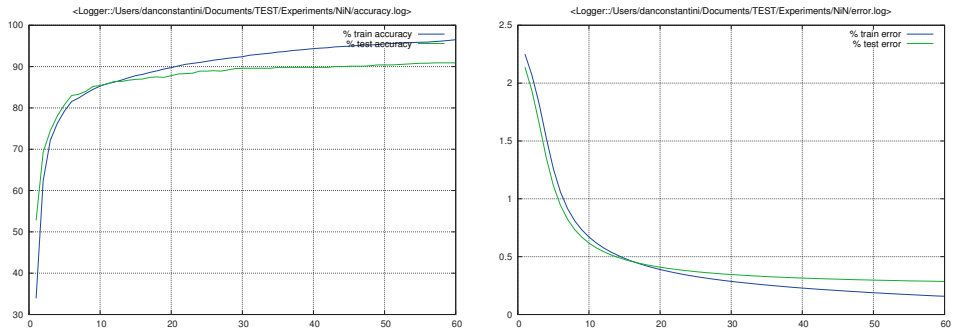

Figure 17: Précision & Erreur

Convolution

References

[1] Y. Bengio. "Learning Deep Architecture for AI". In: *Foundations and Trends in Marchine Learning* (2009), pp. 1–127.

[2] "Deep Networks with Stochastic Depth". In: *CoRR* (2016).

[3] M. Telgarsky. "Benefits of depth in neural networks". In: *CoRR* (2016).

[4] S. Zagoruyko. *Network-in-Network trained in Torch7*. https://gist.github.com/szagoruyko/0f5b4c5e2d2b18472854. [Online; accessed 05-May-2016].