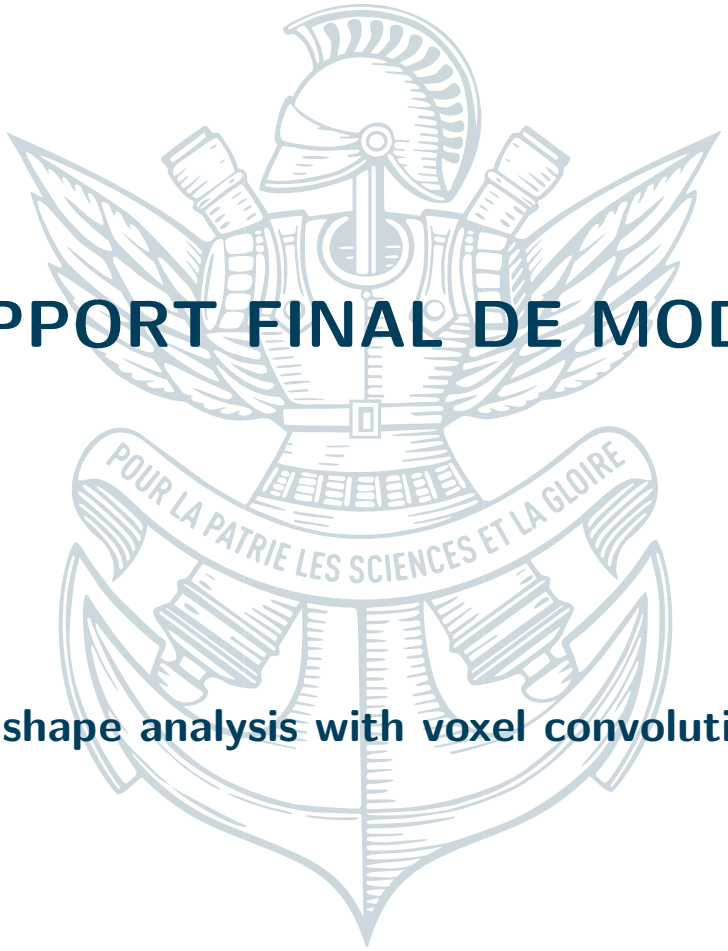

RAPPORT FINAL DE MODAL

3D shape analysis with voxel convolutions

1er juin 2019

Ahmed Bennani, Julien Descamps

1 L'ARCHITECTURE VOXNET : DESCRIPTION DU PROJET

La reconnaissance d'objets en 3 dimensions est un enjeu crucial pour la recherche, en particulier pour la mise en œuvre de robots opérants de manière autonome dans un environnement donné. Ces derniers utilisent des capteurs dont le développement récent a permis une réelle amélioration en terme de fiabilité et de performance.

On définit ainsi le "range imaging" comme l'ensemble des techniques permettant l'acquisition d'images en 2 dimensions dont chaque pixel se voit ajouter une donnée supplémentaire associée à la distance. Depuis plusieurs dizaines d'années, la télédétection par laser (LiDAR) permet de produire ce type de données, mais c'est la production par Microsoft de la caméra Kinect qui a véritablement permis un essor conséquent de cette discipline. Ces progrès technologiques ont ainsi permis la constitution d'un certain nombre de bases de données d'images en 3D que nous avons été amenés à manipuler pour ce projet.

La littérature est riche en méthode permettant la classification d'objet en 3D. Cependant, de nombreuses architectures ne sont pas efficaces lorsqu'il s'agit de départager les différentes instances, ces dernières étant lues sous forme de nuage de points et donc lourdes en information. En 2015, D. Maturana and S. Scherer proposent l'architecture VoxNet, une méthode qui permet pour la première fois d'étiqueter des centaines d'instances par seconde tout en ayant une précision dépassant l'état de l'art de l'époque. L'apport de cet algorithme réside dans la constitution d'une grille d'occupation volumétrique et dans l'utilisation de réseau neuronal convolutif en trois dimensions (3D-CNN).

Il est important de définir deux concepts clefs de cette étude : d'abord, dans cette étude, nous sommes amenés à manipuler des 'occupacy grid' : des matrices représentant l'environnement 3D sous forme de variables aléatoires. Ces dernières, les voxels, sont repérées par leurs trois coordonnées dans l'espace auquel s'ajoute une probabilité de présence.

Depuis VoxNet, un certain nombre de projets ont atteint une précision bien supérieure. Nous les introduirons brièvement à la fin de ce rapport.

2
IMPLÉMENTATION

Lorsqu'est proposée l'architecture VoxNet, la plupart des approches existantes étaient fondées sur l'extraction de caractéristiques fabriquées à la main, qui étaient ensuite apprises à un classificateur de type SVM. Le développement du Deep Learning, qui apprend simultanément quelles sont les caractéristiques et comment les classifier, a permis d'obtention de résultats bien meilleurs.

2.1 3D-CNN, principe général

Si le Deep Learning est la clef d'une telle performance, c'est bien l'utilisation de réseau neuronal convolutif qui distingue véritablement l'architecture VoxNet. Il n'est pourtant pas évident qu'avec un tel type de données, bien plus lourdes que de simples images, le CNN reste l'approche la plus efficace. Pourtant, elle reste pertinente en 3D pour plusieurs raisons :

- Les réseaux neuronaux convolutifs permettent d'apprendre des filtrages spatiaux utiles dans notre cas puisqu'ils exploitent la structure spatiale des différentes données. Ils encodent ainsi des plats ou des coins selon différentes orientations, des caractéristiques essentielles pour distinguer les différentes classes proposées à l'algorithme.
- Ensuite, en superposant plusieurs couches, le réseau va pouvoir construire une hierarchie de caractéristiques plus complexes représentants des zones plus grandes dans l'espace, permettant finalement d'aboutir à une étiquette pour la donnée en entrée.
- Enfin, les inférences sont des 'feed forward', les minimisations de gradient peuvent être faites facilement avec des tenseurs et peuvent donc être menées de façon optimale par le GPU.

Les convolutions à 3 dimensions ont trois directions (x, y, z) de calcul et renvoient une matrice ayant elle aussi 3 dimensions.

2.2 Couches choisies

- Input Layer : L'architecture VoxNet accepte en entrée des grilles de 32x32x32 voxels. La valeur de chaque cellule de la grille étant obtenue d'après la grille d'occupation.

- Convolutional Layers : $Conv(f, d, s)$.

  Cette couche créée f feature maps par convolution de l'entrée par f filtres de taille $dxdxdxf'$ avec d=32 dans notre cas et f' le nombre de feature maps en entrée. On utilise ainsi :
  conv3d(depth=d, height=d, width=d, in_channels=f', out_channels=f).
  Les convolutions peuvent être appliquée avec un 'spatial stride' s, la dimension de sortie sera alors (I/J/K - d + 2*padding)/s + 1

  On applique en sortie un Leaky ReLU, c'est à dire la fonction suivante : $f : x \longmapsto \begin{cases} x & \text{si } x \geq 0 \\ 0.1x & \text{si } x \leq 0 \end{cases}$

- Pooling Layers : $Pool(m)$.
  Les couches de pooling (« mise en commun ») sont une forme de sous-échantillonnage de l'image. L'image d'entrée est découpée en une série de cubes de n pixels de côté ne se chevauchant pas (pooling). On renvoie alors ici le maximum de chaque cube.

— Fully Connected Layer : $FC(n)$.
Les neurones dans une couche entièrement connectée ont des connexions vers toutes les sorties de la couche précédente (comme on le voit régulièrement dans les réseaux réguliers de neurones). La sortie de chaque neurone est une combinaison linéaire des sorties de chacune des sorties de la couche précédente.
— Output Layer : La sortie est un vecteur dont la taille correspond au nombre de classe en entrée et les différents éléments à des probabilités. À l'aide de la fonction non linéaire softmax, on peut renvoyer une classe.
On a finalement dans notre cas :

$$Conv(32, 5, 2) \to Conv(32, 3, 1) \to Pool(2) \to FC(128) \to FC(K)$$

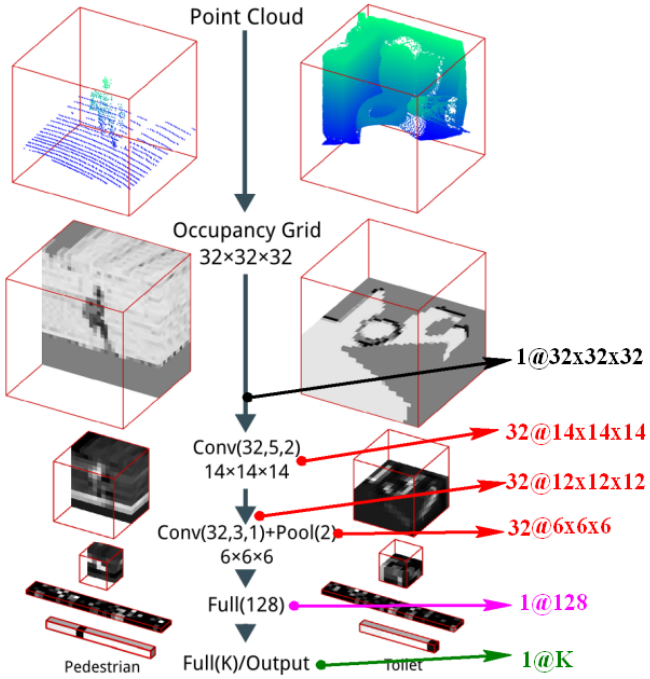

Figure 1 – L'architecture de VoxNet. Conv(f,d,s) signifie f filtres de taille d et de stride s. Pool(m) signifie un pooling de taille m et Full(n) signifie une couche entièrement connectée avec n sorties

2.3 Architecture proposée

À partir de ces différentes couches et de ces différents hyper-paramètres, il existe un très grand nombre d'architectures possibles. Pour les différencier, les auteurs ont procédé à une recherche stochastique extensive sur une base de données limitées. Finalement, les auteurs se sont restreints à un nombre limités de paramètres en partant du principe que cela augmenterait la rapidité de l'étiquetage, permettant à l'algorithme d'apprendre plus vite.

2.4 Data sets

Les auteurs de l'architecture VoxNet ont testé leur algorithme sur trois bases de données distinctes :

— LiDar data - Sydney Urban Objects : ce dataset est constitué des scanners de 631 objets urbains divisés en 26 catégories.
— CAD data - ModelNet : ModelNet40 est constitué de 151.128 représentations 3D classées en 40 catégories. Les auteurs fournissent les modèles 3D ainsi que des versions pré-voxelisées. C'est sur ce dataset et sur sa version réduite Modelnet10 que nous avons concentré l'essentiel de notre travaux. Le découpage correspond à $\frac{3}{4}$ / $\frac{1}{4}$ pour trainset / test set.
— RGBD data - NYUv2 : ce dataset est plus complexe à mettre en œuvre pour des questions de dimension.

Pour augmenter le dataset, les auteurs ont créé 12 copies de chaque éléments correspondants à des rotations de $\frac{360}{12}$ autour de l'axe z du training set. Pour tester la précision de l'algorithme, on calcule la prédiction du réseau sur les n copies de chaque instance du testset. Ces rotations augmentent la robustesse de l'algorithme. En effet, ses prédictions deviennent invariante par rotation.

Pour mener cette étude, nous nous sommes concentrés sur l'implémentation de ModelNet10 et ModelNet40. En effet, en plus d'être disponibles en ligne, les données sont pré-voxélisées. Lors de nos différents tests, nous avons pu tester l'architecture VoxNet sur ModelNet10, ModelNet20 et ModelNet30. Toutefois, ModelNet40 requiert une mémoire vive trop importante pour la machine virtuelle Azure : nous avons du nous limiter aux $\frac{3}{4}$ des données d'entraînement. Notre performance est par conséquent limitée.

Pour téléverser nos données sur notre machine virtuelle Azure, nous avons construit localement un fichier texte contenant une succession de matrices dont le premier élément est le label et le second la matrice à trois dimensions correspondantes. Cependant, en raison de la grande taille du dataset (151.128 données en raison des rotations), nos fichiers textes dépassaient 6 Giga. Nous avons donc choisi d'exploiter la structure de nos données : plutôt que de stocker des matrices 32x32x32, nous avons transmis les coordonnées des points non nuls uniquement. Ce faisant, nos fichiers ne dépassaient pas le giga. Il nous a ensuite été très simple de reconstruire les différentes matrices lors du traitement des données.

Après avoir testé l'algorithme sur ModelNet, nous avons tenté d'essayer une autre base de données. Malheureusement, celle-ci sont bien plus complexes à exploiter et lourdes, relativement au nombre de données disponibles.

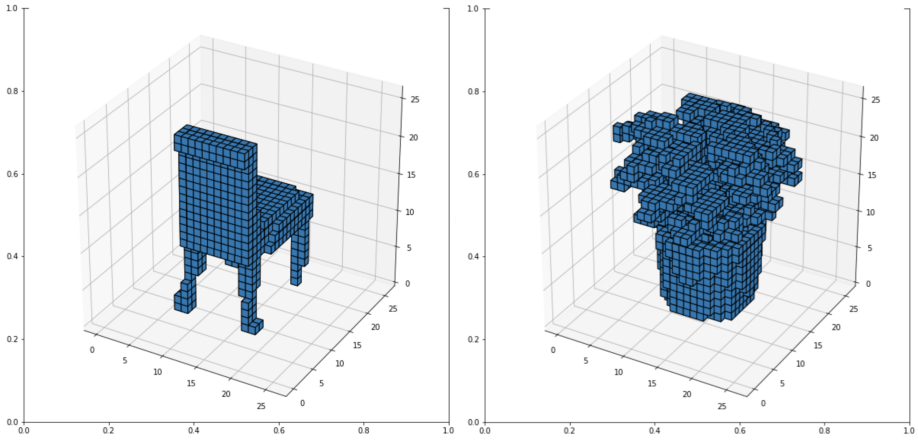

Figure 2 – Exemple de données du dataset ModelNet : ici une chaise et un pot de fleur.

3 RÉSULTATS

3.1 Apport de l'optimiseur et de l'augmentation de données

Nous avons correctement implémenté l'algorithme et obtenu des résultats identiques. Dans un deuxième temps, nous avons changé l'optimiseur 'stochastique gradient descent' pour Adam. Nous obtenons les résultats comparatifs suivants pour le dataset augmenté par rotation :

	ModelNet10	ModelNet20	ModelNet30	ModelNet40
SGD	0.92	/	/	0.83
Adam	0.96	0.94	0.88	0.83*

Ces résultats montrent un réel apport d'Adam.

Il peut être intéressant d'étudier les effets de l'augmentation de la taille du dataset par rotation :

Augmentation	Optimiseur	ModelNet10
Oui	SGD	0.83
Oui	Adam	0.83*
Non	SGD	0.69
Non	Adam	0.81

* À noter : comme expliqué plus tôt, les résultats pour ModelNet40 et l'optimiseur Adam sont obtenues à partir de 3/4 du dataset uniquement.

Il est intéressant de tracer l'évolution de la précision au cours de l'entraînement :

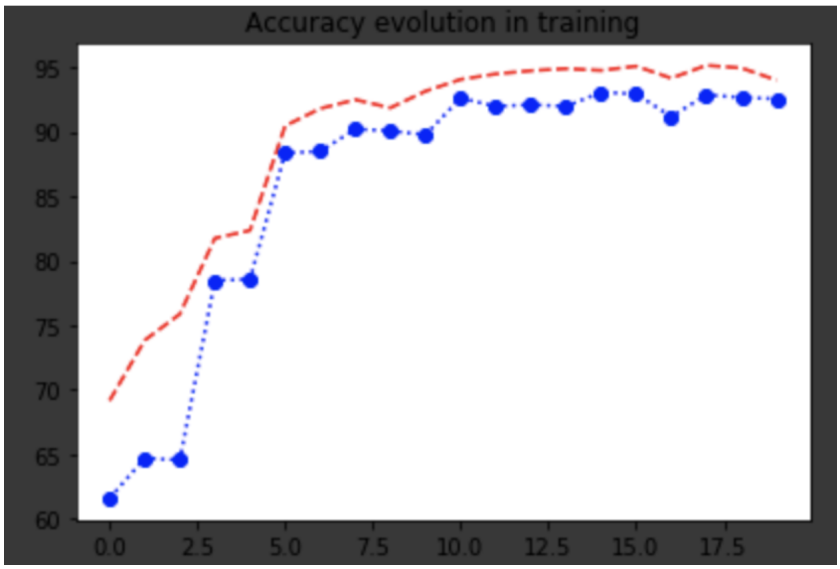

Figure 3 – Evolution de la précision au cours de l'entraînement, ici pour ModelNet10. En rouge l'évolution sur le trainset, en bleu sur le testset.

3.2 Etude des features map

Nous avons par la suite essayé d'interpréter nos résultats afin de mieux comprendre la boite noire entre l'objet et sa prédiction. Voici quelques exemples des différentes visualisations que nous avons obtenu.

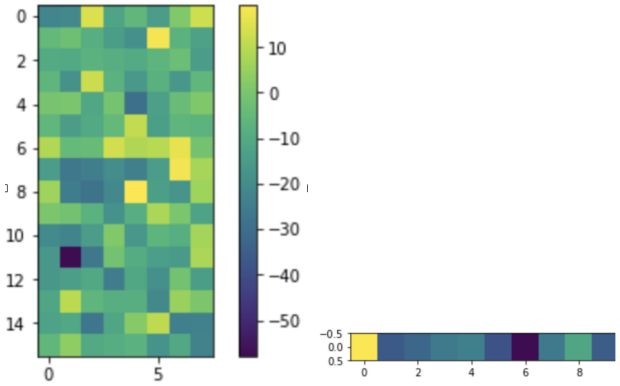

Figure 4 – Activation des neurones des deux couches denses de VoxNet sur un échantillon de ModelNet (un avion, ce qui correspond à label=0)

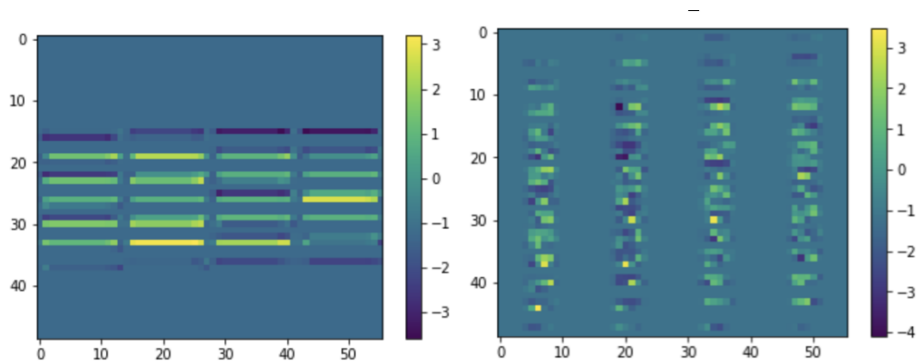

Figure 5 – Résultat de la première couche de convolution 3D. Les tenseurs ont été aplatis.

On remarque des différences fondamentales dans l'interprétation que le réseau de neurones a des objets qu'on lui présente, sans pouvoir en dire davantage cependant.

3.3 Invariance par rotation

Les auteurs prétendent que leur architecture est approximativement invariante par rotation, grâce à l'augmentation des données. Nous avons souhaité tester cette affirmation.

En ordonnée on a la rotation de l'image effectuée : chaque i correspond à une rotation de 360*i/12. On remarque que les valeurs varient peu verticalement (chaque colonne représente un neurone) , ce qui signifie que l'activation d'un neurone quelconque ne dépend pas de l'orientation de l'objet. Ceci corrobore bien l'affirmation des auteurs : L'architecture Voxnet est bien approximativement invariante par rotation !

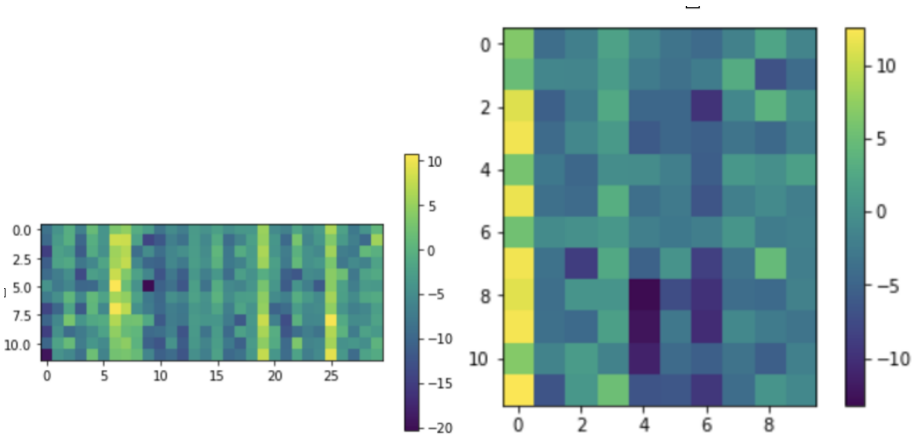

Figure 6 – Activation des neurones de la couche dense 1 et couche dense 2 en fonction de la rotation effectuée pour ModelNet10. Pour inférer, on prend la classe la plus représentée parmi les 12 différentes prédictions. Échantillon : un avion.

4 CONCLUSION ET EXTENSIONS POSSIBLES

L'architecture VoxNet est intéressante puisqu'elle est l'une des premières à employer les réseaux neuronaux convolutifs en trois dimensions afin de classifier des objets. Son implémentation nous a permis de comparer l'influence sur les performances de différents éléments : l'optimiseur, l'augmentation de données.

Outre ces éléments, ce projet nous a permis de comprendre qu'en effet, en deep learning, la partie la plus complexe est dans la gestion des données !

Concernant les améliorations possibles : en s'inspirant de la démarche présentée au cours 9, nous avons essayé de faire une classification des objets utilisant le résultat de la dernière couche, c'est à dire un vecteur de probabilités de l'objet être dans chacune des classes. La démarche est ensuite de séparer spatialement cet ensemble de tenseurs (qui sont dans un espace de dimension le nombre de classes), en réduisant sa dimension à 2 (pour pouvoir le représenter). Malheureusement, les contraintes de mémoire nous ont empêchées de traiter ce point.

À noter que depuis 2015, d'autres architectures bien plus complexes ont dépassé les performances de notre étude. Elles sont regroupées et classées à l'adresse suivante : http ://modelnet.cs.princeton.edu/.