Apprentissage statistique

TP2 – Deep pour les images: ImageNet et au-delà

Olivier Schwander <olivier.schwander@sorbonne-universite.fr>

2023-2024

Exercice 1 - *Introduction aux réseaux convolutifs*

Les réseaux de neurones convolutionnels (CNN) sont devenus les architectures état-de-l'art dans la quasi-totalité des tâches de machine learning appliquées aux images. Ces réseaux diffèrent des réseaux plus classiques par les couches de base utilisées dans leur architecture. On y retrouve souvent deux types de couches différents : la convolution et le pooling.

Couches de convolution

Entrée: Ces couches prennent en entrée un ensemble de D feature maps (c'est-à-dire un tenseur soit l'équivalent en 3 dimensions d'une matrice), chaque feature map étant une matrice de taille $n_x \times n_y$. On a donc une entrée de taille $n_x \times x_y \times D$

Sortie: Sur cette entrée, on applique C convolutions, chacune avec un filtre de convolution de taille $w \times h \times D$, on a généralement $w = h = k$ qu'on appelle *taille du kernel*. Ce sont ces C filtres qui constituent les paramètres que l'on va apprendre. Chaque convolution avec un filtre produit une feature map de sortie, on a donc en sortie un ensemble de feature maps de taille $n'_x \times n'_y \times C$, où n'_x et n'_y dépendent de la façon dont on réalise la convolution.

[Exemples de convolution et de pooling]figures/cnn.png)

Hyperparamètres: En plus du nombre de filtres et de la taille du kernel, il y a deux hyperparmètres courants pour la convolution, le **padding** et le **stride**. Le padding indique combien de rangées de 0 on ajoute autour de l'entrée. Le stride indique de combien de pas on se déplace entre deux calculs de la convolution. La convolution *classique* a une sortie plus petite que l'entrée à cause de la taille du filtre que l'on doit placer à l'intérieur de la feature map d'entrée. Ajouter du padding permet par exemple de retrouver la taille initiale. Ajouter un stride permet de sauter des valeurs, cela correspond à faire du sous-échantillonnage de la sortie.

Références: Une démonstration de la convolution est visible à l'adresse http://cs231n.github.io/assets/conv-demo/index.html

Max Pooling

Principe: Le pooling est une fonction de sous-échantillonnage spatial. Elle prend toujours en entrée des feature maps de taille $n_x \times n_y \times D$, et réduit les deux

premières dimensions spatiales. Le calcul s'effectue selon le même principe qu'une convolution, mais s'applique sur chaque feature map indépendamment. On parcourt donc chaque feature map avec une fenêtre glissante, mais au lieu de réaliser un produit de convolution entre la fenêtre et un filtre, on réalise une opération de pooling sur la fenêtre d'entrée pour produire une valeur. On obtient donc une sortie de taille $n'_x \times n'_y \times D$ avec n'_x et n'_y significativement plus petits que n_x et n_y (souvent d'un facteur 2).

Hyperparamètres : Une couche de pooling a une taille de kernel (définissant la taille de la fenêtre), un stride et du padding qui se comportent comme pour la convolution, et une opération de pooling, les plus courantes étant le *max pooling* (on garde la valeur maximale dans la fenêtre) et l'*average pooling* (on prend la valeur moyenne de la fenêtre). Un pooling très courant est un max pooling avec kernel de taille 2, stride de taille 2 et sans padding.

Exemple d'architecture

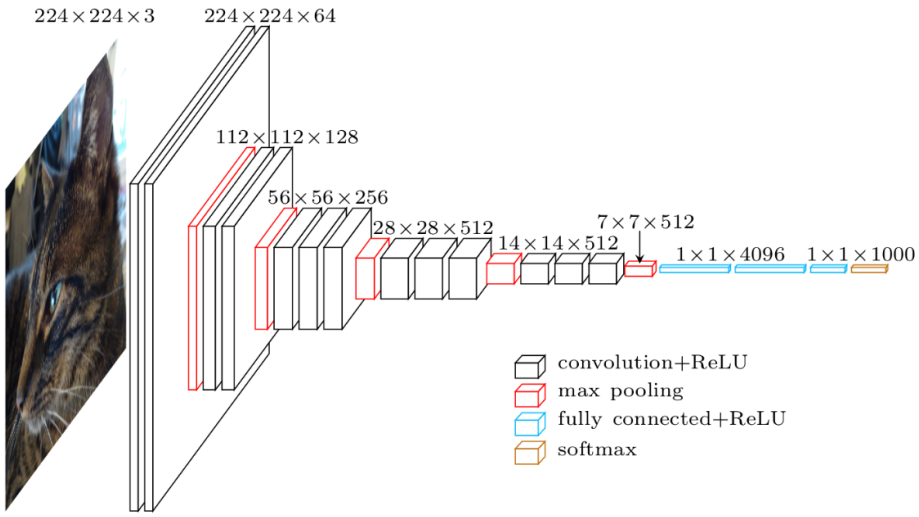

Figure 1: Réseau VGG16

Les réseaux de neurones convolutionnels classiques sont généralement composés d'une succession de couches de convolutions (avec ReLU) avec de plus en plus de filtres, et dont la dimension spatiale est progressivement réduite par des couches de max pooling possiblement jusqu'à aggregation totale des dimensions spatiales, il ne reste donc plus que la *profondeur* correspondant au nombre de filtres appliqués par la dernière convolution ($1 \times 1 \times C$). On y ajoute enfin généralement une ou quelques couches linéaires (appelées *fully-connected*).

Un exemple de ce type d'architecture est le réseau VGG16. Depuis, des architectures plus complexes se sont développées, notamment les architectures Inception, ResNet ou VisionTransformers.

Question 1

Considérant un seul filtre de convolution de padding p, de stride s et de taille de kernel k, pour une entrée de taille $x \times y \times z$, quelle sera la taille de sortie ?

Combien y a-t-il de poids à apprendre ?

Combien de poids aurait-il fallu apprendre si une couche fully-connected devait produire une sortie de la même taille ?

Question 2

Quels avantages apporte la convolution par rapport à des couches *fully-connected* ?

Question 3

On appelle champ récepteur (*receptive field*) d'un neurone l'ensemble des pixels de l'image dont la sortie de ce neurone dépend. Quelles sont les tailles des receptive fields des neurones de la première et de la deuxième couche de convolution ? Pouvez-vous imaginer ce qu'il se passe pour les couches plus profondes ? Comment l'interpréter ?

Question 4

Quel interêt voyez-vous à l'usage du pooling spatial ?

Exercice 2 - *S'adapter à d'autres entrées: réseaux complètement convolutionnels*

Question 5

La plupart des modèles sont entraînés à partir d'ImageNet. Observez quelques exemples de ce dataset, que remarquez-vous ?

Comparez avec quelques images plus naturelles, qu'en concluez-vous ?

Question 6

Supposons qu'on essaye de calculer la sortie d'un réseau convolutionnel classique (par exemple celui en Figure 2) pour une image d'entrée plus grande que la taille initialement prévue (224×224 dans l'exemple). Peut-on (sans modifier l'image) calculer tout ou une partie des couches du réseau ?

Question 7

Montrer que l'on peut voir les couches fully-connected comme des convolutions particulières.

Question 8

Supposons que l'on remplace donc les fully-connected par leur équivalent en convolutions, répondre à nouveau à la question 4. Si on peut calculer la sortie, quelle est sa forme et son intérêt ?

Exercice 3 - *S'adapter à d'autres tâches: fine-tuning*

Les 1000 classes d'ImageNet sont rarement intéressantes pour une application données.

On s'intéresse à une tâche de classification binaire sur des images d'abeilles et de fourmis. La base d'entraînement contient 120 exemples par classe.

Les données sont disponibles à l'adresse: https://download.pytorch.org/tutorial/hymenoptera_data.zip

Question 9

A-t-on une chance de pouvoir entraîner un réseau de neurone efficace sur ces données ?

Question 10

Avec si peu de données, peut-on appliquer une autre méthode de classification ? (SVM, forêt aléatoire, etc)

Question 11

Quelles sont les stratégies qu'on pourrait utiliser pour augmenter la quantité de données disponible ?

Question 12

Comment utiliser un réseau entraîné sur ImageNet pour l'adapter à cette tâche ?

Question 13

Dans quel cas est-il approprié d'utiliser du fine-tuning ?

Question 14

Quels sont les endroits les plus pertinents pour couper le réseau ? Que faire de la partie que l'on garde ?

Question 15

Adaptez une réseau Resnet pré-entraîné pour cette tâche.

Question 16

Adaptez un réseau VGG pré-entraîné pour cette tâche.