# Polytech - Deep Learning - TP 7
# Transfer Learning par extraction de features dans un CNN

Nicolas Thome Rémy Sun Corentin Dancette Matthieu Cord

**Comptes rendus à rendre**

— Un compte-rendu de suivi dans la soirée après la séance
(par mail à `nicolas.thome@sorbonne-universite.fr` et `remy.sun@sorbonne-universite.fr`).
Ce compte rendu contiendra en quelques lignes : (1) les réponses aux questions précédées par ★ ; (2) votre avancement dans le sujet, ce que vous avez réussi ou non ; (3) si vous voulez, des commentaires, remarques sur ce que vous avez compris ou non, etc.

— Un compte-rendu à l'issue du dernier TP. par mail, dont le but est de reprendre au propre le travail effectué pendant ces quatres séances de TP (7, 8, 9, 10), en prenant du recul sur ce qui a été fait.
Ce rapport doit être écrit en francais ou en anglais et faire moins de 10 pages. Ce rapport doit contenir :

- Des parties rédigés et organisées librement (pas forcément en suivant l'ordre des differents TP) dans lesquels vous devez contextualiser les TP, décrire mais aussi discuter les méthodes utilisées pour montrer que vous les avez comprises, et prendre du recul sur ce que vous avez fait : comparer les méthodes, les mettre en lien, discuter leur pertinence, leurs limites, expliquer les visualisations, faire des rapprochements avec d'autres méthodes (vues en cours par exemple), ouvrir à d'autres problématiques,
- Des sections servant à répondre explicitement à toutes les questions présentes dans les sujets, en indiquant clairement le numéro de la question à laquelle vous répondez avec le numéro du TP et le numéro de la question (ex : "Q3.1. L'ensemble de train sert à..."). Faites des réponses claires et synthétiques, allez à l'essentiel (mais ne faites pas des réponses vagues pour autant).

## Objectifs

L'objectif de ce TP est de se familiariser avec l'architecture d'un réseau de convolution de taille significative et couramment utilisé en pratique : VGG16 (**?**). Ensuite une méthode décrite par **?** sera utilisée avec ce réseau sur la base *15 Scene*. On s'intéressera particulièrement à la stratégie visant à extraire des descripteurs *deep* grâce à un réseau pré-entrainé et à les utiliser dans un schéma standard de classification avec des SVM linéaires.

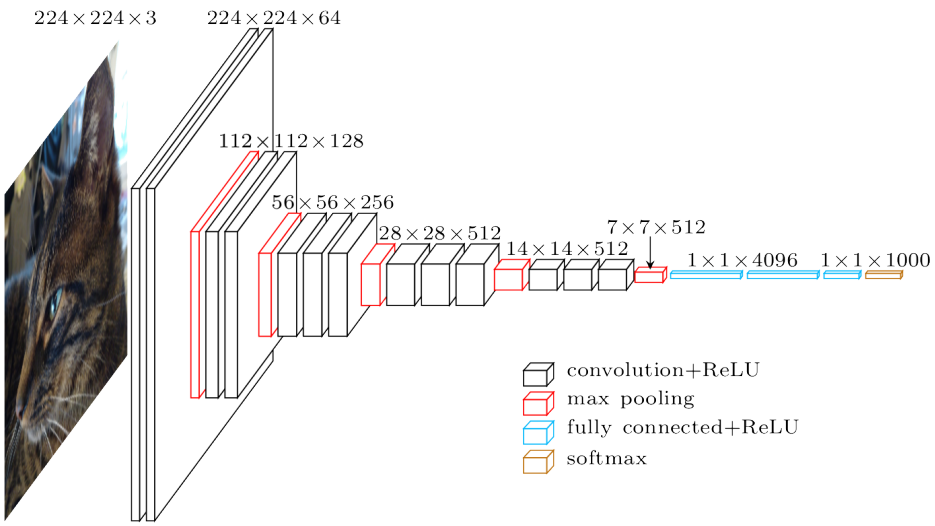


FIGURE 1 – Réseau VGG16

## Partie 1 – Architecture VGG16

Le réseau de convolution VGG16 (?, voir Figure 1) est un modèle de classification appris sur la base d'images ImageNet (?) contenant plus d'un million d'images réparties en 1 000 classes. Son architecture est composée de cinq blocs chacun constitué de :

— deux ou trois couches de convolution avec les même dimensions spatiales ;

— une couche de pooling divisant les dimensions spatiales par deux.

Ensuite le nombre de feature maps est doublé à la couche suivante. La fin du réseau comporte trois couches fully connected.

Le réseau VGG16 pré-appris sur ImageNet peut etre chargé en PyTorch avec la commande :

```
import torchvision
vgg16 = torchvision.models.vgg16(pretrained=True)
```

Les images données en entrée de ce réseau doivent être redimensionnées à la taille $224 \times 224$ pixels, et normalisées avec la moyenne $\mu = [0.485, 0.456, 0.406]$ et l'écart-type $\sigma = [0.229, 0.224, 0.225]$ (calculés sur l'ensemble d'apprentissage d'ImageNet). La liste des noms des 1 000 classes d'ImageNet s'obtient dans un dictionnaire fourni. Le chargement du dictionnaire et d'une image peut être fait en utilisant le code ci-dessous à compléter.

```
import pickle
from PIL import Image
import numpy as np

imagenet_classes = pickle.load(open('imagenet_classes.pkl', 'rb')) # chargement des
↪  classes

img = Image.open("cat.jpg")
img = img.resize((224, 224), Image.BILINEAR)
img = np.array(img, dtype=np.float32) / 255
img = img.transpose((2, 0, 1))
# TODO preprocess image
img = np.expand_dims(img, 0) # transformer en batch contenant une image
```

```
x = torch.Tensor(img)
y = ... # TODO calcul forward
y = y.numpy() # transformation en array numpy
# TODO récupérer la classe prédite et son score de confiance
```

**Questions**

1. ★ Sachant que les couches fully connected comptent la majorité des paramètres du modèle, estimer grossièrement le nombre de paramètres de VGG16 (en utilisant les tailles données sur la Figure 1).
2. ★ Quelle est la taille de sortie de la dernière couche de VGG16 ? À quoi correspond-elle ?
3. **Bonus :** Appliquer le réseau sur plusieurs images de votre choix et commenter les résultats de classification.
4. **Bonus :** Afficher des images correspondant à différentes cartes obtenues après la première convolution. Comment interpréter ces cartes ?

# Partie 2 – Transfer Learning avec VGG16 sur 15 Scene

## 2.1 Principe de la démarche

**Principe général** Le principe de la démarche est assez simple et se compose de deux parties : on utilisera d'abord un réseau de neurones pré-appris pour faire de la *feature extraction* à la suite de laquelle on entraine un modèle de classification.

La première partie peut se voir comme une alternative à l'approche SIFT + Bag of Words (BoW). L'objectif est d'obtenir pour chaque image une représentation vectorielle de l'image décrivant son contenu, que l'on pourra par la suite utiliser pour accomplir diverses tâches, en particulier la classification d'image.

Le principe de l'approche de *feature extraction* est de produire cette représentation des images à partir d'un réseau de neurones convolutionnel appris pour résoudre un problème autre que celui auquel on s'intéresse (par exemple dans notre cas, un réseau appris pour une tâche de classification des images de la base ImageNet). Pour produire la représentation d'une image, on conservera la sortie d'une des couches intermédiaires du réseau de neurones que l'on choisira.

**Principe pour notre TP** Dans le cas de notre TP, nous utiliserons le réseau VGG16, et nous représenterons chaque image par la sortie vectorielle de la couche `relu7` (sortie de la ReLU juste avant la couche de classification). On apprendra ensuite un classifieur SVM pour chaque classe du jeu de données *15 Scene* pour répondre à notre tâche de classification.

**Questions**

5. ★ Pourquoi ne pas directement apprendre VGG16 sur 15 Scene ?
6. ★ En quoi le pré-apprentissage sur ImageNet peut aider à la classification de 15 Scene ?
7. Quelles sont les limites de cette approche par feature extraction ?

## 2.2 Extraction des features de VGG16

Pour extraire des features à la couche *relu7* du réseau VGG16, créez une nouvelle classe `VGG16relu7` où vous copierez les couches de VGG16 jusque la couche *relu7* (voir le code ci-dessous). Les sorties de ce nouveau réseau seront bien les features voulues. Pour extraire des features à une autre couche, il suffira de modifier le champ `classifier` (ou éventuellement `features`) pour supprimer davantage de couches.

Les images en entrée du réseau devront être redimmensionnées à la bonne taille (et avec le bon nombre de canaux !) et être normalisées avec la moyenne et l'écart-type donnés plus haut.

À partir de cette procédure, constituez les matrices $\mathbf{X}_{train}$ et $\mathbf{X}_{test}$ où les features (éventuellement vectorisées) seront stockées par ligne. Chaque vecteur de feature sera également normalisé en norme L2.

**Questions**

8. Quelle est l'influence de la couche à laquelle les features sont extraites ?
9. Les images de 15 Scene sont en noir et blanc, alors que VGG16 attend des images RGB. Comment contourner ce problème ?

```
class VGG16relu7(nn.Module):
    def __init__(self):
        super(VGG16relu7, self).__init__()
        # recopier toute la partie convolutionnelle
```

```
        self.features = nn.Sequential( *list(vgg16.features.children()))
        # garder une partie du classifieur, -2 pour s'arrêter à relu7
        self.classifier = nn.Sequential(*list(vgg16.classifier.children())[:-2])

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

## 2.3 Apprentissage de classifieurs SVM

À partir des *features* $(\mathbf{X}_{train}, \mathbf{X}_{test})$ et des *targets* $(y_{train}, y_{test})$ associées, nous allons apprendre un classifieur multi-classe (constitué de plusieurs classifieurs binaires SVM linéaires en *one-versus-all*) avec ces éléments. Ce travail ayant déjà été réalisé dans un TP précédent, vous utiliserez la classe `sklearn.svm.LinearSVC` de Scikit Learn.

```
from sklearn.svm import LinearSVC
svm = LinearSVC(C=1.0)
```

Entrainez un classifieur SVM multi-classe sur la base d'apprentissage avec $C = 1$ et evaluez son score de classificiation (*accuracy*) sur la base de test en utilisant les *features deep* extraites précédemment. Pour cela, les fonctions `fit` et `score` de `LinearSVC` seront utilisées pour apprendre les SVM et calculer leur précision.

**Questions**

10. Plutôt que d'apprendre un classifieur indépendant, est-il possible de n'utiliser que le réseau de neurones ? Si oui, expliquer comment.

## 2.4 Aller plus loin

Une fois ce schéma de classification opérationnel, vous pourrez étudier l'effet de deux (ou plus) choix sur les performances ou le temps d'apprentissage. Quelques exemples possibles sont listés ci-dessous mais vous êtes libres d'en trouver d'autres !

— Changer la couche à laquelle sont extraites les *features*. Quelle est l'importance de la profondeur de la couche ? Quelle est la taille de la représentation et que cela change-t-il ?

— Essayer d'autre réseaux pré-appris disponibles. Quelles sont les différences entre ces réseaux ?

— Régler la valeur du paramètre $C$ pour améliorer les performances.

— Plutôt que d'apprendre un SVM, remplacer la dernière couche de VGG16 par une nouvelle couche fully connected et continuer l'apprentissage du réseau sur 15 Scene (en propageant ou non les gradients au reste du réseau).

— Étudier des méthodes de réduction de dimension avant la classification et leurs impacts sur les performances et le temps d'exécution.

**Question**

11. Pour chaque amélioration testée, expliquer ses justifications et commenter les résultats obtenus.