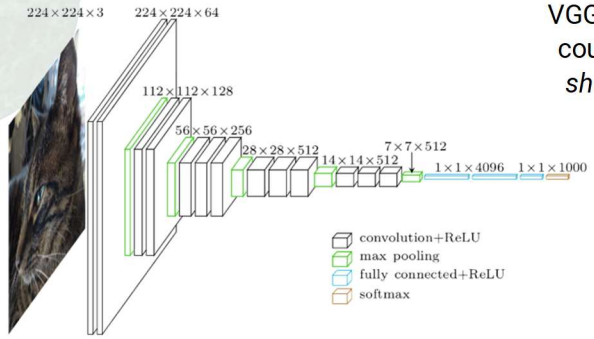
Valentyn Gerushta – ~~3708905~~
Léane Salais – ~~16003610~~
Rapport TME n°7 – 27 janvier

rdfia

transfer learning

On a constaté précédemment comme l'entraînement d'un réseau de neurones était chronophage. Or, si accroître la profondeur améliore la performance des réseaux, cela fait aussi exploser le nombre de paramètres et donc la longueur de l'apprentissage *from scratch*. On voit donc l'intérêt pratique d'une réutilisation de poids déjà appris pour une autre tâche : c'est le *transfer learning*.
Mais le gain de temps n'en est en fait qu'une conséquence, et on en détaillera la mécanique ici.

Architecture VGG16



VGG est un réseau de convolution profond (ici 16 couches), de la génération qui suit les réseaux dits *shallow* vus plus tôt. Il est disponible directement dans `torchvision`, préentraîné sur la base ImageNet à plus d'un million d'images. (Il faut un dataset conséquent pour entraîner un réseau de classification profond et vice-versa.)

← *Figure 1 – Architecture VGG à 16 couches utilisée dans le TME.*

1 Sachant que les couches fully-connected comptent la majorité des paramètres du modèle, estimer grossièrement le nombre de paramètres de VGG16.

Le réseau VGG16 comporte trois couches FC :
la première réceptionne la sortie des couches convolutionnelles, de taille 7*7*512 (soit 25088 pixels) et comporte 4096 neurones, ce qui fait en multipliant un total de 102,760,448 poids à apprendre + 4096 biais ;
la suivante en compte aussi 4096, ce qui fait 16,777,216 poids à apprendre + 4096 biais ;
la dernière en compte 1000, ce qui fait 4,096,000 + 1000 biais.
En sommant, on obtient 123,633,664 poids, soit un ordre de grandeur de 100 millions (cent fois plus que dans le réseau du TME précédent ; le prix d'un gain considérable en expressivité).

2 Quelle est la taille de sortie de la dernière couche de VGG16 ? A quoi correspond-elle ?

Comme décrit sur le schéma, la sortie de la dernière FC passe par une fonction Softmax à mille éléments. **On obtient donc un vecteur de $[0,1]^{1000}$, qui représente une estimation d'une distribution de probabilité sur les mille classes possibles d'ImageNet (cf. TME 4 sur les réseaux de neurones).**

NB : l'implémentation `torchvision` ne contient pas le SoftMax (il doit être ajouté manuellement). En sortie du modèle préentraîné, on a donc des scores de plausibilité qui sont ordonnés, mais non normalisés.

Figure 2 – Images fournies et prédictions →

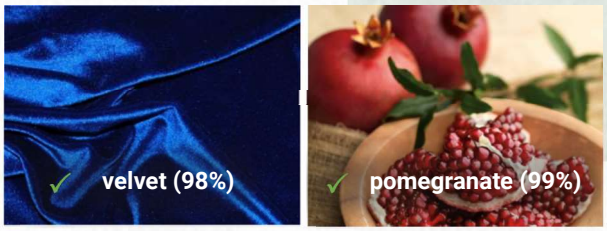
La prédiction de VGG16, en blanc, est juste pour les images fournies (un chat, un chien). Même les classes top5 correspondent à des races plausibles, à l'exception de l'item *carton* pour le chat – lié au beige en bas de l'image.

On note quand même que les prédictions top1 ne sont pas d'une certitude absolue (30,96% pour le chat, 20,77% pour le chien), ce qui est dû au partage des possibilités entre plusieurs races ; après vérification en ligne, on se rend compte d'ailleurs que celle prédite n'est pas forcément la bonne. (On peut donc regretter le design d'ImageNet sans étiquette sémantique globale comme « cat ».)

3 Appliquer le réseau sur des images de votre choix et commenter les résultats de classification.

Nous avons traité huit images avec le réseau, avec des difficultés croissantes (cf. Notebook). Le format de départ importe peu car nous incluons un processus de recadrage – comme déjà pour les images ci-dessus.

Figure 3 (a) – Images type ImageNet et prédictions ↓

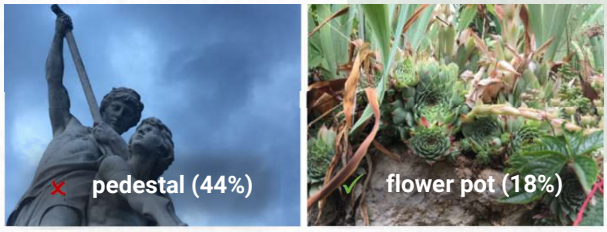


NIVEAU 1 : images type ImageNet

Un tirage aléatoire dans les classes d'ImageNet a renvoyé *velvet* et *pomegranate*. On a fourni des images type de ces deux éléments et le réseau a su les reconnaître avec une confiance maximale. Les classes top5 étaient elles aussi pertinentes malgré leur probabilité négligeable (par exemple, accessoires de mode et ver tropical pour *velvet*).

Ces images nous semblent cependant « faciles », trop propres, avec un cadre trop restreint. Cette artificialité du setting est voulue dans ImageNet car ce n'est qu'un premier pas vers la reconnaissance générale. **Nous avons donc essayé des photographies personnelles pour avoir quelque chose de plus naturel, en prédisant des ratés : nous savons que le modèle n'est pas fait pour ça et nous essayons de le piéger.**

Figure 3 (b) – Images avec classe unique et prédictions ↓



NIVEAU 2 : classe unique

Difficulté théorique sur la statue : les visages. Comme la base d'entraînement n'en contient pas, VGG16 y voit un *piédestal* (la pierre ?) ; en top5, il a aussi repéré la *barre*… et un *chimpanzé* (1%).
Difficulté théorique sur la végétation : distribution sur toute l'image. Là, seul le top2 est acceptable, mais le réseau reconnaît bien *pot de fleurs* et *serre*.

Figure 3 (c) – Images avec contexte et prédictions ↓

NIVEAU 3 : éléments de contexte

Difficulté théorique sur l'oiseau : le fond. Mais le modèle l'a isolé et identifié comme *caille*. Le top5 montre d'autres espèces d'oiseaux – et les *pots*.
Difficulté théorique sur la voiture : partie cachée. Le top5 est bien composé de véhicules, mais du mauvais type et avec des probabilités très faibles.

Figure 3 (d) – Scènes générales et prédictions ↓

NIVEAU 4 : scènes chargées

Difficulté théorique sur la cuisine : fouillis. Mais en top5, le modèle reconnaît bien ce qui s'y trouve (*table à dîner*, *cuisinière*, *cabinet*, *volets*).
Difficulté théorique sur la ville : exotisme, fouillis. Le top5 compte des éléments architecturaux dont *château* : le modèle s'en est sorti en généralisant sur la vue, il n'a pas séparé les objets cette fois.

[A cause de la dernière image, on s'est demandé *si* le modèle était biaisé vers des éléments d'architecture occidentale ; mais on a obtenu la prédiction *porte-avions d'attaque* avec la vue de la tour 15 sur Paris…]

CONCLUSION : VGG16 a eu besoin de ses conditions de train pour présenter une performance convaincante et de bons scores de confiance. Pourtant, pour tous les tests, les éléments apparaissant dans les images étaient toujours présents dans le top5 – même avec une certitude faible – dès lors qu'ils avaient été vus en train. Cela laisse la possibilité de détourner facilement le modèle pour le multiclasses et la segmentation.

Les prédictions de VGG16 sont toujours crédibles malgré sa méconnaissance des visages humains et sa sensibilité aux objets tronqués. Ces défauts peuvent être dus au design d'ImageNet qui ne contient pas ce genre d'obstacles. L'entraîner avec une autre base permettrait de passer outre.

4 Afficher des images correspondant à différentes cartes obtenues après la première convolution. Comment interpréter ces cartes ?

On affiche les cartes associées aux 64 premiers filtres de convolution en noir et blanc (cf. Notebook pour les 64 au complet). Ils n'ont qu'un channel, la notion de couleur n'a donc plus de sens.
Ces cartes servent à analyser les features jugées intéressantes par chaque filtre de convolution appris. Pour les interpréter, il faut noter que plus un pixel est « important » (grand poids), plus il est clair.
Sur un échantillon de quatre (figure 4(a)), on constate comme les filtres de la même couche réagissent différemment au même input, en repérant les contours, les surfaces colorées…
Ici, la première carte ressemble à l'original avec moins de contraste, on semble avoir appris l'identité ; la deuxième souligne des contours sous un certain angle ; la troisième fait ressortir le même angle, mais avec des informations de surface (couleur ?) ; et la dernière met l'accent sur les éléments les plus sombres.

Il est important de remarquer que le traitement est le même quoiqu'on change d'input, vu que ce sont les mêmes poids appris. En Figure 4(b), la réaction des mêmes filtres à l'image du chat est similaire.

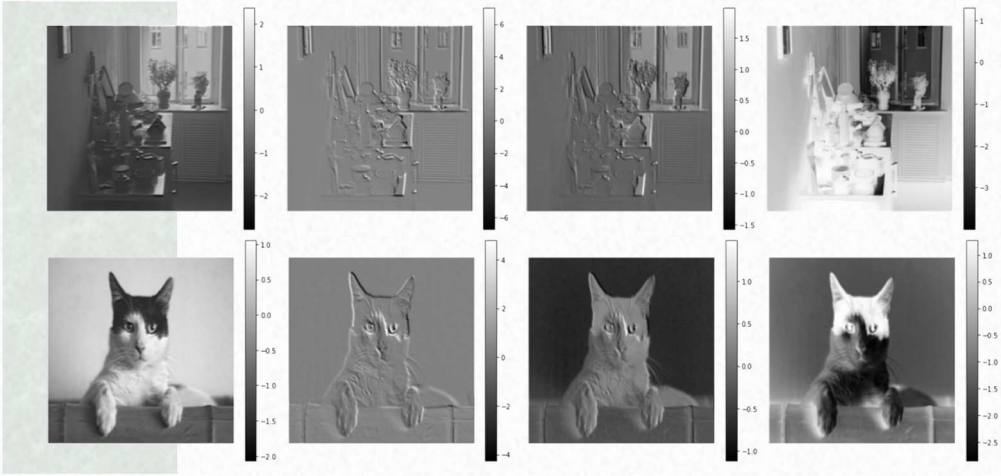


Figure 4 (a) – Feature maps à la sortie de la première convolution

Figure 4 (b) – Même sortie pour une autre image

La prise en compte de l'activation ReLU modifie l'affichage (toujours les quatre mêmes filtres) : les parties sombres (peu importantes) s'assombrissent encore, ce qui fait ressortir les parties claires. En ramenant toute valeur négative à 0, la ReLU augmente le contraste et met en lumière les points d'intérêt à valeurs positives.

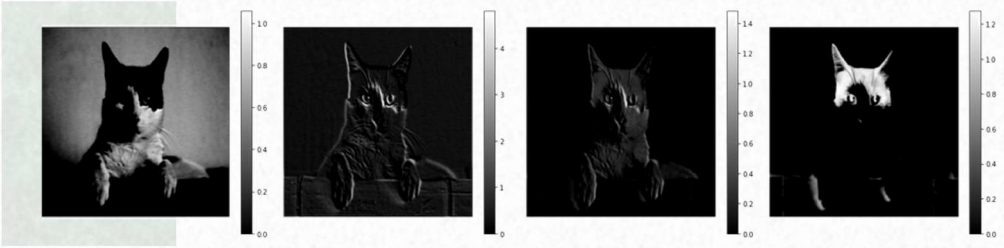


Figure 4 (c) – Impact de l'activation ReLU

Le mécanisme mis en lumière ici se répète à chaque convolution + ReLU du réseau : les filtres isolent des features et l'activation les souligne pour la couche suivante. Cependant, en avançant dans les couches, la dimension de la sortie diminue. On ne conserve donc plus les détails de l'image originale et commenter les cartes de sortie pixellisées devient plus difficile que pour la première. Exemple :

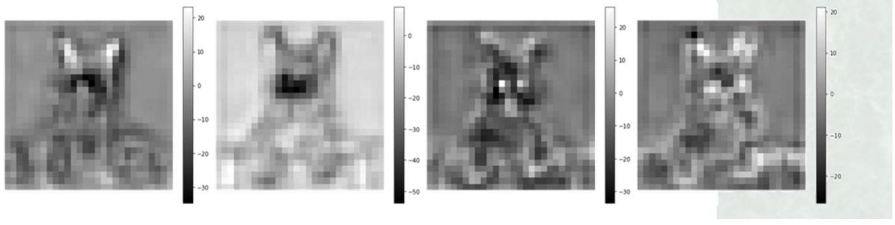


Figure 4 (d) – Feature maps en sortie de douze couches de convolution et poolings

En profondeur, la sensibilité des filtres dépend d'une *combinaison* de caractères appris en amont, laquelle combinaison peut décrire des éléments très spécifiques. Ceux-ci par exemple repèrent bien les oreilles d'un animal ou ses yeux. La pixellisation empêche d'être plus précis dans la description.

Principe du transfer learning

On propose d'utiliser les *features* extraites par VGG16 depuis ImageNet, pour faire fonctionner un réseau sur un autre dataset : par exemple 15Scene, où il n'est plus question d'objets seuls. Comme on l'a vu en Q4, les premières couches d'un réseau apprennent d'abord des *features* très basiques des images (filtres de Gabor) et le concept repose sur l'exploitabilité de ces *features* partout ailleurs.

5 **Pourquoi ne pas directement apprendre VGG16 sur 15Scene ?**

→ **Une valeur codée en dur dans VGG16 empêche l'apprentissage direct : le nombre de classes en sortie. L'architecture classique de VGG16 est prévue pour en reconnaître mille, or il n'y en a que quinze pour 15Scene**. (Par contre, la taille des images ne pose pas problème si on les redimensionne comme plus haut.)
→ Mais adapter ce nombre d'éléments en sortie ne suffirait pas. En fait, **on risque aussi un surapprentissage** dans la mesure où l'échelle de 15Scene, 10000 fois plus petite et à cent fois moins de classes, n'est pas comparable avec celle d'ImageNet. Entraîner une architecture optimisée pour une très grande base dessus serait donc contre-productif pour la généralisation.

6 **En quoi le préapprentissage sur ImageNet peut-il aider à la classification de 15Scene ?**

Tout réseau convolutionnel a une première partie qui peut être vue comme un extracteur de *features*. Certaines de ces *features*, surtout celles extraites par les couches d'entrée, sont valables pour absolument toute représentation par image : des contours, des angles, des valeurs de couleurs.
Reprendre un réseau préentraîné, c'est donc économiser une bonne partie de l'entraînement d'un nouveau réseau. Celui-ci aurait en effet retrouvé des *features* similaires, même dans des données différentes : les lui fournir en avance, c'est gagner du temps et profiter d'un potentiel discriminant qui a déjà fait ses preuves. Selon ce qu'on ajoute comme mécanisme de classification, le nouveau réseau pourra s'adapter au dataset dont il a la charge, et réutiliser les *features* récupérées d'une façon qui sied aux classes à reconnaître.

7 **Quelles sont les limites de cette approche par feature extraction ?**

Au-delà des premières couches, la feature extraction commence à être plus spécifique à un style d'images. Les couches se spécialisent dans la détection de structures complexes qui peuvent être très typiques du dataset d'entraînement. Récupérer des features issues de ces couches profondes pour une tâche sur un dataset trop différent risque donc de compromettre la généralisation et d'induire le modèle en erreur : si l'on veut transférer ces features-là, il faut donc être sûr que les données sont assez similaires.

Extraction des features de VGG16

Mise en œuvre de la stratégie de *transfer learning* décrite.

8 **Quelle est l'influence de la couche à laquelle les features sont extraites ?**

En Q7, on a rappelé que les features reconnues par les couches se complexifiaient avec la profondeur. Il y a donc un juste milieu entre le profit lié à leur réutilisation et la mise en difficulté du modèle du fait d'une trop grande spécificité. On peut choisir de conserver plus ou moins de couches pour équilibrer ces effets.

Si l'on n'en garde que quelques-unes, alors le classifieur qui suit se servira de features très simples ; on aura donc quelque chose de très robuste, mais de peu informatif. Si l'on en conserve davantage, on construira la suite sur une sortie plus complexe. On aura donc un modèle très vulnérable aux variations de style, mais il marchera particulièrement bien sur des images similaires à celles de la première base.

Dans le TP, nous avons même un exemple extrême ; on conserve non seulement toutes les couches de feature extraction, mais on pousse aussi jusqu'aux couches de classification (les FC), en remplaçant seulement la dernière qui fixait le nombre de classes (voir Q5). On commence donc la discrimination des classes *sur les mêmes fondements que pour ImageNet*. Nous verrons en Q11 si cette idée était justifiable.

9 Les images de 15Scene sont en noir et blanc, alors que VGG16 attend des images RGB. Comment contourner ce problème ?

La question de la couleur se règle vite en dupliquant trois fois le seul channel des images noir et blanc. L'information est ainsi conservée, juste redondante, et on obtient le bon encodage. Dans l'autre sens, on aurait pu aplatir l'image sur un seul channel (par moyennage par exemple).

Classifieurs SVM

Comme expliqué en Q6, un classifieur spécifique au nouveau dataset doit de toute façon être créé pour utiliser les features extraites. Par défaut, le sujet de TME nous demande d'utiliser une SVM.

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

Il n'est pas nécessaire de connecter un SVC au réseau. Pour adapter directement VGG16 à 15Scene (Q5), il suffit de retirer la troisième couche FC et de la remplacer par une ou plusieurs autres couches ; la seule condition est que la dernière doit être FC et avoir le bon nombre de classes en sortie (15 au lieu de 1000).

N'avoir ainsi qu'un réseau de neurones permettrait (en contexte d'entraînement) d'effectuer une backprop globale sur toutes ses couches. Si on ne souhaite pas réapprendre la totalité des poids, on peut même freezer ceux des premières grâce aux fonctionnalités de PyTorch.
Pourquoi ne le fait-on pas ? Ici, notre choix des SVC s'explique par celui de ne pas réentraîner la structure. Dans ce cadre, il suffit de paramétrer les SVC avec la constante C pour optimiser la qualité des résultats.

La dernière couche FC de VGG16 est donc remplacée par une SVM linéaire. La précision obtenue par défaut est de **87.2%**. Une fois les features calculées (4096 en sortie de VGG16), le temps d'une passe à travers les données, la réponse du classifieur se fait en moins d'une seconde.

11 Tester des améliorations ; pour chacune, expliquer ses justifications et commenter les résultats.

Voir le tableau page suivante pour un récapitulatif de tous les résultats (précision en test, temps de calcul).

→ **FINE-TUNING DE LA SVM LINEAIRE → important**
On commence par faire varier la constante de régularisation de la SVM sur le modèle implémenté. Cela augmente la précision de 2% pour atteindre **89.2%** quand C=1. C'est le fine-tuning classique, or il n'a pas un impact énorme : le résultat donné par défaut est donc déjà un bon indicateur de la performance finale.

→ **MODIFICATION DU TYPE DE KERNEL DE LA SVM → chronophage**
En essayant des kernels non linéaires, on constate que le gaussien améliore le score : **89.6%** pour C=10. Mais en l'état, la classification est 50 fois plus lente, ce qui rend cet avantage négligeable pour le moment.

→ **PROFONDEUR DU RESEAU PRE-SVM : COUCHES FC → important**
En théorie, la classification sur 15Scene est indépendante de l'extraction de features et n'a pas à se faire sur les mêmes fondements que pour ImageNet. On peut donc penser à réduire les traitements FC communs entre le VGG16 initial et notre nouveau réseau (voir fin de la Q8).
Qu'on retire seulement une couche FC de plus ou bien les deux restantes, on obtient des précisions similaires (**91%** quand C=1), un peu meilleures que pour le modèle initial à C optimale : la décision était donc légitime. Cependant, les temps de traitement changent à cause de la taille de représentation et donnent l'avantage à la première de ces deux configurations. En supprimant une seule couche FC, l'input de la SVM comptera 4096 features comme au départ ; en les supprimant toutes, la SVM devra gérer directement la sortie du dernier max pooling qui est de taille 25088. C'est trop. Sauf à appliquer une réduction de dimensionalité (voir plus bas), on veillera donc à garder au moins une couche FC dans le réseau.

→ PROFONDEUR DU RESEAU PRE-SVM : FEATURE EXTRACTION → à éviter

Quitte à supprimer plus de couches, on a essayé d'empiéter sur les convolutions, l'hypothèse étant que les features issues des couches profondes de VGG16 peuvent être trop adaptées à ImageNet et donc invalides sur 15Scene (cf. Q7). Mais il apparaît qu'il vaut mieux les maintenir toutes car le score obtenu n'est pas bon. Note : conserver le max pooling final a permis de réduire les *features* à traiter (il y en a plus de 100 000 sinon) et d'avoir un temps de calcul acceptable ; mais on n'arrive quand même pas à un score comparable au meilleur score courant. **S'il a besoin des mêmes features, cela veut peut-être dire que notre réseau apprend en fait à classifier les scènes de 15Scene sur la base exacte des objets présents (qui apparaissent dans ImageNet).**

→ REDUCTION DE DIMENSIONALITE → **bonne idée selon le kernel**

Le problème étant souvent le nombre de features, nous avons pensé à ajouter une PCA avant la classification pour réduire le bruit à l'entrée des SVM. En complément de l'essai précédent, conserver 95% de variance expliquée (soit 1256 features sur 25088) n'améliore pas le score mais divise le temps de traitement par 10. Ayant vu plus tôt l'intérêt du kernel gaussien, on a utilisé une PCA pour pouvoir le lancer dans un temps raisonnable, et il a notamment amélioré notre meilleur score courant (en vert dans le tableau).

→ SUPPRESSION DE LA SVM → à éviter

Enfin, comme suggéré par la Q10, nous avons essayé de remplacer la dernière couche de VGG16 par une FC adaptée et de continuer l'apprentissage avec ou sans backpropagation complète.
Si l'on n'apprend que les poids de la couche ajoutée (backpropagation restreinte), il faut quand même 20 epochs pour parvenir au niveau de la toute première SVM linéaire dans le tableau ; et apparemment plus de 50 pour dépasser nos meilleures précisions. C'est plus de 26 minutes sur le même GPU.
Etendre l'apprentissage aux autres couches (backpropagation complète) n'améliore pas particulièrement cette situation. Les scores sont similaires et le temps mis par epoch aussi.
Le choix des SVM dans ce contexte était donc judicieux et nous épargnait un réentraînement fastidieux.

---------- modèle construit ---------- *---------- paramètres ----------* *---------- résultats ----------*

INCLUANT SVM	C optimisée	noyau SVM	PCA	temps SVM	accuracy
Configuration 0 : modèle de départ, VGG16[-1] + SVM	non	linéaire	non	0.8 seconde	87.2%
Configuration 0 : modèle de départ, VGG16[-1] + SVM	oui	linéaire	non	0.8 seconde	89.2%
Configuration 0 : modèle de départ, VGG16[-1] + SVM	oui	sigmoïde	non	37 secondes	89.1%
Configuration 0 : modèle de départ, VGG16[-1] + SVM	oui	gaussien	non	40 secondes	89.6%
Configuration 1 : retrait d'une autre FC avant SVM	oui	linéaire	non	**1 seconde**	**91.0%**
Configuration 1 : retrait d'une autre FC avant SVM	oui	gaussien	oui	**9 secondes**	**91.3%**
Configuration 2 : retrait de toutes les FC avant la SVM	oui	linéaire	non	3.1 secondes	90.9%
Configuration 3 : retrait des FC *et* de deux convolutions	oui	linéaire	non	26 secondes	88.5%
Configuration 4 : configuration 3 + max pooling	oui	linéaire	non	11 secondes	90.1%
Configuration 4 : configuration 3 + max pooling	oui	linéaire	oui	1 seconde	89.9%

FC EN REMPLACEMENT DE LA SVM	learning rate	précision à 50 epochs
Config. 5 : freezing des poids transférés, retrain partiel	0.0005	88.6%
Configuration 6 : pas de freezing, retrain complet	0.0005	88.9%

Figure 5 – Résultats de toutes les expériences

Conclusion

VGG16 sait reconnaître des objets. En récupérant sa *représentation* d'un ensemble d'images, nous avons été capables de construire un système de classification pour des *scènes* qui les contiennent. Le *transfer learning* est en fait un excellent moyen d'obtenir des résultats rapides sur un dataset proche du dataset courant. Ici, on a pu économiser l'entraînement d'un nouveau réseau – en profitant d'un calcul de features adaptatif grâce à un réseau Deep tout prêt, mais en s'en remettant ensuite à la classification simple, rapide et et efficace des modèles de ML basiques.
Cela dit, le fine-tuning d'un réseau transféré se présente sous une forme différente : pour vraiment optimiser les résultats, les possibilités de combinaison sont infinies.