



Guide de l'étudiant

APP2– S7

Réseaux de neurones convolutifs en traitement
d'images

Faculté de génie
Université de Sherbrooke

Hiver 2021

Copyright © 2021, Faculté de génie
Université de Sherbrooke

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Document guide
Rédigé par François Grondin, Jonathan Vincent, Jean-Samuel Lauzon, Marc-Antoine Maheux, Hiver 2021

Copyright © 2021, Faculté de génie, Université de Sherbrooke

Table des matières

1 Activités pédagogiques et compétences	5
2 Synthèse de l'évaluation	5
3 Qualités de l'ingénieur	6
4 Énoncé de la problématique	7
5 Connaissances nouvelles	10
6 Guide de lecture	11
6.1 Références obligatoires	11
6.2 Séquence de lecture suggérée	11
6.2.1 Préparation au 1 ^{er} prodécural	11
6.2.2 Préparation au 1 ^{er} laboratoire	11
6.2.3 Préparation au 2 ^e procédural	11
6.2.4 Préparation au 2 ^e laboratoire	11
7 Logiciels	11
8 Sommaire des activités liées à l'unité	12
9 Productions à remettre	13
9.1 Formation des équipes	13
9.2 Livrables	13
10 Évaluations	15
10.1 Rapport d'APP	15
10.2 Validation	15
10.3 Évaluation sommative	15
10.4 Évaluation finale	15
11 Formation à la pratique procédurale 1	17
11.1 Buts de l'activité	17
11.2 Guide de lecture	17
11.3 Contenu	17
12 Formation à la pratique en laboratoire 1	19
12.1 Buts de l'activité	19
12.2 Guide de lecture	19
12.3 Contenu	19
13 Formation à la pratique procédurale 2	22
13.1 Buts de l'activité	22
13.2 Guide de lecture	22
13.3 Contenu	22
14 Formation à la pratique en laboratoire 2	24
14.1 Buts de l'activité	24
14.2 Guide de lecture	24
14.3 Contenu	24

15 Notes de cours	28
15.1 Tâches en traitement d'images	28
15.1.1 Définitions	28
15.1.2 Métriques	28
15.2 Couches	30
15.2.1 Convolution	30
15.2.2 Convolution transposée	33
15.2.3 Mise en commun maximale	34
15.3 Connexion résiduelle	35
15.4 Champ réceptif	36
15.5 Architectures populaires	36
15.5.1 AlexNet	36
15.5.2 ResNet	36
15.5.3 YOLO	37
15.5.4 U-Net	41

1 Activités pédagogiques et compétences

GRO721: Réseaux de neurones convolutifs en traitement d'images

1. Concevoir et mettre en oeuvre des réseaux de neurones convolutifs.
2. Mettre en oeuvre un réseau convolutif pour une application de traitement d'images.

2 Synthèse de l'évaluation

La note attribuée aux activités pédagogiques de l'APP est une note individuelle, sauf pour la validation et le rapport d'APP qui est une note par équipe de deux. **Tous les membres de l'équipe doivent contribuer à la résolution de la problématique.** En cas de disparité importante dans les contributions, une note individuelle pourrait être attribuée durant la validation. L'évaluation porte sur les compétences figurant dans la description des activités pédagogiques de l'APP à la section 1. Ces compétences, ainsi que la pondération de chacune d'entre elles dans l'évaluation de l'unité, sont :

Activité pédagogique Compétences	GRO721	
	C1	C2
Rapport d'APP et validation	60	60
Évaluation sommative	120	120
Évaluation finale	120	120

Tableau 1: Pondération des points

Le pointage obtenu est ensuite utilisé pour attribuer des cotes selon cette grille :

E	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
<50%	50%	53.5%	57%	60.5%	64%	67.5%	71%	74.5%	78%	81.5%	85%

Tableau 2: Grille de notes selon le pointage

3 Qualités de l'ingénieur

Les qualités de l'ingénieur visées par cette unité d'APP sont les suivantes. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité d'APP.

	Q01	Q02	Q03	Q04	Q05	Q06	Q07	Q08	Q09	Q10	Q11	Q12
Touchée	×	×	×		×							
Évaluée	×	×	×		×							

Tableau 3: Qualités de l'ingénieur visées

Les qualités de l'ingénieur sont les suivantes. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant : <http://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg/>.

Qualité	Libellé
Q01	Connaissances en génie
Q02	Analyse de problèmes
Q03	Investigation
Q04	Conception
Q05	Utilisation d'outils d'ingénierie
Q06	Travail individuel et en équipe
Q07	Communication
Q08	Professionnalisme
Q09	Impact du génie sur la société et l'environnement
Q10	Déontologie et équité
Q11	Économie et gestion de projets
Q12	Apprentissage continu

Tableau 4: Liste des qualités de l'ingénieur

4 Énoncé de la problématique

Vous êtes embauché dans une entreprise qui conçoit des systèmes d'examen par balayage (*scanner*) pour bagages. L'entreprise désire offrir un nouveau logiciel qui permettra d'automatiser le processus d'examen pour faire face à la compétition grandissante dans ce domaine. Votre employeur vous donne le mandat de faire une preuve de concept qui sera basée sur des techniques d'apprentissage profond. Le but est de déterminer s'il est réaliste de développer un logiciel commercial basé sur ce type de techniques dans des délais et avec des fonds limités. La figure 1 illustre un exemple d'image provenant d'un balayeuse.



Figure 1: Exemple d'image provenant d'un balayeuse (*scanner*)

L'ingénieur expert en apprentissage machine de l'entreprise a déjà fait une revue sommaire des différentes techniques et de leurs applications. Il vous indique que pour parvenir à résoudre ce problème, vous devrez utiliser des réseaux de neurones convolutifs. Il vous demande également d'utiliser la bibliothèque logicielle PyTorch pour concevoir et entraîner votre réseau de neurones. Cette bibliothèque logicielle a été choisie puisqu'elle est déjà utilisée pour d'autres projets au sein de l'entreprise. Elle est également facile d'utilisation et adéquate pour du prototypage. Vous utiliserez le module Torchvision de PyTorch qui offre des outils spécialisés pour le traitement d'image.

On vous explique également qu'il vous faut réaliser un programme qui permet de classifier, détecter et segmenter différentes formes dans une image. Puisque ceci est une preuve de concept, on vous demande de fournir une architecture indépendante pour chaque tâche. Ceci vous permettra de vérifier la faisabilité de l'implémentation de chacune de ces techniques dans un projet commercial.

Pour vous aider, on vous donne accès à un ensemble de données dont l'entreprise est propriétaire et qui permet de valider les algorithmes de traitement d'images. Vous avez également accès à une coquille logicielle, qui comprend entre autres un fichier dénommé `dataset.py` qui contient déjà la classe qui vous permettra d'utiliser l'ensemble de données. Cet ensemble de données contient des images contenant des formes simples (cercle, triangle et croix). Les images de l'ensemble de données possèdent les caractéristiques suivantes :

- Un maximum de trois formes par image;
- Un maximum d'une instance de chaque forme par image;

- Images de dimensions 53 pixels par 53 pixels, en niveaux de gris;
- Le niveau de gris est aléatoire et indépendant de chaque forme;
- L'arrière-plan est bruité et tend vers le noir.

La figure 2 illustre un exemple d'image provenant de l'ensemble de données.

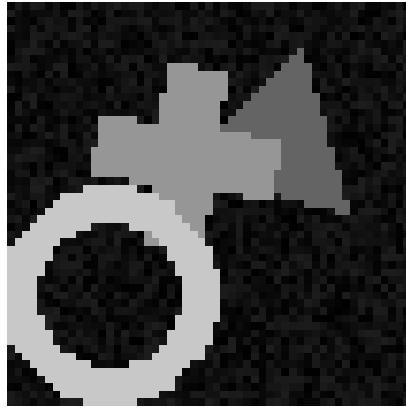


Figure 2: Image avec une instance de chaque forme

Une ébauche du projet vous est fournie. Le fichier `metrics.py` contient des classes permettant d'évaluer la performance de chaque tâche. Le fichier `visualizer.py` effectue l'affichage de la courbe de la fonction de coût de chacun des réseaux et affiche la prédiction d'un réseau sur une image de l'ensemble de test. Le fichier `main.py` permet de lancer l'entraînement des trois réseaux. Ce fichier doit être complété. Vous devez également fournir un fichier par architecture soit `classification_network.py`, `detection_network.py` et `segmentation_network.py`.

Les figures 3, 4 et 5 illustrent des exemples pour les tâches de classification, de détection et de segmentation sémantique.

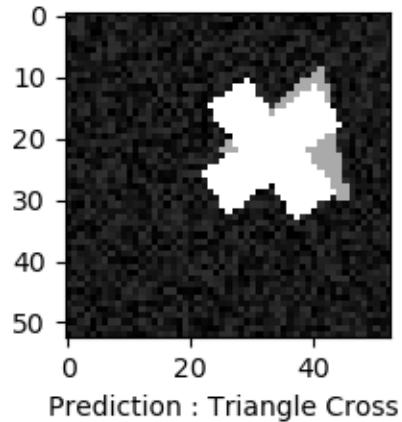


Figure 3: Prédiction avec un réseau de classification

L'algorithme sera éventuellement intégré dans des systèmes limités en termes de mémoire et de capacité de calculs. Il faudra donc également faire attention à la taille de l'architecture du réseau de neurones proposé. Chaque architecture ne devra pas excéder 1 million de paramètres.

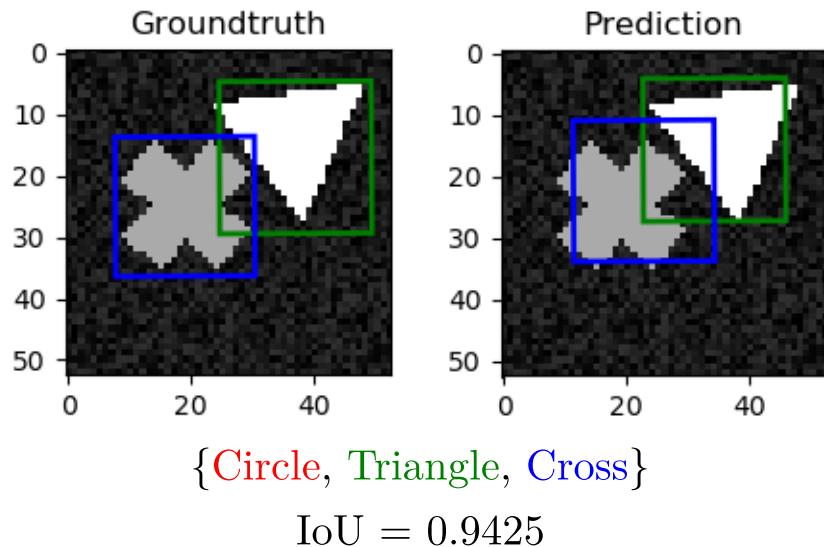


Figure 4: Prédiction avec un réseau de détection d'objets

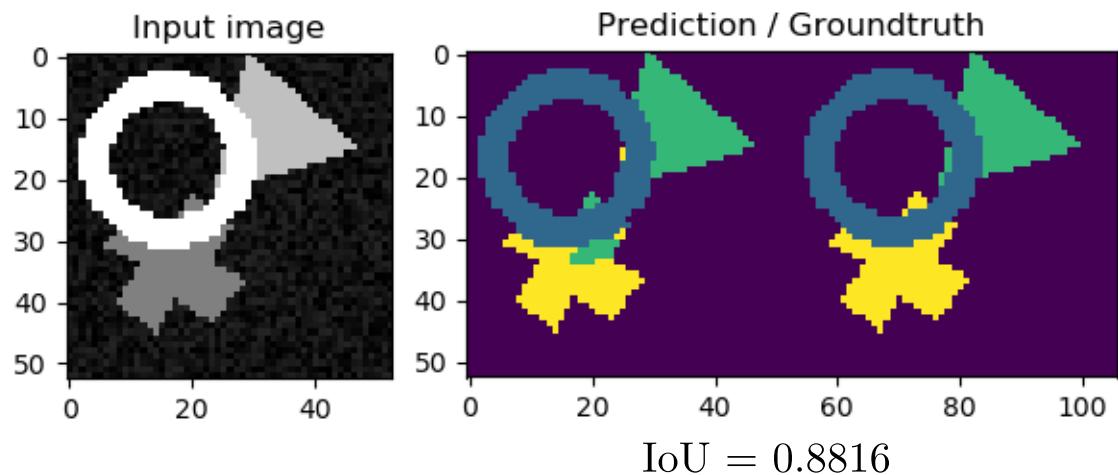


Figure 5: Prédiction avec un réseau de segmentation sémantique d'objets

5 Connaissances nouvelles

Connaissances déclaratives : Quoi

- Approches en traitement d'images : classification, localisation, détection et segmentation sémantique
- Métriques en traitement d'images : précision moyenne et intersection sur union
- Structure d'une couche convulsive : noyau, nombre de filtres, remplissage et foulée
- Couche de sélection du maximum
- Structures de réseaux convolutifs influents en traitement d'images : AlexNet, ResNet, YOLO et U-Net

Connaissances procédurales : Comment

- Définir l'architecture d'un réseau de neurones avec un cadre logiciel
- Charger un ensemble de données grâce à un cadre logiciel
- Calculer automatiquement la propagation des gradients à l'aide d'un cadre logiciel
- Effectuer l'entraînement d'un réseau de neurones avec un processeur dédié
- Effectuer du transfert d'apprentissage à partir de réseaux préentraînés

Connaissances conditionnelles : Quand

- Choisir l'approche appropriée (classification, localisation, détection et segmentation sémantique) pour effectuer une tâche en traitement d'images
- Choisir quelle fonction de coût utiliser pour chaque tâche en traitement d'images

6 Guide de lecture

6.1 Références obligatoires

Voici les références qui sera à l'étude pour cet APP :

The Hundred-Page Machine Learning Book de Andriy Burkov.

Ce livre est disponible en version française, et a été utilisé pour GRO-720 et le sera pour GRO-722.

Des notes de cours sont également disponibles à la fin de ce guide à la section 15 et des ressources en ligne seront suggérées dans la séquence de lecture.

6.2 Séquence de lecture suggérée

6.2.1 Préparation au 1^{er} procédural

- Livre – Chapitre 6 : Réseaux de neurones et apprentissage profond, pp. 69–81
- Notes – Section 15.1 : Tâches en traitement d’images
- Notes – Section 15.2 : Couches

6.2.2 Préparation au 1^{er} laboratoire

- Ressource en ligne : https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
- Ressource en ligne : https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html
- Ressource en ligne : https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html
- Ressource en ligne : https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
- Ressource en ligne : https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

6.2.3 Préparation au 2^e procédural

- Notes – Section 15.3 : Connexions résiduelles
- Notes – Section 15.5 : Architectures populaires

6.2.4 Préparation au 2^e laboratoire

- Notes – Section 15.5 : Architectures populaires

7 Logiciels

Pour cet APP, vous aurez besoin d’installer l’environnement de Python 3, disponible gratuitement en ligne pour tous les systèmes d’exploitation (www.python.org). Il est également suggéré d’utiliser le gestionnaire de progiciel Pip pour installer Pytorch (<https://pytorch.org/>). Il est recommandé d’utiliser l’IDE PyCharm (<https://www.jetbrains.com/fr-fr/pycharm/>).

8 Sommaire des activités liées à l’unité

- Tutorat d’ouverture
- Études personnelles et préparation à la première activité procédurale
- Activité procédurale 1
- Activité en laboratoire 1
- Activité procédurale 2
- Activité en laboratoire 2
- Validation de la problématique
- Tutorat de fermeture
- Évaluation sommative

9 Productions à remettre

- Validation pratique de la solution en laboratoire. Vous devrez présenter votre solution en équipe lors de cette rencontre. **La présence des deux membres de l'équipe est obligatoire.** L'équipe professorale vous présentera un sous-ensemble de test et vous devrez utiliser vos réseaux de neurones entraînés pour effectuer les tâches de reconnaissances de la problématique.
- Rapport d'APP à remettre avant 9h00 le jour du deuxième tutorat. Les consignes de réaction de l'unité d'APP sont données à la section 9.2.

Tout retard sur la remise de livrable entraîne une pénalité de 20% par jour.

9.1 Formation des équipes

La taille des équipes pour l'APP est fixée à 2. Aucune exception ne sera accordée. Veuillez utiliser le formulaire disponible sur le site de la session pour inscrire votre équipe.

9.2 Livrables

Votre solution à la problématique sera évaluée lors d'une séance de validation, mais surtout par un rapport qui décrira vos choix technologiques et le code que vous aurez développé. On attend donc une paire de livrables (rapport et code) par équipe de 2 étudiants.

Rapport

Le rapport, de 10 pages **au maximum**, devra contenir :

- Une page de présentation
- Une introduction pour expliquer la problématique
- Une description de chaque tâche à accomplir (classification, détection et segmentation sémantique)
- Une description des modules pour charger les données d'entraînement et de validation
 - Une description du rôle des méthodes `_len_` et `_getitem_` de la classe `ConveyorSimulator`
 - Une explication portant sur la manière d'implémenter la classe `ConveyorSimulator`
 - Une explication de haut niveau de la classe `torch.utils.data.DataLoader` en expliquant l'effet des paramètres `shuffle`, `batch_size` et `num_workers`
- Pour chaque réseau de neurones :
 - Une justification de l'architecture choisie
 - Un schéma-bloc des différentes couches du réseau
 - Le nombre total de paramètres à entraîner dans le réseau
 - Une description des hyperparamètres utilisés pour l'entraînement : fonction de coût, taux d'apprentissage, taille des lots, etc.
 - Un graphique de l'erreur en fonction des époques pour les sous-ensembles d'entraînement et de validation
 - Un graphique de la précision moyenne ou de l'intersection sur union, selon la tâche du réseau, en fonction des époques pour les sous-ensembles d'entraînement et de validation
 - La valeur de la précision moyenne ou de l'intersection sur union, selon la tâche du réseau, pour l'ensemble de test
 - Un exemple de prédiction (image) avec le sous-ensemble de test pour chaque tâche
- Une conclusion qui fait un retour sur les performances obtenues avec les architectures proposées

Fichiers de code

Vous devez également remettre le code complet de votre solution. Vous pouvez inclure seulement les fichiers que vous avez modifiés (ou ajoutés) aux fichiers fournis pour l'APP. Votre code devra être commenté et organisé clairement.

Procédure de dépôt

Vous devrez combiner en une seule archive ZIP votre rapport **en format PDF** et vos fichiers de code pour le dépôt. Le lien vers la plateforme de dépôt sera disponible sur le site web de la session.

10 Évaluations

10.1 Rapport d'APP

Le contenu du rapport sera évalué selon cette pondération :

Activité pédagogique Compétences	GRO721	
	C1	C2
Rapport	40	40
Description de chaque tâche à accomplir	10	–
Modules de chargement des données	10	–
Justification de l'architecture de chaque réseau	10	–
Schéma-bloc de chaque réseau, incluant le nombre de paramètres	10	–
Description des hyperparamètres	–	10
Graphique de la fonction de coût en fonction des époques	–	10
Graphique de la précision moyenne/intersection sur union en fonction des époques	–	10
Exemples de résultats avec le sous-ensemble de test	–	10
Validation	20	20
Total	60	60

Tableau 5: Pondération des points

10.2 Validation

Votre solution sera également validée en personne lors de la séance de validation. Elle sera évaluée de façon critériée. Vous n'avez rien à ajouter au rapport en lien avec la validation, mais le pointage résultant de l'évaluation y sera ajouté lors de la correction. Le tableau 6 présente la grille d'évaluation pour la validation.

10.3 Évaluation sommative

L'évaluation sommative porte sur tous les objectifs d'apprentissage de l'unité. C'est un examen théorique qui se fera **sans documentation**. Des informations pertinentes seront ajoutées en annexe à l'examen.

10.4 Évaluation finale

L'évaluation finale se fera par activité pédagogique et portera sur tous les objectifs d'apprentissage de cette activité pédagogique. C'est un examen théorique qui se fera **sans documentation**. Des informations pertinentes seront ajoutées en annexe à l'examen.

Qualité	Q01	Q02	Q03	Q05
Compétence	C1	C2	C2	C1
Pondération	10	10	10	10
Excellent (4) – 100%	Connaît adéquatement les architectures de réseaux de neurones convolutifs	Applique efficacement la procédure d'entraînement de réseaux de neurones convolutifs et analyse parfaitement les résultats	Investigue parfaitement l'impact des hyperparamètres sur les performances du réseau	Les modules pour constituer le réseau de neurones sont parfaitement implémentés
Cible (3) – 85%	Connaît la plupart des architectures de réseaux de neurones convolutifs	Applique la procédure d'entraînement de réseaux de neurones convolutifs et analyse correctement les résultats	Investigue correctement l'impact des hyperparamètres sur les performances du réseau	Les modules pour constituer le réseau de neurones sont correctement implémentés
Seuil (2) – 60%	Connaît les architectures essentielles des réseaux de neurones convolutifs	Applique en bonne partie la procédure d'entraînement de réseaux de neurones convolutifs et analyse les résultats avec quelques erreurs mineures	Investigue partiellement l'impact des hyperparamètres sur les performances du réseau avec des erreurs mineures	Les modules pour constituer le réseau de neurones sont partiellement implémentés (les fonctionnalités essentielles sont présentes)
Insuffisant (1) – 25%	Connaît superficiellement les architectures de réseaux de neurones convolutifs	Applique minimalement la procédure d'entraînement de réseaux de neurones et analyse les résultats avec plusieurs erreurs importantes	Investigue minimalement l'impact des hyperparamètres sur les performances du réseau avec des erreurs majeures	Les modules pour constituer le réseau de neurones sont partiellement implémentés (les fonctionnalités essentielles sont absentes)
Non initié (0) – 0%	Ne connaît pas les architectures des réseaux de neurones convolutifs	N'applique pas la procédure d'entraînement de réseaux de neurones convolutifs et n'est pas en mesure d'analyser les résultats	N'investigue pas l'impact des hyperparamètres sur les performances du réseau	Les modules pour constituer le réseau de neurones ne sont pas implémentés

Tableau 6: Grille de validation

11 Formation à la pratique procédurale 1

11.1 Buts de l'activité

- Identifier les tâches de traitement d'images.
- Se familiariser avec l'architecture d'un réseau de neurones convolutif.

11.2 Guide de lecture

The Hundred-Page Machine Learning Book

- Chapitre 6 : Réseaux de neurones et apprentissage profond, pp. 69–81

Notes de cours

- Section 15.1 : Tâches en traitement d'images
- Section 15.2 : Couches

11.3 Contenu

Question 1 – Un premier réseau de neurones est entraîné pour effectuer de la classification d'images avec trois classes. Les courbes de précision et rappel sont illustrées à la figure 6. Calculez la précision moyenne (mAP) pour ce réseau.

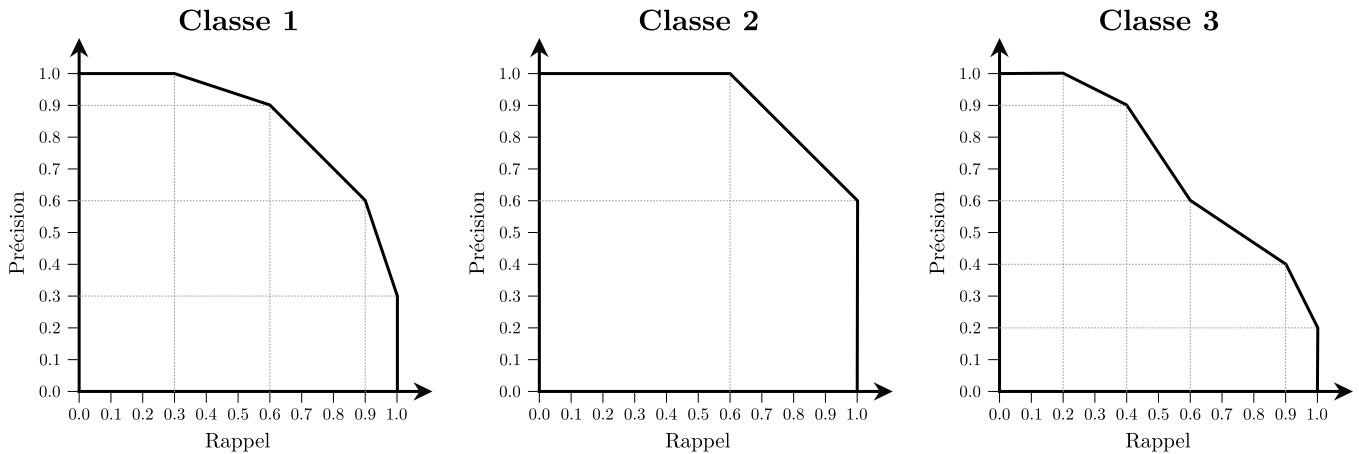


Figure 6: Courbes de précision et rappel

Question 2 – Supposons que nous avons une image d'une hauteur de 10 pixels par une largeur de 10 pixels, avec 1 canal (donc un tenseur de $1 \times 10 \times 10$). Chaque valeur représente un niveau de gris compris dans l'intervalle $[0, 1]$ (0 pour représenter le noir, et 1 le blanc). On peut donc représenter une image d'un carré par la matrice suivante :

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

Les noyaux pour un réseau convolutif sont normalement appris durant la phase d'entraînement. Assumons ici que nous avons les deux noyaux suivants et que nous n'avons pas de biais dans notre couche convulsive:

1. Calculez le résultat pour un balayage avec aucun remplissage et une foulée de 1 pour les deux noyaux $1 \times 3 \times 3$ suivants sur l'image (1) :

$$\begin{bmatrix} -1 & -1 & -1 \\ +1 & +1 & +1 \\ -1 & -1 & -1 \end{bmatrix} \quad (2)$$

$$\begin{bmatrix} -1 & +1 & -1 \\ -1 & +1 & -1 \\ -1 & +1 & -1 \end{bmatrix} \quad (3)$$

2. Appliquez une couche de maximum avec un noyau 2×2 et une foulée de 2 sur le résultat du numéro 1.
3. Calculez le résultat pour un balayage avec aucun remplissage et une foulé de 1 pour les quatre noyaux $2 \times 2 \times 2$ suivants sur le résultat du numéro 2 :

$$\left[\begin{bmatrix} +0 & +1 \\ +0 & +0 \end{bmatrix}, \begin{bmatrix} +0 & +0 \\ +1 & +0 \end{bmatrix} \right] \quad (4)$$

$$\left[\begin{bmatrix} +0 & +0 \\ +1 & +0 \end{bmatrix}, \begin{bmatrix} +0 & +1 \\ +0 & +0 \end{bmatrix} \right] \quad (5)$$

$$\left[\begin{bmatrix} +1 & +0 \\ +0 & +0 \end{bmatrix}, \begin{bmatrix} +0 & +0 \\ +0 & +1 \end{bmatrix} \right] \quad (6)$$

$$\left[\begin{bmatrix} +0 & +0 \\ +0 & +1 \end{bmatrix}, \begin{bmatrix} +1 & +0 \\ +0 & +0 \end{bmatrix} \right] \quad (7)$$

4. En observant les résultats, que permettent d'accomplir ces opérations?

Question 3 – Pour les couches convolutives suivantes et les tenseurs d'entrée, déterminez quelle sera la taille du tenseur de sortie. Indiquez également le nombre de paramètres.

1. Tenseur d'entrée : $3 \times 12 \times 7$, 5 noyaux de dimensions $3 \times 3 \times 4$, foulée de 1 et aucun remplissage.
2. Tenseur d'entrée : $5 \times 5 \times 5$, 3 noyaux de dimensions $5 \times 4 \times 4$, foulée de 1 et un remplissage de 3.
3. Tenseur d'entrée : $4 \times 20 \times 20$, 10 noyaux de dimensions $4 \times 2 \times 2$, foulée de 2 et un remplissage de 1.

Question 4 – Nous avons un tenseur d'entrée de $10 \times 1 \times 1$ et 40 noyaux de dimensions $10 \times 1 \times 1$. Quel lien peut-on faire avec une couche entièrement connectée?

12 Formation à la pratique en laboratoire 1

12.1 Buts de l'activité

- Se familiariser avec la bibliothèque logicielle PyTorch (définition de modèles, chargement d'ensemble de données, entraînements, etc.).

12.2 Guide de lecture

Aide de la bibliothèque logicielle PyTorch

- https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
- https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html
- https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html
- https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

12.3 Contenu

Dans ce laboratoire, vous travaillerez avec l'ensemble de données de chiffres écrit à la main MNIST. Vous comparerez différentes architectures pour accomplir une tâche de classification. Le dossier distribution contient les fichiers `main.py` et `models/net.py` qui devront être complétés.

Question 1 – Veuillez compléter l'initialisation des objets gérant les ensembles de données en effectuant les étapes suivantes :

- Créez les instances de la classe `datasets.MNIST` pour que les données se trouvent dans le sous-dossier `data/` (variable `data_path`).
- Séparez l'ensemble d'entraînement de MNIST en un ensemble d'entraînement et un autre de validation avec `torch.utils.data.random_split` selon les variables `n_train_samples` et `n_val_samples`.

Ceci devrait s'afficher lors de l'exécution du script après les modifications :

```
Number of training samples : 42000
Number of validation samples : 18000
Number of test samples : 10000
```

Afin de visualiser une image de l'ensemble de validation, vous pouvez ajouter les lignes suivantes après les lignes modifiées :

```
img, label = dataset_val[0]
plt.imshow(img[0, :, :].cpu().numpy(), cmap='gray')
plt.title(str(label))
plt.show()
```

Les références suivantes vous seront utiles :

- <https://pytorch.org/docs/stable/torchvision/datasets.html#mnist>
- https://pytorch.org/docs/stable/data.html#torch.utils.data.random_split

Question 2 – Créez les instances de la classe `torch.utils.data.DataLoader` correspondants à chaque ensemble de données instancié à la question 1. Vous devez utiliser les variables `batch_size` et `num_workers`.

Quelle est la fonction de cette classe?

Pour tester vos modifications, vous pouvez ajouter les lignes suivantes afin de visualiser la taille des tenseurs de chaque lot.

```
img, label = next(iter(train_loader))
print('image tensor shape: ', img.shape)
print('label tensor shape: ', label.shape)
```

La référence suivante vous sera utile :

- <https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader>

Question 3 – Écrivez le code permettant d'effectuer la passe avant du réseau de neurones lors de l'entraînement. Vous devez utiliser les variables `model`, `loss_criterion` et `optimizer`. Vous devez accumuler la valeur de la fonction de coût dans la variable `running_loss`.

Pour tester le code modifié, vous devez mettre la variable `train` égale à `True`.

Les références suivantes vous seront utiles :

- https://pytorch.org/tutorials/beginner/examples_nn/polynomial_optim.html
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Question 4 – Écrivez le code permettant d'effectuer la passe avant du réseau de neurones lors de la validation et du test. Vous devez utiliser les variables `model` et `loss_criterion`. Vous devez accumuler la valeur de la fonction de coût pour la validation dans la variable `val_loss`. Vous devez accumuler la valeur de la fonction de coût pour le test dans la variable `test_loss`. Vous devez calculer la justesse et l'assigner dans la variable `accuracy`.

Pour tester le code modifié, vous devez mettre la variable `test` égale à `True`.

Les références suivantes vous seront utiles :

- https://pytorch.org/tutorials/beginner/examples_nn/polynomial_optim.html
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Question 5 – Modifiez le réseau défini dans le fichier `models/net.py` pour que le calcul de la couche pleinement connectée s'effectue à l'aide la classe `torch.nn.Conv2d`.

Comparez les résultats.

Les références suivantes vous seront utiles :

- <https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear>
- <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d>

Question 6 – Modifiez le réseau défini dans le fichier `models/net.py` pour qu'il corresponde à l'architecture du tableau 7.

#	Couche	Paramètres
1	Convolution	4 noyaux de 3×3 avec un remplissage de 1 et une foulée de 1
2	Normalisation de lot	
3	ReLU	
4	Mise en commun maximale	Noyau de 2×2 avec foulée de 2
5	Convolution	2 noyaux de 3×3 avec un remplissage de 1 une foulée de 1
6	Normalisation de lot	
7	ReLU	
8	Mise en commun maximale	Noyau de 2×2 avec foulée de 2
9	Pleinement connectée	À déterminer

Tableau 7: Architecture du réseau

Les références suivantes vous seront utiles :

- <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d>
- <https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html#torch.nn.BatchNorm2d>
- <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html#torch.nn.ReLU>
- <https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.relu>
- <https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d>
- https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.max_pool2d

13 Formation à la pratique procédurale 2

13.1 Buts de l'activité

- Se familiariser avec les architectures populaires en traitement d'images (AlexNet, ResNet, YOLO et U-Net).
- Comprendre comment appliquer ces architectures à des problèmes en traitement d'images.

13.2 Guide de lecture

Notes de cours

- Section 15.3 : Connexions résiduelles
- Section 15.5 : Architectures populaires

13.3 Contenu

Question 1 – Le réseau AlexNet permet d'effectuer une classification d'images. Quel est le rôle des couches convolutives qui reçoivent l'image en entrée? Quel est le rôle des couches entièrement connectées?

Question 2 – Vous avez entraîné pendant plusieurs jours le réseau AlexNet avec l'ensemble de données ImageNet qui contient plus de 14 millions d'images. Vous aimerez maintenant utiliser ce réseau et y ajouter deux nouvelles classes (chaque classe contient une centaine d'images). Y a-t-il une façon de procéder sans relancer un entraînement complet qui nécessitera plusieurs jours de calculs?

Question 3 – Le réseau YOLO est utilisé pour détecter un objet dans une image, et les paramètres prédits pour un élément de la grille (2^e colonne, 6^e rangée) sont $x = 0.6$, $y = 0.5$, $w = 0.3$ et $h = 0.4$. L'image possède une taille de 448×448 pixels. La figure 7 illustre cette configuration. Calculez les coordonnées du coin supérieur gauche et inférieur droit de la boîte englobante, en assumant que le premier pixel dans le coin supérieur gauche de l'image correspond aux coordonnées $(0, 0)$.

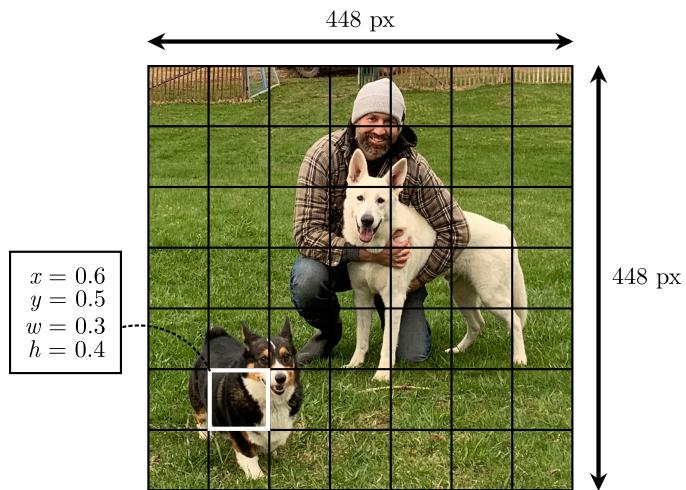


Figure 7: Prédiction sur la grille générée par YOLO

Question 4 – Le réseau U-Net est typiquement utilisé pour effectuer une tâche de segmentation sémantique. Dans les notes de cours, il est question de générer un masque binaire. Comment pourrait être modifié le réseau si nous désirons faire de la segmentation sémantique pour les classes suivantes : chien, chat, humain et voiture, en supposant que chaque pixel est assigné à une seule classe? De plus, qu'arrive-t-il si nous supprimons les connexions entre l'encodeur et le décodeur qui mènent aux blocs de concaténation?

Question 5 – Soit une connexion résiduelle telle qu'illustrée à la figure 8. Calculez la propagation du gradient à partie de la sortie vers l'entrée. En quoi est-ce que cette structure est avantageuse?

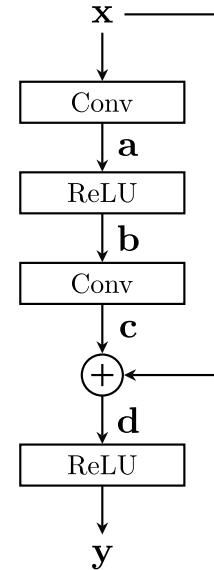


Figure 8: Connexion résiduelle

Vous pouvez assumer que les couches sont modélisées par les fonctions suivantes, et que nous connaissons leurs dérivées: $\mathbf{a} = h_1(\mathbf{x})$, $\mathbf{b} = h_2(\mathbf{a})$, $\mathbf{c} = h_3(\mathbf{b})$, $\mathbf{d} = \mathbf{c} + \mathbf{x}$, $\mathbf{y} = h_4(\mathbf{d})$.

14 Formation à la pratique en laboratoire 2

14.1 Buts de l'activité

- Se familiariser avec les bases de données, l'apprentissage par transfert et différentes architectures de réseaux de neurones convolutifs.

14.2 Guide de lecture

Notes de cours

- Section 15.3 : Connexions résiduelles
- Section 15.5 : Architectures populaires

14.3 Contenu

Dans ce laboratoire, vous allez travailler avec l'ensemble de données PASCAL VOC pour faire de l'apprentissage par transfert avec un réseau ResNet18 afin d'accomplir une tâche de classification d'objets multiples. Vous allez également programmer une architecture U-Net pour utiliser des poids préentraînés sur la base de données COCO pour accomplir une tâche de segmentation sémantique.

Les fichiers `transfer_learning_resnet.py`, `voc_classification_dataset.py` et `models/unet.py` devront être modifiés.

Question 1 – Veuillez compléter le fichier `voc_classification_dataset.py` pour prétraiter la cible.

La cible doit être un vecteur qui a autant d'éléments que le nombre de classes de PASCAL VOC. Vous devez assigner les bonnes valeurs à la variable `multi_hot`. L'élément i de ce vecteur est égal à 1 si un objet de la classe i est présent dans l'image, sinon il est égal à 0. L'attribut `self.VOC_CLASSES_2_ID` permet d'obtenir l'index d'une classe à partir de son nom. Les métadonnées de l'image courante sont dans la variable `target_metadata`. Cette variable est un dictionnaire ayant le format suivant :

```
{  
    "annotation": {  
        "filename": "2008_001329.jpg",  
        "folder": "VOC2012",  
        "object": [  
            {  
                "bndbox": {  
                    "xmax": "342",  
                    "xmin": "123",  
                    "ymax": "389",  
                    "ymin": "194"  
                },  
                "difficult": "0",  
                "name": "car",  
                "occluded": "0",  
                "pose": "Frontal",  
                "truncated": "0"  
            },  
            {  
                "bndbox": {  
                    "xmax": "160",  
                    "xmin": "115",  
                    "ymax": "286",  
                    "ymin": "184"  
                },  
                "difficult": "0",  
                "name": "person",  
                "occluded": "0",  
                "pose": "Frontal",  
                "truncated": "1"  
            }  
,  
        "segmented": "0",  
        "size": {  
            "depth": "3",  
            "height": "500",  
            "width": "375"  
        },  
    },  
}
```

```

        "source": {
            "annotation": "PASCAL_VOC2008",
            "database": "The_VOC2008_Database",
            "image": "flickr"
        }
    }
}

```

Pour tester les modifications, exécutez le fichier `voc_classification_dataset.py`. Vous devriez obtenir un résultat ressemblant à la figure 9. Les classes des objets présents dans l'image sont affichées au-dessus de la figure.



Figure 9: Classification multiobjets

La référence suivante vous sera utile :

- <https://pytorch.org/vision/stable/datasets.html#torchvision.datasets.VOCDetection>

Question 2 – Instanciez et configurez un réseau de type ResNet18 dans le fichier `transfer_learning_resnet.py` pour l'utiliser en apprentissage par transfert. Les paramètres du réseau de type ResNet18 doivent être figés, sauf pour la dernière couche. Vous devez aussi ajouter une couche pleinement connectée et les fonctions d'activation manquantes pour effectuer la classification multiclasse.

Pour tester les modifications, exécutez le fichier `transfer_learning_resnet.py` sur une ou deux époques. Vous devriez obtenir une figure ressemblant à la figure 10.

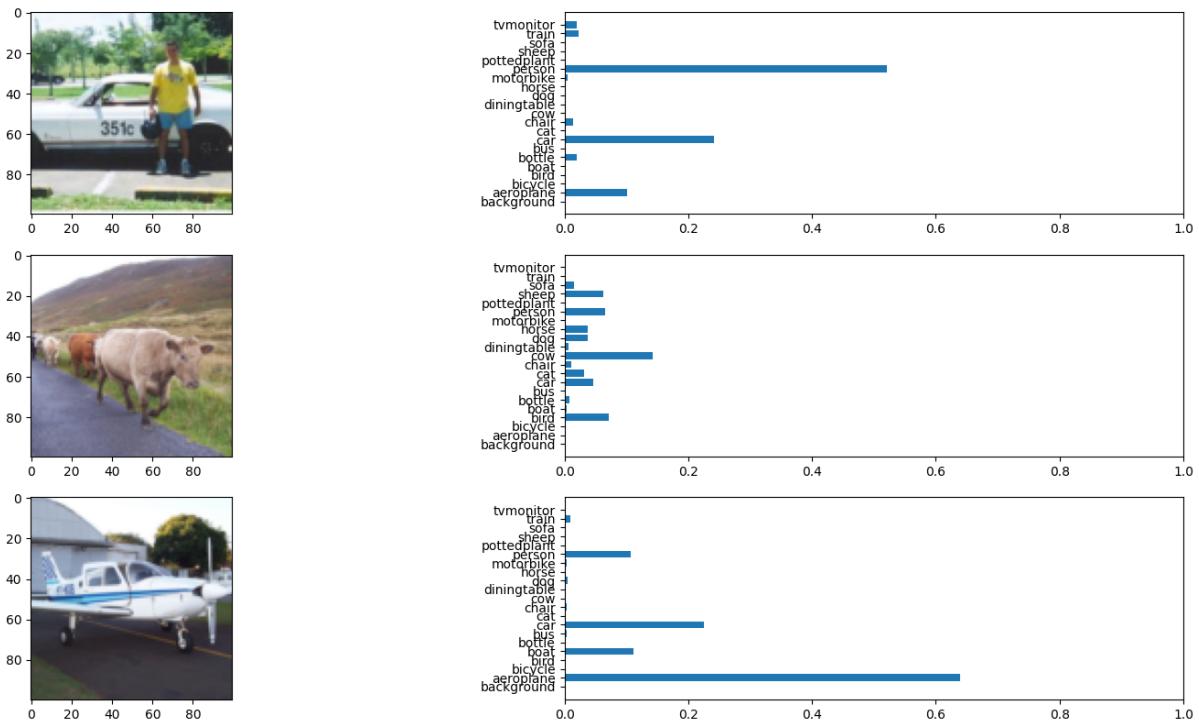


Figure 10: Résultats attendus - ResNet18

Quel est l'avantage de l'apprentissage par transfert?

Les références suivantes vous seront utiles :

- <https://pytorch.org/vision/stable/models.html#torchvision.models.resnet18>
- https://pytorch.org/vision/stable/_modules/torchvision/models/resnet.html

Question 3 – Complétez le calcul de la précision moyenne (mAP) sur les données de validation dans le fichier `transfer_learning_resnet.py`.

Vous devez utiliser la variable `thresholds` pour les seuils de décision.

Question 4 – Programmez l'architecture du réseau U-Net des notes de cours présentée à la figure 33 dans le fichier `models/UNet.py`.

Le réseau doit faire de la segmentation sémantique sur 3 classes (arrière-plan, personne et chien), donc la dernière couche convulsive doit avoir 3 canaux de sortie et elle n'est pas suivie d'une fonction sigmoïde. Les poids de ce réseau ont déjà été entraînés, le fichier `inference_unet.py` instancie la classe `UNet` et charge ces poids. Par conséquent, vous devez implémenter exactement la même architecture que celle dans les notes de cours. L'ordre des concaténations est la suivante : le tenseur provenant de la convolution transposée, suivi de celui provenant de l'encodeur.

Pour tester les modifications, exécutez le fichier `inference_unet.py`. Vous devriez obtenir une figure ressemblant à la figure 11.

Input image



person

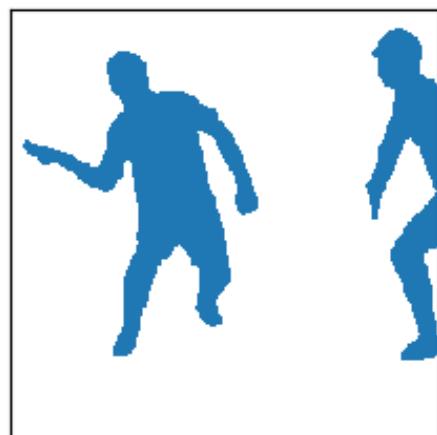


Figure 11: Résultat attendu - U-Net

15 Notes de cours

Cette section introduit certains concepts non couverts ou couverts rapidement dans l'ouvrage de référence proposé pour ce module.

15.1 Tâches en traitement d'images

15.1.1 Définitions

Il existe plusieurs tâches en traitement d'images.

- **Classification** – La classification consiste à déterminer quelle(s) classe(s) est/sont présente(s) dans une photo.
- **Localisation** – La localisation consiste à déterminer une boîte englobante qui encadre un élément de l'image. Cet élément appartient à une seule classe parmi les classes. Pour chaque image, une seule boîte est générée et la classification de l'élément à l'intérieur de l'objet est facultative. La classe peut aussi être fixe. Par exemple, il faut toujours localiser l'humain dans l'image.
- **Détection** – La détection consiste à identifier plusieurs boîtes englobantes qui encadrent des éléments de l'image. Chaque élément appartient à une seule classe parmi les classes. Il peut y avoir plusieurs instances de chaque classe et il est nécessaire de générer une boîte qui encadre chaque instance.
- **Segmentation sémantique** – La segmentation sémantique sépare les images pixel par pixel pour identifier les zones qui sont occupées par une classe spécifique.

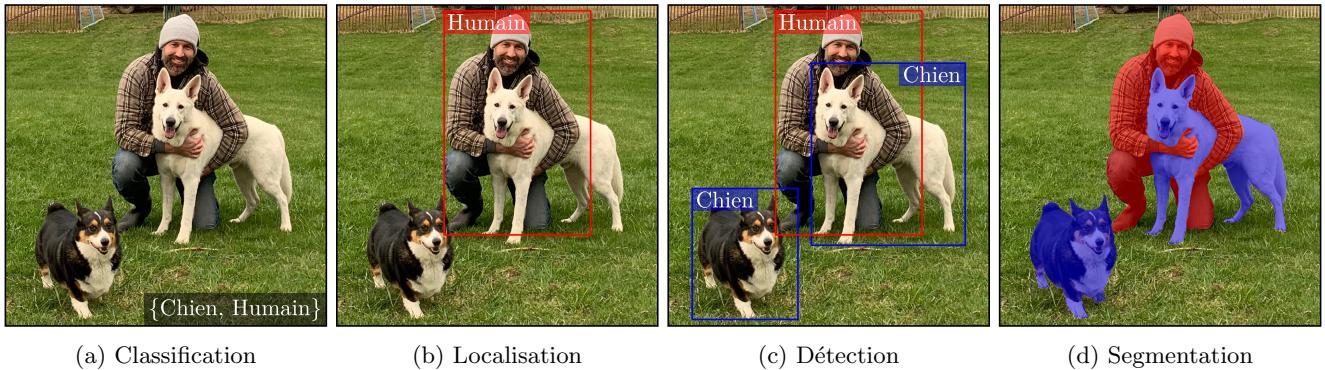


Figure 12: Tâches en traitement d'images

15.1.2 Métriques

Il existe plusieurs métriques qui permettent de mesurer la performance d'une tâche de classification, de localisation, de détection et de segmentation sémantique en traitement d'images.

Intersection sur union Lorsqu'il est question de segmentation sémantique ou de localisation, il est commun de déterminer les performances par ce qu'on appelle l'intersection sur union (ou en anglais *Intersection over Union*, avec l'acronyme *IoU*). Dans le cas d'une localisation, il s'agit de calculer l'aire d'intersection entre la boîte englobante obtenue comme prédiction et la cible, et de diviser par l'aire totale si on procède à l'union des deux boîtes englobantes, tel qu'illustré à la figure 13. Si la tâche de classification classe aussi l'objet, il faut mettre l'intersection sur union à 0 si la classe prédite est différente de la classe cible. Dans un contexte de localisation avec classification, il peut être intéressant d'utiliser une métrique de classification comme la justesse en plus de l'intersection sur union. De la même manière, il est possible de calculer l'aire d'intersection entre les pixels obtenus pour la segmentation sémantique durant la prédiction et la cible, et de diviser par l'aire totale si on procède à l'union des deux masques, tel qu'illustré à la figure 14.

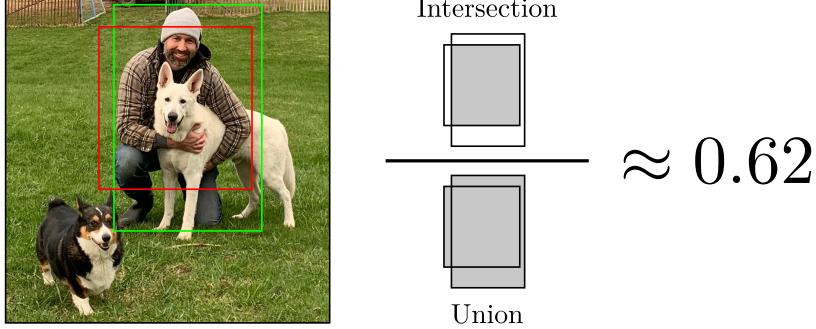


Figure 13: Intersection sur union pour une tâche de localisation

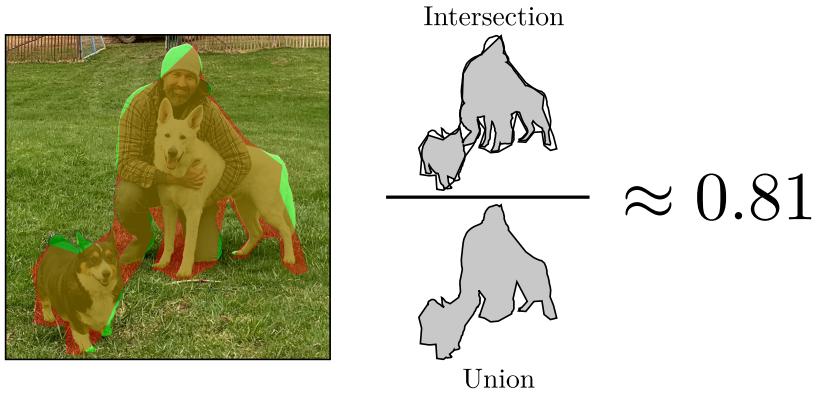


Figure 14: Intersection sur union pour une tâche de segmentation sémantique

Précision moyenne La précision moyenne (*mean Average Precision*, avec l’acronyme *mAP*) permet de mesurer les performances pour une tâche de classification. Le tableau 8 suivant résume les concepts de vrais positifs (VPs), vrais négatifs (VNs), faux positifs (FPs) et faux négatifs (FNs).

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	Vrai négatif (VN)	Faux positif (FP)
$y = 1$	Faux négatif (FN)	Vrai positif (VP)

Tableau 8: Vrais/Faux positifs/négatifs pour une classification binaire

À partir de ces mesures, il est possible de calculer la précision et le rappel, tel que démontré en (8) et (9).

$$Precision = \frac{VP}{VP + FP} \quad (8)$$

$$Rappel = \frac{VP}{VP + FN} \quad (9)$$

Il est possible d’utiliser ces métriques (précision et rappel) pour mesurer la performance du réseau pour effectuer la classification d’une classe spécifique, pour un seuil de décision fixé à une valeur précise. Cependant, en pratique, on cherche à calculer une métrique qui s’applique à plus d’une classe, et qui permet de déterminer à quel point un positif et un négatif peuvent être discriminés efficacement avec un seuil choisi sur mesure. C’est pour cette raison

que nous utilisons plutôt la précision moyenne.

Supposons que nous avons une classification binaire, et que la distribution des scores de la prédiction pour les échantillons dont les cibles sont $y = 0$ et $y = 1$ suivent deux distributions gaussiennes avec des moyennes différentes. En balayant le seuil en partant de 0.0 vers 1.0, la précision augmente progressivement (il y a de moins en moins de faux positifs) tandis que le rappel diminue (il y a de plus en plus de faux négatifs). On peut ensuite représenter sur une courbe la précision en fonction du rappel. Dans un scénario idéal, la précision et le rappel seront à 1 en même temps. En pratique cependant, il est commun que la précision diminue lorsque le rappel augmente, et vice-versa. Il est donc commun de calculer l'aire sous la courbe de précision et rappel, et de l'utiliser comme métrique de performance. Dans l'idéal, l'aire sous la courbe est de 1, et en pratique elle y sera inférieure. La figure 15 illustre ces concepts avec différentes distributions pour les scores de classification.

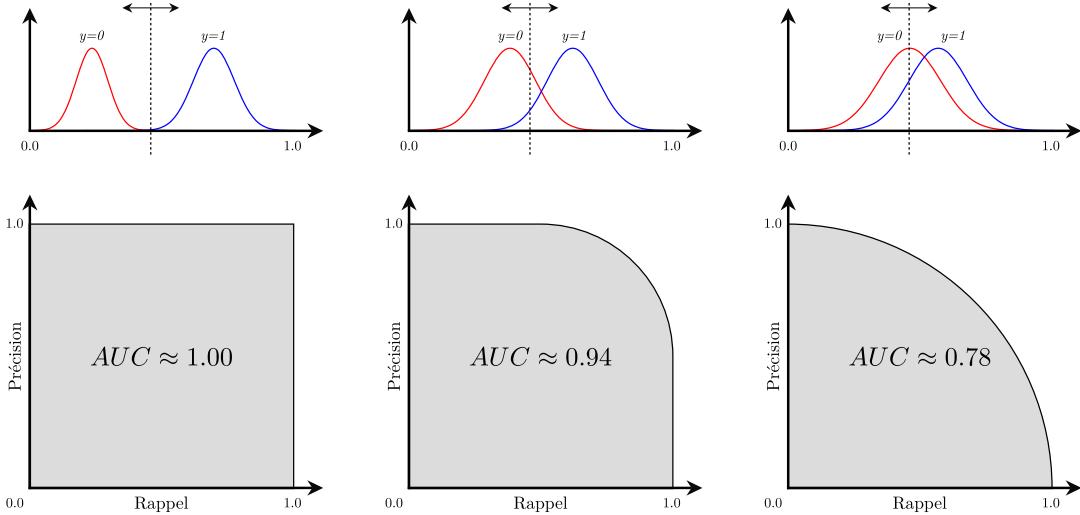


Figure 15: Aire sous la courbe pour une courbe précision et rappel

Dans une tâche de classification en traitement d'images, il y a normalement plusieurs classes, et chacune correspond à une décision binaire. En d'autres mots, si nous voulons déterminer si sur une image nous avons la présence ou non d'un chien, d'un chat et d'une personne, nous nous retrouvons avec 3 classes, et chacune peut être active ou inactive. Puisque nous avons une courbe précision et rappel différente pour chaque classe, nous calculons l'aire sous la courbe (AUC) pour chaque classe individuellement, et faisons finalement la moyenne. C'est ce que nous appelons la précision moyenne (mAP), comme démontré dans l'équation (10), où la variable C représente le nombre de classes au total et c_i , la courbe de la précision en fonction du rappel pour la classe i .

$$mAP = \frac{1}{C} \sum_{i=1}^C AUC(c_i) \quad (10)$$

La précision moyenne est aussi utilisée pour la tâche de détection. Pour décider si la prédiction correspond à la cible, on calcule l'intersection sur l'union et vérifie si elle excède un seuil préétabli. Il est nécessaire de fixer un seuil pour l'intersection sur union T_{IoU} qui a souvent la valeur de 0.5. Le tableau 9 présente les règles de décision pour détecter le bon cas de figure.

15.2 Couches

15.2.1 Convolution

Une couche convective comprend un ou plusieurs noyaux qui contiennent des poids, et le tenseur d'entrée est balayé par chaque noyau dans les sens horizontal et vertical. Nous allons d'abord introduire les différents paramètres pour une convolution en deux dimensions avec un seul canal en entrée et un seul noyau, et ensuite présenter des expressions mathématiques plus générales. La convolution en 2D consiste à faire glisser un noyau sur le tenseur d'entrée, et pour chaque alignement faire la somme des produits des paramètres du noyau avec les éléments du tenseur qui

Cas	Règles de décision
Vrai positif (VP)	$IoU > T_{IoU}$ et la bonne classe est prédite
Faux positif (FP)	$IoU < T_{IoU}$ ou la boîte englobante est dupliquée
Vrai négatif (VN)	Aucune prédiction ou la mauvaise classe est prédite
Faux négatif (FN)	Non comptabilisé

Tableau 9: Vrais/Faux positifs/négatifs pour une détection

sont alignés. Par exemple, si nous avons un tenseur d'entrée 7×7 , et que nous le balayons avec un noyau 3×3 , nous obtenons un tenseur de sortie 5×5 comme illustré à la figure 16.

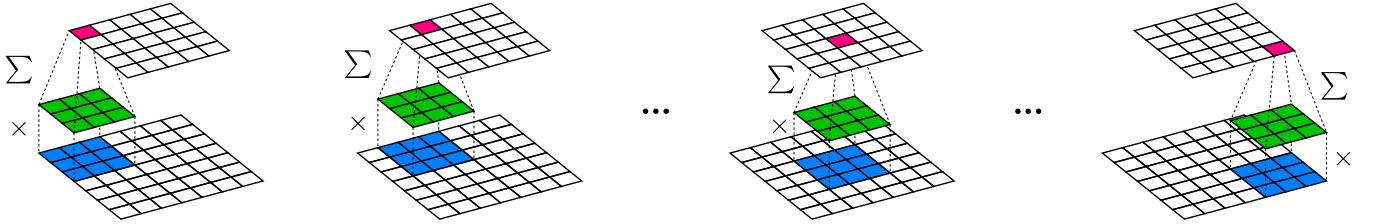


Figure 16: Convolution 2D ($P_H = 0, P_W = 0, S_H = 1, S_W = 1$)

Il est également possible de glisser le noyau en faisant des foulées (*strides*) plus grandes que 1. Ceci a pour effet entre autres de générer un tenseur de sortie avec moins d'éléments, comme le montre la figure 17 pour une foulée de 2, qui génère un tenseur de sortie de 3×3 . Les variables S_H et S_W représentent la foulée dans le sens de la hauteur et de la largeur, respectivement.

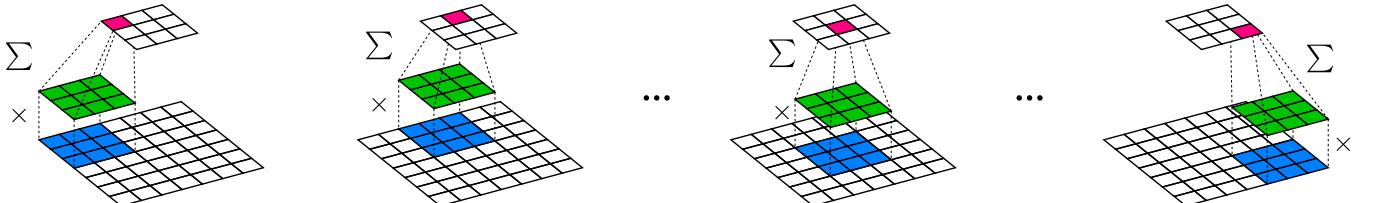


Figure 17: Convolution 2D ($P_H = 0, P_W = 0, S_H = 2, S_W = 2$)

Il est également possible d'ajouter des valeurs de remplissage nulles sur le périmètre du tenseur. Ceci peut entre autres permettre de générer un tenseur de sortie de mêmes dimensions que le tenseur d'entrée, comme dans la figure 18 où un remplissage de 1 permet de conserver les dimensions de 7×7 en sortie. Les variables P_H et P_W représentent le remplissage dans le sens de la hauteur et de la largeur, respectivement.

Dans les exemples précédents, le tenseur d'entrée possède une hauteur de H_{in} éléments et une largeur de W_{in} éléments. Il a été assumé qu'il y avait un seul canal d'entrée et un seul noyau. De manière générale, il peut y avoir plusieurs canaux et plusieurs noyaux, ce qui fait qu'en réalité les tenseurs possèdent trois axes. Il est possible de définir mathématiquement une couche convulsive. Le tenseur d'entrée est défini comme suit, où C correspond au nombre de canaux en entrée:

$$\mathbf{x} \in \mathbb{R}^{C \times H_{in} \times W_{in}} \quad (11)$$

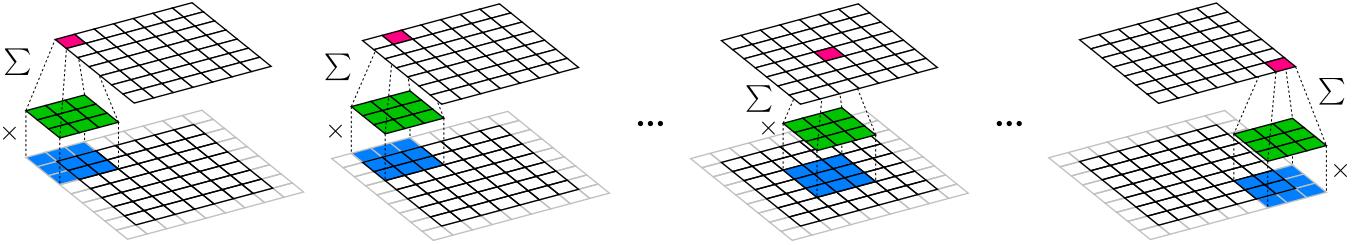


Figure 18: Convolution 2D ($P_H = 1$, $P_W = 1$, $S_H = 1$, $S_W = 1$,)

Le tenseur de sortie possède une hauteur de H_{out} éléments et une largeur de W_{out} éléments, et F canaux (qui correspond au nombre de noyaux):

$$\mathbf{y} \in \mathbb{R}^{F \times H_{out} \times W_{out}} \quad (12)$$

L'ensemble des paramètres dans les noyaux sont regroupés dans le tenseur \mathbf{W} , qui est défini comme suit :

$$\mathbf{W} \in \mathbb{R}^{F \times C \times K_H \times K_W} \quad (13)$$

Il est également intéressant de noter qu'un biais est ajouté pour chaque noyau :

$$\mathbf{b} \in \mathbb{R}^F \quad (14)$$

$$\mathbf{b} = [b_1 \ b_2 \ \dots \ b_F] \quad (15)$$

Afin de simplifier l'équation, on peut également définir une matrice en deux dimensions pour chaque canal c du tenseur d'entrée :

$$\mathbf{x}_c \in \mathbb{R}^{H_{in} \times W_{in}} \quad (16)$$

$$\mathbf{x} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_C] \quad (17)$$

De la même manière, on définit une matrice en deux dimensions pour chaque canal f (donc chaque noyau) du tenseur de sortie :

$$\mathbf{y}_f \in \mathbb{R}^{H_{out} \times W_{out}} \quad (18)$$

$$\mathbf{y} = [\mathbf{y}_1 \ \mathbf{y}_2 \ \dots \ \mathbf{y}_F] \quad (19)$$

On peut finalement définir une matrice en deux dimensions pour chaque paire de canaux d'entrée c et de canal de sortie f :

$$\mathbf{W}_{f,c} \in \mathbb{R}^{K_H \times K_W} \quad (20)$$

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{1,1} & \mathbf{W}_{1,2} & \dots & \mathbf{W}_{1,C} \\ \mathbf{W}_{2,1} & \mathbf{W}_{2,2} & \dots & \mathbf{W}_{2,C} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{W}_{F,1} & \mathbf{W}_{F,2} & \dots & \mathbf{W}_{F,C} \end{bmatrix} \quad (21)$$

Il est donc possible de calculer la prédiction d'une couche convulsive en utilisant l'équation suivante, avec l'opérateur \star qui représente la convolution en deux dimensions :

$$\mathbf{y}_f = b_f + \sum_{c=1}^C \mathbf{W}_{f,c} \star \mathbf{x}_c \quad (22)$$

Note : En apprentissage profond, le terme convolution est différent de l'opération effectuée en traitement de signal. Il s'agit en fait d'une corrélation selon le vocabulaire du traitement de signal.

Les dimensions du tenseur de sortie peuvent être calculées selon les dimensions du tenseur d'entrée, les dimensions du noyau, et les propriétés de la convolution (remplissage et foulée) :

$$H_{out} = \left\lfloor \frac{H_{in} + 2P_H - K_H}{S_H} + 1 \right\rfloor \quad (23)$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2P_W - K_W}{S_W} + 1 \right\rfloor \quad (24)$$

Une couche convective en 2-D peut être implémentée avec Pytorch en utilisant la classe `torch.nn.Conv2d`.

15.2.2 Convolution transposée

La convolution transposée consiste à balayer un noyau, le multiplier par chaque élément du tenseur d'entrée, et accumuler le résultat sur une zone spécifique du tenseur de sortie. Les noyaux multipliés par chaque élément se chevauchent donc dans le tenseur de sortie. Par exemple, si nous avons un tenseur d'entrée de 5×5 avec un noyau de 3×3 , nous obtenons un tenseur de sortie de 7×7 . La figure 19 illustre cette opération. Il est intéressant de noter que le tenseur de sortie est plus grand que le tenseur d'entrée.

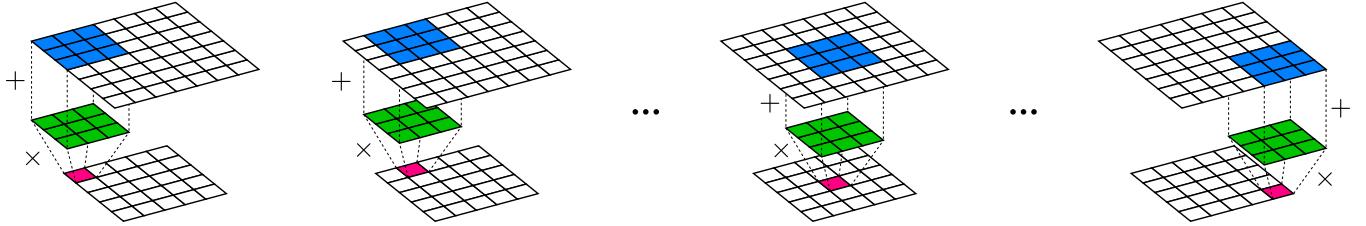


Figure 19: Convolution 2D transposée ($P_H = 0, P_W = 0, S_H = 1, S_W = 1$)

Tout comme avec une convolution, il est possible de modifier la foulée. Par exemple, si nous avons un tenseur d'entrée de 3×3 avec un noyau de 3×3 et une foulée de 2, nous obtenons un tenseur de sortie de 7×7 , tel qu'ilustré à la figure 20.

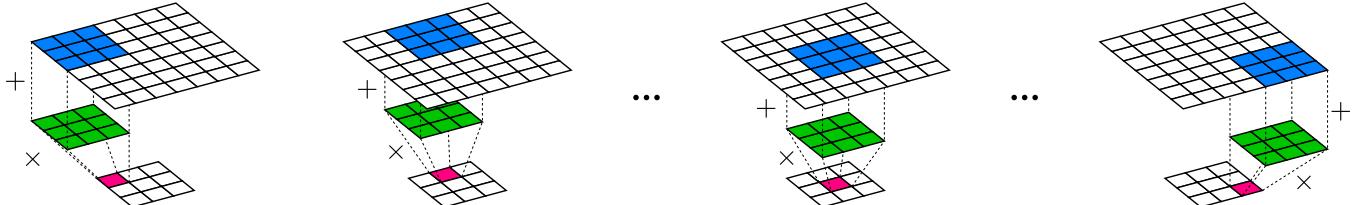


Figure 20: Convolution 2D transposée ($P_H = 0, P_W = 0, S_H = 2, S_W = 2$)

Finalement, il est également possible d'ajouter du remplissage. Dans ce cas précis, le remplissage consiste en fait à ignorer les éléments sur la bordure extérieure du tenseur de sortie afin de réduire la taille. La figure 21 illustre un exemple avec un tenseur d'entrée de 7×7 et un noyau de 3×3 avec un remplissage de 1, ce qui donne un tenseur de sortie de 7×7 .

Dans les exemples précédents, le tenseur d'entrée possède une hauteur de H_{in} éléments et une largeur de W_{in} éléments. Il a été assumé qu'il y avait un seul canal d'entrée et un seul noyau. De manière générale, il peut y avoir

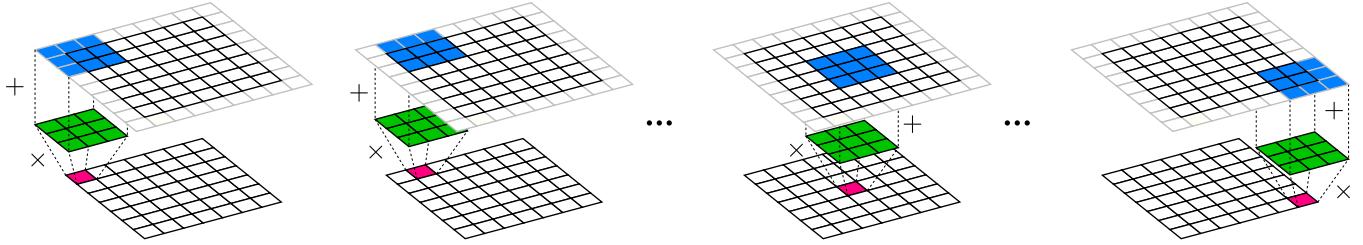


Figure 21: Convolution 2D transposée ($P_H = 1, P_W = 1, S_H = 1, S_W = 1$)

plusieurs canaux et plusieurs noyaux, ce qui fait qu'en réalité les tenseurs possèdent trois axes. Il est possible de définir mathématiquement une couche convective. Le tenseur d'entrée est défini comme suit, où C_{in} correspond au nombre de canaux en entrée :

$$\mathbf{x} \in \mathbb{R}^{C \times H_{in} \times W_{in}} \quad (25)$$

Le tenseur de sortie possède une hauteur de H_{out} éléments et une largeur de W_{out} éléments, et C canaux (qui correspond au nombre de noyaux) :

$$\mathbf{y} \in \mathbb{R}^{F \times H_{out} \times W_{out}} \quad (26)$$

Les dimensions du tenseur en sortie sont obtenues comme suit :

$$H_{out} = (H_{in} - 1)S_H - 2P_H + K_H \quad (27)$$

$$W_{out} = (W_{in} - 1)S_W - 2P_W + K_W \quad (28)$$

Une couche convective en 2-D peut être implémentée avec Pytorch en utilisant la classe `torch.nn.ConvTranspose2d`.

15.2.3 Mise en commun maximale

La couche de mise en commun maximale (*max pooling*) se compare à la couche de convolution, mais au lieu de multiplier les éléments du tenseur par un noyau et d'en faire la somme, nous cherchons à extraire la valeur maximale. Les principes de foulée et de remplissage sont les mêmes que pour la couche de convolution. Cette couche génère un tenseur de mêmes dimensions que pour une couche de convolution. Les figures 22, 23 et 24 illustrent le fonctionnement de la couche de mise en commun maximale.

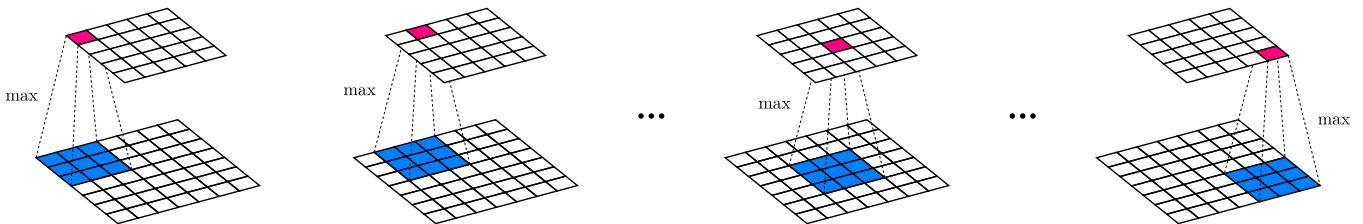


Figure 22: Mise en commun maximale 2D ($P_H = 0, P_W = 0, S_H = 1, S_W = 1$)

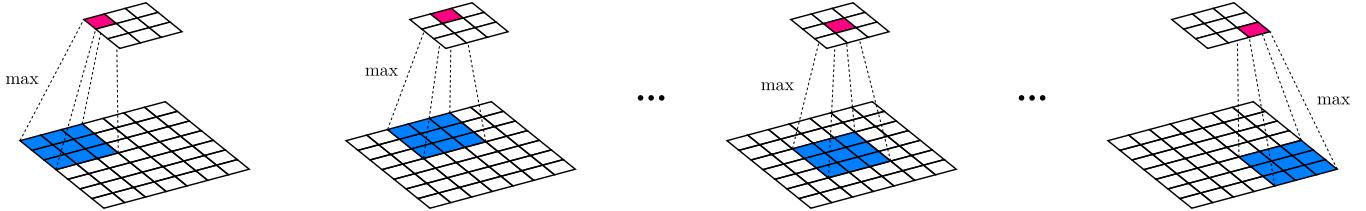


Figure 23: Mise en commun maximale 2D ($P_H = 0, P_W = 0, S_H = 2, S_W = 2$)

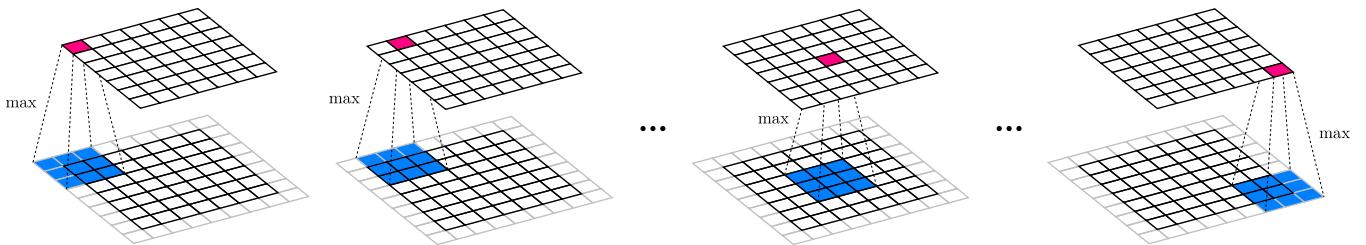


Figure 24: Mise en commun maximale 2D ($P_H = 1, P_W = 1, S_H = 1, S_W = 1$)

Une couche de mise en commun maximale en 2-D peut être implémentée avec Pytorch en utilisant la classe `torch.nn.MaxPool2d`.

15.3 Connexion résiduelle

Lorsqu'un réseau de neurones convolutif possède une quantité importante de couches convolutives, il se produit ce que nous appelons la disparition du gradient. Ceci est dû au fait que le gradient est propagé par multiplication d'une couche à l'autre, et finit par tendre vers zéro, ce qui rend l'entraînement particulièrement difficile. Pour y remédier, le principe d'une connexion résiduelle est introduit. L'idée derrière ce type de connexion est d'apprendre la différence entre la sortie attendue et l'entrée, plutôt que la sortie elle-même. La sortie est ensuite obtenue en additionnant l'entrée à la différence obtenue. C'est pour cette raison qu'on utilise le terme *résiduel* pour ce type de connexion.

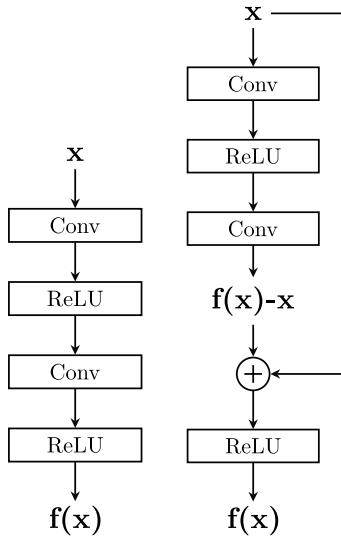


Figure 25: Connexion directe (gauche) et connexion résiduelle (droite). Ces deux architectures permettent de générer la même prédiction, mais la connexion résiduelle facilite grandement l'entraînement puisque les couches convolutives doivent seulement prédire le résidu.

15.4 Champ réceptif

Le champ réceptif correspond à la portion de l'image qui influence une seule valeur d'un tenseur à une couche donnée. Les équations suivantes permettent de calculer les dimensions du champ réceptif pour une série de couches convolutives et de mise en commun maximale.

$$\begin{aligned} J_H^{(0)} &= 1 \\ J_H^{(i)} &= J_H^{(i-1)} S_H^{(i)} \\ R_H^{(0)} &= 1 \\ R_H^{(i)} &= R_H^{(i-1)} + (K_H^{(i)} - 1)J_H^{(i-1)} \end{aligned} \tag{29}$$

$$\begin{aligned} J_W^{(0)} &= 1 \\ J_W^{(i)} &= J_W^{(i-1)} S_W^{(i)} \\ R_W^{(0)} &= 1 \\ R_W^{(i)} &= R_W^{(i-1)} + (K_W^{(i)} - 1)J_W^{(i-1)} \end{aligned} \tag{30}$$

$R_H^{(0)}$ et $R_W^{(0)}$ sont respectivement la hauteur et la largeur du champ réceptif à l'entrée du réseau de neurones. $R_H^{(i)}$ et $R_W^{(i)}$ sont respectivement la hauteur et la largeur du champ réceptif après la couche i . $S_H^{(i)}$ et $S_W^{(i)}$ correspondent aux foulées de la couche i , tandis que $K_H^{(i)}$ et $K_W^{(i)}$ sont les dimensions des noyaux de la couche i .

Puisque les couches convolutives et de mise en commun maximale utilisent seulement une partie de l'information pour calculer une valeur de sortie, il est intéressant de calculer les dimensions du champ réceptif à une certaine couche pour déterminer la quantité de contexte utilisé pour calculer les valeurs de sortie de cette couche.

15.5 Architectures populaires

15.5.1 AlexNet

Le réseau AlexNet a été conçu afin d'effectuer de la classification d'images. Le réseau reçoit une image en entrée et génère un score pour chaque classe, et le plus grand score correspond à l'image détectée, tel qu'ilustré à la figure 26. La figure 27 illustre l'architecture du réseau.

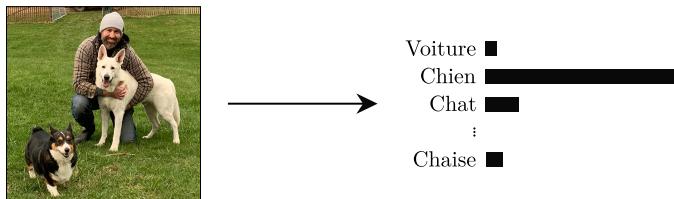


Figure 26: Classification d'images avec AlexNet

Il reçoit en entrée des images de 224×224 pixels, en couleur (donc avec 3 canaux, pour RVB) (l'entrée est donc $\mathbf{x} \in \mathbb{R}^{3 \times 224 \times 224}$). Le réseau produit à la sortie un vecteur 1 parmi n, avec 1000 classes (la sortie est donc $\hat{\mathbf{y}} \in \mathbb{R}^{1000}$). Les premières couches utilisent des convolutions pour extraire les caractéristiques visuelles dans l'image. Les dernières couches utilisent des couches entièrement connectées (linéaires) pour effectuer une classification. Les couches de mise en commun maximale permettent deux choses : 1) augmenter le champ réceptif, 2) réduire les dimensions des tenseurs et 3) rendre les caractéristiques robustes à de légères translations. Notez les dimensions des tenseurs d'entrées et de sorties pour les couches convolutives et de mise en commun maximale en (23) et (24). La couche *Flatten* fait simplement réorganiser les éléments du tenseur $\mathbb{R}^{256 \times 6 \times 6}$ en un tenseur sous forme de vecteur \mathbb{R}^{9216} .

15.5.2 ResNet

Pour faciliter l'entraînement d'un réseau convolutif avec de nombreuses couches, il est possible d'utiliser les connexions résiduelles comme expliquées dans la section 15.3. Il existe plusieurs architectures avec des profondeurs différentes, mais à des fins de compréhension nous allons uniquement présenter l'architecture ResNet18, qui contient 18 couches

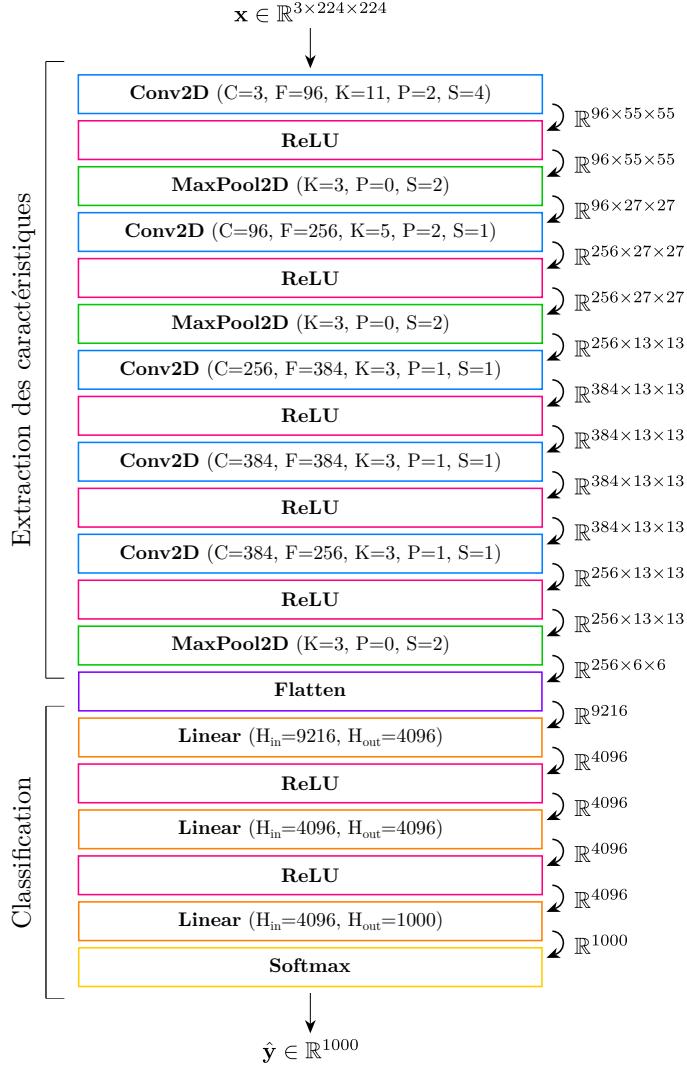


Figure 27: Architecture du réseau AlexNet

convolutives (il existe également ResNet34, ResNet50, ResNet101, etc.). Cette architecture prend en entrée des images de 224×224 pixels, en couleur (donc avec 3 canaux, pour RGB) (l'entrée est donc $\mathbf{x} \in \mathbb{R}^{3 \times 224 \times 224}$). Le réseau produit à la sortie un vecteur 1 parmi n, avec 1000 classes (la sortie est donc $\hat{\mathbf{y}} \in \mathbb{R}^{1000}$). La figure 28 illustre l'architecture de ResNet18.

15.5.3 YOLO

Le réseau YOLO (You Only Look Once) vise à effectuer de la détection d'images en utilisant un réseau convolutif qui génère une grille de 7×7 , où chaque élément de la grille contient des informations sur les objets à proximité. Chaque élément de la grille peut représenter jusqu'à deux boîtes, dont les centres sont situés à l'intérieur de cet élément de la grille. Chaque boîte est composée des coordonnées en x et y normalisées par rapport au coin supérieur gauche de l'élément dans la grille, et la largeur w et hauteur h de la boîte normalisée par rapport aux dimensions de l'image. Finalement, un indice de confiance est généré pour chaque boîte, et correspond à la probabilité qu'un objet soit présent multiplié par l'intersection sur l'union de la prédiction et la cible. Si aucune boîte n'est présente dans un élément de la grille, la confiance est simplement fixée à zéro. Finalement, dans chaque élément de la grille, pour chaque classe, une valeur entre 0 et 1 représente la probabilité que cet élément de la grille contienne l'image de cette classe. Dans la première version de YOLO, il y avait 2 boîtes par élément de grille et 20 classes, ce qui donne un vecteur de 30 dimensions par élément de la grille. Le tenseur de sortie a donc le format $\mathbb{R}^{30 \times 7 \times 7}$. Ces concepts sont

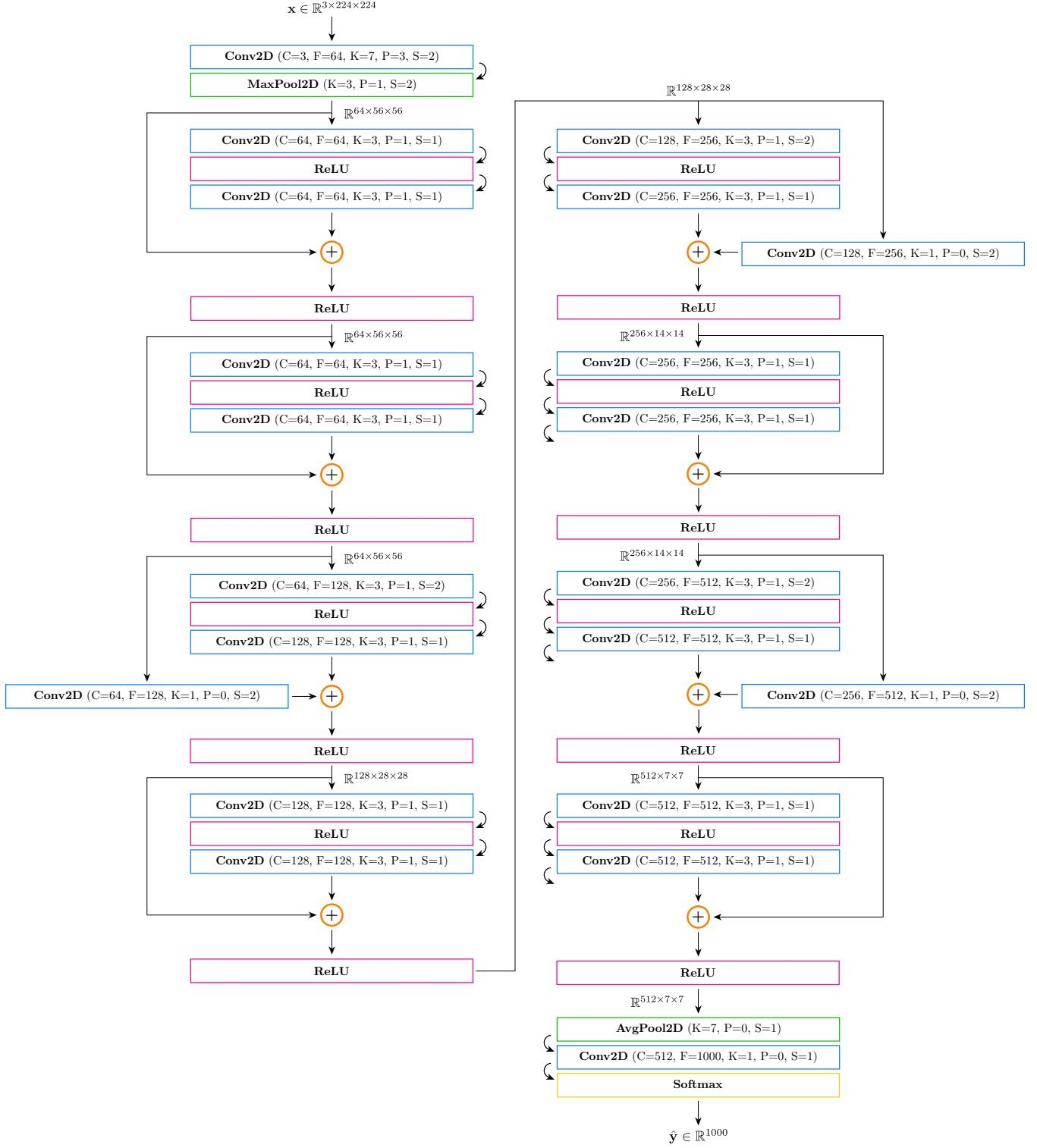


Figure 28: Architecture du réseau ResNet18

illustrés dans la figure 29.

L'architecture du réseau est présentée à la figure 30. Notez que ce réseau consiste essentiellement en des couches convolutives, mais que les fonctions d'activation sont pour la plupart un rectificateur avec fuite (*Leaky ReLU*). Cette

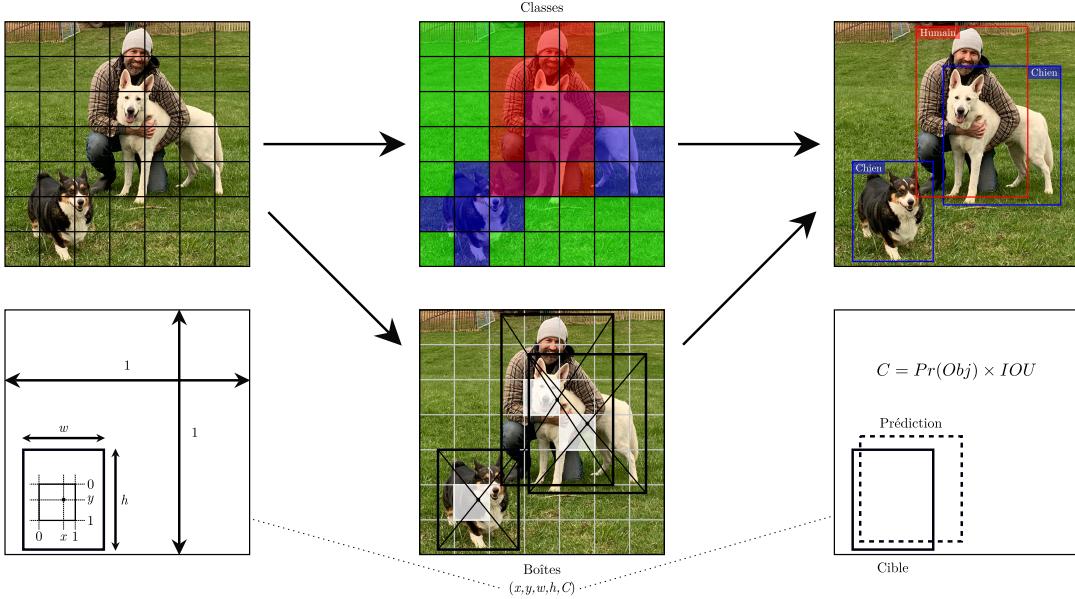


Figure 29: Détection d'images avec YOLO

fonction d'activation diffère d'une fonction ReLU ordinaire (voir (31)) par la définition en (32). Dans l'article original, les auteurs indiquent de l'utilisation d'un rectificateur avec fuite améliore l'entraînement du réseau.

$$y = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \quad (31)$$

$$y = \begin{cases} 0.1x & x < 0 \\ x & x \geq 0 \end{cases} \quad (32)$$

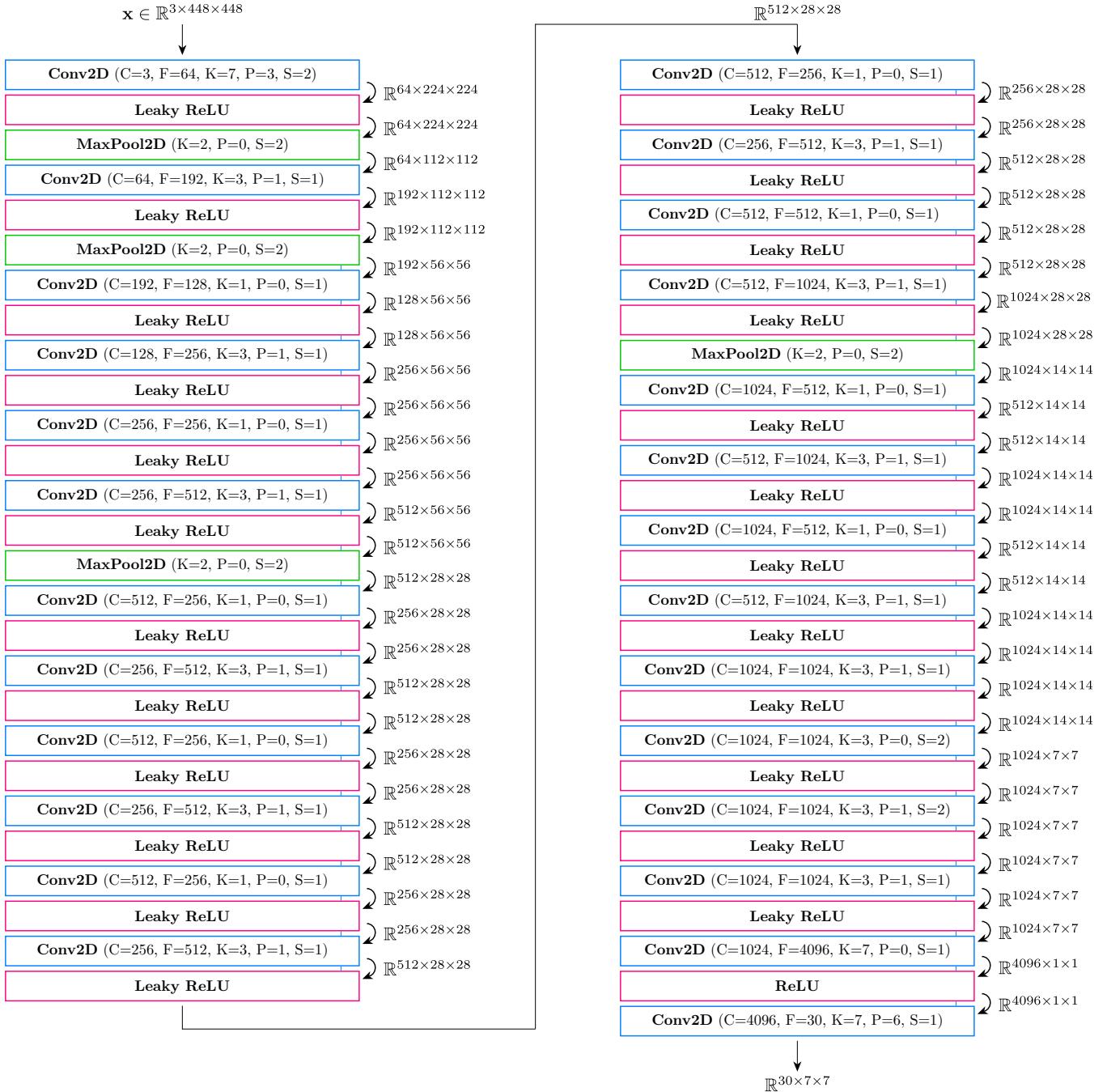


Figure 30: Architecture du réseau YOLO

15.5.4 U-Net

Le réseau U-Net est particulièrement adapté à des tâches de segmentations sémantiques. Par exemple, nous recevons une image en entrée, et désirons générer un masque binaire de segmentation sémantique pour séparer deux classes : êtres vivants (animaux et humains), et le reste, comme illustré à la figure 31.

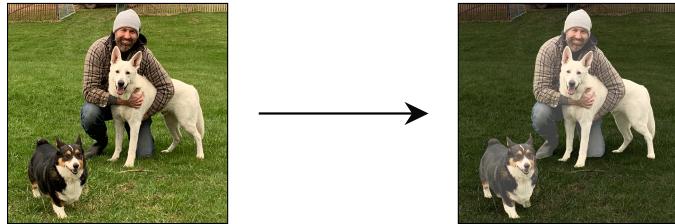


Figure 31: Segmentation sémantique d'images avec U-Net

Lorsqu'on utilise un réseau convolutif, les couches de mise en commun maximales réduisent la résolution spatiale de l'image, tout en encodant des informations de plus haut niveau à propos du contenu de l'image. L'idée derrière l'architecture U-Net est de concaténer un tenseur qui contient une bonne résolution spatiale avec un autre qui a une moins bonne résolution spatiale, mais des informations contextuelles de plus haut niveau, tel qu'illustré à la figure 32.

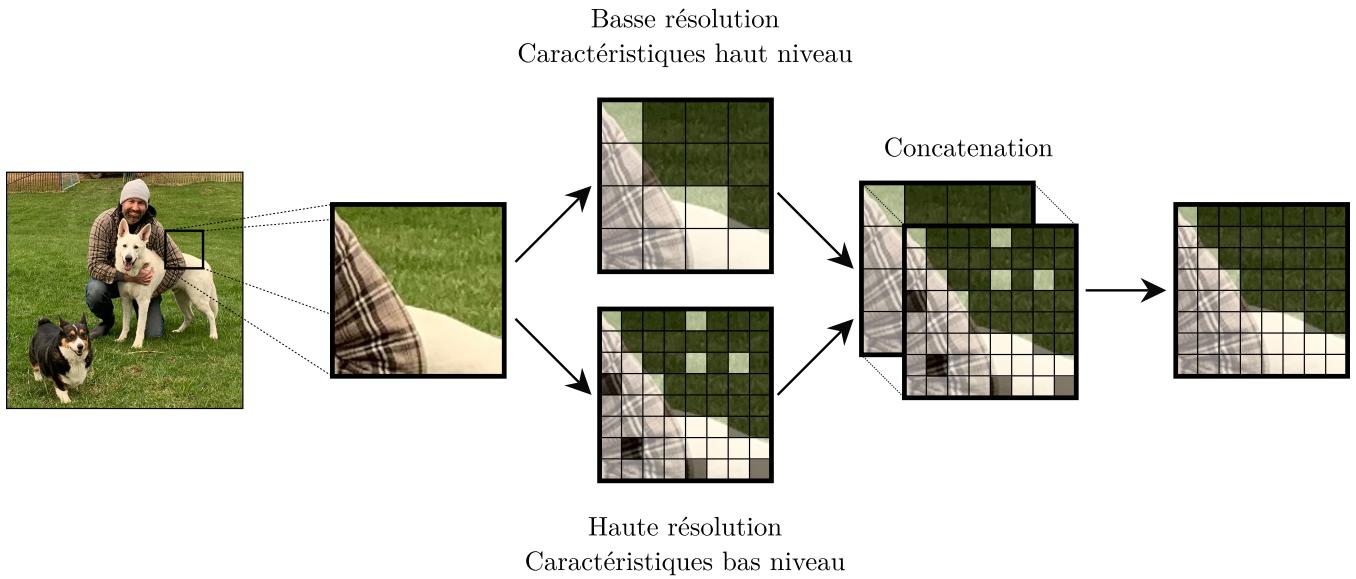


Figure 32: Concaténation de tenseurs

Il existe plusieurs variantes de ce réseau : nous proposons à la figure 33 un exemple qui diffère légèrement de l'article scientifique qui a introduit la première version de U-Net, mais illustre clairement le principe. Le réseau reçoit en entrée une image de 256×256 pixels, en couleur (donc avec 3 canaux, pour RVB) (l'entrée est donc $\mathbf{x} \in \mathbb{R}^{3 \times 256 \times 256}$). Dans cet exemple, le réseau retourne un masque binaire pour segmenter l'image, et ce masque est représenté par le tenseur $\hat{\mathbf{y}} \in [0, 1]^{1 \times 256 \times 256}$. La partie gauche du réseau encode l'information abstraite de plus haut niveau, tandis que la partie droite décode et génère le masque. La particularité du réseau réside dans les connexions de gauche à droite et la concaténation des tenseurs dans le décodeur. Ce processus permet de conserver la résolution plus fine de l'image qui serait autrement ignorée suite aux couches de mise en commun maximales. Le nom du réseau provient d'ailleurs de la forme en U du schéma du réseau.

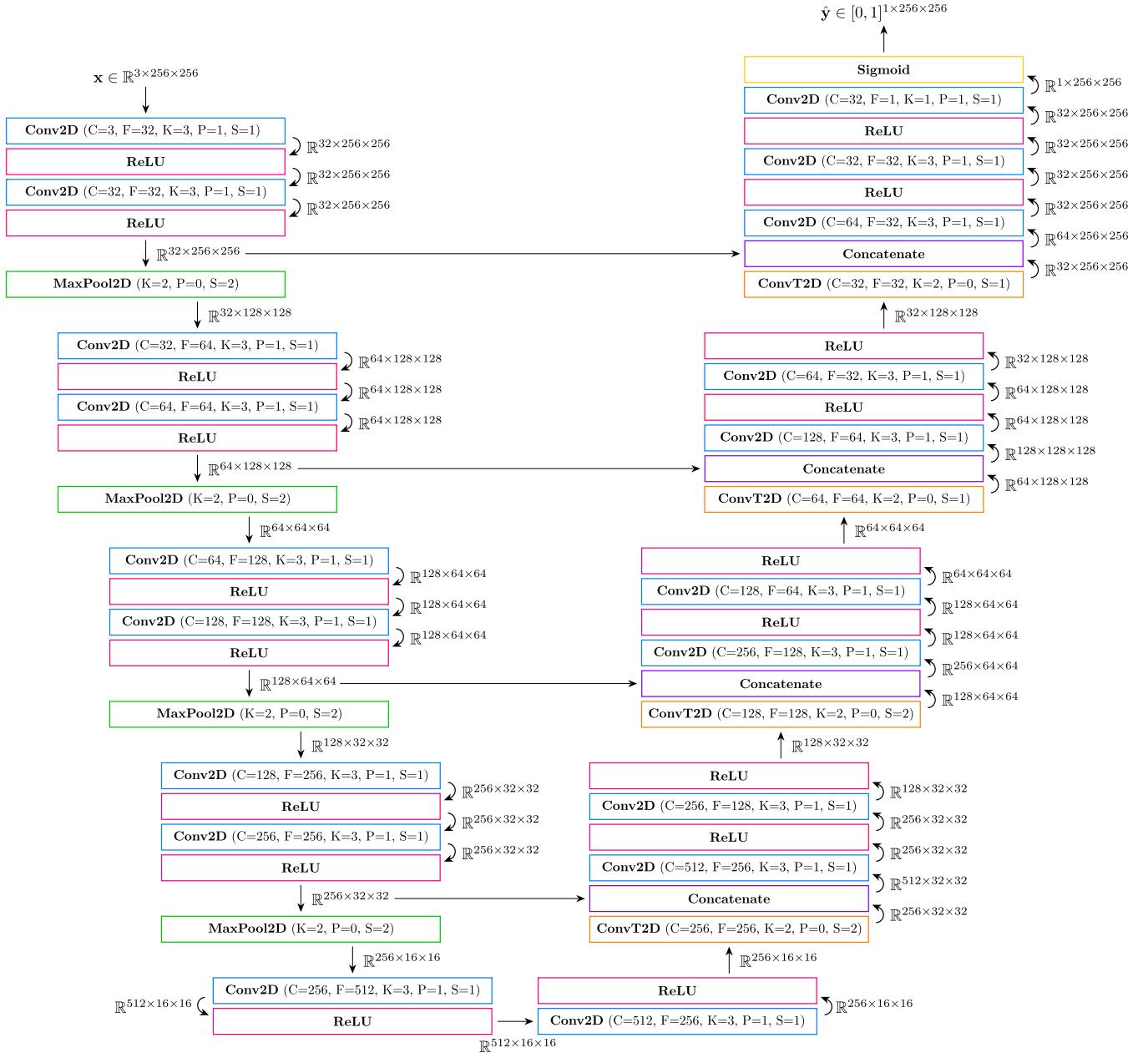


Figure 33: Architecture du réseau U-Net