

Détection de fumée avec YOLOv5 et comparaison de performance avec MVMNet

Chu Ka Hei², Renaud Lespérance¹, Morgan Péju² and Gabriel Poupart¹

¹Département de génie électrique, Polytechnique Montréal

²Département de génie informatique et logiciel, Polytechnique Montréal

Abstract

Les feux de forêt peuvent être causés par toute sorte de choses (mégot de cigarette, foudre, dépôt d'ordures, etc). Il est possible d'éliminer un feu de forêt et de minimiser son impact si nous sommes capables de détecter la fumée qu'il produit. Dans cet article, nous proposons un modèle de détection de fumée basée sur YOLOv5 [1]. Notre modèle est entraîné sur trois bases de données provenant d'images du High Performance Wireless Research and Education Network(HPWREN)[2]. Notre modèle nous donne des résultats de 79.26% de précision, 73.09% de rappel, 77.95% de mAP0.5 et 37.89% de mAP0.5 :0.95. Nous avons comparé nos résultats avec celui du modèle MVM-Net, présenté dans l'article "Fast forest fire smoke detection using MVMNet"[3] qui nous a inspiré pour ce projet. Notre modèle offre une performance inférieure à celle de MVMNet, mais bénéficie d'une instanciation rapide grâce à YOLOv5 .

1 Introduction

Les feux de forêt sont un danger écologique qui se produisent de plus en plus fréquemment. Au Canada, environ 8000 feux de forêt se déclarent chaque année[4]. La moyenne annuelle des superficies brûlées par les feux de forêt est de 2,5 millions d'hectares par année, ce qui équivaut à la moitié de la superficie de la Nouvelle-Écosse[4]. Globalement, plus de 300 000 personnes sont tuées chaque année par des feux de forêt[5]. La fumée est le premier indice d'un feu. Si nous sommes capables de la détecter, nous pouvons réduire les risques avant qu'il ne soit trop tard. Nous nous sommes inspirés de l'article "Fast forest fire smoke detection using MVMNet"[3] de Yaowen Hu et Jialei Zhan. L'idée est donc de découvrir si nous pouvons réaliser un modèle semblable de détection de fumée avec des techniques que nous avons vues dans le cours INF8225 donné à Polytechnique Montréal. Pour cette raison, nous avons décidé d'entraîner notre propre modèle de détection de fumée avec YOLOv5[1]. YOLOv5 est sélectionné parce qu'il exige un temps relativement court pour l'entraîner comparativement à d'autres techniques dans le domaine de la détection d'objet. Nous avons

entraîné notre modèle avec trois bases de données (Wildfire Smoke v1, Wildfire Smoke v2 et Cloud dataset) qui proviennent d'images du High Performance Wireless Research and Education Network (HPWREN) [2] qui ont été labélisées par AI For Mankind[6]. On retrouve les liens vers ces trois bases de données sur la page GitHub de AI For Mankind [7].

Le code d'entraînement et de test relié à cet article est disponible sur le GitHub suivant : https://github.com/MorganPeju/inf8225_project.

2 Travaux antérieurs

Dans la littérature, on retrouve diverses approches de détection de feux de forêt. Ici nous allons en définir sommairement certaines, et finir notre analyse des travaux antérieurs en parlant de méthodes de détection d'objet général pouvant être utilisées pour détecter une multitude d'objets.

2.1 Approche de détection de feux de forêt

Une première idée intéressante est d'utilisation des drones peu coûteux pour pouvoir survoler des territoires où l'on n'a pas nécessairement de caméras installées [8]. Cette méthode offre une perspective différente des caméras terrestres et peu permettent une détection plus rapide que des systèmes satellites comme de plus petits feux peuvent être détectés.

Une deuxième approche plus classique et qui permet de couvrir des régions vastes est d'utiliser les images provenant des satellites. Certaines méthodes utilisent des satellites pouvant couvrir de larges bandes de fréquences, partant du spectre visible jusqu'à l'infrarouge [9]. Cette méthode est utilisée pour suivre beaucoup de facteurs environnementaux différents comme la couverture forestière, les masses d'eau ou encore les feux de forêt. Une autre méthode plus récente est d'utiliser la réflectance des nuages de fumée pour effectuer un meilleur suivi des grands feux de forêt [10].

Une troisième approche est basée sur l'utilisation de senseur terrestre. Ce type de détection pourrait être utilisé proche de zone que l'on sait déjà à risque. Ces senseurs peuvent être utilisés pour détecter des feux à plus de 1 km [11], ce qui peut en faire des bons outils de suivi, mais difficilement de détection dans des zones peu développées.

La dernière approche que nous avons décidé d'étudier est l'utilisation d'images provenant de caméras installées sur des stations d'observation forestière ou sur des autres structures implantées en nature. Cette approche est celle étudiée dans

l'article qui a inspiré ce travail [3]. En utilisant ces images on peut avoir une méthode relativement peu coûteuse de détection des feux de forêt, tout en conservant une bonne précision grâce aux avancées des dernières années dans le domaine de la détection d'objet.

2.2 Méthode de détection d'objet

Comme la méthode étudiée dans cet article est basée sur des images dans le spectre visible nous n'allons pas détailler de méthode de détection d'objet pour les autres approches mentionnées plus tôt.

On peut séparer en deux grandes catégories les méthodes de détection d'objet. Soit les détecteurs à deux niveaux, soit les détecteurs à un niveau [12].

La première catégorie utilisant deux niveaux fonctionne avec un premier modèle permettant de créer de proposition de boîte englobante où il peut potentiellement y avoir un objet et passé ces propositions au deuxième niveau. Le deuxième niveau a comme tâche la classification d'image. Par ce principe, on peut obtenir de très bonne détection en combinant différent premier et deuxième stage. Cependant, ces méthodes ont tendance à être relativement lentes et non utilisables pour des systèmes temps réel, comme cela peut prendre plusieurs secondes dans un certain cas pour faire la détection d'un objet. Dans cette famille on retrouve les méthodes RCNN [13], fast - RCNN [14], EfficientNets [15] et bien d'autres. Leur performance est très bonne, mais peu intéressante dans un cas comme le nôtre où chaque seconde compte lorsqu'un feu s'embrace. Elles ne seront donc pas plus détaillées.

La deuxième catégorie utilise un seul réseau profond pour effectuer la détection des boîtes englobante et la classification des objets. C'est pour cela qu'ils sont en général plus rapides que les détecteurs à deux niveaux. Dans cette famille on compte les modèles YOLO[16], RetinaNet [17], SSD [18] et bien d'autres encore.

Les réseaux RetinaNet sont basés sur un ResNet suivi d'une pyramide d'extraction d'attribut et utilisent une fonction de perte pour la classification de type "focal loss". C'est à cause de cette perte qui permet de focaliser sur les exemples négatifs que le réseau porte ce nom.

Le réseau SSD (Single Shot Detector) est lui basé sur le réseau VGG-16 suivi de couche de convolution. Ces couches de convolution proposent un ensemble de boîtes et de classification qui sont filtrées par un mécanisme NMS (non-maximum suppression).

Les réseaux de type YOLO qui seront plus détaillés dans ce rapport offrent de bonnes performances avec une architecture relativement simple. Les dernières versions YOLOv5[1], YOLOv4 [19] et YOLOv3[20] sont tous basées sur un réseau Darknet-53 suivie de couche de convolution. Plutôt que d'utiliser un mécanisme d'entraînement négatif ou de prédire beaucoup de boîtes de tailles différentes, les deux dernières versions préfèrent effectuer trois tailles de détection de boîte englobante, sur lesquels les classifications seront faites.

2.3 Ajout intéressant dans le contexte de détection de feux de forêt

Il a été trouvé dans la littérature que l'on peut décider de combiner notre système de détection d'objet avec un système

ne permettant de suivre le déplacement d'un objet [21]. Dans notre cas, cela pourrait être intéressant pour effectuer le suivi d'une colonne de fumée qui se déplace avec le vent ou encore le déplacement de la limite d'un feu de forêt. Un autre ajout qui pourrait être pertinent serait de rendre la détection de feux de forêt moins dépendante des conditions météorologiques [22]. Il est possible qu'un feu de forêt commence lors de courtes pluies avec la présence d'éclairs, mais si notre modèle a été entraîné uniquement par beau temps il risque d'avoir plus de difficulté à faire une bonne détection.

Ces deux méthodes pourraient être intéressantes pour des travaux futurs, mais ne seront pas implémentées dans notre projet.

3 Approche théorique

3.1 Qu'est-ce que YOLO ?

L'algorithme "You Only Look Once", abrégé YOLO, a fait son irruption dans le domaine de la vision par ordinateur et des réseaux de neurones en 2016 [16]. Il s'est fait remarquer par ses performances et sa capacité à détecter des objets en temps réel. YOLO repose sur les trois points suivants :

- La division en grille : l'image fournie au réseau est divisé par une grille de dimension $S \times S$
- Les boîtes englobantes (Bounding Box)
- L'intersection sur l'union (IoU)

La division en grille

Tout d'abord, l'image est redimensionnée selon une dimension prédéfinie puis divisée en plusieurs grilles. Chaque grille a une dimension de $S \times S$.

Bounding Box et IoU

Pour chacune des cases de la grille, l'algorithme cherchera à détecter les objets qu'elle contient en essayant de prédire une boîte englobante. Il s'agit d'un cadre qui met en évidence un objet en l'entourant. Une boîte est donc définie par sa largeur, sa hauteur, son centre et la classe de l'objet qu'elle entoure avec le score de confiance d'appartenance à cette classe. De plus, l'intersection sur l'union (IoU) permet de mesurer le chevauchement entre la boîte prédite et la boîte réelle.

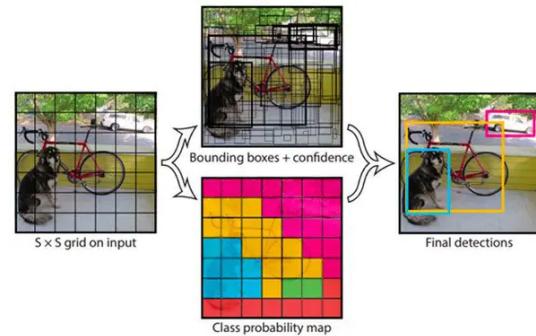


FIGURE 1 – A gauche, l'image divisée par une grille. En haut, la détection des bounding box. En bas, chaque case est colorée selon la $P(\text{Classe} | \text{Objet})$. A droite le résultat final.[16]

$$IoU = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

où A est la boîte prédite et B est la boîte réelle. On calcule ainsi l'aire de l'intersection sur l'aire de l'union des deux boîtes. Si IoU vaut 1 alors la boîte prédite est la même que la boîte réelle. Le modèle calcule ensuite c probabilités conditionnelles $P(\text{Classe} | \text{Objet})$ évaluant la probabilité de la classe sachant qu'il y a un objet (ou non) dans la boîte. A partir de ces éléments, un score de confiance est calculé comme suit :

$$\begin{aligned} \text{Confiance} &= P(\text{Classe} | \text{Objet}) * P(\text{Objet}) * IoU \\ &= P(\text{Classe}) * IoU \end{aligned} \quad (2)$$

où $P(\text{Objet})$ est la probabilité que la case contienne un objet (1 ou 0). Cette score de confiance sert à refléter à quel point le modèle est sûr que la boîte contient un objet et à quel point il pense que la boîte est bien placée.

L'ensemble de ces étapes sont décrisées dans la figure 1. On remarque que pour le résultat final (à droite de la figure 1), YOLO fait un choix sur les boîtes à garder. Cela repose sur le "Non-Maximum Suppression" (NMS) qui est une méthode permettant de ne sélectionner qu'une seule entité parmi toutes celles qui se chevauchent. Il est également à noter que toutes les prédictions d'objets sont effectués simultanément grâce à un réseau de neurones convolutif que l'on décrira par la suite.

3.2 YOLOv5

Améliorations apportées

YOLOv5[1] fait suite à YOLOv4 [19] qui apportait son lot d'améliorations majeurs au modèle initial. En effet, YOLOv4 intégrait des stratégies d'entraînement (BoF pour "bag of freebies") avec des techniques d'augmentation de données mais aussi de plusieurs méthodes de post-traitement (BoS pour "bag of specials"). Les BoF améliorent la précision du détecteur, sans augmenter le temps d'inférence. Les BoS augmentent légèrement le coût d'inférence mais améliorent significativement la précision de la détection d'objets. YOLOv5 est apparu seulement deux mois après YOLOv4. Il s'agit en fait d'une implémentations PyTorch de la version 4. Les changements principaux sont l'augmentation de données par mosaïques et la mise en place de différentes tailles de modèle avec une profondeur et une largeur de réseau (tailles nano, small, medium, large, extralarge).

Architecture

L'architecture est décrite sur la figure 2. YOLO et notamment YOLOv5 se décompose en trois parties :

- **Backbone** : Le backbone du modèle est utilisé pour extraire des caractéristiques importantes de l'image d'entrée. Dans YOLO v5, il s'agit d'un réseau CSP-Darknet53 qui effectue l'extraction des attributs grâce à plusieurs couches convolutives.
- **Neck** : Le cou est une pyramide de maxpooling qui permet de faire un pré-traitement des attributs extraits.
- **Head** : La tête, identique à celle de YOLOv3 permet de déterminer les boîtes englobantes des objets et de les classifier.

De plus, YOLOv5 utilise comme fonction d'activation les fonctions Leaky ReLU et Sigmoïde. La première est utilisée dans les couches intermédiaires et la fonction sigmoïde est utilisée dans la couche de détection finale. Le modèle permet également de choisir entre la méthode SGD ou Adam pour la mise à jour des poids lors de la rétropropagation. Enfin, la fonction de perte mise en place par les auteurs de YOLOv5 est l'entropie croisée binaire de PyTorch[23] :

$$\begin{aligned} l(x, y) &= (l_1, \dots, l_N)^T, \\ l_n &= -w_n[y_n \log(\sigma(x_n)) + (1 - y_n) * \log(1 - \sigma(x_n))] \end{aligned} \quad (3)$$

3.3 Différences entre YOLOv5 et MVMNet

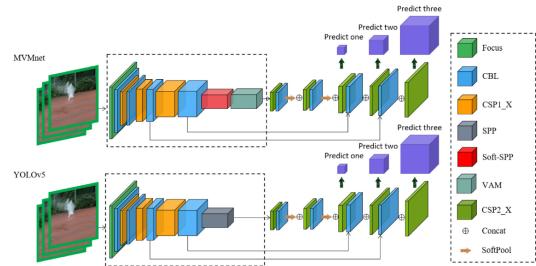


FIGURE 2 – Architecture de YOLOv5 et MVMNet.[3]

La différence avec MVMNet, modèle avec lequel nous comparerons nos résultats, réside principalement dans les modules présents dans l'architecture du réseau. En effet, comme nous pouvons le voir sur la figure 2, le modèle MVMNet utilise un module Soft-SPP plutôt qu'un SPP. SPP est une couche convulsive qui utilise la mise en commun de pyramides spatiales pour supprimer la contrainte de taille fixe du réseau. Autrement dit, la couche SPP permet de générer une représentation de longueur fixe indépendamment de la taille initiale de l'image. MVMNet utilise également un VAM qui est une couche d'attention qui utilise les informations sur les textures et couleurs et de texture de l'image. Cela est notamment dû au fait que l'objectif d'utiliser MVMNet pour la détection de fumée est aussi de déterminer la direction de la fumée de manière la plus précise. C'est pourquoi, les bounding box prédites par ce modèle intègrent également un paramètre θ , angle décrivant l'orientation de la fumée sur l'image. Ainsi, la fonction de perte de MVMNet prend en compte ces différences avec YOLOv5 en combinant la fonction de perte de régression, la fonction de perte du VAM et celle liée à l'angle.

3.4 Augmentation de donnée

Le but de l'augmentation de données est de créer artificiellement de nouvelles données à travers différentes techniques. Nous en avons utilisé certaines dans le cadre de notre projet afin de développer un modèle plus robuste et plus performant. Voici les techniques que nous avons utilisé aux cours de nos expériences :

- **Mosaïque** : Il s'agit de faire une combinaison de 4 images en 1, où chacune des images à un ratio différent

- pour certaines images durant l'entraînement selon une probabilité fixée ;
- **Mix-Up** : Il s'agit de faire un mélange pondéré d'images pour certaines images selon une probabilité fixée ;
 - **Copier-Coller** : Il s'agit de copier des portions l'image correspondant aux éléments à détecter puis à les coller sur d'autres images. Ceci est effectué pour certaines images durant l'entraînement selon une probabilité fixée ;
 - **Rotation** : Il s'agit d'effectuer une rotation d'un degré θ de l'image ;
 - **Translation** : Il s'agit d'effectuer une translation δ de l'image ;
 - **L'augmentation/diminution d'échelle** : Il s'agit de redimensionner l'image selon un ratio ;
 - **Cisaillement (Shearing)** : Il s'agit d'effectuer une transformation affine en décalant dans des directions opposées le haut et le bas de l'image et la droite et la gauche de celle-ci ;
 - **Perspective** : Il s'agit d'ajouter de la perspective à l'image selon une fraction ρ .

4 Expériences et analyse des résultat

4.1 Test de taille du modèle et choix de l'optimiseur

L'objectif de cette expérience est de choisir la taille optimale du modèle, autant en largeur qu'en profondeur. L'architecture YOLOv5 propose 5 tailles différentes. Le modèle large est considéré comme le modèle complet avec un facteur de mise à l'échelle pour la largeur et la profondeur de 1. Les modèles large et extralarge ne pourront pas être analysés ici comme les cartes graphiques à notre disposition n'ont pas assez (NVIDIA Tesla P100 PCIe 16 GB) de VRAM pour contenir le modèle. Il nous reste donc le modèle nano, petit et moyen qui peuvent rentrer dans la mémoire de notre carte graphique. Nous avons décidé de comparer les modèles petits et moyens, qui ont respectivement un facteur de mise à l'échelle de profondeur de 0.33 et 0.67 et de largeur de 0.5 et 0.75. La deuxième partie de cette expérience est de comparer les méthodes d'optimisation des paramètres. On a testé la descente de gradient stochastique (SGD) et l'estimation adaptative du moment (Adam). Ces tests ont été effectués selon les paramètres suivants :

- 100 epochs
- Taille de batch de 32
- Le premier dataset a été utilisé

Les résultats obtenus sont dans les tableau 1 et tableau 2 ci-dessous.

Tests	Précision	Rappel	mAP 0.5	mAP 0.5 :0.95
SGD petit	92.09%	94.59%	97.44%	56.65%
SGD moyen	95.45%	85.10%	95.48%	52.04%
Adam petit	94.59%	94.54%	96.12%	55.22%

TABLE 1 – Taille du modèle et choix de l'optimiseur - performance

Tests	Entraînement		Validation	
	Boites	Objets	Boites	objets
SGD petit	0.006461	0.02531	0.02919	0.005412
SGD moyen	0.009499	0.03479	0.03237	0.006053
Adam petit	0.007623	0.02981	0.03081	0.005220

TABLE 2 – Taille du modèle et choix de l'optimiseur - perte

Les courbes de performance et de perte des modèles entraînés d'où sont prises, les valeurs des tables précédentes sont disponibles en annexe. Pour la première partie on a pu remarquer que le modèle de petite taille offre de meilleures performances que le modèle de taille moyenne sur toute la durée d'entraînement pour ce qui est du rappel et des précisions mAP 0.5, mAP 0.5 :0.95. Le seul point où le modèle de taille moyenne semble avoir un avantage est pour ce qui est de la précision, mais il s'agit du pic ponctuel et non d'une tendance. Cependant, il est intéressant de remarquer que le modèle moyen prend environ deux fois plus de temps à entraîner pour un gain relativement faible. Il est donc de décider de continuer notre analyse en utilisant le modèle de petite taille. En utilisant l'algorithme d'optimisation de paramètre Adam avec les paramètres par défaut de YOLOv5 on remarque qu'il est moins bon dans toutes les courbes comparées à SGD sauf pour la précision. Tous comme dans l'analyse de la taille du modèle SGD semblent plus constants et offrent une convergence plus rapide. On décide donc de prendre SGD avec un modèle de petite taille dans le reste de nos tests et pour le modèle final.

4.2 Test Couches de convolution concentrée

L'objectif de cette expérience est de tester deux types de couches convolutives concentrée dans les réseaux afin de choisir celles offrant les performances et pertes optimales. Une couche convolution concentrée-complète applique des convolutions asymétriques avant la convolution standard afin de compenser la perte d'information liée à cette dernière. Pour cela, nous avons testé deux modules de convolutions concentrées : C3 et C3Ghost. Ce test a été effectué selon les paramètres suivants :

- 50 epochs
- Taille de batch de 32
- Le premier dataset a été utilisé

Les résultats obtenus sont dans les Table 3 et 4 ci-dessous.

Tests	Précision	Rappel	mAP 0.5	mAP 0.5 :0.95
C3	93.04%	90.54%	96.52%	57.23%
C3Ghost	94.18%	91.89%	95.12%	46.12%

TABLE 3 – Choix du type de couches de convolution - performance

Tests	Entraînement		Validation	
	Boites	Objets	Boites	objets
C3	0.0286	0.00758	0.0306	0.00545
C3Ghost	0.0377	0.00987	0.0345	0.00599

TABLE 4 – Choix du type de couches de convolution - perte

Les courbes dont sont issus les résultats présentés ci-dessus sont disponibles en annexe de ce rapport. Nous remarquons qu'une couche convective de type C3Ghost offre de meilleure performance entre terme de précision et de rappel. De plus, le temps de calcul était moindre qu'avec C3 avec 14min32s contre 15min2s. Néanmoins, une couche standard C3 de convolution concentrée permet de mieux minimiser la perte de détection et obtient une meilleure performance en termes de mAP. C'est pourquoi nous avons fait le choix de conserver C3 pour notre architecture finale de YOLOv5 et la suite des expériences.

4.3 Méthode d'augmentation de données

L'objectif de cette expérience est d'analyser l'impact des différentes méthodes d'augmentation de données afin de choisir une combinaison qui maximise la performance de notre modèle. Pour cela, nous avons testé les 8 techniques suivantes : Mosaïque, Mix-Up, Copier-Coller, Rotation, Translation, L'augmentation/diminution d'échelle(Scale), Cisaillement(Shearing) et Perspective. Nous avons d'abord testé notre modèle avec ces hyperparamètres tous égaux à zéro. Ensuite, nous avons assigné une nouvelle valeur à chacun des hyperparamètres et les testés indépendamment. Ce test a été effectué selon les paramètres suivants :

- 50 epochs
- Taille de batch de 32
- Le premier dataset a été utilisé

Les résultats obtenus sont dans les Table 5 et 6 ci-dessous.

Tests	Précision	Rappel	mAP 0.5	mAP 0.5 :0.95
Base(Tous à zéro)	83.46%	77.71%	83.73%	33.17%
Mosaïque=1.0	92.38%	78.46%	89.49%	38.68%
Mixup=0.3	83.46%	77.71%	83.73%	33.17%
Copier-Coller=0.3	83.46%	77.71%	83.73%	33.17%
Rotation=2.5	91.72%	78.57%	88.37%	37.04%
Translation=0.1	92.03%	79.07%	89.06%	39.07%
Échelle=0.5	65.70%	75.25%	75.04%	25.55%
Cisaillement=2.5	84.04%	71.12%	81.18%	31.83%
Perspective=0.0005	93.99%	76.33%	89.70%	38.96%

TABLE 5 – Tests indépendants des méthodes d'augmentation de données - performance

Tests	Entraînement		Validation	
	Boites	Objets	Boites	objets
Base(Tous à zéro)	0.03445	0.005561	0.03999	0.00729
Mosaïque=1.0	0.04013	0.009535	0.03845	0.00704
Mixup=0.3	0.03445	0.005561	0.03999	0.00729
Copier-Coller=0.3	0.03445	0.005561	0.03999	0.00729
Rotation=2.5	0.03682	0.005879	0.03873	0.00701
Translation=0.1	0.03983	0.006187	0.03791	0.00713
Échelle=0.5	0.04620	0.006916	0.04452	0.00758
Cisaillement=2.5	0.03885	0.006200	0.042100	0.00761
Perspective=0.0005	0.03876	0.006009	0.03974	0.00708

TABLE 6 – Tests indépendants des méthodes d'augmentation de données - perte

Les courbes dont sont issus les résultats présentés ci-dessus sont disponibles en annexe de ce rapport. Nous remarquons

que seulement 4 techniques (Mosaïque, Rotation, Translation, Perspective) parmi les 8 ont un impact positif sur la performance et la perte (validation) de notre modèle. Les 8 techniques ont un impact négatif sur la perte (entraînement) de notre modèle. Les techniques Mixup et Copier-coller n'ont aucun impact sur notre modèle en performance et en perte. Et les techniques échelle et cisaillement ont un impact négatif sur notre modèle en performance et en perte. Pour trouver la meilleure combinaison des techniques, nous avons choisi trois techniques qui ont le grand impact positif (Mosaïque, Perspective et Rotation) et testé leurs combinaisons dans le but de trouver une combinaison qui maximise la performance de notre modèle.

Tests	Précision	Rappel	mAP 0.5	mAP 0.5 :0.95
Mos & Rota	90.44%	86.77%	91.98%	40.53%
Pers & Rota	78.82%	80.00%	82.95%	31.64%
Pers & Mos	89.94%	88.10%	92.68%	38.35%
Pers,Mos & Rota	87.36%	84.88%	89.77%	36.68%

TABLE 7 – Tests de combinaison des méthodes d'augmentation de données - performance

Tests	Entraînement		Validation	
	Boites	Objets	Boites	objets
Mos & Rota	0.04045	0.009785	0.03736	0.006692
Pers & Rota	0.04074	0.006426	0.04091	0.007536
Pers & Mos	0.04190	0.010490	0.03744	0.006863
Pers,Mos & Rota	0.04240	0.010400	0.04084	0.007118

TABLE 8 – Tests de combinaison des méthodes d'augmentation de données - perte

Les courbes dont sont issus les résultats présentés ci-dessus sont disponibles en annexe de ce rapport. Nous remarquons que la combinaison des techniques Perspective et Mosaïque nous donne la meilleure performance parce que cette combinaison nous donne le plus haut rappel et une précision un peu plus bas que la combinaison de Mosaïque et Rotation. Cependant, cette combinaison donne une moins bonne perte pour notre modèle.

4.4 Datasets multiples

L'objectif de cette expérience est de tester l'utilisation de différents datasets de fumée et observer l'effet de ceux-ci tant sur la qualité des résultats que sur le temps d'entraînement afin de pouvoir trouver si l'ajout de divers datasets améliore ou diminue la qualité de notre réseau. On a trouvé 3 datasets pertinents à notre projet, soit smoke_dataset_v1, smoke_dataset_v2 et cloud_dataset. smoke_dataset_v1 et smoke_dataset_v2 contiennent des images identifiées en forêt contenant des étiquettes de fumée. cloud_dataset contient des images non-identifiées de fumée en forêt et sans fumée (surtout des nuages).

Pour cela, on entraîne notre réseau avec les hyperparamètres optimaux trouvés précédemment. Comme expériences, on a l'utilisation du dataset smoke_dataset_v1 (nommé v1) seul pour l'entraînement, la validation et le test. Ensuite, on a l'utilisation du dataset smoke_dataset_v1 et

smoke_dataset_v2 (nommé v1 et v2) pour l'entraînement, la validation et le test. Notre dernière expérience est l'utilisation des datasets smoke_dataset_v1, smoke_dataset_v2 et cloud_dataset pour l'entraînement et la validation et les datasets smoke_dataset_v1, smoke_dataset_v2 pour le test (nommé v1, v2 et cloud). On effectue les tests avec les mêmes databases pour l'expérience avec 2 databases et celle avec 3 databases afin de pouvoir les comparer plus facilement. Finalement, après avoir observé les résultats de notre expérience avec seulement un dataset, on a effectué une expérience pour évaluer les vraies performances du réseau, en ajoutant des images différentes dans la validation pour éviter le sur-apprentissage. Ce test a été effectué avec les paramètres suivants :

- 100 epochs
- Taille de batch de 32
- Hyperparamètres optimaux

Les résultats obtenus sont dans la Table 9 ci-dessous.

Datasets	Précision	Rappel	mAP 0.5	mAP 0.5 : 0.95
v1	0.9595	0.9594	0.9866	0.6012
Actuel 1 : v1	0.7112	0.4601	0.4947	0.2397
v1, v2	0.7926	0.7309	0.7795	0.3789
v1, v2, clouds	0.7860	0.7221	0.7485	0.3558

TABLE 9 – Choix des datasets utilisés pour l'entraînement - performance

Nos entraînements utilisant nos divers datasets ont pris un temps très différent. D'abord, avec un dataset, cela a pris environ 30 minutes de temps d'exécution avec Google Colab Pro. Pour deux datasets, cela a pris environ 2 heures. Finalement, pour les trois datasets, cela a pris environ 5 heures. Ainsi, ajouter plus de données augmente considérablement notre temps d'exécution.

On remarque que notre entraînement avec seulement le dataset v1 est trop précis, ce qui est inquiétant. En effet, après avoir analysé les photos du dataset, plusieurs images se ressemblent extrêmement, ce qui rend notre modèle surentraîné, malgré le fait que les images soient différentes. Aussi, puisque le nombre de données est assez petit, on peut s'attendre à des résultats qui ne se généralisent pas très bien. Notre dernière expérience avec une validation qui comprend des images hors du dataset v1 nous indique qu'on a eu en effet affaire à une situation de sur-apprentissage, car le réseau ne s'adapte pas bien à de nouvelles données.

Pour nos entraînements avec 2 et 3 datasets, on obtient des résultats très similaires, ce qui nous indique que les nuages dans les images avec de la fumée sont suffisants pour que notre réseau ne confonde pas les nuages avec la fumée.

On a aussi appliqué nos réseaux entraînés sur des données externes aux trois datasets afin d'évaluer leurs performances. On a trouvé que pour une image qui est d'un point de vue similaire à nos images d'entraînement, on a des résultats assez similaires, sauf pour notre modèle entraîné avec v1 uniquement qui parfois manque de détecter une fumée, tel que montré à la figure 3. Pour une image complètement différente [24], le seul modèle qui avait des résultats décents était celui de l'expérience avec deux datasets (v1 et v2), tel que montré

à la figure 4.



FIGURE 3 – Application du réseau sur une image similaire aux images. Images de gauche à droite : Dataset v1; Dataset v1 et v2;Dataset v1, v2 et cloud



FIGURE 4 – Application du réseau sur une image différente aux images. Images de gauche à droite : Dataset v1; Dataset v1 et v2;Dataset v1, v2 et cloud

5 Analyse critique de l'approche

5.1 Comparaison MVMNet

Avec l'ensemble des expériences que nous avons réalisé, nous pouvons comparer nos résultats avec l'article sur MVM-Net [3]. Bien que nos datasets soient différents de ceux utilisés dans l'article, nous remarquons que nous avons réussi à reproduire le niveau de performance pour YOLOv5 selon le critère mAP0.5. Cependant MVMNet conserve l'avantage avec un mAP0.5 de 88.05% contre 78% pour notre YOLOv5. Ce résultat était attendu, comme MVMNet est une version de YOLO amélioré pour la détection de boîte en diagonale. Ce type de détection s'avère être un atout pour reconnaître une colonne de fumée.

5.2 Avantages de la méthode

On a obtenu des résultats très intéressants avec notre réseau, avec des temps d'exécutions très bas (entre 0.015s et 0.020s, ce qui donne entre 50 et 60 fps). On a donc une méthode portable qui permet d'avoir de bonnes performances en temps réel. De plus, cette méthode analyse image par image, ce qui permet d'éviter des erreurs liées à un déplacement de la caméra, comme avec une technique de HOG ou de superposition d'images.

La taille finale du modèle entraîné pourrait être un avantage significatif pour un certain type d'application. Dans les comparaisons de YOLOv5 disponibles sur leur GitHub [1] on remarque que les modèles large et extra large ont une performance largement supérieure au modèle de petite taille. Cependant, nous avons remarqué que cet avantage ne tient plus lorsque nous avons qu'une ou deux classes où il devient possible d'avoir des résultats comparables entre les modèles large et petit. Dans notre cas particulier les poids et biais de notre modèle ne prennent que 14MB en comparaison avec la moyenne de 166MB du modèle extra large, tout en offrant de bonne performance.

5.3 Amélioration possible

Quant aux datasets utilisés, puisque notre domaine d'application est très niche, il est difficile de trouver une grande quantité d'images variées pour entraîner notre réseau. En effet, on remarque que nos datasets utilisés ont pratiquement tous un angle de vue provenant d'une station météo et ont une résolution similaire. Les photos sont également prises avec une caméra avec un objectif fish-eye . Ainsi, notre modèle se généralise mal à un nouvel environnement, mais peut parfois avoir des résultats décents, tel qu'exploré dans la figure 4. Dans notre domaine d'application, il serait ainsi intéressant de créer de nouveaux datasets avec des expériences nouvelles recueillies par la même caméra qui utilise notre réseau afin de l'améliorer constamment. Aussi, quant à l'utilisation d'un dataset de nuages sans fumée pour augmenter nos données, on s'est rendu compte que cette approche augmentait considérablement nos temps d'entraînement sans augmenter nos performances, ce qui nous indique que cette méthode n'est pas nécessairement appropriée à notre réseau, considérant la quantité énorme de nuages déjà présents dans nos datasets avec fumée. Il pourrait cependant être intéressant de combiner un réseau de détection de fumée avec un réseau de détection de nuages afin de comparer manuellement les boîtes englobantes et utiliser ces résultats dans l'entraînement.

Une deuxième possibilité pour améliorer les performances de notre modèle serait de faire une sélection des hyperparamètres de façon autonome. Pour l'instant, les hyperparamètre qui ont été tester ainsi que leur valeur ont été choisis manuellement et la sélection du modèle final a été fait incrémentales pour sauver du temps par rapport à une recherche exhaustive de toutes les combinaisons possibles. Depuis longtemps la science s'intéresse à ce qui se passe dans la nature et essaye de s'en inspirer dans plusieurs domaines technologiques. De cela, une méthode de recherche d'hyperparamètres basé sur des algorithmes de recherche génétique a été développée [25] [26]. Ces algorithmes bien connus en recherche opérationnelle se prêtent bien à la recherche d'hyperparamètre. L'idée de base est de générer de nouvelle combinaison en croisant des parents provenant d'une sélection de solutions prometteuses pris dans une population de solution. Ce croisement inclut aussi une composante aléatoire sous forme de mutation. Chaque parent ou enfant équivaut à un vecteur des valeurs des hyperparamètres et la population est l'ensemble des combinaisons possibles. Cette méthode va donc de façons semi-aléatoires chercher au travers des combinaisons possibles et a de bonnes chances de converger vers une bonne combinaison plus vite qu'une recherche purement exhaustive. Cette idée a été implémenté dans la version de YOLOv5 disponible sur leur GitHub [1]. Nous avons cependant décidé de nous contenter d'une recherche manuelle, comme selon les témoignages des auteurs de YOLOv5 converger vers une bonne combinaison d'hyperparamètre avec l'algorithme génétique peut prendre plusieurs centaines d'heures de calcul GPU.

6 Conclusion

En conclusion, plusieurs méthodes ont été testées pour augmenter les performances de nos instances de base. La combinaison de multiples augmentation de données et de plusieurs bases de données, certaines de fumée et d'autres de nuage, nous avons permis d'obtenir suffisamment de données d'entraînement et de test. La sélection de la taille du modèle et du type de couche de convolution nous a permis d'obtenir un modèle entraînable dans un temps réaliste, maximum 5h pour le modèle final. Notre ajustement des hyperparamètres nous a permis d'obtenir des résultats comparables au modèle de comparaison de l'article MVMNet [3].

On a travaillé avec plusieurs contraintes, tant sur nos ressources disponibles que sur la durée restreinte du projet. Malgré nos résultats prometteurs, il y a encore des pistes non explorées qui pourraient permettre d'améliorer nos résultats, comme choisir les hyperparamètres avec un algorithme de recherche génétique ou encore de crée une base de données supplémentaire plus variée avec des images trouvées sur internet.

Une application intéressante de notre modèle serait de pouvoir l'implanter dans un système de caméras de surveillance d'une station météorologique afin de faire fonctionner ce système de détection en temps réelle. Le point clé de la famille de modèle YOLO est la vitesse à laquelle le modèle est capable de donner de bonne prédition. Cette application pourrait être explorée dans des travaux futurs pour permettre une détection préventive de feu de forêt.

7 Bibliographie

- [1] *ultralytics/yolov5*. original-date: 2020-05-18T03:45:11Z. 29 avr. 2022. URL : <https://github.com/ultralytics/yolov5> (visité le 29/04/2022).
- [2] San Diego Supercomputer CENTER. *High Performance Wireless Research and Education Network*. HP-WREN. URL : <http://hpwren.ucsd.edu/> (visité le 27/03/2022).
- [3] Yaowen HU et al. “Fast forest fire smoke detection using MVMNet”. In : *Knowledge-Based Systems* 241 (avr. 2022), p. 108219. ISSN : 09507051. DOI : 10.1016/j.knosys.2022.108219. URL : <https://linkinghub.elsevier.com/retrieve/pii/S0950705122000612> (visité le 27/03/2022).
- [4] *How Is Climate Change Affecting Canada?* en-US. Août 2020. URL : <https://www.worldatlas.com/articles/how-is-climate-change-affecting-canada.html> (visité le 13/04/2022).
- [5] *Les incendies de forêt feraient plus de 300.000 morts chaque année.* fr. Section: Planète. Fév. 2012. URL : https://www.maxisciences.com/feu-de-foret/les-incendies-de-foret-feraient-plus-de-300-000-morts-chaque-annee_art21879.html (visité le 30/04/2022).
- [6] WEI SHUNG CHUNG. *AI for Mankind*. AI For Mankind. URL : <https://aiformankind.org/> (visité le 27/03/2022).

- [7] AI for MANKIND. *Open Wildfire Smoke Datasets*. original-date: 2019-10-04T05:16:33Z. 24 mars 2022. URL : <https://github.com/aiformankind/wildfire-smoke-dataset> (visité le 13/04/2022).
- [8] Ayşegül YANIK et al. *Machine Learning Based Early Fire Detection System using a Low-Cost Drone*. 2021. DOI : 10.48550/ARXIV.2101.09362. URL : <https://arxiv.org/abs/2101.09362>.
- [9] H. KAUFMANN et al. “EnMAP A Hyperspectral Sensor for Environmental Mapping and Analysis”. In : *2006 IEEE International Symposium on Geoscience and Remote Sensing*. 2006, p. 1617-1619. DOI : 10.1109/IGARSS.2006.417.
- [10] Rehan SIDDIQUI et al. *Efficient application of the Radiance Enhancement method for detection of the forest fires due to combustion-originated reflectance*. 2021. DOI : 10.48550/ARXIV.2102.02136. URL : <https://arxiv.org/abs/2102.02136>.
- [11] G CHARPAK et al. “Progress in the development of a S-RETGEM-based detector for an early forest fire warning system”. In : *Journal of Instrumentation* 4.12 (déc. 2009), P12007-P12007. DOI : 10.1088/1748-0221/4/12/p12007. URL : <https://doi.org/10.1088/2F1748-0221%2F4%2F12%2Fp12007>.
- [12] Bobby CHEN. *Object Detection and Tracking*. original-date: 2019-04-18T05:51:23Z. 29 avr. 2022. URL : <https://github.com/yehengchen/Object-Detection-and-Tracking/blob/85a13b9957c07b12e299d9320640f01df33be869/Two-stage%20vs%20One-stage%20Detectors.md> (visité le 29/04/2022).
- [13] Ross GIRSHICK et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. DOI : 10.48550/ARXIV.1311.2524. URL : <https://arxiv.org/abs/1311.2524>.
- [14] Ross GIRSHICK. *Fast R-CNN*. 2015. DOI : 10.48550/ARXIV.1504.08083. URL : <https://arxiv.org/abs/1504.08083>.
- [15] Mingxing TAN et Quoc V. LE. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In : (2019). DOI : 10.48550/ARXIV.1905.11946. URL : <https://arxiv.org/abs/1905.11946>.
- [16] Joseph REDMON et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI : 10.48550/ARXIV.1506.02640. URL : <https://arxiv.org/abs/1506.02640>.
- [17] Tsung-Yi LIN et al. *Focal Loss for Dense Object Detection*. 2017. DOI : 10.48550/ARXIV.1708.02002. URL : <https://arxiv.org/abs/1708.02002>.
- [18] Wei LIU et al. “SSD: Single Shot MultiBox Detector”. In : *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, p. 21-37. DOI : 10.1007/978-3-319-46448-0_2. URL : https://doi.org/10.1007/2F978-3-319-46448-0_2.
- [19] Alexey BOCHKOVSKIY, Chien-Yao WANG et Hong-Yuan Mark LIAO. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. DOI : 10.48550/ARXIV.2004.10934. URL : <https://arxiv.org/abs/2004.10934>.
- [20] Joseph REDMON et Ali FARHADI. *YOLOv3: An Incremental Improvement*. 2018. DOI : 10.48550/ARXIV.1804.02767. URL : <https://arxiv.org/abs/1804.02767>.
- [21] Ruize HAN et al. *Single Object Tracking Research: A Survey*. 2022. DOI : 10.48550/ARXIV.2204.11410. URL : <https://arxiv.org/abs/2204.11410>.
- [22] Saket S. CHATURVEDI, Lan ZHANG et Xiaoyong YUAN. *Pay “Attention” to Adverse Weather: Weather-aware Attention-based Object Detection*. 2022. DOI : 10.48550/ARXIV.2204.10803. URL : <https://arxiv.org/abs/2204.10803>.
- [23] *BCEWITHLOGITSLOSS*. original-date: 29 avr. 2022. URL : <https://pytorch.org/docs/master/generated/torch.nn.BCEWithLogitsLoss.html> (visité le 29/04/2022).
- [24] *Wildfire smoke and air quality*. en. Mai 2021. URL : <https://www.pca.state.mn.us/featured/wildfire-smoke-and-air-quality> (visité le 30/04/2022).
- [25] Chen LI et al. *Genetic Algorithm based hyperparameters optimization for transfer Convolutional Neural Network*. 2021. DOI : 10.48550/ARXIV.2103.03875. URL : <https://arxiv.org/abs/2103.03875>.
- [26] Meng ZHOU. *Heuristic Hyperparameter Optimization for Convolutional Neural Networks using Genetic Algorithm*. 2021. DOI : 10.48550/ARXIV.2112.07087. URL : <https://arxiv.org/abs/2112.07087>.

8 Annexe

8.1 Test de taille du modèle et choix de l'optimiseur

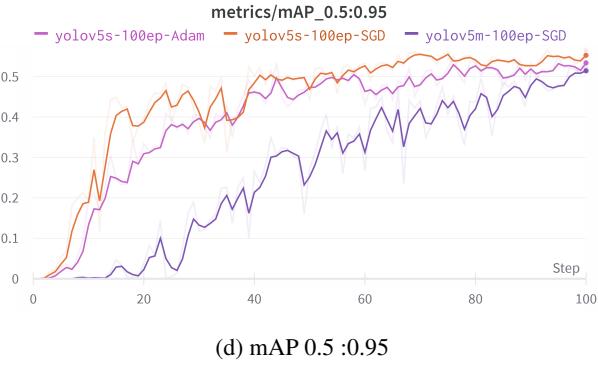
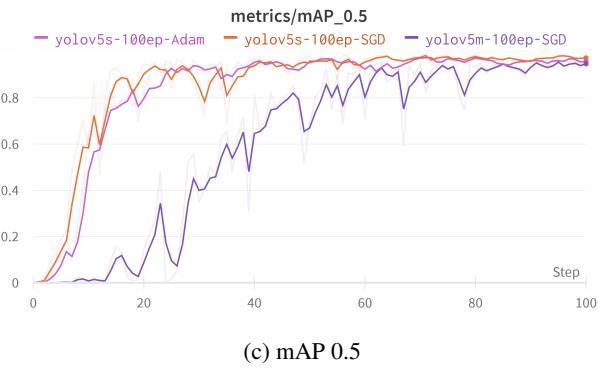
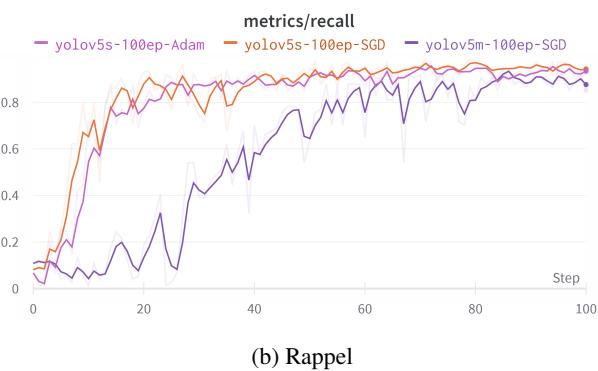
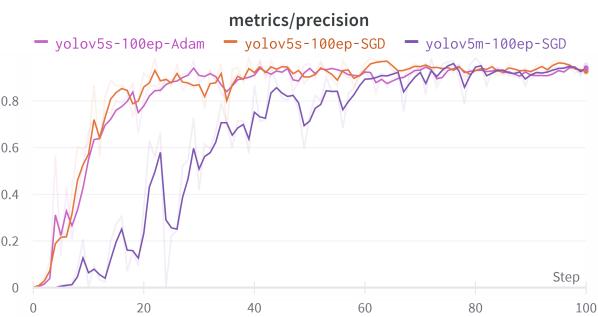


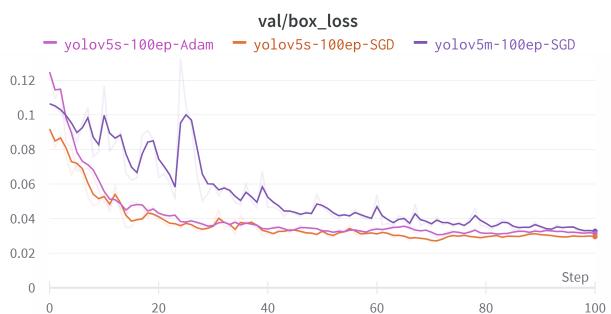
FIGURE 5 – Taille du modèle et choix de l'optimiseur - performance



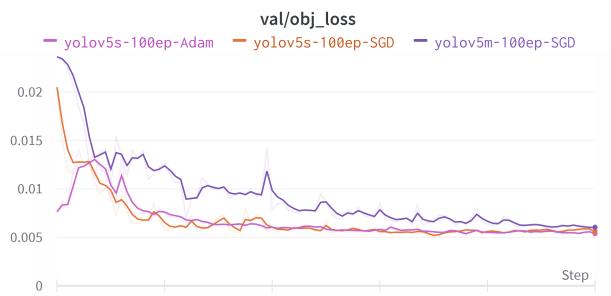
(a) Entraînement - Perte des boîtes englobantes



(b) Entraînement - Perte de classification d'objet



(c) Validation - Perte des boîtes englobantes



(d) Validation - Perte de classification d'objet

FIGURE 6 – Taille du modèle et choix de l'optimiseur - perte

8.2 Test Couches de convolution concentrée

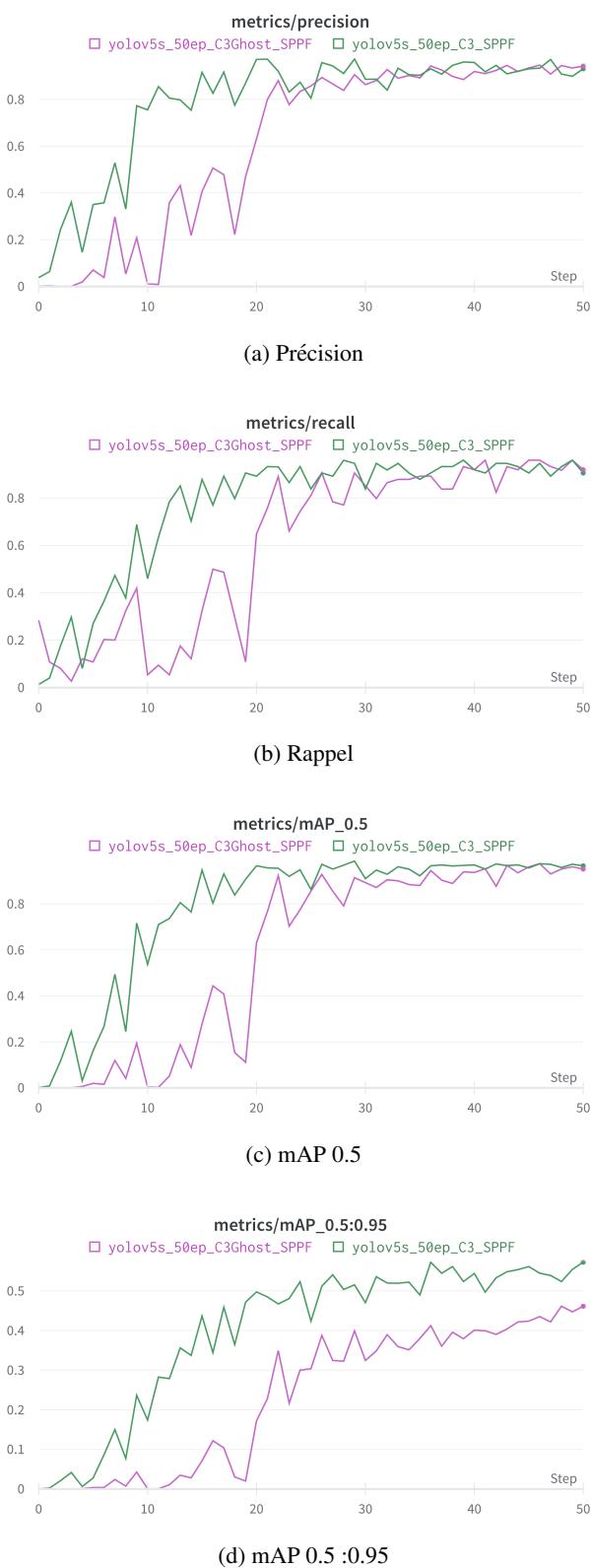


FIGURE 7 – Choix du type de couches de convolution - performance

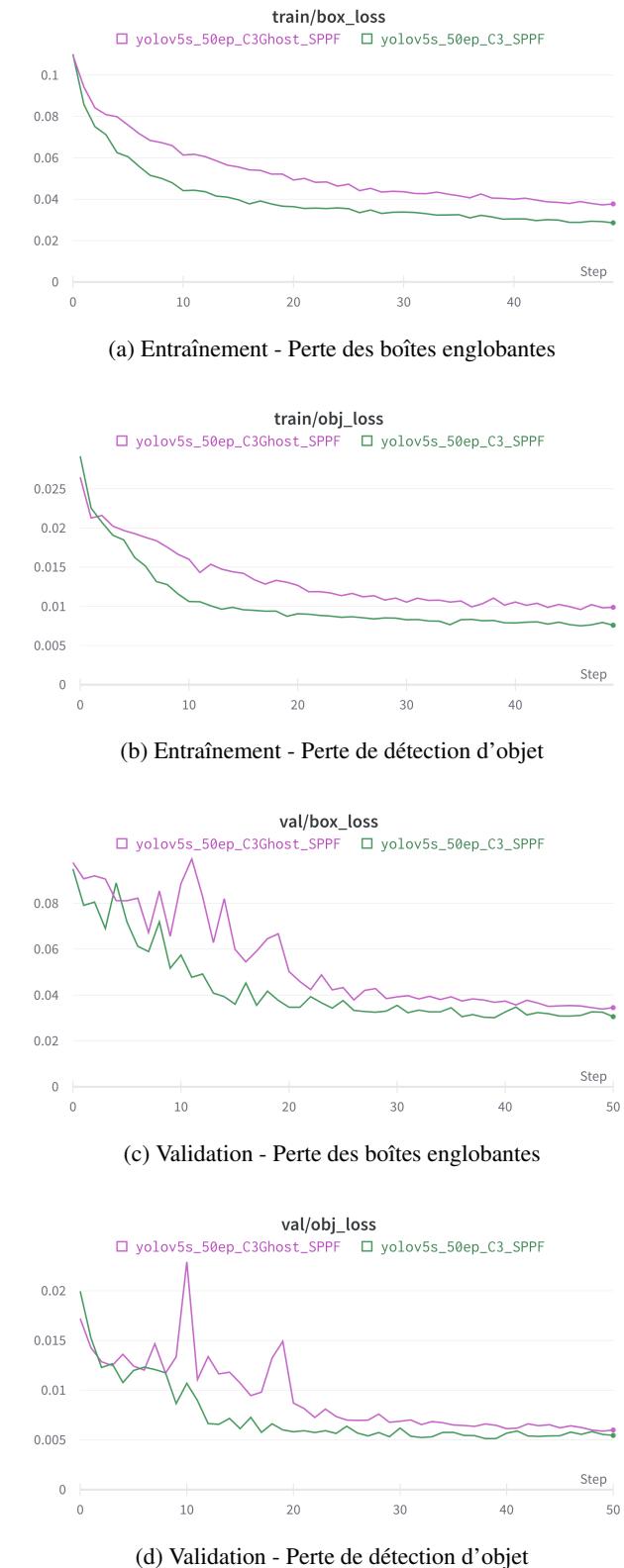


FIGURE 8 – Choix du type de couches de convolution - perte

8.3 Augmentation de données - Test séparé

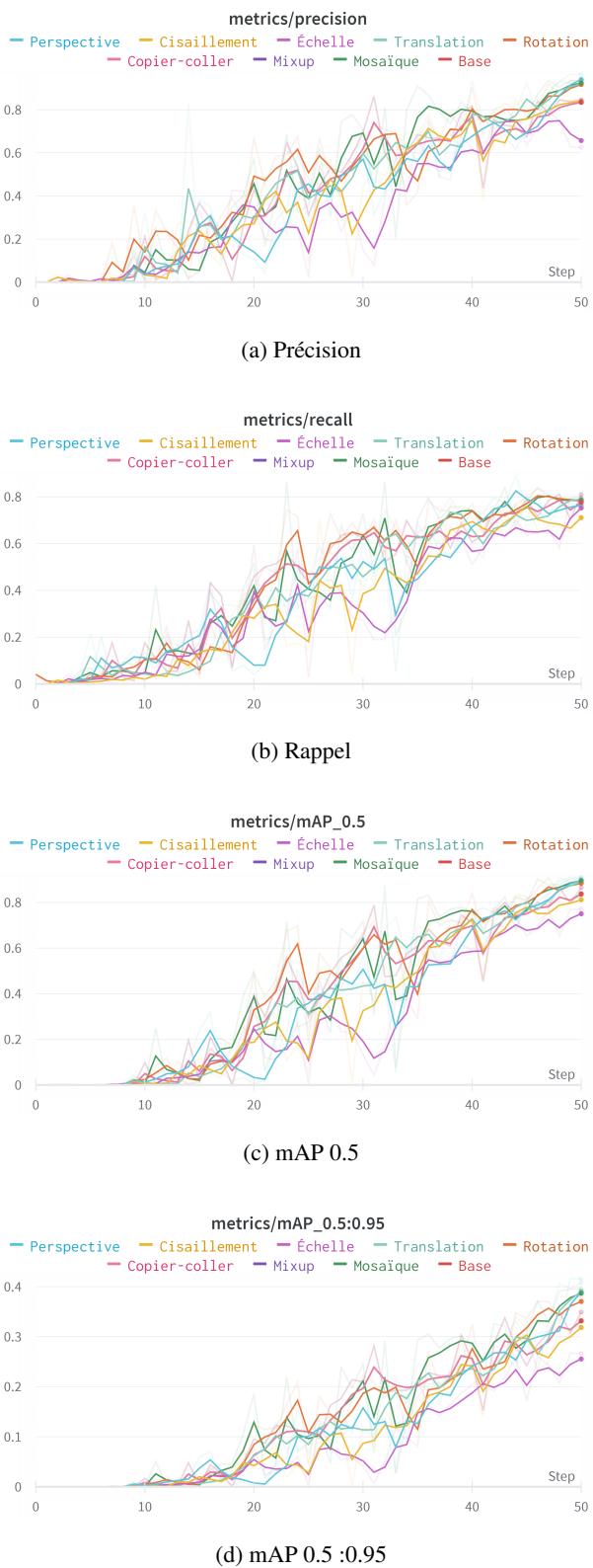


FIGURE 9 – Tests indépendants des méthodes d'augmentation de données - performance

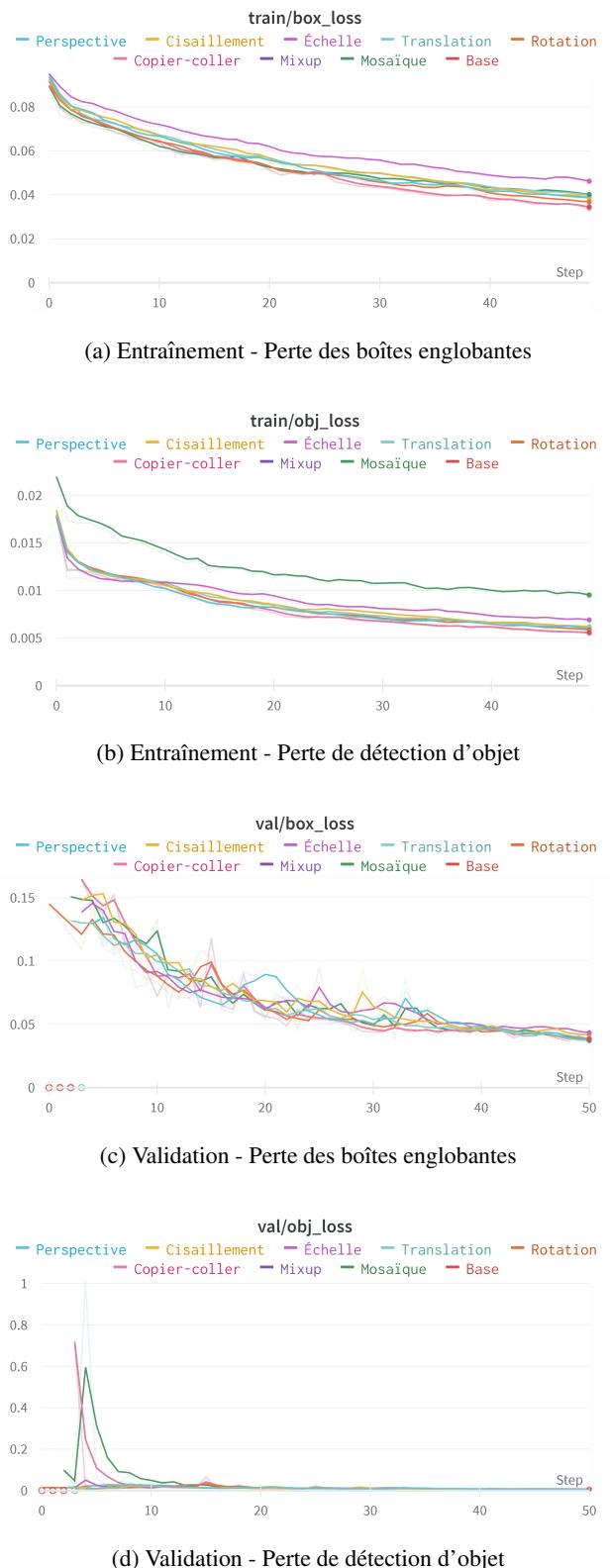


FIGURE 10 – Tests indépendants des méthodes d'augmentation de données - perte

8.4 Augmentation de données - Test en combinaison

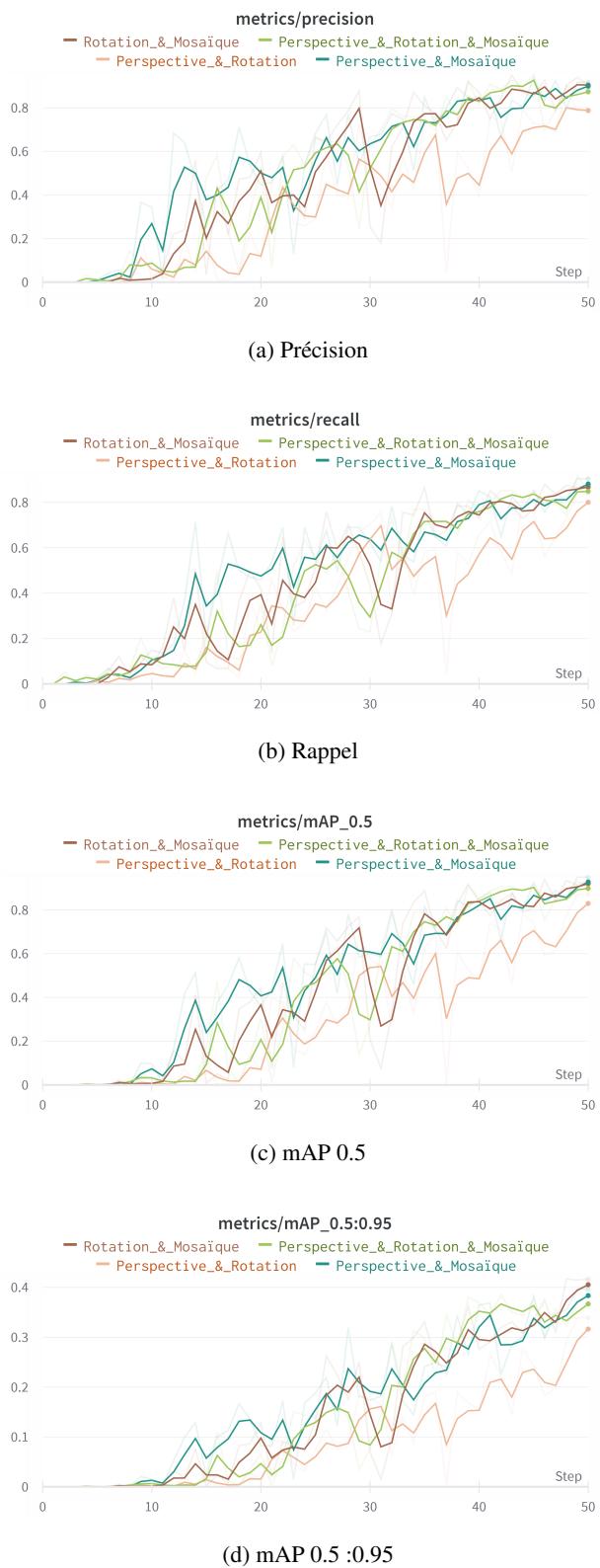


FIGURE 11 – Tests de combinaison des méthodes d'augmentation de données - performance

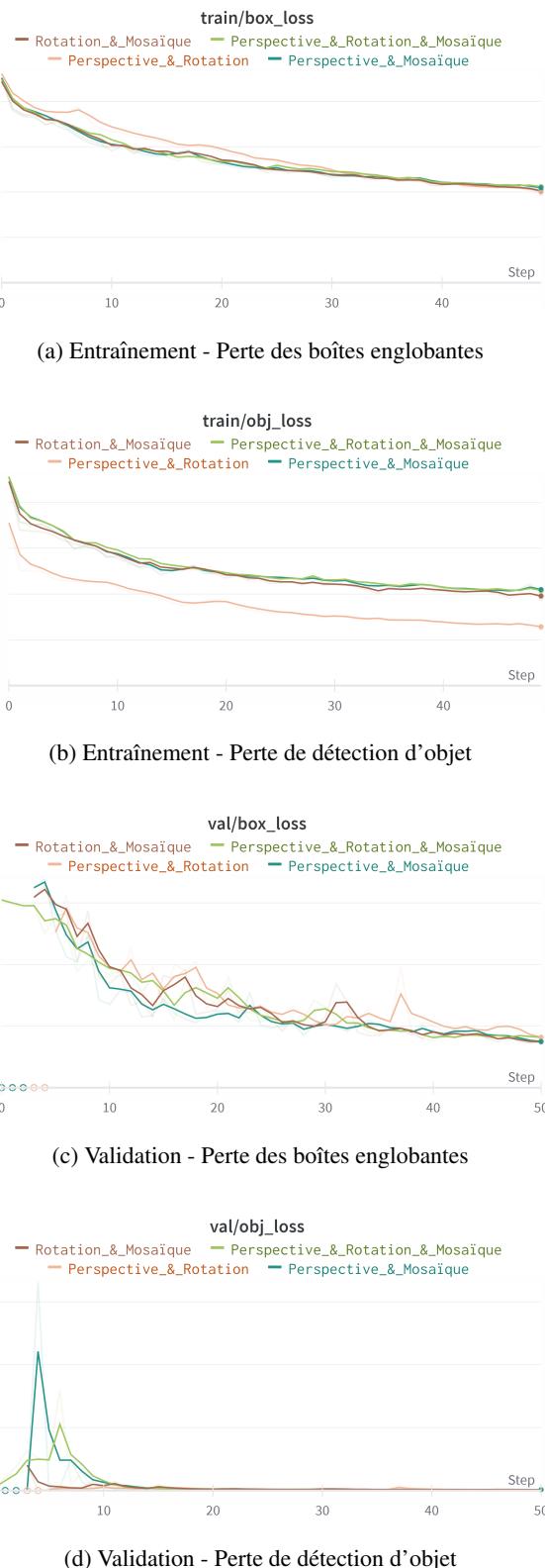
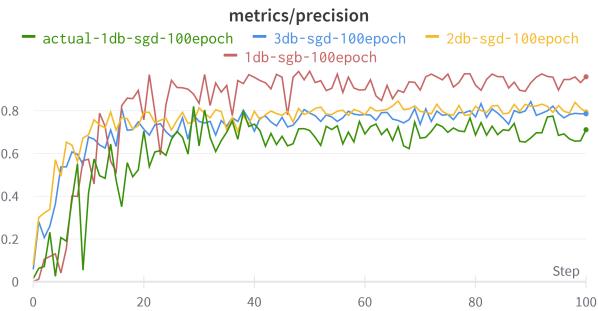
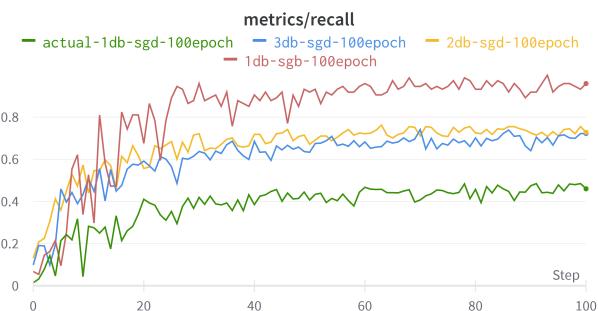


FIGURE 12 – Tests de combinaison des méthodes d'augmentation de données - perte

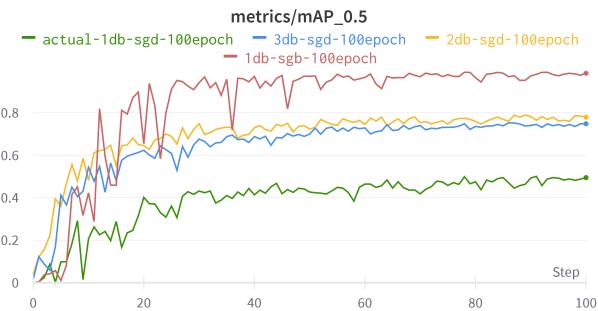
8.5 Test sur les différents datasets



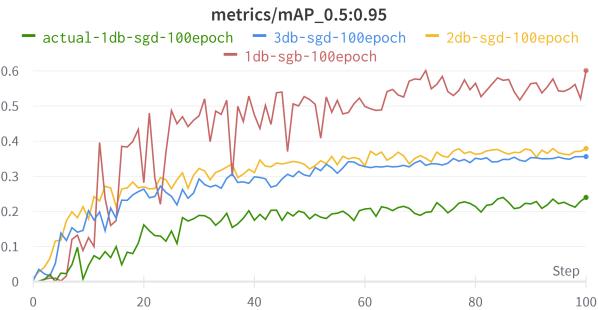
(a) Précision



(b) Rappel

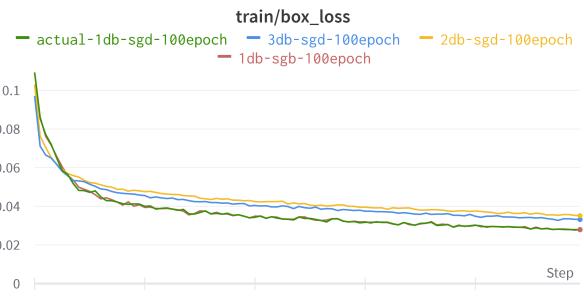


(c) mAP 0.5

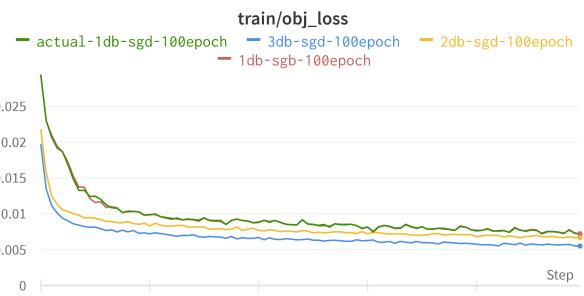


(d) mAP 0.5 :0.95

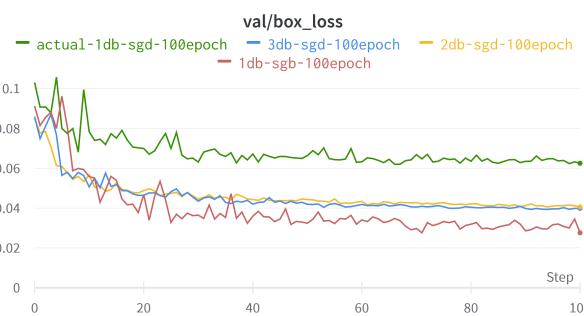
FIGURE 13 – Choix des datasets utilisés pour l'entraînement - performance



(a) Entraînement - Perte des boîtes englobantes



(b) Entraînement - Perte de détection d'objet



(c) Validation - Perte des boîtes englobantes



(d) Validation - Perte de détection d'objet

FIGURE 14 – Choix du type de couches de convolution - perte