

You asked:

fait le 21 a partire des info a faire avec yolo

✦ Gemini Replied:

Absolument ! Voici une marche à suivre détaillée pour réaliser le défi n°21, "Le Nimbus 3000", en utilisant un réseau de neurones de type YOLO, comme vous l'avez suggéré.

L'objectif est de comparer plusieurs "optimizers" pour déterminer lequel est le plus performant pour votre modèle.

✦ Plan d'Action pour le Défi "Le Nimbus 3000"

Voici les étapes pour mener à bien ce benchmark et rédiger votre rapport de recherche.

Étape 1 : Préparation du Réseau de Neurones (YOLO)

Le défi vous permet d'utiliser un réseau de neurones déjà existant. YOLO (You Only Look Once) est un excellent choix pour la reconnaissance d'objets, ce qui correspond bien à l'esprit du défi précédent, "Is it you Harry?".

- **Choix du modèle :** Vous pouvez utiliser une version pré-entraînée de YOLO (par exemple, YOLOv8 de la bibliothèque Ultralytics).
- **Jeu de données (Dataset) :** Pour que la comparaison soit pertinente, vous devriez utiliser le jeu de données que vous avez créé pour le défi n°20, avec les images des 10 personnages de la saga Harry Potter.

Étape 2 : Mise en Place de l'Environnement de Benchmark

Pour comparer efficacement les optimizers, il vous faut un environnement de travail solide.

- **Outils principaux :**
 - Python avec un framework de deep learning comme PyTorch ou TensorFlow.
 - Jupyter Notebook pour l'expérimentation.
- **Suivi des expériences :** Le document mentionne l'utilité de logiciels tiers comme "Weight and Bias" (probablement Weights & Biases). Cet outil est très puissant pour enregistrer et visualiser les performances de chaque entraînement (perte, précision, etc.), ce qui sera crucial pour votre rapport.

Étape 3 : Lancement des Tests (Le Benchmark)

C'est le cœur du défi. Vous allez devoir entraîner votre modèle YOLO plusieurs fois, en ne changeant qu'une seule variable à chaque fois : l'optimizer.

1. **Configurez un entraînement de base** pour votre modèle YOLO sur votre dataset.
2. **Dupliquez cet entraînement** pour chaque optimizer de la liste :
 - Adadelta
 - Adagrad
 - RMSProp
 - Adam
 - SGD
 - Adan
 - AdamW
3. **Lancez chaque entraînement** et enregistrez méticuleusement les résultats à l'aide de Weights & Biases. Les métriques importantes à suivre sont :
 - La vitesse de convergence (à quelle vitesse la "perte" diminue).
 - La précision finale du modèle.
 - L'utilisation des ressources (temps de calcul, mémoire).

Étape 4 : Rédaction du Rapport de Recherche

Le livrable final est une documentation de type "papier de recherche". La consigne "générée à l'aide de votre baguette magique" peut être interprétée comme le fait d'utiliser des outils pour générer automatiquement vos graphiques et tableaux à partir des données collectées.

Votre document doit contenir :

- **Introduction** : Présentez l'objectif du benchmark, le réseau de neurones utilisé (YOLO) et le jeu de données.
- **Méthodologie** : Décrivez précisément votre protocole de test. Comment avez-vous configuré chaque entraînement ? Quels hyperparamètres sont restés constants ?
- **Résultats** : Présentez vos résultats sous forme de tableaux comparatifs et de graphiques (par exemple, les courbes de perte et de précision pour chaque optimizer, générées depuis Weights & Biases).

- **Analyse** : Comparez les performances. Lequel est le plus rapide ? Le plus précis ? Le plus stable ? Y a-t-il un compromis à faire ?
- **Conclusion** : Démontrez quels sont les optimizers les plus adaptés à votre réseau de neurones et pourquoi.

Astuce : En utilisant les fonctions de rapportage de *Weights & Biases*, vous pouvez intégrer directement vos graphiques et analyses dans un document, ce qui correspond bien à l'idée de "générer avec votre baguette magique".

Souhaitez-vous que je détaille une de ces étapes, par exemple comment structurer concrètement le code pour tester les différents optimizers en Python ?

You asked:

fait le scripte

✦ Gemini Replied:

Bien sûr ! Voici un script Python complet et commenté pour réaliser le benchmark du défi n°21.

Ce script utilise la bibliothèque **Ultralytics** pour le modèle YOLOv8 et **Weights & Biases** pour le suivi des expériences, comme suggéré dans l'énoncé.

📋 Prérequis

1. **Installer les bibliothèques** : Ouvrez votre terminal et installez les paquets nécessaires.

Bash

```
pip install ultralytics wandb
```

2. **Créer un compte Weights & Biases** :

- Rendez-vous sur le site wandb.ai pour créer un compte gratuit.
- Connectez votre machine en tapant cette commande dans le terminal et en suivant les instructions pour coller votre clé API :

Bash

```
wandb login
```

3. **Préparer votre jeu de données** : Le script suppose que vous avez un fichier de configuration pour votre dataset (un fichier `.yaml`). **N'oubliez pas de modifier la variable `DATASET_YAML`**

dans le script pour pointer vers votre propre fichier.

Le Script de Benchmark

Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Script pour le défi n°21 : "LE NIMBUS 3000"
-----

Ce script réalise un benchmark de différents optimizers sur un modèle YOLOv8.
Il entraîne le modèle successivement avec chaque optimizer et enregistre les
résultats sur Weights & Biases (wandb) pour une analyse comparative.

Auteur: Gemini
Date: 16/10/2025
"""

import wandb
from ultralytics import YOLO

# --- CONFIGURATION DU BENCHMARK ---
# Modifiez ces paramètres selon vos besoins

# ⚠ IMPORTANT : Modifiez cette ligne pour indiquer le chemin vers le fichier .yaml
# de votre jeu de données Harry Potter (créé pour le défi n°20).
# Pour l'exemple, nous utilisons 'coco128.yaml' qui est un dataset de démo.
DATASET_YAML = 'coco128.yaml'

# Nom du projet sur Weights & Biases où les résultats seront sauvegardés.
WANDB_PROJECT_NAME = "Nimbus_3000_Benchmark"

# Modèle YOLO à utiliser. 'yolov8n.pt' est le plus petit et le plus rapide.
MODEL_TO_USE = 'yolov8n.pt'

# Nombre d'époques pour chaque entraînement.
# Augmentez cette valeur pour un benchmark plus précis (ex: 50, 100).
NUM_EPOCHS = 10

# Taille des images pour l'entraînement.
IMAGE_SIZE = 640

# [cite_start]Liste des optimizers à tester, conformément à l'énoncé du défi[cite: 307, 308, 309]
OPTIMIZERS_TO_TEST = [
    'SGD',
    'Adam',
    'AdamW',
    'RMSProp',
    'Adagrad',
    'Adadelata',
    # 'Adan' peut nécessiter une installation/configuration spécifique
    # s'il n'est pas inclus par défaut dans la version de PyTorch utilisée
    # par ultralytics. Testez les autres en premier lieu.
]

def run_benchmark():
```

```

"""Fonction principale pour lancer le benchmark."""

print("🚀 Démarrage du benchmark 'Le Nimbus 3000'...")

for optimizer_name in OPTIMIZERS_TO_TEST:
    print(f"\n{'='*50}")
    print(f"🔧 Test de l'optimizer : {optimizer_name}")
    print(f"{'='*50}")

    try:
        # 1. Initialiser une nouvelle session sur Weights & Biases pour cet optimizer
        wandb.init(
            project=WANDB_PROJECT_NAME,
            name=f"{MODEL_TO_USE.split('.')[0]}_{optimizer_name}", # Ex: "yolov8n_Adam"
            reinit=True, # Permet de réinitialiser wandb à chaque tour de boucle
            config={
                "model": MODEL_TO_USE,
                "optimizer": optimizer_name,
                "epochs": NUM_EPOCHS,
                "image_size": IMAGE_SIZE,
                "dataset": DATASET_YAML,
            }
        )

        # 2. Charger le modèle YOLO.
        # Il est important de le recharger à chaque fois pour repartir
        # des poids pré-entraînés et non de l'entraînement précédent.
        model = YOLO(MODEL_TO_USE)

        # 3. Lancer l'entraînement
        # La bibliothèque 'ultralytics' s'intègre automatiquement avec wandb
        # et enverra les métriques (perte, précision, etc.).
        model.train(
            data=DATASET_YAML,
            epochs=NUM_EPOCHS,
            imgsz=IMAGE_SIZE,
            optimizer=optimizer_name,
            # Les deux lignes suivantes sont pour que les logs aillent au bon endroit,
            # mais l'intégration wandb est souvent automatique.
            project=WANDB_PROJECT_NAME,
            name=f"{MODEL_TO_USE.split('.')[0]}_{optimizer_name}"
        )

        print(f"✅ Entraînement avec {optimizer_name} terminé avec succès.")

    except Exception as e:
        print(f"❌ Une erreur est survenue lors de l'entraînement avec {optimizer_name}: {e}")

    finally:
        # 4. Fermer la session wandb pour passer à la suivante
        wandb.finish()

print("\n🌟 Benchmark 'Le Nimbus 3000' terminé ! 🌟")

```

```
print(f"Consultez les résultats sur votre dashboard Weights & Biases: https://wandb.ai/{WAN
```

```
if __name__ == "__main__":  
    run_benchmark()
```

⚡ Comment l'utiliser

1. **Enregistrez** le code ci-dessus dans un fichier nommé `benchmark_nimbus.py` .
2. **Placez** votre fichier de configuration de dataset (`.yaml`) dans le même dossier ou mettez le chemin complet dans la variable `DATASET_YAML` .
3. **Exécutez** le script depuis votre terminal :

Bash

```
python benchmark_nimbus.py
```

4. **Analysez les résultats** : Pendant que le script tourne, vous pouvez suivre les entraînements en direct sur votre dashboard Weights & Biases. Une fois terminé, vous pourrez facilement comparer les graphiques de performance de chaque optimizer pour rédiger votre rapport.

Ce script vous fournira toutes les données nécessaires pour analyser et conclure quel optimizer est le plus performant pour votre cas d'usage, comme demandé dans le défi.

You asked:

Skip to main contentmerlin-mordrede-

Projects

Nimbus_3000_Benchmark

Runs

yolov8n_SGD

arthur simoes

Project

Workspace

Runs

Automat.

Sweeps

Reports

Artifacts

yolov8n_SGD


OverviewLogsFiles

2025-10-16 13:38:30

2025-10-16 13:38:30

Validating /content/Nimbus_3000_Benchmark/yolov8n_SGD/weights/best.pt...

2025-10-16 13:38:30

Ultralytics 8.3.215  Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)

2025-10-16 13:38:30

Model summary (fused): 72 layers, 3,008,183 parameters, 0 gradients, 8.1 GFLOPs

2025-10-16 13:38:30

	Class	Images	Instances	Box(P	R	mAP50	mAP50-
95): 100%	_____			2/2	6.8it/s	0.3s	0.6s

2025-10-16 13:38:33

	all	47	59	0.602	0.842	0.817	0.464
--	-----	----	----	-------	-------	-------	-------

2025-10-16 13:38:33

	ALBUS-DUMBLEDORE		5	5	0.634	1	0.995	0.557
--	------------------	--	---	---	-------	---	-------	-------

2025-10-16 13:38:33

	DRACO-MALFOY		7	7	0.88	1	0.995	0.732
--	--------------	--	---	---	------	---	-------	-------

2025-10-16 13:38:33

	GINNY-WEASLEY		7	16	0.684	0.543	0.735	0.435
--	---------------	--	---	----	-------	-------	-------	-------

2025-10-16 13:38:33

	HARRY-POTTER		4	4	0.6	1	0.995	0.639
--	--------------	--	---	---	-----	---	-------	-------

2025-10-16 13:38:33

HERMIONE-GRANGER	5	5	0.175	0.4	0.212	0.143
------------------	---	---	-------	-----	-------	-------

2025-10-16 13:38:33

LORD-VOLDEMORT	3	5	0.762	1	0.995	0.496
----------------	---	---	-------	---	-------	-------

2025-10-16 13:38:33

LUNA-LOVEGOOD	2	2	0.388	0.5	0.695	0.19
---------------	---	---	-------	-----	-------	------

2025-10-16 13:38:33

MINERVA-MCGONAGALL	3	3	0.591	0.667	0.753	0.501
--------------------	---	---	-------	-------	-------	-------

2025-10-16 13:38:33

MOLLY-WEASLEY	1	1	0.399	1	0.497	0.298
---------------	---	---	-------	---	-------	-------

2025-10-16 13:38:33

RON-WEASLEY	3	4	0.697	1	0.945	0.667
-------------	---	---	-------	---	-------	-------

2025-10-16 13:38:33

RUBEUS-HAGRID	4	4	0.619	1	0.995	0.36
---------------	---	---	-------	---	-------	------

2025-10-16 13:38:33

SEVERUS-SNAPE	3	3	0.796	1	0.995	0.548
---------------	---	---	-------	---	-------	-------

2025-10-16 13:38:33

Speed: 0.2ms preprocess, 2.0ms inference, 0.0ms loss, 1.7ms postprocess per image

2025-10-16 13:38:33

Results saved to /content/Nimbus_3000_Benchmark/yolov8n_SGD

2025-10-16 13:38:35

✔ Entraînement avec SGD terminé avec succès.

✦ Gemini Replied:

Excellent, the first test of your benchmark is complete! Here's a breakdown of what these results for the **SGD optimizer** mean.

In short, the model's performance is quite good overall, but it excels at recognizing some characters while struggling significantly with others.

📊 Key Metrics Explained

To understand the results, let's quickly define the main columns:

- **Precision (P):** When the model predicts a character, how often is it correct? High precision means few false alarms.
 - **Recall (R):** Of all the actual characters in the images, how many did the model successfully find? High recall means the model doesn't miss much.
 - **mAP50:** The main performance score. It's the average precision calculated when the predicted box must overlap the true box by at least 50%. A score of **0.817 (or 81.7%)** is a solid result.
 - **mAP50-95:** A stricter version of mAP that averages the score across multiple overlap thresholds (from 50% to 95%). This score is lower (**0.464**) because it penalizes boxes that aren't perfectly placed.
-

📈 Performance Analysis for SGD

Overall Score

Your model achieved an overall **mAP50 score of 81.7%**. This is a strong baseline and shows that the SGD optimizer was effective at training the model to recognize most characters.

The Star Pupils ✨ (Strengths)

The model is **extremely effective** at identifying several key characters.

- **Top Performer: Draco Malfoy** is the best-recognized character with an almost perfect **mAP50** (0.995) and the highest strict **mAP50-95** score (0.732).
- **Excellent Recognition: Albus Dumbledore, Harry Potter, Lord Voldemort, Ron Weasley, Rubeus Hagrid, and Severus Snape** are also detected with near-perfect accuracy and recall. The model rarely misses them and is very confident in its predictions.

Needs Improvement 🛠️ (Weaknesses)

The model has clear difficulties with two characters, which brings down the average score.

- **Biggest Challenge: Hermione Granger** is the model's primary weakness. With a **mAP50** of only **0.212**, the model struggles significantly. The low **Precision** (0.175) and **Recall** (0.4) indicate that the model both makes many incorrect guesses for her and misses her presence in images about 60% of the time.
 - **Also Struggling: Luna Lovegood** is also poorly detected, with the second-lowest **mAP50** score (0.19).
-

🧙 Next Steps

This is just one piece of the puzzle! The goal of the "Nimbus 3000" challenge is to compare optimizers.

1. **Let the Benchmark Finish:** Wait for the tests with Adam, AdamW, RMSProp, and the others to complete.
2. **Compare on Weights & Biases:** Use your W&B dashboard to create charts comparing the `mAP50` and `mAP50-95` scores for *all* optimizers.
3. **Identify the Best Optimizer:** The one that yields the highest overall scores (and perhaps better scores for weaker characters like Hermione) will be the winner of your benchmark.

These results for SGD are your baseline. Now you can see if another optimizer can do better!

You asked:

mais les matricule des resultat des modell ne remonte pas

💡 Gemini Replied:

Ah, c'est un problème frustrant mais assez courant ! Le script termine son travail, mais les graphiques et les chiffres n'apparaissent pas sur votre tableau de bord Weights & Biases.

Ne vous inquiétez pas, nous allons régler ça. Voici une liste de vérifications à effectuer, de la cause la plus probable à la plus rare.

✨ Plan de Dépannage : Faire Apparaître les Métriques

1. Vérification n°1 : La Connexion à Weights & Biases (Cause la plus fréquente)

Souvent, le script n'arrive tout simplement pas à s'authentifier correctement.

- **Action :** Dans votre terminal (là où vous lancez le script), exécutez à nouveau la commande de connexion et suivez les instructions pour recoller votre clé API.

Bash

```
wandb login
```

- **Vérification :** Relancez le script pour un seul optimizer (pour aller plus vite). Au tout début de l'exécution, vous devriez voir des messages de `wandb` qui ressemblent à ceci :

```
wandb: Currently logged in as: votre-nom-utilisateur (votre-equipe). Use `wandb login --relogin` to force login.  
wandb: Tracking run with wandb version x.xx.x  
wandb: Run data is saved locally in /path/to/your/project/wandb/run-xxxxxxx-xxxxxxx  
wandb: Syncing run yolov8n_SGD  
wandb: 🌟 View project at https://wandb.ai/votre-equipe/Nimbus_3000_Benchmark  
wandb: ✅ View run at https://wandb.ai/votre-equipe/Nimbus_3000_Benchmark/runs/xxxxxxx
```

Si vous ne voyez pas ces lignes avec les liens, c'est que la connexion est le problème.

2. Vérification n°2 : L'Intégration Automatique d'Ultralytics

Le script repose sur le fait que la bibliothèque `ultralytics` détecte `wandb` et lui envoie les données automatiquement. Parfois, cette intégration peut être désactivée.

- **Action :** Dans votre terminal, lancez la commande suivante pour vérifier les paramètres de `ultralytics` :

Bash

```
yolo settings
```

- **Vérification :** Assurez-vous que dans la liste qui s'affiche, `wandb` n'est pas dans une section `disabled_loggers` ou que les paramètres de logging sont bien activés.

3. Vérification n°3 : Le Script Lui-même

Assurez-vous que les lignes essentielles du script que je vous ai fourni n'ont pas été modifiées ou supprimées par erreur.

- **Vérification :** Contrôlez que les lignes `wandb.init(...)` et `wandb.finish()` sont bien présentes et non mises en commentaire (`#`). Le paramètre `reinit=True` dans `wandb.init` est crucial pour que la boucle fonctionne.

4. Le Plan B : Ajouter un "Callback" Manuel

Si l'intégration automatique ne fonctionne vraiment pas, nous pouvons forcer l'envoi des données manuellement. C'est comme utiliser un sortilège de Convocation (`Accio Métriques !`).

Modifiez la boucle d'entraînement dans le script pour y ajouter des **callbacks**, qui sont de petites fonctions qui s'exécutent à des moments clés de l'entraînement.

Voici à quoi ressemblerait la nouvelle boucle :

Python

```
# (Dans la fonction run_benchmark(), à l'intérieur de la boucle for)

# ... après la ligne model = YOLO(MODEL_TO_USE)

# --- DEBUT DE LA MODIFICATION ---

# Définir une fonction qui enverra les métriques à la fin de chaque époque
def log_metrics_to_wandb(trainer):
    # trainer.metrics contient toutes les données de performance
    if trainer.metrics:
        wandb.log(trainer.metrics)

# Attacher notre fonction à l'événement 'on_epoch_end'
model.add_callback("on_epoch_end", log_metrics_to_wandb)

# --- FIN DE LA MODIFICATION ---

# 3. Lancer l'entraînement (le reste ne change pas)
model.train(
    data=DATASET_YAML,
    # ... autres paramètres ...
)

# ...
```

Cette méthode est plus robuste car elle ne dépend pas de l'intégration automatique. Elle dit explicitement au script : "À la fin de chaque époque, prends toutes les métriques et envoie-les à Weights & Biases."

Commencez par la **vérification n°1**, car elle résout le problème dans 90% des cas. Si cela ne fonctionne pas, les autres étapes devraient vous mener à la solution.

You asked:

peut tu faire le scripte en entier

💡 Gemini Replied:

Absolument. Voici la version complète et mise à jour du script.

Cette version intègre la solution de "callback" manuel que nous avons discutée. C'est la méthode la plus fiable pour s'assurer que vos métriques sont bien envoyées à Weights & Biases, même si l'intégration automatique rencontre un problème.

J'ai ajouté des commentaires pour expliquer précisément les nouvelles lignes de code.

Script de Benchmark (Version Fiable avec Callback)

Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
Script pour le défi n°21 : "LE NIMBUS 3000" (Version Robuste)
-----
Ce script réalise un benchmark de différents optimizers sur un modèle YOLOv8.
Il utilise un "callback" manuel pour garantir que les métriques de performance
sont envoyées à Weights & Biases (wandb) à la fin de chaque époque.

Auteur: Gemini
Date: 16/10/2025
"""

import wandb
from ultralytics import YOLO

# --- CONFIGURATION DU BENCHMARK ---
# Modifiez ces paramètres selon vos besoins

# ⚠ IMPORTANT : Modifiez cette ligne pour indiquer le chemin vers le fichier .yaml
# de votre jeu de données Harry Potter.
DATASET_YAML = 'coco128.yaml'

# Nom du projet sur Weights & Biases où les résultats seront sauvegardés.
WANDB_PROJECT_NAME = "Nimbus_3000_Benchmark"

# Modèle YOLO à utiliser. 'yolov8n.pt' est le plus petit et le plus rapide.
MODEL_TO_USE = 'yolov8n.pt'

# Nombre d'époques pour chaque entraînement.
NUM_EPOCHS = 10

# Taille des images pour l'entraînement.
IMAGE_SIZE = 640

# Liste des optimizers à tester, conformément à l'énoncé du défi.
OPTIMIZERS_TO_TEST = [
    'SGD',
    'Adam',
    'AdamW',
    'RMSProp',
    'Adagrad',
    'Adadelata',
]

def run_benchmark():
    """Fonction principale pour lancer le benchmark."""

    print("🚀 Démarrage du benchmark 'Le Nimbus 3000' (version avec callback)...")

    for optimizer_name in OPTIMIZERS_TO_TEST:
```



```

print(f"\n{'='*50}")
print(f"🔧 Test de l'optimizer : {optimizer_name}")
print(f"{'='*50}")

try:
    # 1. Initialiser une nouvelle session sur Weights & Biases
    wandb.init(
        project=WANDB_PROJECT_NAME,
        name=f"{MODEL_TO_USE.split('.')[0]}_{optimizer_name}",
        reinit=True,
        config={
            "model": MODEL_TO_USE,
            "optimizer": optimizer_name,
            "epochs": NUM_EPOCHS,
            "image_size": IMAGE_SIZE,
            "dataset": DATASET_YAML,
        }
    )

    # 2. Charger le modèle YOLO.
    model = YOLO(MODEL_TO_USE)

    # --- NOUVELLE PARTIE : CALLBACK MANUEL ---
    # On définit une fonction qui sera appelée à la fin de chaque époque
    # pour envoyer les métriques à wandb.
    def log_metrics_to_wandb(trainer):
        """Callback pour logger les métriques de l'entraîneur."""
        if trainer.metrics:
            wandb.log(trainer.metrics)

    # On "attache" notre fonction à l'événement 'on_epoch_end' de l'entraînement.
    model.add_callback("on_epoch_end", log_metrics_to_wandb)
    # --- FIN DE LA NOUVELLE PARTIE ---

    # 3. Lancer l'entraînement
    model.train(
        data=DATASET_YAML,
        epochs=NUM_EPOCHS,
        imgsz=IMAGE_SIZE,
        optimizer=optimizer_name,
    )

    print(f"✅ Entraînement avec {optimizer_name} terminé avec succès.")

except Exception as e:
    print(f"❌ Une erreur est survenue lors de l'entraînement avec {optimizer_name}: {e}")

finally:
    # 4. Fermer la session wandb pour passer à la suivante
    wandb.finish()

print("\n🌟 Benchmark 'Le Nimbus 3000' terminé ! 🌟")
print(f"Consultez les résultats sur votre dashboard: https://wandb.ai/{WANDB_PROJECT_NAME.s

```

```
if __name__ == "__main__":  
    run_benchmark()
```

Comment l'utiliser

1. **Assurez-vous d'être connecté** : Exécutez `wandb login` dans votre terminal une dernière fois.
2. **Copiez et collez** l'intégralité de ce script dans votre fichier `benchmark_nimbus.py`.
3. **Modifiez la variable** `DATASET_YAML` pour qu'elle corresponde à votre jeu de données.
4. **Exécutez le script** comme avant : `python benchmark_nimbus.py`.

Avec cette version, vous devriez voir les graphiques de performance (perte, mAP, etc.) se mettre à jour sur votre tableau de bord Weights & Biases après chaque époque d'entraînement.

You asked:

voici le resultat "🚀 Démarrage du benchmark 'Le Nimbus 3000' (version avec callback)...

=====

🔧 Test de l'optimizer : SGD

=====

Tracking run with wandb version 0.22.2

Run data is saved locally in /content/wandb/run-20251016_134908-yg4hlfgv

Syncing run yolov8n_SGD to Weights & Biases (docs)

View project at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

View run at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/yg4hlfgv

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB) engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugument, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, compile=False, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=/content/harry-potter-character-detection-1-1/data.yaml, degrees=0.0, deterministic=True, device=None, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript,

fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=640, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yolov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=train, nbs=64, nms=False, opset=None, optimize=False, optimizer=SGD, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=None, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=/content/runs/detect/train, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

Overriding model.yaml nc=80 with nc=13

from n params module arguments

0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]
1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]
2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]
3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]
4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]
5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]
6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]
8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]
9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]
10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]
13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]
15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]
16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]

17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]

18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]

19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]






20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]

21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]

22 [15, 18, 21] 1 753847 ultralytics.nn.modules.head.Detect [13, [64, 128, 256]]

Model summary: 129 layers, 3,013,383 parameters, 3,013,367 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'AMP: running Automatic Mixed Precision (AMP)
checks...AMP: checks passed  train: Fast image access  (ping: 0.0±0.0 ms, read: 250.4±98.9 MB/s, size: 6.6 KB)train: Scanning /content/harry-potter-character-detection-1-1/train/labels.cache...
1722 images, 0 backgrounds, 0 corrupt: 100%  1722/1722 1.7Mit/s
0.0salbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)),
ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=
(1.0, 4.0), tile_grid_size=(8, 8))val: Fast image access  (ping: 0.0±0.0 ms, read: 228.3±119.8 MB/s, size: 8.8 KB)val: Scanning /content/harry-potter-character-detection-1-1/valid/labels.cache...
47 images, 0 backgrounds, 0 corrupt: 100%  47/47 10.8Kit/s 0.0s

Plotting labels to /content/runs/detect/train/labels.jpg... optimizer: SGD(lr=0.01, momentum=0.937)
with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)


Image sizes 640 train, 640 val


Using 8 dataloader workers

Logging results to /content/runs/detect/train

Starting training for 10 epochs...

Closing dataloader mosaicalbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01,
blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average', num_output_channels=3),
CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size 1/10 2.08G 1.593 4.824 1.966 10 640:
100%  108/108 5.8it/s 18.5s

Class Images Instances Box(P R mAP50 mAP50-95): 100%  2/2 7.4it/s
0.3s

all 47 59 0.562 0.103 0.0655 0.0298

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
2/10 2.54G 1.464 3.946 1.741 10 640: 100% 108/108 6.6it/s 16.3s
Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.7it/s
0.3s
all 47 59 0.175 0.267 0.152 0.0706

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
3/10 2.54G 1.47 3.475 1.74 11 640: 100% 108/108 6.7it/s 16.0s
Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.5it/s
0.3s
all 47 59 0.609 0.303 0.397 0.215

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
4/10 2.54G 1.474 3.156 1.717 10 640: 100% 108/108 6.8it/s 15.8s
Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.9it/s
0.3s
all 47 59 0.213 0.411 0.345 0.162

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
5/10 2.54G 1.417 2.744 1.632 12 640: 100% 108/108 6.8it/s 15.8s
Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.7it/s
0.3s
all 47 59 0.572 0.439 0.488 0.284

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
6/10 2.54G 1.383 2.44 1.584 10 640: 100% 108/108 6.8it/s 15.8s
Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.9it/s
0.3s

all 47 59 0.702 0.55 0.649 0.348

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

7/10 2.54G 1.348 2.155 1.542 10 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.9it/s 0.3s

all 47 59 0.592 0.75 0.689 0.39

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

8/10 2.54G 1.284 1.922 1.485 10 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.6it/s 0.3s

all 47 59 0.571 0.661 0.717 0.398

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

9/10 2.54G 1.236 1.724 1.441 10 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.8it/s 0.3s

all 47 59 0.59 0.769 0.742 0.394

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

10/10 2.54G 1.186 1.585 1.411 12 640: 100% 108/108 6.9it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 8.0it/s 0.2s

all 47 59 0.601 0.842 0.817 0.464

10 epochs completed in 0.046 hours.

Optimizer stripped from /content/runs/detect/train/weights/last.pt, 6.2MB

Optimizer stripped from /content/runs/detect/train/weights/best.pt, 6.2MB

Validating /content/runs/detect/train/weights/best.pt...

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)

Model summary (fused): 72 layers, 3,008,183 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 6.7it/s
0.3s

all 47 59 0.602 0.842 0.817 0.464

ALBUS-DUMBLEDORE 5 5 0.634 1 0.995 0.557

DRACO-MALFOY 7 7 0.88 1 0.995 0.732

GINNY-WEASLEY 7 16 0.684 0.543 0.735 0.435

HARRY-POTTER 4 4 0.6 1 0.995 0.639

HERMIONE-GRANGER 5 5 0.175 0.4 0.212 0.143

LORD-VOLDEMORT 3 5 0.762 1 0.995 0.496

LUNA-LOVEGOOD 2 2 0.388 0.5 0.695 0.19

MINERVA-MCGONAGALL 3 3 0.591 0.667 0.753 0.501

MOLLY-WEASLEY 1 1 0.399 1 0.497 0.298

RON-WEASLEY 3 4 0.697 1 0.945 0.667

RUBEUS-HAGRID 4 4 0.619 1 0.995 0.36

SEVERUS-SNAPE 3 3 0.796 1 0.995 0.548

Speed: 0.2ms preprocess, 2.0ms inference, 0.0ms loss, 1.7ms postprocess per image

Results saved to /content/runs/detect/train

✅ Entraînement avec SGD terminé avec succès.

View run yolov8n_SGD at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/yg4hlfgv

View project at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: ./wandb/run-20251016_134908-yg4hlfgv/logs

=====

🔧 Test de l'optimizer : Adam

=====

Tracking run with wandb version 0.22.2

Run data is saved locally in /content/wandb/run-20251016_135208-fomhwpt8

Syncing run yolov8n_Adam to Weights & Biases (docs)

View project at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

View run at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/fomhwpt8

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugument, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, compile=False, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=/content/harry-potter-character-detection-1-1/data.yaml, degrees=0.0, deterministic=True, device=None, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=640, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yolov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=train2, nbs=64, nms=False, opset=None, optimize=False, optimizer=Adam, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=None, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=/content/runs/detect/train2, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

Overriding model.yaml nc=80 with nc=13

from n params module arguments

0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]




1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]

2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]

3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]
 4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]
 5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]
 6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
 7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]
 8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]
 9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]
 10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
 11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
 12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]
 13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
 14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]
 15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]
 16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]
 17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]
 18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]
 19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]
 20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]
 21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]
 22 [15, 18, 21] 1 753847 ultralytics.nn.modules.head.Detect [13, [64, 128, 256]]

Model summary: 129 layers, 3,013,383 parameters, 3,013,367 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'AMP: running Automatic Mixed Precision (AMP)
 checks...AMP: checks passed  train: Fast image access  (ping: 0.0±0.0 ms, read: 343.9±96.8 MB/s, size: 6.6 KB)train: Scanning /content/harry-potter-character-detection-1-1/train/labels.cache...
 1722 images, 0 backgrounds, 0 corrupt: 100% ————— 1722/1722 2.0Mit/s
 0.0salbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)),
 ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=
 (1.0, 4.0), tile_grid_size=(8, 8))val: Fast image access  (ping: 0.0±0.0 ms, read: 148.6±45.1 MB/s,

size: 8.8 KB)val: Scanning /content/harry-potter-character-detection-1-1/valid/labels.cache... 47
images, 0 backgrounds, 0 corrupt: 100% ————— 47/47 10.8Kit/s 0.0s

Plotting labels to /content/runs/detect/train2/labels.jpg... optimizer: Adam(lr=0.01, momentum=0.937)
with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)

Image sizes 640 train, 640 val

Using 8 dataloader workers

Logging results to /content/runs/detect/train2

Starting training for 10 epochs...

Closing dataloader mosaicalbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01,
blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average', num_output_channels=3),
CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size 1/10 2.08G 1.993 3.753 2.27 10 640:
100% ————— 108/108 6.1it/s 17.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 3.1it/s
0.6s

all 47 59 0.25 0.0536 0.000907 0.000129

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

2/10 2.56G 1.912 3.048 2.155 10 640: 100% ————— 108/108 6.6it/s 16.3s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 4.3it/s
0.5s

all 47 59 0.0938 0.491 0.0401 0.0138

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

3/10 2.58G 1.808 2.897 2.031 11 640: 100% ————— 108/108 6.7it/s 16.1s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.5it/s
0.3s

all 47 59 0.272 0.36 0.116 0.054

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

4/10 2.58G 1.732 2.713 1.949 10 640: 100% ————— 108/108 6.8it/s 15.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.8it/s
0.3s

all 47 59 0.263 0.448 0.248 0.103

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

5/10 2.58G 1.716 2.601 1.905 12 640: 100% ————— 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.9it/s
0.3s

all 47 59 0.29 0.292 0.251 0.115

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

6/10 2.6G 1.629 2.479 1.825 10 640: 100% ————— 108/108 6.8it/s 15.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.7it/s
0.3s

all 47 59 0.24 0.298 0.197 0.108

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

7/10 2.6G 1.552 2.364 1.749 10 640: 100% ————— 108/108 6.8it/s 15.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.7it/s
0.3s

all 47 59 0.273 0.401 0.277 0.114

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

8/10 2.6G 1.509 2.22 1.702 10 640: 100% ————— 108/108 6.8it/s 16.0s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.8it/s
0.3s

all 47 59 0.201 0.353 0.305 0.138

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

9/10 2.6G 1.476 2.111 1.668 10 640: 100% ————— 108/108 6.5it/s 16.5s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.8it/s
0.3s

all 47 59 0.193 0.506 0.366 0.194

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

10/10 2.6G 1.411 2.017 1.62 12 640: 100% ————— 108/108 6.8it/s 15.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.7it/s
0.3s

all 47 59 0.347 0.582 0.492 0.257

10 epochs completed in 0.047 hours.

Optimizer stripped from /content/runs/detect/train2/weights/last.pt, 6.2MB

Optimizer stripped from /content/runs/detect/train2/weights/best.pt, 6.2MB

Validating /content/runs/detect/train2/weights/best.pt...

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)

Model summary (fused): 72 layers, 3,008,183 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 6.8it/s
0.3s

all 47 59 0.347 0.582 0.492 0.257

ALBUS-DUMBLEDORE 5 5 0.394 1 0.995 0.414

DRACO-MALFOY 7 7 0.793 0.551 0.837 0.646

GINNY-WEASLEY 7 16 0.456 0.438 0.489 0.243

HARRY-POTTER 4 4 0.308 0.75 0.745 0.462

HERMIONE-GRANGER 5 5 0.0679 0.2 0.105 0.0633

LORD-VOLDEMORT 3 5 1 0.395 0.697 0.343

LUNA-LOVEGOOD 2 2 0.0984 0.5 0.194 0.0479

MINERVA-MCGONAGALL 3 3 0.24 0.667 0.284 0.171

MOLLY-WEASLEY 1 1 0.364 1 0.497 0.298

RON-WEASLEY 3 4 0.103 0.154 0.132 0.0812

RUBEUS-HAGRID 4 4 0.261 1 0.816 0.256

SEVERUS-SNAPE 3 3 0.081 0.333 0.116 0.0612

Speed: 0.2ms preprocess, 2.0ms inference, 0.0ms loss, 1.8ms postprocess per image

Results saved to /content/runs/detect/train2

✅ Entraînement avec Adam terminé avec succès.

View run yolov8n_Adam at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/fomhwpt8

View project at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: ./wandb/run-20251016_135208-fomhwpt8/logs

=====

🔧 Test de l'optimizer : AdamW

=====

Tracking run with wandb version 0.22.2

Run data is saved locally in /content/wandb/run-20251016_135509-1pumzm94

Syncing run yolov8n_AdamW to Weights & Biases (docs)

View project at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

View run at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/1pumzm94

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugument, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, compile=False, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=/content/harry-potter-character-detection-1-1/data.yaml, degrees=0.0, deterministic=True, device=None, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=640, int8=False,

iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yolov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=train3, nbs=64, nms=False, opset=None, optimize=False, optimizer=AdamW, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=None, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=/content/runs/detect/train3, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

Overriding model.yaml nc=80 with nc=13

from n params module arguments

```
0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]
1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]
2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]
3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]
4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]
5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]
6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]
8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]
9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]
10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]
13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]
15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]
16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]
```

17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]

18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]

19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]






20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]

21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]

22 [15, 18, 21] 1 753847 ultralytics.nn.modules.head.Detect [13, [64, 128, 256]]

Model summary: 129 layers, 3,013,383 parameters, 3,013,367 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'AMP: running Automatic Mixed Precision (AMP)
checks...AMP: checks passed  train: Fast image access  (ping: 0.0±0.0 ms, read: 349.1±96.1 MB/s, size: 6.6 KB)train: Scanning /content/harry-potter-character-detection-1-1/train/labels.cache...
1722 images, 0 backgrounds, 0 corrupt: 100%  1722/1722 2.0Mit/s
0.0salbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)),
ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=
(1.0, 4.0), tile_grid_size=(8, 8))val: Fast image access  (ping: 0.0±0.0 ms, read: 139.6±57.4 MB/s, size: 8.8 KB)val: Scanning /content/harry-potter-character-detection-1-1/valid/labels.cache... 47
images, 0 backgrounds, 0 corrupt: 100%  47/47 32.7Kit/s 0.0s

Plotting labels to /content/runs/detect/train3/labels.jpg... optimizer: AdamW(lr=0.01,
momentum=0.937) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63
bias(decay=0.0)


Image sizes 640 train, 640 val


Using 8 dataloader workers

Logging results to /content/runs/detect/train3

Starting training for 10 epochs...

Closing dataloader mosaicalbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01,
blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average', num_output_channels=3),
CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size 1/10 2.17G 1.996 3.786 2.285 10 640:
100%  108/108 6.0it/s 17.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100%  2/2 4.1it/s
0.5s

all 47 59 0.0916 0.0516 0.0118 0.00318

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

2/10 2.63G 1.973 3.164 2.215 10 640: 100% 108/108 6.6it/s 16.4s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.6it/s 0.3s

all 47 59 0.313 0.171 0.0387 0.0147

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

3/10 2.63G 1.847 2.886 2.051 11 640: 100% 108/108 6.7it/s 16.1s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.3it/s 0.3s

all 47 59 0.0639 0.205 0.106 0.0502

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

4/10 2.63G 1.719 2.63 1.939 10 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.6it/s 0.3s

all 47 59 0.171 0.326 0.155 0.0692

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

5/10 2.63G 1.672 2.547 1.883 12 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.8it/s 0.3s

all 47 59 0.295 0.19 0.175 0.0835

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

6/10 2.65G 1.604 2.431 1.805 10 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 8.0it/s 0.3s

all 47 59 0.158 0.478 0.229 0.117

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

7/10 2.65G 1.53 2.272 1.745 10 640: 100% 108/108 6.8it/s 15.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.9it/s 0.3s

all 47 59 0.197 0.484 0.358 0.19

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

8/10 2.66G 1.467 2.153 1.664 10 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 8.2it/s 0.2s

all 47 59 0.266 0.395 0.334 0.161

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

9/10 2.67G 1.438 2.064 1.649 10 640: 100% 108/108 6.8it/s 16.0s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.9it/s 0.3s

all 47 59 0.22 0.445 0.36 0.174

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

10/10 2.67G 1.406 1.965 1.623 12 640: 100% 108/108 6.8it/s 15.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 8.0it/s 0.2s

all 47 59 0.27 0.633 0.417 0.21

10 epochs completed in 0.047 hours.

Optimizer stripped from /content/runs/detect/train3/weights/last.pt, 6.2MB

Optimizer stripped from /content/runs/detect/train3/weights/best.pt, 6.2MB

Validating /content/runs/detect/train3/weights/best.pt...

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)

Model summary (fused): 72 layers, 3,008,183 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100% ————— 2/2 7.1it/s
0.3s

all 47 59 0.271 0.633 0.417 0.21

ALBUS-DUMBLEDORE 5 5 0.421 1 0.886 0.388

DRACO-MALFOY 7 7 0.441 0.714 0.515 0.351

GINNY-WEASLEY 7 16 0.436 0.5 0.44 0.214

HARRY-POTTER 4 4 0.291 1 0.808 0.534

HERMIONE-GRANGER 5 5 0.0817 0.4 0.107 0.0601

LORD-VOLDEMORT 3 5 0.739 0.4 0.598 0.254

LUNA-LOVEGOOD 2 2 0.0739 0.5 0.215 0.0479

MINERVA-MCGONAGALL 3 3 0.178 0.667 0.204 0.133

MOLLY-WEASLEY 1 1 0 0 0.166 0.0995

RON-WEASLEY 3 4 0.187 0.747 0.174 0.109

RUBEUS-HAGRID 4 4 0.265 1 0.62 0.225

SEVERUS-SNAPE 3 3 0.138 0.667 0.277 0.107

Speed: 0.2ms preprocess, 2.0ms inference, 0.0ms loss, 1.7ms postprocess per image

Results saved to /content/runs/detect/train3

✅ Entraînement avec AdamW terminé avec succès.

View run yolov8n_AdamW at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/1pumzm94

View project at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `./wandb/run-20251016_135509-1pumzm94/logs`

=====

🔧 Test de l'optimizer : RMSProp

=====

Tracking run with wandb version 0.22.2

Run data is saved locally in `/content/wandb/run-20251016_135810-qgz6qx0l`

Syncing run `yolov8n_RMSProp` to [Weights & Biases \(docs\)](#)

View project at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

View run at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/qgz6qx0l

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB) engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugument, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, compile=False, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=/content/harry-potter-character-detection-1-1/data.yaml, degrees=0.0, deterministic=True, device=None, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=640, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yolov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=train4, nbs=64, nms=False, opset=None, optimize=False, optimizer=RMSProp, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=None, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=/content/runs/detect/train4, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

Overriding model.yaml nc=80 with nc=13



from n params module arguments



0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]

1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]
 2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]
 3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]
 4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]
 5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]
 6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
 7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]
 8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]
 9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]
 10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
 11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
 12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]
 13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
 14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]
 15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]
 16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]
 17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]
 18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]
 19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]
 20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]
 21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]
 22 [15, 18, 21] 1 753847 ultralytics.nn.modules.head.Detect [13, [64, 128, 256]]

Model summary: 129 layers, 3,013,383 parameters, 3,013,367 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'AMP: running Automatic Mixed Precision (AMP)
 checks...AMP: checks passed  train: Fast image access  (ping: 0.0±0.0 ms, read: 345.6±103.8 MB/s, size: 6.6 KB)train: Scanning /content/harry-potter-character-detection-1-1/train/labels.cache...
 1722 images, 0 backgrounds, 0 corrupt: 100% ————— 1722/1722 1.7Mit/s

0.0salbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))val: Fast image access  (ping: 0.0±0.0 ms, read: 155.6±67.4 MB/s, size: 8.8 KB)val: Scanning /content/harry-potter-character-detection-1-1/valid/labels.cache... 47 images, 0 backgrounds, 0 corrupt: 100%  47/47 9.5Kit/s 0.0s

Plotting labels to /content/runs/detect/train4/labels.jpg... optimizer: RMSprop(lr=0.01, momentum=0.937) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)


Image sizes 640 train, 640 val


Using 8 dataloader workers

Logging results to /content/runs/detect/train4

Starting training for 10 epochs...

Closing dataloader mosaicalbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))


Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size 1/10 2.09G 2.996 4.77 3.662 10 640: 100%  108/108 6.0it/s 17.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100%  2/2 5.2it/s 0.4s

all 47 59 0.417 0.1 0.000647 0.000111

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size


2/10 2.56G 2.796 3.964 3.134 10 640: 100%  108/108 6.7it/s 16.2s

Class Images Instances Box(P R mAP50 mAP50-95): 100%  2/2 8.2it/s 0.2s

all 47 59 0.00201 0.223 0.0087 0.0021

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

3/10 2.56G 2.683 4.006 3.024 11 640: 100%  108/108 6.8it/s 15.9s

Class Images Instances Box(P R mAP50 mAP50-95): 100%  2/2 7.8it/s 0.3s

all 47 59 0.0294 0.261 0.0296 0.00773

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

4/10 2.56G 2.51 3.72 2.929 10 640: 100% 108/108 6.9it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.5it/s 0.3s

all 47 59 0.505 0.0208 0.0188 0.00423

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

5/10 2.57G 2.475 3.645 2.871 12 640: 100% 108/108 6.9it/s 15.7s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.3it/s 0.3s

all 47 59 0.919 0.0278 0.0159 0.00388

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

6/10 2.58G 2.424 3.485 2.831 10 640: 100% 108/108 6.9it/s 15.6s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 8.0it/s 0.2s

all 47 59 0.73 0.144 0.0475 0.0184

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

7/10 2.58G 2.364 3.44 2.777 10 640: 100% 108/108 6.9it/s 15.7s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 8.4it/s 0.2s

all 47 59 0.00978 0.326 0.0267 0.00638

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

8/10 2.59G 2.363 3.389 2.722 10 640: 100% 108/108 6.9it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.5it/s 0.3s

all 47 59 0.836 0.0208 0.00292 0.000526

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

9/10 2.59G 2.289 3.325 2.696 10 640: 100% 108/108 6.8it/s 15.8s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 7.6it/s 0.3s

all 47 59 0.00867 0.49 0.0598 0.0234

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size

10/10 2.59G 2.255 3.284 2.675 12 640: 100% 108/108 6.6it/s 16.3s

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 8.0it/s 0.2s

all 47 59 0.00904 0.836 0.0591 0.0252

10 epochs completed in 0.046 hours.

Optimizer stripped from /content/runs/detect/train4/weights/last.pt, 6.2MB

Optimizer stripped from /content/runs/detect/train4/weights/best.pt, 6.2MB

Validating /content/runs/detect/train4/weights/best.pt...

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)

Model summary (fused): 72 layers, 3,008,183 parameters, 0 gradients, 8.1 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100% 2/2 6.8it/s 0.3s

all 47 59 0.00902 0.836 0.0592 0.025

ALBUS-DUMBLEDORE 5 5 0.0127 0.8 0.165 0.0563

DRACO-MALFOY 7 7 0.0133 0.714 0.0423 0.0198

GINNY-WEASLEY 7 16 0.0235 0.5 0.0391 0.0155

HARRY-POTTER 4 4 0.00653 1 0.201 0.108

HERMIONE-GRANGER 5 5 0.00666 0.8 0.0172 0.00557

LORD-VOLDEMORT 3 5 0.00666 0.8 0.0154 0.00333

LUNA-LOVEGOOD 2 2 0.00601 1 0.0348 0.00982

MINERVA-MCGONAGALL 3 3 0.0145 1 0.0478 0.0176

MOLLY-WEASLEY 1 1 0.00174 1 0.005 0.00255

RON-WEASLEY 3 4 0.00746 0.75 0.0763 0.0414

RUBEUS-HAGRID 4 4 0.00615 1 0.0156 0.00534

SEVERUS-SNAPE 3 3 0.00303 0.667 0.0501 0.015

Speed: 0.2ms preprocess, 2.0ms inference, 0.0ms loss, 1.9ms postprocess per image

Results saved to /content/runs/detect/train4

✅ Entraînement avec RMSProp terminé avec succès.

View run yolov8n_RMSProp at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/qgz6qx0l

View project at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: ./wandb/run-20251016_135810-qgz6qx0l/logs

=====

🔧 Test de l'optimizer : Adagrad

=====

Tracking run with wandb version 0.22.2

Run data is saved locally in /content/wandb/run-20251016_140110-53njlehn

Syncing run yolov8n_Adagrad to Weights & Biases (docs)

View project at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

View run at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/53njlehn

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, compile=False, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=/content/harry-potter-character-detection-1-1/data.yaml, degrees=0.0, deterministic=True, device=None, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, fliplr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=640, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yolov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=train5, nbs=64, nms=False, opset=None, optimize=False, optimizer=Adagrad, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=None, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=/content/runs/detect/train5, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

Overriding model.yaml nc=80 with nc=13




from n params module arguments

```
0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]
1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]
2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]
3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]
4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]
5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]
6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]
8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]
9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]
10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
```


12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]
13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]
15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]
16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]
17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]
18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]
19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]
20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]
21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]
22 [15, 18, 21] 1 753847 ultralytics.nn.modules.head.Detect [13, [64, 128, 256]]

Model summary: 129 layers, 3,013,383 parameters, 3,013,367 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'AMP: running Automatic Mixed Precision (AMP)
checks...AMP: checks passed  train: Fast image access  (ping: 0.0±0.0 ms, read: 349.8±104.9 MB/s, size: 6.6 KB)train: Scanning /content/harry-potter-character-detection-1-1/train/labels.cache...
1722 images, 0 backgrounds, 0 corrupt: 100% ————— 1722/1722 1.7Mit/s
0.0salbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)),
ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=
(1.0, 4.0), tile_grid_size=(8, 8))val: Fast image access  (ping: 0.0±0.0 ms, read: 84.1±52.1 MB/s, size: 8.8 KB)val: Scanning /content/harry-potter-character-detection-1-1/valid/labels.cache... 47
images, 0 backgrounds, 0 corrupt: 100% ————— 47/47 9.2Kit/s 0.0s

Plotting labels to /content/runs/detect/train5/labels.jpg...

 Une erreur est survenue lors de l'entraînement avec Adagrad: Optimizer 'None' not found in list of available optimizers {'AdamW', 'Adamax', 'NAdam', 'Adam', 'auto', 'RMSProp', 'SGD', 'RAdam'}. Request support for addition optimizers at <https://github.com/ultralytics/ultralytics>.

View run yolov8n_Adagrad at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/53njlehn

View project at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `./wandb/run-20251016_140110-53njlehn/logs`

=====

🔧 Test de l'optimizer : Adadelata

=====

Tracking run with wandb version 0.22.2

Run data is saved locally in `/content/wandb/run-20251016_140118-cr7q6hf7`

Syncing run `yolov8n_Adadelata` to Weights & Biases (docs)

View project at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

View run at https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/cr7q6hf7

Ultralytics 8.3.215 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB) engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugument, batch=16, bgr=0.0, box=7.5, cache=False, cfg=None, classes=None, close_mosaic=10, cls=0.5, compile=False, conf=None, copy_paste=0.0, copy_paste_mode=flip, cos_lr=False, cutmix=0.0, data=/content/harry-potter-character-detection-1-1/data.yaml, degrees=0.0, deterministic=True, device=None, dfl=1.5, dnn=False, dropout=0.0, dynamic=False, embed=None, epochs=10, erasing=0.4, exist_ok=False, flip_lr=0.5, flipud=0.0, format=torchscript, fraction=1.0, freeze=None, half=False, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, imgsz=640, int8=False, iou=0.7, keras=False, kobj=1.0, line_width=None, lr0=0.01, lrf=0.01, mask_ratio=4, max_det=300, mixup=0.0, mode=train, model=yolov8n.pt, momentum=0.937, mosaic=1.0, multi_scale=False, name=train6, nbs=64, nms=False, opset=None, optimize=False, optimizer=Adadelata, overlap_mask=True, patience=100, perspective=0.0, plots=True, pose=12.0, pretrained=True, profile=False, project=None, rect=False, resume=False, retina_masks=False, save=True, save_conf=False, save_crop=False, save_dir=/content/runs/detect/train6, save_frames=False, save_json=False, save_period=-1, save_txt=False, scale=0.5, seed=0, shear=0.0, show=False, show_boxes=True, show_conf=True, show_labels=True, simplify=True, single_cls=False, source=None, split=val, stream_buffer=False, task=detect, time=None, tracker=botsort.yaml, translate=0.1, val=True, verbose=True, vid_stride=1, visualize=False, warmup_bias_lr=0.1, warmup_epochs=3.0, warmup_momentum=0.8, weight_decay=0.0005, workers=8, workspace=None

Overriding model.yaml nc=80 with nc=13

from n params module arguments

0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]

1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]

2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]

3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]

4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]

5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]

6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]

7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]

8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]

9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]

10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]

12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]

13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]

15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]

16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]

17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]

18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]

19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]




20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]


21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]

22 [15, 18, 21] 1 753847 ultralytics.nn.modules.head.Detect [13, [64, 128, 256]]


Model summary: 129 layers, 3,013,383 parameters, 3,013,367 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights

Freezing layer 'model.22.dfl.conv.weight'AMP: running Automatic Mixed Precision (AMP)
 checks...AMP: checks passed  train: Fast image access  (ping: 0.0±0.0 ms, read: 356.4±96.4 MB/s, size: 6.6 KB)train: Scanning /content/harry-potter-character-detection-1-1/train/labels.cache...
 1722 images, 0 backgrounds, 0 corrupt: 100%  1722/1722 1.8Mit/s
 0.0salbumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)),

ToGray(p=0.01, method='weighted_average', num_output_channels=3), CLAHE(p=0.01, clip_limit=(1.0, 4.0), tile_grid_size=(8, 8))val: Fast image access  (ping: 0.0±0.0 ms, read: 106.5±34.7 MB/s, size: 8.8 KB)val: Scanning /content/harry-potter-character-detection-1-1/valid/labels.cache... 47 images, 0 backgrounds, 0 corrupt: 100% ————— 47/47 7.1Kit/s 0.0s

Plotting labels to /content/runs/detect/train6/labels.jpg...

 Une erreur est survenue lors de l'entraînement avec Adadelta: Optimizer 'None' not found in list of available optimizers {'AdamW', 'Adamax', 'NAdam', 'Adam', 'auto', 'RMSProp', 'SGD', 'RAdam'}. Request support for addition optimizers at <https://github.com/ultralytics/ultralytics>.

View run yolov8n_Adadelta at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2/runs/cr7q6hf7

View project at: https://wandb.ai/merlin-mordrede-/Nimbus_3000_Benchmark-2

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: ./wandb/run-20251016_140118-cr7q6hf7/logs

🌟 Benchmark 'Le Nimbus 3000' terminé ! 🌟

Consultez les résultats sur votre dashboard: https://wandb.ai//Nimbus_3000_Benchmark-2" peut tu faire le document

Gemini Replied:

Absolument. Félicitations pour avoir mené le benchmark jusqu'au bout !

Voici le document de recherche complet, rédigé à partir de vos résultats. Ce rapport est structuré pour répondre parfaitement aux exigences du défi "Le Nimbus 3000".

Rapport de Benchmark : Sélection de l'Optimizer Optimal pour la Détection de Personnages (Projet Nimbus 3000)

- **Auteur** : merlin-mordrede
- **Date** : 16 octobre 2025
- **Projet** : Défi n°21, Workshop "Poudlard à l'EPSI/WIS"

1. Introduction

Ce document présente les résultats d'une étude comparative visant à déterminer l'optimizer le plus performant pour l'entraînement d'un réseau de neurones convolutif **YOLOv8n**. L'objectif est la détection de personnages de la saga Harry Potter sur un jeu de données personnalisé.

Conformément au cahier des charges du défi, plusieurs optimizers populaires ont été testés : **SGD**, **Adam**, **AdamW**, **RMSProp**, **Adagrad**, et **Adadelata**. L'évaluation se base sur la métrique de performance **mAP50** (mean Average Precision), qui mesure la précision de la détection.


2. Méthodologie

- **Modèle** : YOLOv8n, pré-entraîné sur le dataset COCO.
- **Dataset** : Un jeu de données personnalisé contenant 1722 images pour l'entraînement et 47 pour la validation, réparties sur 13 classes de personnages.
- **Environnement** : L'entraînement a été réalisé sur un GPU NVIDIA Tesla T4 avec le framework PyTorch et la bibliothèque Ultralytics.
- **Protocole de Test** :
 - Chaque optimizer a été utilisé pour entraîner le modèle pendant **10 époques**.
 - Les hyperparamètres, notamment le taux d'apprentissage (learning rate), ont été laissés à leur valeur par défaut fournie par la bibliothèque pour garantir une comparaison équitable des algorithmes eux-mêmes.
 - Les métriques de performance ont été enregistrées à l'aide de Weights & Biases.

3. Résultats

Les entraînements ont révélé des différences de performance très significatives entre les optimizers testés. De plus, des problèmes de compatibilité ont été rencontrés avec certains d'entre eux.

Le tableau ci-dessous synthétise les scores finaux obtenus pour chaque test.

Optimizer	Score mAP50	Score mAP50-95	Statut
 SGD	81.7%	46.4%	Succès
Adam	49.2%	25.7%	Succès
AdamW	41.7%	21.0%	Succès
RMSProp	5.9%	2.5%	Succès (performance très faible)
Adagrad	-	-	Échec (non supporté)
Adadelata	-	-	Échec (non supporté)

4. Analyse et Discussion

Analyse des Performances

Les résultats sont sans appel. L'optimizer **SGD (Stochastic Gradient Descent)** avec momentum a surpassé tous les autres de manière spectaculaire, atteignant un score mAP50 de **81.7%**.

- Les optimizers adaptatifs de la famille Adam (**Adam** et **AdamW**) ont produit des résultats médiocres en comparaison, avec des scores respectifs de 49.2% et 41.7%. Bien que ces optimizers soient souvent réputés pour leur vitesse de convergence, ils se sont ici installés dans un minimum local de performance inférieure.
- L'optimizer **RMSProp** a été un échec total sur le plan de la performance. Avec un score de 5.9%, le modèle n'a pratiquement rien appris, indiquant que les hyperparamètres par défaut de cet optimizer sont totalement inadaptés à cette tâche.

Analyse des Échecs Techniques

Les optimizers **Adagrad** et **Adadelta** ont provoqué une erreur au lancement de l'entraînement. Le log d'erreur `"Optimizer 'None' not found in list of available optimizers"` indique qu'ils ne sont pas directement supportés par l'implémentation de la bibliothèque Ultralytics.

5. Conclusion

Pour la tâche de détection des personnages d'Harry Potter avec le modèle YOLOv8n et les hyperparamètres par défaut, l'optimizer **SGD est sans équivoque le choix le plus performant.**

Il a non seulement permis au modèle d'atteindre une précision élevée (81.7% mAP50), mais il a également démontré une stabilité et une efficacité largement supérieures aux alternatives testées. Les optimizers adaptatifs, bien que populaires, se sont avérés être un mauvais choix dans ce contexte précis.

Ce benchmark démontre l'importance de ne pas se fier aveuglément aux choix par défaut populaires et de toujours valider expérimentalement les meilleures options pour une tâche donnée.

You asked:

peut tu faire le document en entier

💡 Gemini Replied:

Bien sûr, voici à nouveau le rapport de recherche complet, structuré et rédigé à partir des résultats que vous avez obtenus.

Rapport de Benchmark : Sélection de l'Optimizer Optimal pour la Détection de Personnages (Projet Nimbus 3000)

- **Auteur** : merlin-mordrede
- **Date** : 16 octobre 2025

1. Introduction

Ce document présente les résultats d'une étude comparative visant à déterminer l'optimizer le plus performant pour l'entraînement d'un réseau de neurones convolutif **YOLOv8n**. L'objectif est la détection de personnages de la saga Harry Potter sur un jeu de données personnalisé.

Conformément au cahier des charges du défi, plusieurs optimizers populaires ont été testés : **SGD**, **Adam**, **AdamW**, **RMSProp**, **Adagrad**, et **Adadelta**. L'évaluation se base sur la métrique de performance **mAP50** (mean Average Precision), qui mesure la précision de la détection.


2. Méthodologie

- **Modèle** : YOLOv8n, pré-entraîné sur le dataset COCO.
 - **Dataset** : Un jeu de données personnalisé contenant 1722 images pour l'entraînement et 47 pour la validation, réparties sur 13 classes de personnages.
 - **Environnement** : L'entraînement a été réalisé sur un GPU NVIDIA Tesla T4 avec le framework PyTorch et la bibliothèque Ultralytics.
 - **Protocole de Test** :
 - Chaque optimizer a été utilisé pour entraîner le modèle pendant **10 époques**.
 - Les hyperparamètres, notamment le taux d'apprentissage (learning rate), ont été laissés à leur valeur par défaut fournie par la bibliothèque pour garantir une comparaison équitable des algorithmes eux-mêmes.
 - Les métriques de performance ont été enregistrées à l'aide de Weights & Biases.
-

3. Résultats

Les entraînements ont révélé des différences de performance très significatives entre les optimizers testés. De plus, des problèmes de compatibilité ont été rencontrés avec certains d'entre eux.

Le tableau ci-dessous synthétise les scores finaux obtenus pour chaque test.

Optimizer	Score mAP50	Score mAP50-95	Statut
 SGD	81.7%	46.4%	Succès
Adam	49.2%	25.7%	Succès
AdamW	41.7%	21.0%	Succès
RMSProp	5.9%	2.5%	Succès (performance très faible)

Optimizer	Score mAP50	Score mAP50-95	Statut
Adagrad	-	-	Échec (non supporté)
Adadelta	-	-	Échec (non supporté)

4. Analyse et Discussion

Analyse des Performances

Les résultats sont sans appel. L'optimizer **SGD (Stochastic Gradient Descent)** avec momentum a surpassé tous les autres de manière spectaculaire, atteignant un score mAP50 de **81.7%**.

- Les optimizers adaptatifs de la famille Adam (**Adam** et **AdamW**) ont produit des résultats médiocres en comparaison, avec des scores respectifs de 49.2% et 41.7%. Bien que ces optimizers soient souvent réputés pour leur vitesse de convergence, ils se sont ici installés dans un minimum local de performance inférieure.
- L'optimizer **RMSProp** a été un échec total sur le plan de la performance. Avec un score de 5.9%, le modèle n'a pratiquement rien appris, indiquant que les hyperparamètres par défaut de cet optimizer sont totalement inadaptés à cette tâche.

Analyse des Échecs Techniques

Les optimizers **Adagrad** et **Adadelta** ont provoqué une erreur au lancement de l'entraînement. Le log d'erreur `"Optimizer 'None' not found in list of available optimizers"` indique qu'ils ne sont pas directement supportés par l'implémentation de la bibliothèque Ultralytics.

5. Conclusion

Pour la tâche de détection des personnages d'Harry Potter avec le modèle YOLOv8n et les hyperparamètres par défaut, l'optimizer **SGD est sans équivoque le choix le plus performant.**

Il a non seulement permis au modèle d'atteindre une précision élevée (81.7% mAP50), mais il a également démontré une stabilité et une efficacité largement supérieures aux alternatives testées. Les optimizers adaptatifs, bien que populaires, se sont avérés être un mauvais choix dans ce contexte précis.

Ce benchmark démontre l'importance de ne pas se fier aveuglément aux choix par défaut populaires et de toujours valider expérimentalement les meilleures options pour une tâche donnée.

Ce format vous convient-il, ou y a-t-il une section spécifique que vous aimeriez que je développe davantage ?

You asked:

peut tu donner un classement et aussi que fait chaque opti et que sont leur avantage et incon

🌟 Gemini Replied:

Absolument. Voici le classement final basé sur vos résultats, suivi d'une explication détaillée de chaque optimizer, avec leurs avantages et inconvénients.

🏆 Classement des Optimizers pour le Projet Nimbus 3000

Basé sur le score de performance principal (`mAP50`), voici le classement de votre benchmark :

1. 🥇 **SGD**

- **Score mAP50 : 81.7%**
- **Verdict :** Vainqueur incontesté. A fourni des résultats très précis et fiables.

2. 🥈 **Adam**

- **Score mAP50 : 49.2%**
- **Verdict :** Performance moyenne. Nettement inférieur à SGD, le modèle a eu plus de mal à apprendre efficacement.

3. 🥉 **AdamW**

- **Score mAP50 : 41.7%**
- **Verdict :** Performance décevante. Légèrement moins bon qu'Adam, il n'a pas montré d'avantage dans ce test.

4. **RMSPProp**

- **Score mAP50 : 5.9%**
- **Verdict :** Échec de l'apprentissage. Le modèle n'a quasiment rien appris, rendant cet optimizer inutilisable avec les réglages par défaut.

Non classés (Échec Technique)

- **Adagrad et Adadelata :** Ces deux optimizers n'ont pas pu être testés car ils ne sont pas directement supportés par la bibliothèque `ultraalytics` utilisée, ce qui a provoqué une erreur au lancement.

🧙 Grimoire des Optimizers : À Quoi Servent-ils ?

Voici une explication de chaque optimizer, de leur fonctionnement et de leurs forces/faiblesses.

1. SGD (Stochastic Gradient Descent)

- **Principe** : C'est l'optimizer le plus "classique". Imaginez que vous êtes au sommet d'une montagne et que vous voulez atteindre la vallée (le point où l'erreur est minimale). SGD vous fait faire de petits pas dans la direction de la pente la plus raide. On y ajoute presque toujours du "**momentum**", qui agit comme une inertie : si vous descendez dans la même direction depuis un moment, vous accélérez.
- **Avantages** :
 - **Meilleure Généralisation** : Il trouve souvent des solutions plus "larges" qui fonctionnent mieux sur des données nouvelles. C'est probablement la raison de son succès dans votre test.
 - **Stabilité** : C'est un algorithme très bien étudié et compris.
 - **Moins de Mémoire** : Il est moins gourmand en ressources que les autres.
- **Inconvénients** :
 - **Convergence Lente** : Il peut prendre plus de temps pour s'entraîner.
 - **Sensible au "Learning Rate"** : Le choix de la "taille des pas" est crucial et difficile à régler.

2. Adam (Adaptive Moment Estimation)

- **Principe** : C'est le "couteau suisse" des optimizers. Au lieu d'utiliser la même taille de pas pour tout le monde, Adam **adapte la vitesse d'apprentissage pour chaque paramètre** du modèle. Il le fait en se basant sur une estimation de la "vitesse" (premier moment) et de l'"accélération" (deuxième moment) des gradients.
- **Avantages** :
 - **Convergence Rapide** : Il est souvent beaucoup plus rapide que SGD, surtout au début de l'entraînement.
 - **Facile à utiliser** : Ses paramètres par défaut fonctionnent bien sur une grande variété de problèmes.
- **Inconvénients** :
 - **Généralisation Parfois Inférieure** : Sa vitesse peut le faire converger vers une solution "pointue" mais moins bonne que celle trouvée par SGD. Ceci peut expliquer pourquoi il a obtenu un moins bon score dans votre test.
 - **Plus Gourmand en Mémoire.**

3. AdamW (Adam with Weight Decay)

- **Principe** : C'est une version améliorée d'Adam. Il corrige une erreur dans la manière dont Adam gère la "régularisation" (une technique pour éviter que le modèle n'apprenne "par cœur").

AdamW applique cette régularisation de manière plus efficace, ce qui aide souvent le modèle à mieux généraliser.

- **Avantages :**

- **Souvent Meilleur qu'Adam :** Il donne généralement de meilleurs résultats finaux qu'Adam, surtout sur les grands modèles.

- **Inconvénients :**

- Partage les mêmes inconvénients potentiels qu'Adam (généralisation parfois moins bonne que SGD).

4. RMSProp (Root Mean Square Propagation)

- **Principe :** C'est un autre optimiseur adaptatif, un peu comme un ancêtre d'Adam. Il adapte aussi le taux d'apprentissage pour chaque paramètre, mais en se basant uniquement sur une moyenne de "l'accélération" des gradients.

- **Avantages :**

- Efficace pour les problèmes où les données changent beaucoup (non-stationnaires).

- **Inconvénients :**

- **Très Sensible aux Paramètres :** Comme vous l'avez vu, ses paramètres par défaut peuvent être catastrophiques pour certaines tâches. Il est aujourd'hui moins utilisé au profit d'Adam/AdamW.

Souhaitez-vous que je développe un concept en particulier, comme le "learning rate" ou le "momentum" ?

You asked:

fait le rendu avec tout cela

✦ Gemini Replied:

Absolument. Voici le rendu final complet qui synthétise l'ensemble de notre travail : le protocole, les résultats, le classement, l'analyse détaillée et le guide théorique des optimizers.

Ce document constitue le livrable complet pour le **Défi n°21 : Le Nimbus 3000**.

Livrable Final : Défi n°21 - Le Nimbus 3000

- **Auteur :** merlin-mordrede
- **Date :** 16 octobre 2025

- **Projet** : Benchmark d'optimizers pour un modèle de détection d'objets YOLOv8n dans le cadre du Workshop "Poudlard à l'EPSI/WIS".

1. Protocole Expérimental

L'objectif de ce projet était de comparer plusieurs algorithmes d'optimisation (optimizers) afin de déterminer le plus performant pour l'entraînement d'un modèle **YOLOv8n** sur une tâche de détection de personnages de la saga Harry Potter.

- **Méthodologie** : Un script Python a été développé pour automatiser l'entraînement du modèle avec chaque optimizer listé ci-dessous. Chaque test a été mené sur **10 époques** avec des hyperparamètres par défaut pour assurer une comparaison équitable.
- **Environnement** : Les entraînements ont été effectués sur un GPU NVIDIA Tesla T4. Le suivi des métriques a été réalisé via la plateforme Weights & Biases.
- **Optimizers Testés** : SGD, Adam, AdamW, RMSProp, Adagrad, Adadelta.

2. Résultats et Classement Final

L'expérimentation a montré des écarts de performance très importants. Le classement suivant est basé sur le score **mAP50**, qui mesure la précision globale de la détection.

🏆 Classement des Performances

1. 🥇 **SGD** : **81.7%** mAP50
2. 🥈 **Adam** : **49.2%** mAP50
3. 🥉 **AdamW** : **41.7%** mAP50
4. **RMSProp** : **5.9%** mAP50

***Note** : Les optimizers Adagrad et Adadelta ont échoué pour des raisons techniques (non-compatibilité avec la bibliothèque) et ne sont donc pas classés.*

Tableau Récapitulatif des Résultats

Optimizer	Score mAP50	Score mAP50-95	Statut
🥇 SGD	81.7%	46.4%	Succès
Adam	49.2%	25.7%	Succès
AdamW	41.7%	21.0%	Succès
RMSProp	5.9%	2.5%	Succès (performance très faible)
Adagrad	-	-	Échec (non supporté)

Optimizer	Score mAP50	Score mAP50-95	Statut
Adadelta	-	-	Échec (non supporté)

3. Rapport d'Analyse

Analyse des Performances

Les résultats sont sans appel : l'optimizer **SGD (Stochastic Gradient Descent)** a offert des performances largement supérieures à toutes les alternatives. Avec un score de **81.7%**, il a démontré sa capacité à guider efficacement le modèle vers une solution très performante.

À l'inverse, les optimizers adaptatifs comme **Adam (49.2%)** et **AdamW (41.7%)**, bien que souvent plus rapides en début d'entraînement, ont convergé vers des solutions de qualité nettement inférieure. Le cas de **RMSProp (5.9%)** est extrême : il n'a pas permis au modèle d'apprendre, ce qui indique une forte incompatibilité de ses réglages par défaut avec notre tâche.

Analyse des Échecs Techniques

Les optimizers **Adagrad** et **Adadelta** n'ont pas pu être évalués en raison d'une incompatibilité avec l'implémentation de la bibliothèque `ultralytics`, qui ne les inclut pas dans sa liste d'options par défaut.

4. Conclusion & Recommandation

Pour la tâche de détection des personnages d'Harry Potter avec le modèle YOLOv8n, l'optimizer SGD est incontestablement le choix optimal.

Ce benchmark souligne un point crucial en deep learning : la popularité ou la complexité d'un algorithme ne garantit pas sa supériorité. Le classique et robuste SGD a prouvé qu'il restait une référence, capable de surpasser des méthodes plus modernes mais moins adaptées à ce contexte spécifique.

5. Annexe : Grimoire des Optimizers

- **SGD (Stochastic Gradient Descent)** : Le plus fondamental. Il fait avancer l'apprentissage pas à pas dans la direction de l'erreur la plus forte. L'ajout de "momentum" (inertie) l'aide à accélérer et à éviter les mauvais chemins. Il est réputé pour trouver des solutions qui généralisent très bien.
- **Adam (Adaptive Moment Estimation)** : Le "couteau suisse". Il adapte la vitesse d'apprentissage pour chaque paramètre du modèle, ce qui le rend très rapide. Cependant, cette vitesse peut parfois l'amener à une solution moins optimale que celle trouvée par SGD.
- **AdamW (Adam with Weight Decay)** : Une version corrigée et améliorée d'Adam. Il gère mieux la régularisation (une technique anti-triche pour l'IA), ce qui conduit souvent à de meilleures performances qu'Adam.

- **RMSProp (Root Mean Square Propagation)** : Un autre optimiseur adaptatif qui ajuste la vitesse d'apprentissage. Il est historiquement important mais a été largement remplacé par Adam et AdamW, car il peut être très sensible aux réglages.

You asked:

peut tu aussi denne des explication par raport au metrique

✦ Gemini Replied:

Bien sûr. Comprendre ces métriques est la clé pour juger de la performance de votre modèle. Voici une explication simple pour chacune d'elles, en partant du concept le plus fondamental.

Le Concept Clé : l'IoU (Intersection over Union)

Avant tout, il faut comprendre l'**IoU**. C'est le score qui mesure à quel point la boîte dessinée par votre IA (la *prédiction*) correspond à la boîte de la vraie réponse (la *vérité terrain*).

- **Principe** : C'est simplement le ratio entre la **surface d'intersection** des deux boîtes et la **surface totale** qu'elles occupent. Le score va de 0 (aucune superposition) à 1 (superposition parfaite).
- **Analogie** : Imaginez que vous demandez à l'IA de colorier un carré sur une feuille. L'IoU mesure à quel point le coloriage de l'IA correspond au vrai carré. Un score de 0.7 signifie que 70% de la zone est correctement coloriée.

Pour décider si une détection est "bonne", on fixe un seuil. Par exemple : "la détection est correcte si l'IoU est supérieur à 0.5".

P — La Précision (Precision)

La question à laquelle elle répond : "Quand le modèle dit qu'il a trouvé un personnage, a-t-il raison ?"

- **Principe** : C'est le pourcentage de détections correctes parmi toutes les détections que le modèle a faites.
- **Analogie (La pêche)** : Vous pêchez des saumons. La **précision** mesure la proportion de saumons dans votre seau. Si vous avez 10 poissons dans votre seau, dont 9 sont des saumons et 1 est une vieille botte, votre précision est de 90%.
- **En résumé** : Une **haute précision** signifie que le modèle ne fait pas beaucoup de "fausses alertes".

R — Le Rappel (Recall)

La question à laquelle elle répond : "Sur tous les personnages qui existent réellement dans l'image, combien le modèle en a-t-il trouvé ?"

- **Principe** : C'est le pourcentage de personnages réels que le modèle a réussi à trouver.
- **Analogie (La pêche, suite)** : Le **rappel** mesure combien de saumons de l'étang vous avez effectivement attrapés. S'il y avait 20 saumons en tout dans l'étang et que vous en avez pêché 9, votre rappel est de 45%. Vous n'avez pas attrapé de bottes (haute précision), mais vous avez manqué beaucoup de saumons (faible rappel).
- **En résumé** : Un **haut rappel** signifie que le modèle "n'oublie" personne et trouve bien tous les personnages présents.

mAP50 (mean Average Precision @ IoU=0.50)

C'est la **métrique de performance principale** pour la détection d'objets. Décortiquons-la :

- **AP (Average Precision)** : C'est un score unique qui combine la Précision et le Rappel pour une seule classe (par exemple, "Harry Potter"). Il donne une vision globale de la performance pour cette classe.
- **mAP (mean Average Precision)** : C'est simplement la **moyenne des scores AP** sur toutes vos classes (Harry, Hermione, Dumbledore, etc.).
- **@50** : C'est la partie la plus importante. Cela signifie que pour qu'une détection soit comptée comme "correcte", son **IoU doit être d'au moins 0.50** (la boîte prédite doit recouvrir au moins 50% de la vraie boîte).

En résumé : Le `mAP50` est le **score de précision global** de votre modèle, avec une condition assez souple sur la qualité du placement des boîtes. C'est le score que l'on regarde le plus souvent pour comparer des modèles.

mAP50-95

- **Principe** : C'est une version **beaucoup plus stricte** du mAP. Au lieu de ne vérifier la qualité qu'à un seuil d'IoU de 0.50, elle calcule le mAP à 10 seuils différents (0.50, 0.55, 0.60, ... jusqu'à 0.95) et fait ensuite la moyenne de ces 10 scores.
- **Ce que ça signifie** : Pour obtenir un bon score `mAP50-95`, le modèle doit non seulement trouver les bons personnages, mais aussi dessiner des boîtes de détection **extrêmement précises**. C'est pourquoi ce score est toujours plus bas que le `mAP50`. Il mesure à la fois la capacité de détection et la précision de la localisation.

You asked:

peut tu lincorporer au document final et le faire en entier

🌟 Gemini Replied:

Absolument. Voici le document final complet et consolidé. Il intègre maintenant le rapport de benchmark, le classement, le guide des optimizers, et le guide détaillé des métriques de performance, le tout en un seul livrable.

Livrable Final : Défi n°21 - Le Nimbus 3000

- **Auteur** : merlin-mordrede
- **Date** : 16 octobre 2025
- **Projet** : Benchmark d'optimizers pour un modèle de détection d'objets YOLOv8n dans le cadre du Workshop "Poudlard à l'EPSI/WIS".

1. Protocole Expérimental

L'objectif de ce projet était de comparer plusieurs algorithmes d'optimisation (optimizers) afin de déterminer le plus performant pour l'entraînement d'un modèle **YOLOv8n** sur une tâche de détection de personnages de la saga Harry Potter.

- **Méthodologie** : Un script Python a été développé pour automatiser l'entraînement du modèle avec chaque optimizer listé ci-dessous. Chaque test a été mené sur **10 époques** avec des hyperparamètres par défaut pour assurer une comparaison équitable.
- **Environnement** : Les entraînements ont été effectués sur un GPU NVIDIA Tesla T4. Le suivi des métriques a été réalisé via la plateforme Weights & Biases.
- **Optimizers Testés** : SGD, Adam, AdamW, RMSProp, Adagrad, Adadelta.

2. Résultats et Classement Final


L'expérimentation a montré des écarts de performance très importants. Le classement suivant est basé sur le score **mAP50**, qui mesure la précision globale de la détection.

🏆 Classement des Performances

1. 🥇 **SGD** : 81.7% mAP50
2. 🥈 **Adam** : 49.2% mAP50
3. 🥉 **AdamW** : 41.7% mAP50
4. **RMSProp** : 5.9% mAP50

Note : Les optimizers Adagrad et Adadelata ont échoué pour des raisons techniques (non-compatibilité avec la bibliothèque) et ne sont donc pas classés.

Tableau Récapitulatif des Résultats

Optimizer	Score mAP50	Score mAP50-95	Statut
 SGD	81.7%	46.4%	Succès
Adam	49.2%	25.7%	Succès
AdamW	41.7%	21.0%	Succès
RMSProp	5.9%	2.5%	Succès (performance très faible)
Adagrad	-	-	Échec (non supporté)
Adadelata	-	-	Échec (non supporté)

3. Rapport d'Analyse

Analyse des Performances

Les résultats sont sans appel : l'optimizer **SGD (Stochastic Gradient Descent)** a offert des performances largement supérieures à toutes les alternatives. Avec un score de **81.7%**, il a démontré sa capacité à guider efficacement le modèle vers une solution très performante.

À l'inverse, les optimizers adaptatifs comme **Adam (49.2%)** et **AdamW (41.7%)**, bien que souvent plus rapides en début d'entraînement, ont convergé vers des solutions de qualité nettement inférieure. Le cas de **RMSProp (5.9%)** est extrême : il n'a pas permis au modèle d'apprendre, ce qui indique une forte incompatibilité de ses réglages par défaut avec notre tâche.

Analyse des Échecs Techniques

Les optimizers **Adagrad** et **Adadelata** n'ont pas pu être évalués en raison d'une incompatibilité avec l'implémentation de la bibliothèque `ultralytics`, qui ne les inclut pas dans sa liste d'options par défaut.

4. Conclusion & Recommandation

Pour la tâche de détection des personnages d'Harry Potter avec le modèle YOLOv8n, l'optimizer SGD est incontestablement le choix optimal.

Ce benchmark souligne un point crucial en deep learning : la popularité ou la complexité d'un algorithme ne garantit pas sa supériorité. Le classique et robuste SGD a prouvé qu'il restait une référence, capable de surpasser des méthodes plus modernes mais moins adaptées à ce contexte spécifique.

Annexe 1 : Grimoire des Optimizers

- **SGD (Stochastic Gradient Descent)** : Le plus fondamental. Il fait avancer l'apprentissage pas à pas dans la direction de l'erreur la plus forte. L'ajout de "momentum" (inertie) l'aide à accélérer et à éviter les mauvais chemins. Il est réputé pour trouver des solutions qui généralisent très bien.
 - **Adam (Adaptive Moment Estimation)** : Le "couteau suisse". Il adapte la vitesse d'apprentissage pour chaque paramètre du modèle, ce qui le rend très rapide. Cependant, cette vitesse peut parfois l'amener à une solution moins optimale que celle trouvée par SGD.
 - **AdamW (Adam with Weight Decay)** : Une version corrigée et améliorée d'Adam. Il gère mieux la régularisation (une technique anti-triche pour l'IA), ce qui conduit souvent à de meilleures performances qu'Adam.
 - **RMSProp (Root Mean Square Propagation)** : Un autre optimizer adaptatif qui ajuste la vitesse d'apprentissage. Il est historiquement important mais a été largement remplacé par Adam et AdamW, car il peut être très sensible aux réglages.
-

Annexe 2 : Guide des Métriques de Performance

IoU (Intersection over Union)

C'est le score de base qui mesure à quel point la boîte dessinée par l'IA (*prédiction*) correspond à la vraie boîte (*vérité terrain*). Il s'agit du ratio entre la surface de superposition des deux boîtes et la surface totale qu'elles occupent (de 0 à 1). Une détection est considérée "correcte" si son IoU dépasse un certain seuil (par ex. 0.5).

P — La Précision (Precision)

- **Question** : "Quand le modèle dit qu'il a trouvé un personnage, a-t-il raison ?"
- **Définition** : Le pourcentage de prédictions correctes parmi toutes les prédictions faites par le modèle.
- **Conclusion** : Une **haute précision** signifie que le modèle ne génère pas de "fausses alertes".

R — Le Rappel (Recall)

- **Question** : "Sur tous les personnages qui existent réellement, combien le modèle en a-t-il trouvé ?"
- **Définition** : Le pourcentage de personnages réels que le modèle a réussi à identifier.
- **Conclusion** : Un **haut rappel** signifie que le modèle oublie peu de personnages.

mAP50 (mean Average Precision @ IoU=0.50)

C'est la **métrique principale** pour évaluer la performance globale.

- **AP (Average Precision)** : Un score unique qui combine Précision et Rappel pour une seule classe.
- **mAP (mean Average Precision)** : La **moyenne des scores AP** sur toutes les classes de personnages.
- **@50** : Indique que le seuil d'IoU pour qu'une détection soit jugée correcte est de **0.50**.
- **Conclusion** : C'est le score de performance global du modèle, avec une condition assez souple sur le placement des boîtes.

mAP50-95

C'est une version **beaucoup plus stricte** du mAP. Elle calcule la moyenne des scores mAP sur 10 seuils d'IoU différents (de 0.50 à 0.95).

- **Conclusion** : Ce score pénalise fortement les boîtes mal placées. Un bon score ici signifie que le modèle est non seulement capable d'identifier les personnages, mais aussi de les localiser avec une **très grande précision**.