



UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Recherche d'objets dans une base de données à partir d'images

Rapport de TO52 – P2024

ALBRECHT Elise

INFO4
DS

Encadrant UTBM
ABBAS-TURKI Abdeljalil

Contents

1	Introduction	2
2	Etat de l'art	3
3	Fonctionnement général	4
4	Présentation du dataset	5
5	Entraînement avec YOLOv8	7
5.1	Présentation de YOLO	7
5.2	Adapter le dataset	7
5.3	Entraînement du model	10
5.3.1	Entraînement avec les ressources de calcul du serveur de l'UTBM	10
5.3.2	Entraînement sur ma carte graphique	10
5.4	Résultats de l'entraînement	11
5.4.1	Interprétation des résultats	11
5.4.2	Résultats	12
6	Base de données avec les images	15
7	Similarités	16
8	Interface et utilisation	18
9	Limites du projet et améliorations	20
9.1	Limites	20
9.2	Améliorations	20
9.2.1	Améliorer le modèle de prédiction	20
9.2.2	Réduire le nombre de comparaisons	21
9.3	Conclusion	22

1 Introduction

Le projet vise à développer un programme permettant d'envoyer une requête sous la forme d'une image accompagnée éventuellement d'options textuelles. Le programme doit ensuite retourner les images d'objets similaires en tenant compte de critères spécifiques ajoutés. Les objets à rechercher sont des vêtements, ce qui implique la reconnaissance du type (t-shirt, pantalon, veste, etc.), du style (couleur, forme, etc.) et de la matière (jean, sequins, dentelle, etc.). Les options saisies manuellement peuvent inclure des contraintes telles que la marque et la taille du vêtement.

Le travail à réaliser comprend plusieurs volets. Tout d'abord, il s'agit de concevoir une interface utilisateur permettant d'envoyer des requêtes sous forme d'image avec texte. Cette interface doit être capable d'afficher les éléments similaires visuellement, dans un ordre de pertinence, tout en respectant les contraintes textuelles. Ensuite, il faut adapter une base de données contenant une variété de vêtements pour qu'elle puisse être utilisée efficacement par le programme. Enfin, un script incluant un algorithme d'apprentissage doit être développé pour détecter les similitudes entre les vêtements. Cet algorithme devra être capable de comparer les images des vêtements envoyées par les utilisateurs avec celles de la base de données, en tenant compte des critères supplémentaires fournis sous forme de texte.

L'objectif final est de connecter ce programme à la plateforme Vinted [1] pour permettre aux utilisateurs de trouver des vêtements similaires en vente. En intégrant cette fonctionnalité à Vinted, les utilisateurs pourraient non seulement identifier des articles similaires dans leur propre garde-robe, mais également découvrir des articles en vente qui correspondent à leurs préférences spécifiques, facilitant ainsi le processus de recherche et d'achat de vêtements sur la plateforme. Cette partie a néanmoins été laissée de côté dans le cadre de cette matière, et fera l'objet d'un travail personnel annexe.

Ma motivation est multiple. Tout d'abord, c'est un projet qui me tient à cœur, car j'apprécie l'application Vinted, mais trouve qu'il est parfois très difficile de trouver un article précis. Ensuite, c'était l'occasion pour moi de me challenger avec un projet concret et complet : le projet couvre de nombreux sujets, de la Data Science, de l'IA, de la visualisation de résultats, le tout avec un cadre semi-professionnel. Bien que le projet soit réalisé pour mes cours, j'ai eu une date de rendu, des meetings avec mon enseignant encadrant, et je devais produire un rendu de mon travail.

2 Etat de l'art

L'utilisation de modèles YOLO (You Only Look Once) pour la détection et la classification des vêtements est un domaine de recherche actif, visant à améliorer la précision et l'efficacité des systèmes de reconnaissance d'images dans le contexte de la mode. L'article "Clothing Detection and Classification with Fine-Tuned YOLO-Based Models" de Hai T. Nguyen et al. présente une approche approfondie utilisant ces modèles pour la détection de vêtements. [2]

L'étude a collecté et annoté plus de 10 000 images contenant 18 types de vêtements différents, allant des t-shirts aux vêtements traditionnels vietnamiens comme l'ao dai. Les chercheurs ont utilisé le modèle YOLOv5 avec des ensembles de paramètres hyperparamétriques ajustés pour améliorer la précision de la détection par rapport aux versions précédentes de YOLO. En particulier, le modèle YOLOv5l a été choisi pour ses capacités de précision et de traitement des images de plus petite taille, optimisant ainsi la reconnaissance des vêtements.

L'étude a comparé plusieurs ensembles d'hyperparamètres, notamment Hyp-default (valeurs par défaut), Hyp-med, Hyp-high et Hyp-evolve. Les résultats ont montré que les ensembles Hyp-high et Hyp-evolve atteignaient les scores F1 les plus élevés, indiquant une meilleure performance en termes de précision et de rappel. Par exemple, l'ensemble Hyp-high a obtenu un score F1 de 0.693 et une précision de détection des objets de 0.933 sur le jeu de données de test.

Les résultats expérimentaux ont montré que le modèle ajusté YOLOv5 avec les hyperparamètres Hyp-high et Hyp-evolve performait mieux que les versions précédentes comme YOLOv4 et les paramètres par défaut de YOLOv5. Le modèle a atteint une précision de 93,3 % pour la reconnaissance des vêtements et une précision de 77,2 % pour la classification avec l'ensemble Hyp-high. Les pertes observées pendant l'entraînement et la validation ont démontré une convergence stable, évitant le surapprentissage grâce à la technique de l'arrêt anticipé.

L'étude a démontré que l'utilisation de modèles YOLOv5 finement ajustés peut significativement améliorer la détection et la classification des vêtements. Bien que les résultats soient prometteurs, il reste des défis à surmonter, notamment la reconnaissance des vêtements de petite taille ou partiellement occultés et la différenciation des vêtements aux couleurs ou formes similaires. Les futures recherches devraient se concentrer sur l'amélioration de la précision pour ces cas complexes afin de permettre une application pratique efficace dans les systèmes de commerce électronique.

3 Fonctionnement général

Le programme réalise le travail suivant : une image d'un vêtement est analysée à l'aide d'un modèle YOLO pour prédire sa catégorie, et cette catégorie est utilisée pour rechercher et retourner des images similaires d'une base de données.

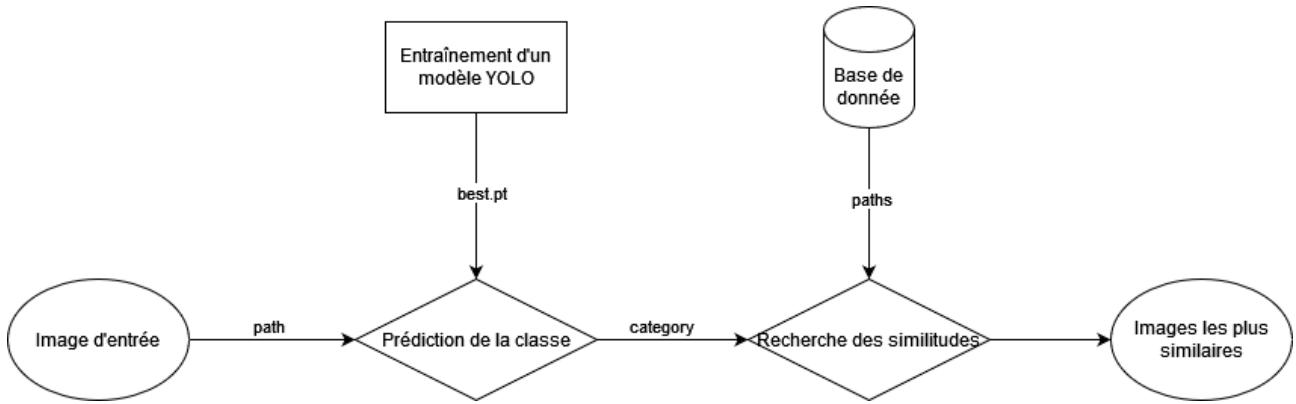


Figure 1: Fonctionnement général du projet

Image d'entrée : Le processus commence par l'acquisition d'une image d'entrée, qui est l'image du vêtement que l'utilisateur souhaite analyser.

Prédition de la classe : L'image d'entrée est ensuite transmise au modèle YOLO préalablement entraîné. Le modèle YOLO, utilisant le fichier best.pt (qui représente les poids du modèle optimisé après l'entraînement), effectue une prédiction pour identifier la classe du vêtement présent dans l'image. La sortie de cette étape est la catégorie du vêtement (par exemple, t-shirt, pantalon, veste).

Recherche des similarités : Une fois que la catégorie du vêtement a été prédite, cette information est utilisée pour interroger une base de données contenant des chemins d'accès à diverses images de vêtements. Le processus de recherche dans la base de données identifie les images qui sont les plus similaires à l'image d'entrée en fonction de la catégorie prédite et d'autres critères de similarité.

Images les plus similaires : Les résultats de la recherche sont les images de la base de données qui ressemblent le plus à l'image d'entrée. Ces images sont ensuite présentées à l'utilisateur comme les options les plus similaires disponibles.

Le travail réalisé se découpe en quatre grandes étapes :

- L'entraînement d'un modèle YOLO pour reconnaître la catégorie de vêtement 5
- La création d'une base de données avec des images de vêtements qui seront comparées à la photo entrée par l'utilisateur 6
- La comparaison entre les images, en les vectorisant 7
- L'interface pour permettre à l'utilisateur d'utiliser facilement le programme 8

4 Présentation du dataset

Tout d'abord, il a fallu trouver un dataset. Ce dernier a deux objectifs majeurs : entraîner notre modèle à reconnaître des types de vêtements et servir de base de donnée pour comparer les images. J'ai choisi d'utiliser le dataset "DeepFashion" [3].



Ce dernier est une vaste collection d'images dédiée à la recherche et au développement de systèmes de reconnaissance et de classification de vêtements. Il contient plus de 800 000 images d'articles de mode variés, englobant des vêtements pour hommes et femmes. Ces images couvrent une large gamme de catégories telles que robes, t-shirts, pantalons, jupes, manteaux, vestes, et bien d'autres types de vêtements. Elles peuvent présenter des habits portés (sur mannequins) ou non.



(a) Exemple de robe



(b) Exemple de top



(c) Exemple de jean

Figure 2: Exemple de photos du dataset DeepFashion

Catégorie	Nombre de photos	Catégorie	Nombre de photos
Anorak	160	Hoodie	4048
Blazer	7495	Jacket	10467
Blouse	24557	Jersey	748
Bomber	309	Parka	676
Button-Down	330	Peacoat	97
Cardigan	13311	Poncho	791
Flannel	324	Sweater	13123
Halter	17	Tank	15429
Henley	716	Tee	36887
Top	10078	Turtleneck	146
Capris	77	Chinos	527
Culottes	486	Cutoffs	1669
Gauchos	49	Jeans	7076
Jeggings	594	Jodhpurs	45
Joggers	4416	Leggings	5013
Sarong	32	Shorts	19666
Skirt	14773	Sweatpants	3048
Sweatshorts	1106	Trunks	386
Caftan	54	Coat	2120
Coverup	17	Dress	72158
Jumpsuit	6153	Kaftan	126
Kimono	2294	Onesie	70
Robe	150	Romper	7408
Total	289222		

Table 1: Nombre d’images par catégories de vêtements

Chaque image du dataset est accompagnée de métadonnées et d’annotations détaillées. Les annotations fournissent des informations sur les catégories de vêtements, les attributs des vêtements comme la couleur, le motif et le matériau. De plus, des annotations pour les boîtes englobantes (bounding boxes) délimitent les zones d’intérêt dans les images.

Le dataset DeepFashion supporte plusieurs tâches cruciales pour la reconnaissance des vêtements. Cela inclut la catégorisation des vêtements, où l’objectif est d’identifier la catégorie d’un vêtement donné, comme une robe ou un t-shirt. En outre, il facilite la détection de points clés sur les vêtements, comme les cols et les manches, et la reconstruction 3D, qui permet de créer des modèles tridimensionnels des vêtements à partir des images bidimensionnelles.

DeepFashion est largement utilisé pour développer et évaluer des algorithmes de vision par ordinateur et d’apprentissage automatique dans le domaine de la mode. Les riches annotations et la diversité des images en font un outil précieux pour les chercheurs et les développeurs cherchant à améliorer la précision et l’efficacité des systèmes de reconnaissance d’images de vêtements, il m’a donc paru être un choix judicieux.

5 Entrainement avec YOLOv8

5.1 Présentation de YOLO

Le modèle de détection d’images Ultralytics YOLOv8 (You Only Look Once version 8) est une évolution des précédentes versions de YOLO, qui sont des modèles de détection d’objets en temps réel. [4]

L’architecture utilisée par YOLOv8 est une architecture de réseau de neurones convolutionnels (CNN) qui prend une image en entrée et la divise en une grille. Chaque cellule de la grille est responsable de la détection des objets dont le centre se trouve dans cette cellule. [5]

Pour chaque cellule de la grille, plusieurs ”boîtes d’ancrage” sont définies, avec différentes tailles et proportions pour couvrir des objets de différentes dimensions. Chaque boîte d’ancrage prédit la probabilité de présence d’un objet et les coordonnées de la boîte englobante.

Pour chaque boîte d’ancrage, le modèle prédit :

- Les coordonnées de la boîte englobante (x, y, largeur, hauteur)
- La confiance de la prédiction, c’est-à-dire la probabilité qu’un objet soit présent dans la boîte
- Les classes d’objets possibles avec leur probabilité respective

Après avoir généré toutes les prédictions, une étape de suppression non maximale est appliquée pour éliminer les prédictions redondantes et ne conserver que les boîtes avec la plus haute probabilité. [6]

YOLOv8 est entraîné sur un ensemble de données annotées avec des boîtes englobantes et des classes d’objets. L’objectif de l’entraînement est de minimiser une fonction de perte qui prend en compte la précision des coordonnées des boîtes et la précision de la classification des objets.

YOLOv8 est conçu pour être rapide et précis, ce qui le rend adapté pour les applications en temps réel. Il peut traiter des vidéos et des flux d’images en direct avec une haute efficacité.

5.2 Adapter le dataset

Pour réaliser son entraînement, YOLO nécessite une organisation particulière des données. [7]

D’abord, le dataset doit être organisé en plusieurs répertoires :

- images/ : qui contiendra toutes les images possédant l’un des formats [’bmp’, ’jpg’, ’jpeg’, ’png’, ’tif’, ’tiff’, ’dng’].
- labels/ : qui contiendra tous les labels associés aux images, au format ’txt’. Chacune d’entre elles possédera un label, avec le même nom (seul le type du fichier change)

On peut encore les découper en 2 sous-dossiers :

- train/ : qui contiendra donc les données d’entraînement
- val/ : qui contiendra les données de validation

Attention, excepté les deux sous-dossiers présentés ici, aucun autre répertoire ne doit être présent, car le modèle ne les parcourra pas : il ne trouvera pas les données.

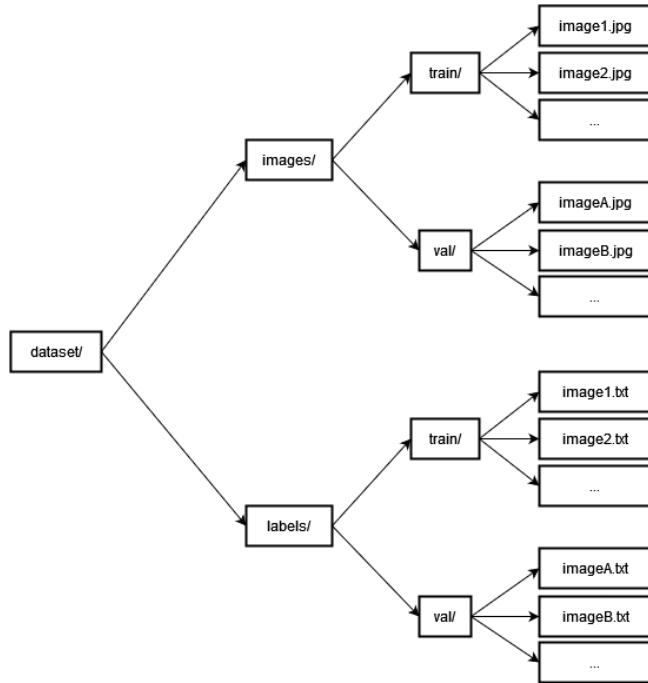


Figure 3: Architecture des répertoires à suivre

Les fichiers textes possédant les labels ont une mise en forme particulière à respecter:

`<class> <x_center> <y_center> <width> <height>`

Il y a autant de lignes qu'il y a d'éléments à observer sur l'image.

De plus, les valeurs numériques sont normalisées. Donc si le centre de l'image est situé à 120x120px pour une image qui fait 240x240px, les valeurs du x et y du centre seront 0.5



Figure 4: Exemple de boîtes englobantes

[7]

Enfin, le modèle fonctionne avec un fichier `.yaml` qui permet de traduire les classes.

```

# Train/val/test sets as 1) dir: path/to/imgs , 2) file: path/to/imgs.txt
path: ../datasets/coco8 # dataset root dir
train: images/train # train images (relative to 'path') 4 images
val: images/val # val images (relative to 'path') 4 images

# Classes (80 Coco classes)
names:
  0: person
  1: bicycle
  2: car
  #
  ...
  77: teddy bear
  78: hair drier
  79: toothbrush

```

La première partie indique au modèle où chercher les données du dataset. Si on reprend notre exemple d'architecture donné dans la figure 3, on aurait :

```

path: ../dataset # dataset root dir
train: images/train # train images (relative to 'path')
val: images/val # val images (relative to 'path')

names:
  0: Anorak
  1: Blazer
  2: Blouse
  ...
  43: Onesie
  44: Robe
  45: Romper

```

Le nom des classes a été remplacé par les catégories de vêtements présentées plus haut.

La toute première étape pour entraîner le modèle est donc de préparer les images et labels, ainsi que la configuration.yaml.

DeepFashion est composé d'images et de labels associés. Ces derniers sont du format suivant

<paths> <x_top> <y_top> <x_bottom> <y_bottom> #valeurs en px

Comme ce n'est pas le format attendu par YOLO, il a fallu modifier les valeurs.

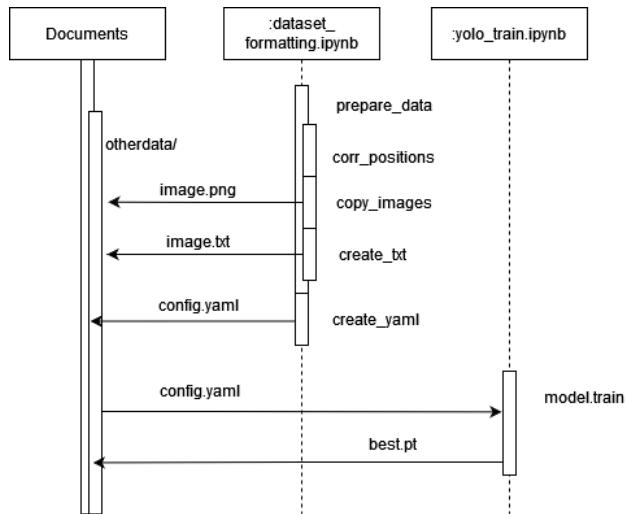


Figure 5: Diagramme de séquence résumant le travail réalisé pour mettre en forme le dataset et entraîner le modèle

Cette figure résume le travail réalisé pour mettre en forme les données à fournir au modèle. On commence par corriger les positions des boîtes englobantes en travaillant avec un DataFrame obtenu du fichier listant toutes les images et leurs labels. On enlève la mention du chemin que l'on remplace avec la classe, et on calcule les nouvelles valeurs des labels. On peut à présent copier l'image et son label vers le dossier qui figurera dans le .yaml. Ce dernier est également créé, en prenant le chemin et les classes

5.3 Entraînement du model

5.3.1 Entraînement avec les ressources de calcul du serveur de l’UTBM

Cette partie du travail, bien que frustrante, n'a pas été vaine. J'ai passé beaucoup de temps à essayer de comprendre comment je pourrais réaliser l'entraînement à distance sur le serveur. J'ai donc appris à me servir de remote SSH de VSCode, et plus généralement des commandes liées à SSH et le travail sur un serveur distant. Cela m'a permis de rafraîchir mes connaissances des commandes linux et des environnements virtuels. J'ai également pu prendre conscience de l'utilité d'un requierements.txt, et de faire tourner JupyterNotebook sur un serveur plutôt qu'en local.

Finalement, une erreur apparaissait au lancement de l'entraînement, sans que je puisse comprendre pourquoi. J'aurais certainement dû recontacter des professeurs pour demander de l'aide.

5.3.2 Entraînement sur ma carte graphique

Comme je possède un bon ordinateur portable avec une carte graphique puissante (la NVidia RTX-4070), une option possible était de réaliser l'entraînement sur mon propre ordinateur. C'est ce que j'ai fait. En revanche, d'autres problèmes sont survenus, car le programme ne trouvait pas la carte graphique, et j'ai encore perdu du temps à résoudre ce problème.

Voilà la configuration qui permet de prendre le GPU comme device pour l'entraînement YOLO :

PyTorch Version : 2.3.0+cu118
CUDA Version : 11.8
cuDNN Version : 8700

Ce dernier a été réalisé avec le modèle yolov8n.yaml, la version nano de YOLO. J'ai choisi cette version, car mes ressources sont tout de même plutôt limitées, et les résultats obtenus satisfaisants dans le cadre de ce projet. [8] L'entraînement a été réalisé sur 32 epochs, et les hyperparamètres étaient ceux par défaut.

5.4 Résultats de l'entraînement

5.4.1 Interprétation des résultats

YOLO offre nativement des indications sur l'entraînement qu'il a réalisé. Dans cette analyse, nous allons utiliser 2 résultats

Premièrement, les matrices de confusion [9]

Les matrices de confusion sont des outils puissants pour évaluer les performances des modèles de classification. Elles permettent de visualiser non seulement la précision du modèle, mais aussi ses erreurs spécifiques de classification. Voici les composantes d'une matrice de confusion :

- Vrais positifs (TP) : Nombre de fois où le modèle a correctement prédit la classe.
- Faux positifs (FP) : Nombre de fois où le modèle a incorrectement prédit une classe.
- Faux négatifs (FN) : Nombre de fois où le modèle a manqué de prédire la classe correcte.
- Vrais négatifs (TN) : Nombre de fois où le modèle a correctement prédit l'absence de la classe.

Secondement, la courbe Precision-Recall [10]

La courbe Precision-Recall montre la relation entre la précision et le rappel pour différentes classes. C'est une métrique utile pour évaluer les modèles de classification déséquilibrés. Une courbe plus proche du coin supérieur droit indique une meilleure performance. Nous utilisons en particulier la métrique mAP (pour mean Average Precision) car c'est une mesure standard pour évaluer les modèles de détection d'objets, qui prend en compte à la fois la précision et le rappel. Elle fournit une valeur unique qui résume la performance globale du modèle. Elle est une approximation de l'accuracy utilisée dans les autres modèles dans le contexte de la détection d'objets, car elle reflète la capacité du modèle à détecter et à classer correctement les objets. Notons cependant que mAP est une métrique plus stricte car elle prend en compte les localisations des objets, en plus de l'accuracy d'un modèle qui mesure la justesse de la prédiction de la classe de l'image.

		Real Label		$\rightarrow \text{Precision} = \frac{\sum \text{TP}}{\sum \text{TP} + \text{FP}}$
		Positive	Negative	
Predicted Label	Positive	True Positive (TP)	False Positive (FP)	
	Negative	False Negative (FN)	True Negative (TN)	
				$\downarrow \text{Recall} = \frac{\sum \text{TP}}{\sum \text{TP} + \text{FN}}$
				$\text{Accuracy} = \frac{\sum \text{TP} + \text{TN}}{\sum \text{TP} + \text{FP} + \text{FN} + \text{TN}}$

Figure 6: Precision, recall et accuracy [11]

5.4.2 Résultats

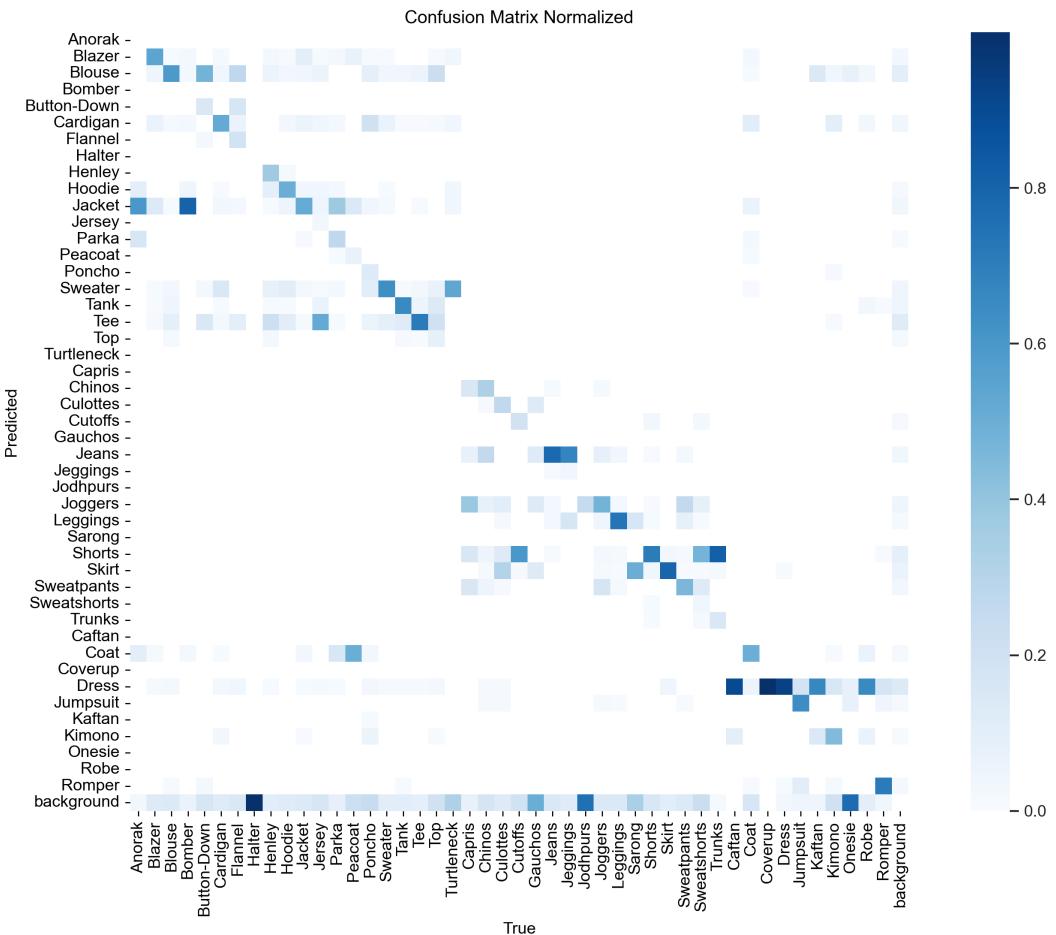


Figure 7: Matrice de confusion

Une matrice de confusion parfaite devrait avoir ses diagonales foncées, et tout le reste clair. On voit ici que ce n'est pas le cas, et qu'il y a une confusion assez importante du modèle. On

voit quatre parties se démarquer : de Anorak à Top, de Culottes à Trunks, de Coat à Romper et Background.

On voit donc que le modèle discerne bien les différents types entre haut, bas et full body, mais confond les classes à l'intérieur des types. Par exemple, les anoraks et bombers seront des jackets (vestes), et beaucoup de hauts tomberont dans la catégorie tee (pour tee-shirt).

Pour les bas, le modèle prédit essentiellement des jeans, joggers, leggings, shorts et skirt (jupes). On retrouve donc les grandes catégories, celles majoritairement portées.

Finalement, les vêtements qui couvrent tout le corps seront considérés par le modèle comme des dress (robes), et parfois des jumpsuits (combinaisons). Encore une fois, on retrouve les essentiels.

Les labels prédits comme "Background" correspondent aux images que le modèle ne réussit pas à prédire. Cela concerne essentiellement les catégories avec moins d'images.

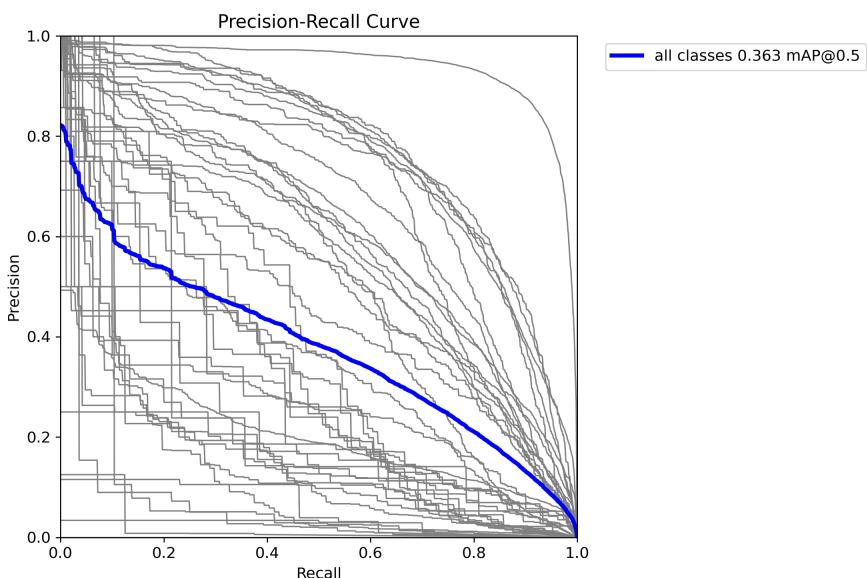


Figure 8: Courbe Precision-recall

On voit ici des courbes très distinctes, qui font écho à la matrice de confusion présentée en haut : certaines classes sont très bien retrouvées (dress, skirt, shorts, jeans, tee, top, sweater, blazer, blouse, etc) qui correspondent aux lignes vers le haut : elles ont une bonne valeur de précision et de recall. Mais l'ensemble est très nettement affaibli par les catégories que le modèle n'arrive pas à trouver (halter, onsie, turtleneck, capris, etc.) ainsi que celles qu'il confond avec d'autres catégories. On obtient donc une accuracy d'environ 36%, ce qui est extrêmement faible.

Les métriques indiquent donc un entraînement mitigé dans l'ensemble, mais on retrouve quand même au final les catégories principales de vêtements utilisées. Je pourrais difficilement faire la différence à l'œil nu entre jacket et bomber.



(a) Exemple de jacket



(b) Exemple de bomber

Figure 9: Exemple de photos du dataset DeepFashion

Les résultats mitigés concernent majoritairement les catégories qui présentent peu d'images à la base. J'aurais dû les supprimer, ou les inclure dans la grande catégorie la plus ressemblante, pour que le modèle rencontre tout de même le type. On obtient donc un modèle qui reste exploitable dans le cadre de ce projet, mais risque de fournir des résultats légèrement moins bons en articles similaires pour les vêtements de niche.

6 Base de données avec les images

Pour pouvoir comparer l'image fournie avec des images témoins, j'ai créé une base de données contenant les images utilisées pour l'entraînement.

Pour éviter de stocker plusieurs fois les images, surtout lorsqu'il y en a plus de 280000, j'ai décidé de ne stocker que le chemin vers les images, puisque ces dernières sont stockées sur mon PC.

L'architecture de la base de données est la suivante :

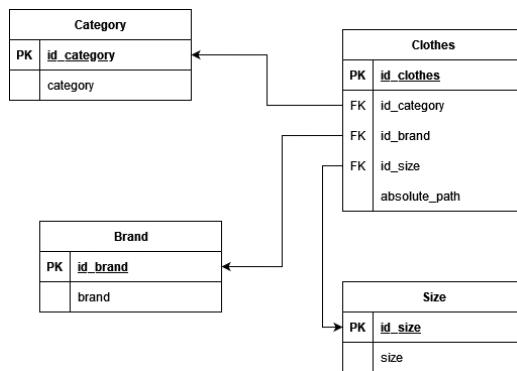


Figure 10: Diagramme entité-relation de la base de données

On a donc 4 tables :

- table Clothes : La table principale, qui possède 289222 entrées, pour chaque image. Chaque vêtement se voit attribuer une catégorie (obtenue avec les labels) ainsi qu'une taille et une marque. Ces deux dernières valeurs ont été attribuées au hasard, pour convenir au sujet initial. Dans l'idéal, cette base de donnée devrait s'appuyer sur des vrais vêtements vendus sur Vinted par exemple. Enfin, le dernier attribut est le chemin d'accès aux images. Ce dernier n'est pas absolu, pour pouvoir être portable : il s'agit du chemin fourni par les labels.
- table Category : Liste les 46 catégories avec leur numéro associé. Ils correspondent aux labels annoncés dans la configuration.yaml
- table Brand : Liste les 10 marques possibles : 'Zara', 'H&M', 'Forever 21', 'Asos', 'Jennifer', 'Promod', 'Uniqlo', 'Pull&Bear', 'ESPRIT' et 'Primark', avec un identifiant associé
- table Size : Liste les 7 tailles possibles : 'XXS', 'XS', 'S', 'M', 'L', 'XL' et 'XXL'

La base de données a été créée avec MySQL [12]. J'ai choisi ce service de base de données, car il est gratuit et open source, simple à utiliser et prendre en main, et fournit une application Workbench pour faciliter la gestion des bases [13]. J'aurais également pu utiliser PostgreSQL, qui propose des services similaires. J'ai déjà travaillé avec PostgreSQL, et ce projet était une bonne occasion pour découvrir un autre outil. Au final, je trouve que les deux se valent, et je pourrais difficilement les départager.

La base de donnée est hébergée en localhost. Pour la remplir avec les données énoncées plus haut, j'ai réalisé des DataFrames que j'ai ensuite enregistrés en .csv, avant de les téléverser dans les tables préalablement créées.

7 Similarités

Pour renvoyer à l'utilisateur les images des vêtements les plus similaires à celle envoyée, j'utilise l'extraction des caractéristiques de l'image, à l'aide d'un modèle VGG16 pré-entraîné et des fonctions de la librairie OpenCV [14]. Pour la recherche d'images similaires, l'objectif étant de capturer les composantes profondes et représentatives d'une image, VGG16 est plus adapté avec ses couches entièrement connectées. C'est un réseau de convolution profond avec 16 couches, ce qui permet d'extraire des caractéristiques riches et complexes à partir des images. [15] Ce modèle a été choisi plutôt que YOLO, car ce dernier, bien qu'il soit rapide, est plus complexe à mettre en œuvre pour l'extraction de caractéristiques globales, notamment parce qu'il est plus difficile de récupérer les couches du modèle.

Le processus permet de trouver et de classer les images les plus similaires à une image cible en utilisant à la fois des caractéristiques profondes extraites par un modèle de réseau neuronal et des informations de couleur. On commence par récupérer les composantes de l'image d'entrée, qui seront ensuite comparées à celles des images de la base de données.

La première fonction, qui extrait les éléments de l'image, se découpe en trois étapes :

- Chargement et prétraitement de l'image : on récupère l'image, qui est ensuite adaptée pour le modèle VGG16 (redimensionnement en 224*224, convertissement en tableau NumPy, normalisation des pixels)
- Extraction des caractéristiques avec VGG16 : Les données prétraitées de l'image sont passées dans le modèle VGG16 pour extraire des caractéristiques à partir de la couche Fully Connected 1 du modèle.
- Extraction de l'histogramme de couleurs : grâce à la fonction calcHist d'OpenCV, je peux récupérer les valeurs HSV [16]:
 - Hue (teinte) : Représente la couleur pure, indépendamment de la luminosité et de la saturation
 - Saturation (saturation) : Représente la pureté de la couleur, de la teinte la plus pure au gris
 - Value (valeur) : Représente la luminosité de la couleur

La séparation de ces composantes permet de traiter les informations de couleur et de luminosité indépendamment, ce qui peut être bénéfique pour des tâches où la lumière et l'ombre varient. Cet espace de couleurs est préférable à RGB, qui est souvent moins pratique en raison de sa sensibilité aux variations d'éclairage et à la corrélation entre ses composantes. Pour des images de vêtements, prises majoritairement par des téléphones portables, et dans des conditions de luminosité pas toujours optimales, les poids HSV sont plus représentatifs pour comparer les images. [17]

Pour comparer les images, j'utilise deux techniques :

- Similarité des caractéristiques obtenues par VGG16 : similarité cosinus. C'est une mesure qui calcule la similarité entre deux vecteurs dans un espace multidimensionnel. Les vecteurs représentent ici les images [18]

-
- Similarité entre les histogrammes de couleurs : méthode de corrélation d'OpenCV. C'est une mesure qui calcule les valeurs des bins des histogrammes [19]

Ces similarités sont combinées (on prend la moyenne des deux), et j'obtiens une valeur entre 0 et 1, 0 signifiant que les images n'ont rien en commun et 1 qu'elles sont parfaitement identiques.

Cette technique, bien qu'efficace, reste tout de même assez longue à mettre en place, puisqu'une comparaison est faite entre l'image d'entrée et chaque image de la base de données. Comme cette dernière est composée de plus de 200 000 images en tout, et allant jusqu'à 72 000 exemplaires de robes par exemple, il serait extrêmement long de réaliser autant de comparaisons. J'ai donc mis une limite d'articles à comparer.

Pour améliorer la recherche de ressemblances, les images ont toutes été réduites, en utilisant les valeurs des boîtes englobantes des labels. Je ne garde donc que la partie intéressante, avec le vêtement, pour faire la comparaison. On évite ainsi d'augmenter la ressemblance si les mannequins se ressemblent ou ont les mêmes poses.

8 Interface et utilisation

les trois parties présentées précédemment sont le corps du projet : à partir d'une image d'entrée, je peux récupérer sa catégorie et récupérer des images de vêtements similaires de ma base de données. Pour lier ce travail et le rendre accessible à un utilisateur, j'ai décidé de faire une interface en utilisant le module TKinter [20].

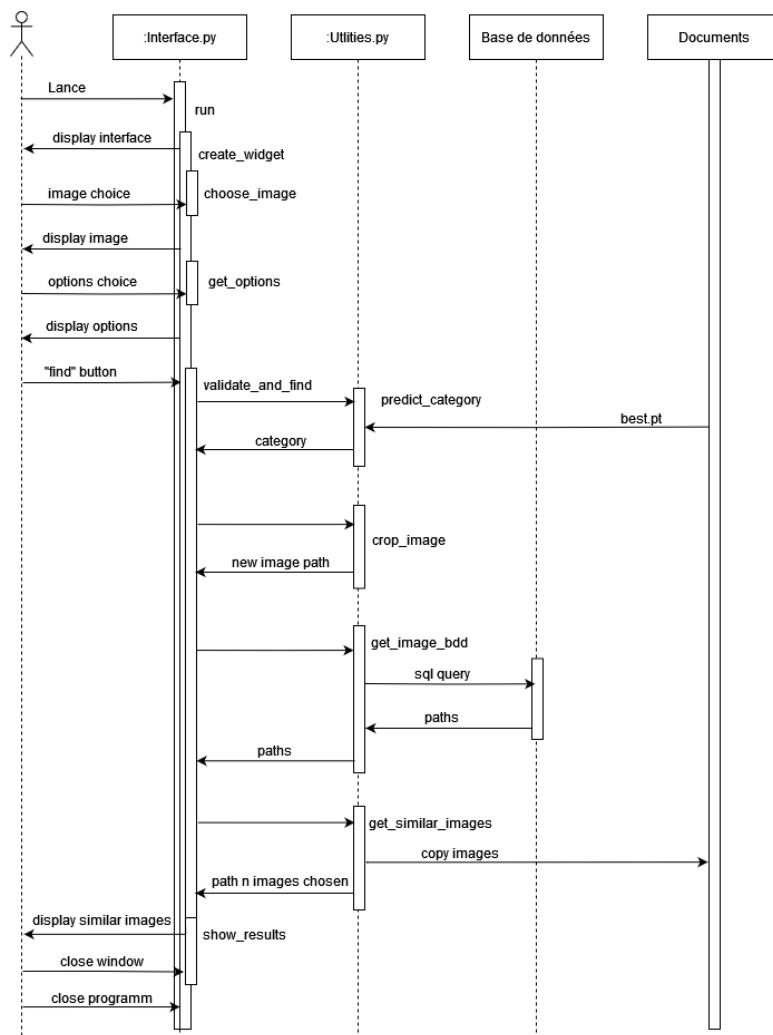


Figure 11: Diagramme de séquence résumant le fonctionnement de l'interface

L'utilisateur commence par lancer le projet. L'interface s'ouvre, et il peut réaliser quatre actions :

- (obligatoire) téléverser une image : elle sera la cible pour trouver les images les plus ressemblantes.
- (facultatif) : choisir la taille et choisir la marque désirée pour le vêtement
- (obligatoire mais pré-rempli) : choisir le nombre d'images similaires à montrer (entre 1 et 12, par défaut 6)

En appuyant ensuite sur le bouton "Find", une autre fenêtre s'ouvre, avec à gauche l'image donnée par l'utilisateur (qui a été réduite en ne gardant que la partie de la boîte englobante prédite par YOLO) et à droite les images similaires trouvées par notre programme. Ces dernières sont accompagnées de trois labels : le pourcentage de ressemblance, la taille et la marque du vêtement.

Au niveau du programme, lorsque l'utilisateur appuie sur le bouton "find", on appelle une méthode qui va réaliser le travail dans l'ordre:

1. Prédire la catégorie de l'image et garder la partie intéressante
2. faire une requête à la base de données pour obtenir n chemins d'images de la catégorie prédite. N est inaccessible à l'utilisateur, mais très facilement modifiable dans le code. Je l'ai mis à 250, car c'est assez pour avoir des résultats corrects sans attendre trop longtemps (moins de 30s). Comme les images semblables sont regroupées dans la base de données (elles ont un chemin d'accès similaire), une condition "ORDER BY RAND()" est ajoutée à la requête, pour récupérer des vêtements plus variés, et donc avoir plus de chance d'en avoir un proche de la cible
3. Tester une à une les images de la base avec celle entrée par l'utilisateur pour obtenir celles les plus similaires. Toutes les images testées sont également copiées dans un dossier dans les documents de l'ordinateur avec comme nom leur pourcentage de ressemblance, pour pouvoir observer ce que le programme considère semblable et pas.
4. Afficher les résultats dans la deuxième fenêtre

Le programme ne s'arrête pas tout seul, l'utilisateur peut réaliser plusieurs essais en gardant les résultats à côté, et avec différentes images.

Si le programme ne détecte aucun vêtement, il indiquera un message d'erreur et permettra à l'utilisateur de téléverser une autre image.

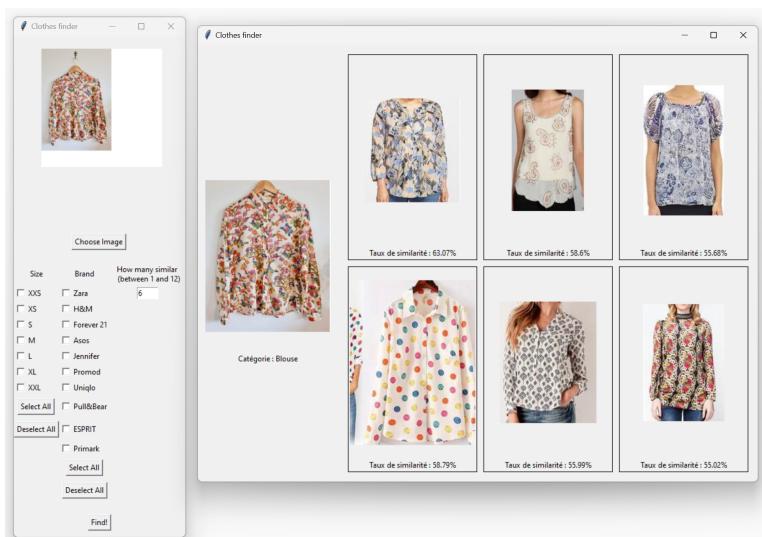


Figure 12: Exemple des deux fenêtres de l'interface : le choix et les résultats

9 Limites du projet et améliorations

9.1 Limites

Le projet, bien que réalisant l'objectif attendu de renvoyer à un utilisateur des images similaires à celle donnée, présente de nombreuses limites.

- Les vêtements du dataset peuvent être portés ou non. Pour ceux portés par un mannequin, on observe très vite que ces derniers sont souvent des femmes blanches et minces. Des vêtements de plus grandes tailles ou portés par des personnes de couleur pourraient être moins bien reconnus par le modèle.
- Les résultats de la recherche de similitudes sont aussi impactés par la différence entre le vêtement porté et non. En effet, lorsqu'on a pour image cible l'un des cas, c'est ce cas également qui sera considéré comme présentant le plus de similitude.
- Comme expliqué précédemment, le dataset utilisé présente des limites pour les articles de niche, et le programme reste donc dans des grandes catégories très utilisées. Si l'utilisateur cherche une catégorie assez peu utilisée de vêtement, le programme lui renverra des images de catégories plus standards.
- Il y a majoritairement des vêtements de femmes, et notre programme ne fait pas la distinction entre les vêtements masculins ou féminins. C'est uniquement lors de la recherche de ressemblance que la différence est observable, avec néanmoins beaucoup de résultats pour femmes lors de recherches de vêtements masculins. Mon projet a toujours été principalement axé sur les habits féminins.
- Le programme ne réalise de comparaisons que sur un petit échantillon d'images de la base de données, car sinon il prendrait beaucoup trop de temps, et pose une importante limite de précision dans les résultats renvoyés à l'utilisateur

9.2 Améliorations

9.2.1 Améliorer le modèle de prédiction

Avoir un modèle de prédiction plus performant aurait permis de chercher dans des catégories plus niches, et donc de renvoyer à l'utilisateur des images plus ressemblantes. Plusieurs pistes existent pour améliorer le modèle :

- Augmenter les données : enrichir le jeu de données avec davantage d'images pour chaque catégorie. Une plus grande quantité d'images pour certaines catégories auraient permis un meilleur apprentissage.
- Améliorer les algorithmes de classification : utiliser des techniques de transfert de style (Transfer Learning) en s'appuyant sur des modèles pré-entraînés sur des bases de données plus larges et générales, puis en les affinant sur notre jeu de données spécifique. [21] On pourrait également faire une prédiction avec différents modèles pour garantir une meilleure reconnaissance, avec par exemple un modèle EfficientNet [22], VGG16 [15], DenseNet [23], ou d'autres modèles utilisés pour de la classification.

-
- Optimiser les hyperparamètres de YOLO
 - Incorporer de techniques de régularisation : utiliser des méthodes de régularisation telles que le Dropout, L2 Regularization, et Batch Normalization pour éviter le surapprentissage (overfitting) et améliorer la généralisation du modèle [24]

9.2.2 Réduire le nombre de comparaisons

La seconde grande limite du projet concerne la comparaison entre les images, qui prend beaucoup de temps et n'est pas toujours efficace si le jeu de test est trop petit et si les chemins pris au hasard ne fournissent pas une image intéressante. Pour y remédier, on pourrait :

- Enrichir des caractéristiques d'entrée : incorporer des métadonnées supplémentaires (telles que la couleur dominante, la texture, les motifs, etc.) en complément des images pour enrichir l'information disponible pour le modèle. On pourrait également utiliser des descripteurs de caractéristiques avancés comme SIFT, SURF ou ORB en combinaison avec des réseaux neuronaux pour capturer des détails fins dans les images. [25]
- Comparer les images entre-elles avant de les comparer avec l'image d'entrée : créer des clusters d'images similaires en utilisant des techniques de regroupement, telles que le clustering K-means ou DBSCAN, basées sur des caractéristiques visuelles comme la couleur, la texture et la forme. Une fois les clusters formés, l'image cible est comparée uniquement avec les images des clusters les plus pertinents. [26] Cela permet de réduire le nombre total de comparaisons à réaliser, car au lieu de comparer l'image d'entrée avec l'ensemble de la base de données, on limite les comparaisons aux images d'un sous-ensemble plus restreint et pertinent. Cette technique améliore non seulement la rapidité du processus de recherche, mais aussi la précision des résultats en se concentrant sur des groupes d'images visuellement cohérents, augmentant ainsi la probabilité de trouver des correspondances pertinentes.

9.3 Conclusion

En conclusion, le projet de recherche d'objets dans une base de données à partir d'images a permis de développer un système capable de prédire la catégorie d'un vêtement et de trouver des images similaires en fonction de cette catégorie. Les résultats obtenus montrent que le modèle est capable de distinguer efficacement les principales catégories de vêtements. Par exemple, les catégories telles que "Blouse", "Tee" et "Dress" sont bien reconnues par le modèle, comme l'indique la forte concentration de prédictions correctes sur la diagonale de la matrice de confusion.

Cependant, certaines catégories présentent des confusions importantes, notamment les vêtements ayant des similarités visuelles fortes, comme "Bomber" et "Jacket", ou "Jeans" et "Jeggings". Ces confusions montrent les limites actuelles du modèle en matière de précision et soulignent la nécessité d'améliorations pour les catégories de niche ou moins représentées dans le jeu de données.

Malgré ces défis, les performances globales du modèle restent satisfaisantes pour les catégories de vêtements les plus courantes, avec des courbes de precision-recall indiquant une bonne précision et un bon rappel pour ces classes.

Pour renforcer encore les performances du modèle, des améliorations sont suggérées, notamment l'enrichissement du jeu de données, l'utilisation de techniques de transfert de style, et l'incorporation de métadonnées supplémentaires comme la couleur et la texture. De plus, la comparaison des images entre elles avant de les comparer avec l'image d'entrée pourrait affiner davantage les résultats et rendre le processus plus efficace.

En mettant en œuvre ces améliorations, le projet pourrait offrir une expérience utilisateur encore plus satisfaisante, permettant de trouver plus facilement des vêtements similaires à ceux recherchés, et ainsi faciliter la recherche et l'achat de vêtements sur des plateformes comme Vinted. Les résultats obtenus sont encourageants et posent les bases d'un système de reconnaissance d'images de vêtements robuste et performant, tout en ouvrant la voie à des améliorations futures pour augmenter sa précision et son efficacité.

References

- [1] [Online]. Available: <https://www.vinted.fr/>
- [2] H. T. Nguyen, K. K. Nguyen, P. T.-N.-Diem, and T. T.-Dien, “Clothing detection and classification with fine-tuned yolo-based models,” in *Advances and Trends in Artificial Intelligence. Theory and Applications*, H. Fujita, Y. Wang, Y. Xiao, and A. Moonis, Eds. Cham: Springer Nature Switzerland, 2023, pp. 127–132. [Online]. Available: https://doi.org/10.1007/978-3-031-36819-6_11
- [3] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang, “Deepfashion: Powering robust clothes recognition and retrieval with rich annotations,” in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. [Online]. Available: <http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html>
- [4] G. Jocher, A. Chaurasia, and J. Qiu, “Ultralytics yolov8,” 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [5] “You only look once (yolo) : Qu'est-ce que c'est ?” [Online]. Available: <https://datascientest.com/you-only-look-once-tout-savoir>
- [6] “What is yolov8? the ultimate guide. [2024].” [Online]. Available: <https://blog.roboflow.com/whats-new-in-yolov8/>
- [7] “Object detection datasets overview.” [Online]. Available: <https://docs.ultralytics.com/datasets/detect/>
- [8] “Yolov8 models overview.” [Online]. Available: <https://docs.ultralytics.com/models/yolov8/>
- [9] “Matrice de confusion : qu'est-ce que c'est et comment l'utiliser ?” [Online]. Available: <https://datascientest.com/matrice-de-confusion>
- [10] “Precision, recall et precision-recall curve.” [Online]. Available: <https://kobia.fr/classification-metrics-precision-recall/>
- [11] “Evaluating ML Models: The Confusion Matrix, Accuracy, Precision, Recall.” [Online]. Available: <https://medium.com/develearn/evaluating-ml-models-the-confusion-matrix-accuracy-precision-recall-b3f8f4431fa5>
- [12] “Mysql.” [Online]. Available: <https://www.mysql.com/fr/>
- [13] “Mysql workbench.” [Online]. Available: <https://www.mysql.com/products/workbench/>
- [14] “Open cv modules.” [Online]. Available: <https://docs.opencv.org/4.x/index.html>
- [15] “Vgg : en quoi consiste ce modèle ? daniel vous dit tout !” [Online]. Available: <https://datascientest.com/quest-ce-que-le-modele-vgg>
- [16] “Color spaces in opencv (c++ / python).” [Online]. Available: <https://learnopencv.com/color-spaces-in-opencv-cpp-python/>

-
- [17] “Hsl and hsv.” [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV
 - [18] “Comprendre la similarité cosinus et son application.” [Online]. Available: <https://ichi.pro/fr/comprendre-la-similarite-cosinus-et-son-application-278464028813674>
 - [19] “Correspondance de modèles à l'aide d'opencv en python.” [Online]. Available: <https://stacklima.com/correspondance-de-modeles-avec-opencv-en-python/>
 - [20] “tkinter — python interface to tcl/t.” [Online]. Available: <https://docs.python.org/3/library/tkinter.html>
 - [21] “Transfer learning : Qu'est-ce que c'est ?” [Online]. Available: <https://datascientest.com/transfer-learning>
 - [22] “Efficientnet: Improving accuracy and efficiency through automl and model scaling.” [Online]. Available: <https://research.google/blog/efficientnet-improving-accuracy-and-efficiency-through-automl-and-model-scaling/>
 - [23] “Réseaux de neurones densenet : tout ce qu'il y a à savoir.” [Online]. Available: <https://datascientest.com/reseaux-de-neurones-densenet>
 - [24] “Comprendre la regularization (l1, l2) en deep learning rapidement !” [Online]. Available: <https://inside-machinelearning.com/regularization-deep-learning/>
 - [25] “A comparison of sift, surf and orb on opencv.” [Online]. Available: <https://mikhail-kennerley.medium.com/a-comparison-of-sift-surf-and-orb-on-opencv-59119b9ec3d0>
 - [26] “Qu'est-ce que le clustering ? les 3 méthodes à connaître.” [Online]. Available: <https://larevueia.fr/clustering-les-3-methodes-a-connaire/>