Deep Learning pour la classification d'images et la reconnaissance multi-objets

Classification d'images

Dans cette première partie, l'objectif est d'utiliser un réseau de neurones profond pour la classification d'images. Le modèle employé est GoogleNet qui est un réseau de neurones convolutif de 22 couches profondes, développé par Google en 2014. C'est la première version du modèle Inception de Google, construit avec des filtres parallèles et du fait, son architecture est large et profonde, ce qui permet d'éviter le surapprentissage.

Une version du modèle est pré-entraînée sur l'ensemble de données ImageNet. Le réseau entraîné sur ImageNet permet la classification des images en 1000 catégories (objets du quotidien, animaux, moyens de transports, etc). Ce modèle a appris différentes représentations des caractéristiques pour une large gamme d'images de taille 224x224.

L'objectif de GoogleNet est de prédire la classe de l'image. Par conséquent, le modèle doit retrouver la classe de l'image dont la probabilité est prédominante. Si plusieurs instances sont présentes dans une même image, le modèle fournit l'étiquette de l'objet le plus précis.

Partie 1 : Chargement et initialisation du modèle pré-entrainé

GoogleNet a été entrainé sur ImageNet pour classifier 1000 catégories d'images. Le modèle est fourni avec ses fichiers de poids, de configuration et des étiquettes des classes.

1) Charger le fichier **classes_names_googlenet.txt** qui contient les noms des classes.
2) Déclarer le modèle avec la classe **Net** et utiliser la fonction **readNet** pour paramétrer le réseau GoogleNet avec ses poids et sa configuration. Le fichier des poids est **bvlc_googlenet.caffemodel** et le fichier de configuration est **bvlc_googlenet.prototxt**.

Partie 2 : Traitement de la vidéo

1) Ouvrir la vidéo **vidTuto41.wmv** et définir la boucle principale de traitement pour capturer les images de la séquence.
2) Appeler la fonction **blobFromImage** pour créer un blob en 4 dimensions à partir de l'image. Cette fonction redimensionne et rogne éventuellement l'image, soustrait les valeurs moyennes, met à l'échelle les valeurs par un facteur d'échelle et permute les canaux bleu et rouge. **blobFromImage** convertit l'image en un blob 4D sous forme 1x3x224x224, des pré-traitements sont nécessaires tels que le redimensionnement de l'image à 224x224, la soustraction de la moyenne (104, 117, 123) et la permutation des canaux bleu et rouge. Par ailleurs, aucune mise à l'échelle n'est appliquée.
3) Appeler la fonction **setInput** pour passer le blob au réseau.
4) Propager les entrées dans le modèle en utilisant la fonction **forward** du modèle.
5) Déterminer la meilleure prédiction de classe. La sortie du réseau contient les probabilités pour les 1000 classes d'images dans une matrice [1 , 1000]. L'objectif est de trouver l'indice de l'image correspondant à la classe ayant la probabilité maximale en utilisant la fonction **minMaxLoc**.

Partie 3 : Affichage et interface utilisateur

1) Afficher la classe et la probabilité de l'image ainsi que le temps de traitement pour chaque propagation du réseau sur le flux vidéo.
2) Ajouter une barre de progression à la fenêtre graphique. Lorsque la vidéo atteint la fin, la barre de progression doit se repositionner au début de la vidéo et n'importe quelle touche clavier permet de redémarrer la vidéo.
3) Programmer les touches claviers suivantes :
    a. Échappe : pour quitter la fenêtre graphique.
    b. Barre espace : pour arrêter/démarrer le flux vidéo.

Reconnaissance multi-objets

L'objectif de ce travail est d'apprendre à utiliser le framework de Deep Learning YOLOv4 qui est un détecteur d'objets très populaire qui permet la détection et l'identification multi-objets.
Le modèle YOLO reconnaît 80 objets différents dans les images et les vidéos, il est performant en temps de calcul (pour son implémentation GPU) et en précision.
YOLO utilise un seul réseau de neurones sur l'image complète, ce réseau divise l'image en régions et prédit des cadres de délimitation et des probabilités pour chaque région. Ces boîtes englobantes sont pondérées par les probabilités prédites.

Partie 1 : Chargement du modèle pré-entrainé

Pour tester YOLOv4 en temps réel sur une vidéo, la version YOLOv4-tiny est utilisée. YOLOv4-tiny est une architecture réduite de YOLOv4 qui est efficace en termes de temps de calcul, notamment pour une implémentation sur CPU. Le modèle YOLOv4-tiny est pré-entraîné sur une base d'images de 80 catégories dont les noms sont enregistrés dans le fichier **classes_names_yolo.txt**.

1) Déclarer le modèle avec la classe **Net** et charger YOLOv4-tiny en utilisant la fonction **readNet**. Les fichiers de poids et de configuration sont respectivement : **yolov4-tiny.weights** et **yolov4-tiny.cfg**.
2) Ouvrir et lire le fichier **classes_names_yolo.txt** contenant les noms des classes.

Partie 2 : Traitement de la vidéo et prétraitement des blobs

L'image en entrée du réseau de neurones doit être dans un certain format appelé **blob**. Une fois que l'image est lue à partir du flux vidéo, elle est transmise à la fonction **blobFromImage** pour la convertir en un objet blob utilisé en entrée du réseau. Dans ce processus, le modèle normalise l'image et redimensionne à une taille donnée. Cette taille doit avoir comme valeur une parmi les résolutions suivantes : 320 × 320, 352 × 352, … jusqu'à 608 × 608 (avec un pas de 32). La moyenne est définie par défaut et le paramètre **swapRB** est initialisé à vrai car OpenCV utilise des images BGR. L'objet blob de sortie est ensuite transmis au réseau par la fonction **setInput** pour obtenir une liste de blobs. Ces blobs sont ensuite traités afin de filtrer ceux dont les scores de confiance sont faibles.

1) Ouvrir la vidéo et capturer les images.
2) Appeler la fonction **blobFromImage** et préciser la taille de l'image à 416 x 416. Par ailleurs le facteur d'échelle est égal à 1 et les canaux rouge et bleu sont interchangés.

3) Appeler la fonction **setInput** du modèle avec comme arguments : les blobs, un facteur d'échelle défini à **0.00392** et une moyenne égale à 0.
4) Appeler la fonction **forward** du modèle pour calculer les prédictions. Cette fonction a besoin de la couche finale puisque c'est cette couche qui fournit les prédictions. Afin d'obtenir le nom de la couche finale, il faut utiliser la fonction **getUnconnectedOutLayers** du modèle qui donne les noms des couches de sortie du réseau.

Partie 4 : Post-traitement

Le modèle produit des boîtes englobantes de sorties qui sont représentées par un vecteur de 5 éléments, plus le nombre de classes (80).

Les 4 premiers éléments représentent le center x, le centre y, la largeur et la hauteur de la boîte englobante. Le cinquième élément représente la certitude de la boîte pour englober un objet.

Le reste des éléments est la confiance associée pour chaque classe d'objet. La boîte englobante est attribuée à la classe correspondant au score le plus élevé.

Les boîtes englobantes sont ensuite filtrées à l'aide du seuil de confiance, elles sont soumises à la suppression non maximale pour éliminer celles qui se chevauchent.

YOLO définit 3 échelles pour manipuler différentes tailles des images. Les blobs de sortie sont :

- 507 (13 x 13 x 3) pour les grands objets
- 2028 (26 x 26 x 3) pour les objets moyens
- 8112 (52 x 52 x 3) pour les petits objets

Définir une fonction appelée **postProcessing** pour réaliser les traitements suivants :

1) Parcourir les blobs de sortie du modèle déterminés par la fonction **forward**. Ces blobs sont représentés par un tenseur 3D : [[8112 , 85], [2028 , 85], [507 , 85]]. YOLO-tiny définit uniquement les blobs [[2028 , 85], [507 , 85]] pour accélérer les traitements.
2) Pour chaque ligne du blob de sortie :
    a. Récupérer les scores pour les différentes classes.
    b. Déterminer le score le plus élevé.
    c. Si ce score est supérieur à 0.35, alors récupérer la boîte englobante (centre, largeur, hauteur) ainsi que l'indice de la classe et la valeur du score.
3) Appeler la fonction **NMSBoxes** pour filtrer les boîtes englobantes qui se chevauchent en fixant le seuil de suppression à 0.5 (0 : filtrage maximal)
4) Définir une fonction qui permet d'afficher les boîtes englobantes retrouvées par **NMSBoxes** ainsi que les scores de confiance et les noms des classes.
5) Appeler la fonction **postProcessing** dans la boucle du traitement vidéo après la fonction **forward**.

Partie 5 : Affichage des résultats

Définir une fonction d'affichage **drawPredictions** qui permet de :

1) Afficher la boîte englobante entourant l'objet détecté.
2) Afficher la classe d'objet et la précision de prédiction.