|  |
| --- |
| OT2 |
| Rapport projet Machine Learning |
| Détection automatique de visage |

|  |
| --- |
| Alexis Strappazzon  [Date] |

# Présentation du projet

L’objectif du projet est d’utiliser l’intelligence artificielle et le machine learning pour distinguer les visages dans une image de taille variable. Pour ce faire, il faudra entraîner puis exploiter un modèle.

Pour entraîner les modèles, une base de données d’images de visage et de non-visage a été à disposition. Comportant 6831 « non-visage » et 797 « visage ».

# Existant

L’approche historique de R. Vaillant, C. Monrocq et Y. Le Cun (Vaillant, Monrocq, & LeCun, 1994) est presque ce que je vais réaliser. Dans cet article, les auteurs entraînent plusieurs modèles qui ont la même architecture mais avec des données différentes pour des objectifs différents. Une localisation dites « rough », une localisation « précise » et une localisation « complète ». Finalement, il utilise le modèle « rough » pour estimer la présence d’un visage puis le modèle « précis » pour trouver le centre des visages dans les hypothèses générées par le modèle « rough ». Les résultats, sont analysés pour détecter les visages (analyses de blobs). Cette technique est celle qui est enseigné en cours et celle que je vais utiliser.

Il existe d’autres techniques comme OverFeat (Sermanet, et al., 2014), R-CNN (Ross, Jeff, Trevor, & Jitendra, 2014), YOLO (Joseph, Santosh, Ross, & Ali, 2016) et d’autres.

# Librairies et développement des outils

Pour développer ce projet, j’ai utilisé les libraries PyTorch (https://pytorch.org/) et OpenCV (<https://opencv.org/>). PyTorch donne les outils nécessaires à la construction de réseau de neurones sous une forme très basique. OpenCV permet de visualiser et de traiter les images. Pour l’hyperparameter tunning j’ai utilisé Ray Tune (<https://docs.ray.io/en/latest/tune/index.html>) qui automatise cela.

J’ai commencé par développer un « Trainer » pour entraîner mes modèles. J’ai développé les fonctionnalités suivantes :

* Sauvegarde automatique des checkpoints (état de l’entraînement, comporte les poids et l’état de l’optimizer)
* Sauvegarde automatique des modèles
* Early-stop en fonction de la précision/loss value d’un des jeux de données (training, validation, test).
* Implémentation du nécessaire pour utiliser Ray Tune.

# Entraînement des modèles

## Puissance de calcul

Pour l’entrainement des modèles, j’utilise mon ordinateur personnel. Les caractéristiques de mon système sont :

* OS : Windows 10
* CPU : Intel I5-8600k
* GPU : Nvidia 1070 GTX
* Capacité RAM : 16Go

Ce système est assez performant pour des entraînements sur des petits jeux de données. Cependant, la carte graphique est assez datée et non-optimal pour de l’entrainement.

## Augmentation des données

Le dataset original a été divisé en 3 jeux de données : training, validation, test. Le dataset de training, sert à l’entraînement du modèle. Celui de validation, sert à mesurer la performance du modèle lors de l’entraînement et permet l’early-stopping. Le faire avec le dataset de test pourrait induire un biais. Le dataset de test lui sert à mesurer la performance du modèle sur des données qu’il n’a jamais vues.

Comme le nombre d’échantillons de chaque classe n’est pas équilibré, les modèles entraînés peuvent être biaisés. Dans l’optique de réduire ça, même si les dataset ne sont pas équilibrés, je montre de manière égale des visages et des non-visages.

Pour augmenter le nombre d’exemple et la capacité de généralisation du modèle, le jeu de données de training a été augmenté. C’est-à-dire que chaque image peut se retrouver légèrement altéré. Les altérations possibles sont de l’ordre de la rotation ou d’une symétrie.

## Architecture retenue

## Entrainement de ce modèle

# Exploitation des modèles

L’exploitation du modèle n’est pas triviale. C’est finalement une partie tout aussi compliquée que les l’entrainement du modèle. Pourtant, ce n’est pas quelque chose dont on parle. Une limitation du dataset est que nous avons des visages en 36x36. Les images de visage sont centrées. Et l’image entière est considérée comme un visage. Ce qui ne permet pas d’entraîner un modèle qui puisse localiser le visage dans une image plus grande sans traitement ultérieur. Plusieurs méthodes existent. La méthode naïve qui consiste à balader une « window » sur l’image de la taille de l’entrée du réseau de neurones (c’est-à-dire 36x36 dans notre cas). Et une méthode moins naïve qui utilise les CNN à leur pleine capacité. A fin, de pouvoir trouver des visages de toutes taille, l’image en entrée est graduellement miniaturisée (downsample) par un certain facteur.

# Méthode 1 : Sliding Window

La première méthode, consiste à déplacer sur toutes les images (l’original et les versions miniaturisée à différent facteur) la fenêtre sans faire d’hypothèse au préalable. Ce qui revient à faire beaucoup de calcul selon le pas (stride) et la taille de l’image. C’est une méthode qui est lente et qui ne peut pas servir à la détection de visage en temps réel. Cependant, elle a l’avantage d’être simple de compréhension.

Le résultat de cette opération est une liste de « heatmaps ». Entre autres, chaque image résulte dans une nouvelle image où les pixels ont la valeur de prédiction d’un visage à cet endroit. Les heatmaps sont plus petites en nombre de pixels que les images d’entrées. On a aussi des images de différentes tailles à cause de la miniaturisation.

La suite du traitement consiste à faire de l’analyse de « blob » pour trouver les visages. Premièrement, je recherche lesdits blobs. Pour cela, j’itère sur chaque pixel de la heatmap (préalablement agrandie à la taille de l’image original). Lorsque je trouve un pixel avec une confiance au-dessus d’un certain seuil, je démarre une méthode récursive. Cette méthode, met ajoute le pixel courant à une liste de pixels qui formera le blob puis met le pixel courant à 0. Ensuite, elle s’appelle mais à une position différente, c’est-à-dire décalé d’un vers le bas, décalé d’un vers la gauche ou la droite. Une fois un blob trouvé, je regarde s’il est possible de fusionner des blobs qui sont proches dans l’espace. Pour cela, je regarde la distance entre les centroïdes de tous les blobs. S’il y en a dessous un certain seuil, ils sont fusionnés. Tous les blobs, même des différentes versions de l’image sont agréger dans la même liste.

A partir de ces blobs, je recherche les « boundings boxes ». Comme, les blobs sont agrégés dans la même liste, il est possible (certain) que des boudings boxes se chevauchent. Afin, de n’avoir plus que des boudings boxes sans chevauchement j’utilise la méthode NMS (Non-Maxima Suppression).

# Méthode 2 : Rough Detection

Cette méthode consiste à utiliser un réseau de neurones dont les couches sont toutes Convolutional. Puisque, les couches types Convolutional n’ont pas d’attente quant à la taille des données qu’elles reçoivent, on peut fournir à ce genre de réseaux de neurones des images de la taille que l’on veut. Il existe des manières de transformer des couches Fully Connected en couche Convolutional.

Cette méthode a bien des avantages, comme une taille d’image variable et une meilleure rapidité car on ne recalcule pas les mêmes opérations à chaque fois à l’inverse de la sliding window. Elle donne comme résultat des heatmaps similaires aux heatmaps que l’ont générais avec la sliding window. Cependant, ces heatmaps sont beaucoup plus petite en taille que celle générées avec la méthode de la sliding window. Car les convolutions faites par le réseau de neurones ainsi que les Pooling layers réduisent la taille de l’image. Cette réduction est en lien directe avec la taille des kernels et des strides de chaque couche.

Cette méthode est utile pour avoir une estimation de la localisation d’un visage dans une image. Mais il faudra une autre méthode pour la localisation précise d’un visage.

# Méthode 3 : Rough Detection + Sliding Window

Une troisième méthode peut naître de l’association des deux méthodes précédemment évoquées. C’est la méthode utilisée par (Vaillant, Monrocq, & LeCun, 1994) dans leur papier. Un premier passage avec la méthode 2 pour estimer toutes les images rapidement. Puis un second passage avec la méthode 1 sur les bouts d’images qui semble contenir un visage.

Cette méthode est un bon compromis entre précision de la localisation et vitesse de calcul.

# Résultats

L’exploitation du modèle fut beaucoup plus dure à implémenter que la partie entrainement du modèle. Puisque PyTorch et d’autres librairies prennent en charge toute cette partie. Tandis que l’exploitation a demandé le développement d’outils adaptés.

De plus, si pendant l’entrainement, le jeu de test montre une précision aux alentours de 98%. Pendant l’exploitation les 2% restant font que le modèle a un taux de faux positifs immense.

J’ai essayé beaucoup de paramètre et de modèle différent, mais je ne suis pas arrivé à concevoir un modèle dont les résultats sont satisfaisants pour la détection de visage. A vrai dire, je ne sais pas ce que je peux améliorer. Est-ce que mes modèles sont trop simple ? A l’inverse sont-ils trop compliqués ? Est-ce qu’il faudrait que j’augmente les données d’une manière différente ?

Le problème est que chaque hypothèse demande de réentraîner le modèle. Mais mon ordinateur n’est pas assez puissant pour entraîner rapidement des modèles. Chaque entrainement demande entre 5 et 10 heures de calculs. Une option aurait été d’utiliser des moyens en ligne comme Google Colab qui permet de disposer d’une plus grande puissance de calcul. Cependant, cela ajoute son lot de problématique. Comme la gestion de l’espace disque et des données.

# Bibliographie

Joseph, R., Santosh, D., Ross, G., & Ali, F. (2016). You Only Look Once: Unified, Real-Time Object Detection. *ArXiv*. doi:You Only Look Once: Unified, Real-Time Object Detection

Ross, G., Jeff, D., Trevor, D., & Jitendra, M. (2014, November 18). Rich feature hierarchies for accurate object detection and semantic segmentation. doi:https://doi.org/10.48550/arXiv.1311.2524

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2014, February). OverFeat : Integrated Recognition, Localization and Detection using Convolutional Networks. *ArXiv*. doi: https://doi.org/10.48550/arXiv.1312.6229

Vaillant, R., Monrocq, C., & LeCun, Y. (1994, August). Original approach for the localisation of objects in images. *IEE Proceedings : Vision, Image and Signal Processing*, 245-250. doi:https://doi.org/10.1049/ip-vis:19941301