

Reconnaissance d'armes à feu sur flux vidéo

Simon Cohen
CentraleSupélec

simon.cohen@student.ecp.fr

Cécile Gontier
CentraleSupélec

cecile.gontier@student.ecp.fr

Romain Pelloie
CentraleSupélec

romain.pelloie@student.ecp.fr

Abstract

Dans le cadre du plan de vidéoprotection de la ville de Paris, la préfecture de police a mis en place des milliers de caméras de surveillance dans les rues et les transports de Paris. Aujourd'hui, aucune détection automatique n'est mise en place sur ces flux vidéos, c'est uniquement manuel. Nous avons donc pensé pouvoir mettre en place une détection automatique et instantanée d'armes à feu. Pour cela, nous avons d'abord exploré des approches de vision par ordinateur, avec des algorithmes de détection de points caractéristiques et de 'feature matching'. Comme les résultats n'étaient pas concluants, nous avons continué avec une approche d'apprentissage profond. Nous avons donc constitué notre propre ensemble de données, et entraîné différents modèles de Tensorflow. Nous avons choisi ssd-mobilenet-v2 car il nous semblait être le meilleur et avons tenté de l'optimiser. Cependant, les paramètres initiaux étaient déjà optimisés, et nous n'avons pas réussi à faire mieux. Nous concluons sur les performances finales du modèle, sur des images et sur des flux vidéos.

1. Introduction

Dans les rues de Paris, la Préfecture de Police a mis en place quelques milliers de caméras dans le cadre du Plan de la VidéoProtection de la ville de Paris [1]. À celles-ci s'ajoutent toutes les caméras détenues par des entreprises privées, ainsi que toutes celles dans le réseau de transport en commun [2]. Cependant, l'énorme quantité de données vidéos provenant de ces caméras de surveillance ne fait l'objet aujourd'hui que d'un contrôle manuel. Ces données aident les agents à prévenir d'un risque en visionnant les enregistrements en direct ou à comprendre un incident a posteriori [3]. Nous voulons aller plus loin et permettre de déceler un potentiel danger avant qu'il ne se produise et ce, de façon automatique.

Nous nous proposons donc d'utiliser les techniques de vision automatique pour détecter ces situations de danger le plus tôt possible. En cas de danger, le système lancerait l'alerte à un agent de sécurité, qui pourrait ensuite vérifier par lui-même.

Ici nous nous intéresserons à la détection d'armes à feu ou armes de poing, sans distinguer les différents types.

Dans une première partie, nous regarderons si les techniques de computer vision permettent de détecter une arme sur une image (*Note : seuls Simon et Cécile ont travaillé sur la partie Computer Vision ; Romain n'ayant pas suivi le cours de Céline Hudelot*), puis nous approfondirons l'étude avec des techniques de Deep Learning.

2. État de l'art en analyse d'images

On distingue différentes techniques d'analyses d'images par ordinateur, que l'on peut classer selon 3 catégories [4] :

- Les techniques basiques dites 'handcrafted' ;
- Les techniques d'apprentissage automatique (Machine Learning) ;
- Les techniques d'apprentissage profond (Deep Learning).

Parmi les techniques 'handcrafted', on trouve la binarisation ou seuillage (application d'un seuil à chacun des pixels de l'image), la détection de coin de Harris et la détection de formes. On peut alors mettre en place les détecteurs de Sobel et de Canny, qui utilisent le seuillage pour faire traiter l'image. On introduit ensuite des techniques plus élaborées, invariantes à l'échelle : SIFT et ses versions plus rapides SURF (fondé sur des points d'intérêts) et FAST. BRIEF est une technique qui utilise la binarisation sur des points particuliers et utilise moins de données. Il existe de nombreuses autres méthodes, notamment ORB, alternative gratuite et open source à SIFT et SURF. Enfin, on parle de feature matching pour la correspondance de caractéristiques entre 2 images.

Viennent ensuite des techniques d'apprentissage auto-

matique, comme HOG-SVM, qui détecte les contours par calcul de gradients, le détecteur de Haar ou LBP.

Enfin, les techniques les plus élaborées sont celles d'apprentissage profond (Deep Learning), qui nécessitent un GPU pour entraîner les modèles. On retrouve notamment SSD et YOLO par exemple, que l'on verra par la suite.

3. État de l'art en détection d'armes

Certains techniques de détection automatique d'armes dans des flux vidéos existent déjà et elles utilisent du Deep Learning, plus précisément des analyses optimisées de réseaux convolutifs par régions (faster_rcnn). C'est une solution très efficace et que l'on développera par la suite. [10, 11]

4. Approche Computer Vision

En première approche, nous avons testé différentes techniques de vision par ordinateur pour reconnaître et détecter une arme à feu.

4.1. Détection par seuillage : filtre de Canny

En calculant le module du gradient d'une image en niveaux de gris, on peut faire apparaître les contours d'objets dans cette image. Un contour est décrit par un gradient de forte intensité. Là où le gradient est fort, le pixel sera blanc, sinon il sera noir. Il s'agit alors de trancher sur le seuil du gradient fort, à partir duquel le pixel est blanc. C'est le seuillage. [5]

Le filtre de Canny utilise une double détection des contours, horizontale et verticale, qui permet de déterminer leur orientation. A partir de ce traitement, on supprime les non-maxima d'intensité, pour ne garder que les contours intéressants. On vient ensuite appliquer le seuillage, qui est ici particulier : c'est un seuillage par hystéresis. Le critère de décision est fondé sur 2 seuils, haut et bas : pour chaque pixel, si l'intensité du gradient est inférieure au seuil bas, le point est noir ; si elle est supérieure au seuil haut, le point est blanc (c'est un contour) ; si elle est entre le seuil bas et le seuil haut, le pixel est blanc si et seulement s'il est adjacent à un point déjà blanc.

4.2. Détection de points caractéristiques : SURF

SURF (Speeded Up Robust Features) est un algorithme qui détecte les caractéristiques d'un objet, en décrivant ses points particuliers. Par exemple, il permet de détecter les angles, les contours externes et internes de l'objet par contraste de couleur, les trous dans l'objet, etc. On peut donc utiliser l'algorithme ainsi : on lui donne une image d'entraînement de pistolet et il va détecter ses points particuliers. Ensuite, on peut lui donner une autre image et il

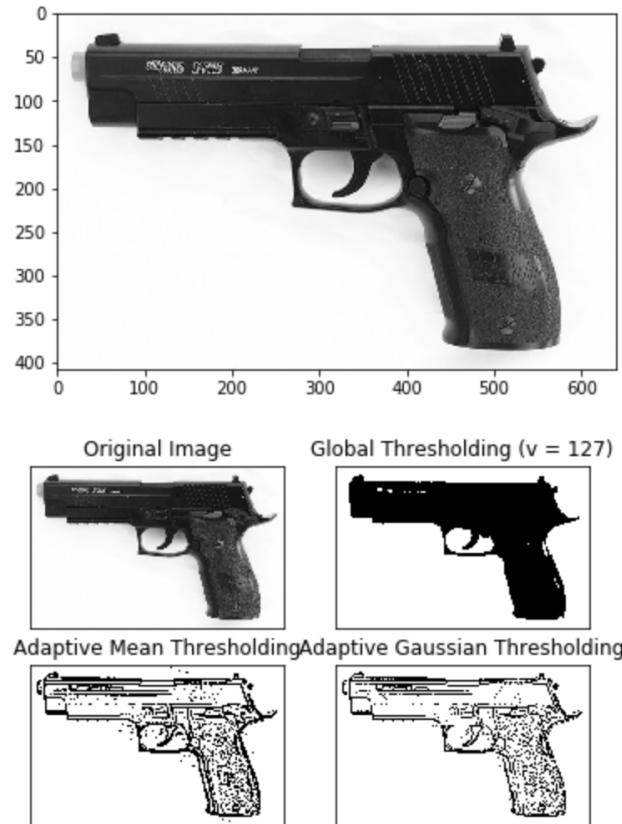


FIGURE 1. Traitements de seuillage pour le filtre de Canny

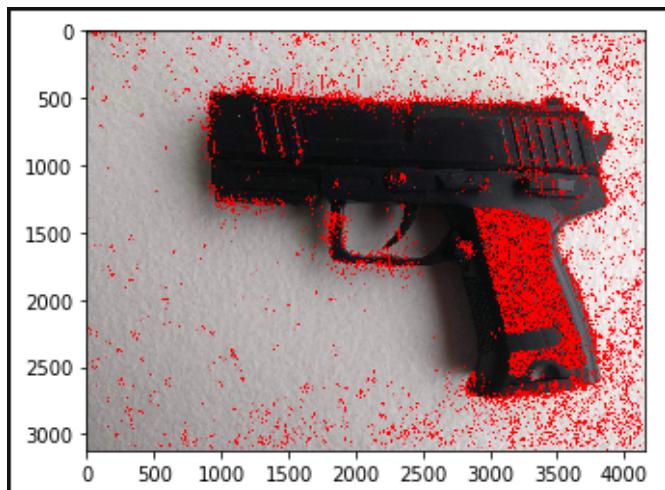


FIGURE 2. Points caractéristiques par l'algorithme SURF

en détermine également les points particuliers. Un exemple est donné en Figure 2. On remarque que l'algorithme détecte très bien les angles, les contours externes mais aussi les particularités à l'intérieur de l'objet (par exemple les reliefs sur la crosse).

4.3. Correspondance de points particuliers

A chacun de ces points particuliers, SURF associe une caractéristique. On peut alors analyser deux images pour voir s'il peut associer deux à deux certaines caractéristiques. Pour cela nous avons comparé deux algorithmes, BFMatcher et FlannMatcher. Ces algorithmes prennent en entrée les descriptions des points caractéristiques de deux images et renvoie une liste des match les plus probables. BFMatcher, pour chaque point de la première image, il va chercher le plus proche voisin parmi les points de la deuxième image, en utilisant la norme donnée sur les descripteurs de points ; FlannMatcher est un algorithme de KNN optimisé pour les hautes dimensions, ce qui est pertinent pour comparer les descriptions des points particuliers. Nous avons constaté que le BFMatcher donnait de meilleurs résultats. Il nous renvoie les deux images côté à côté et les points particuliers correspondants sont reliés. Un exemple est donné en Figure 3. En haut, on a donné à l'algorithme deux fois la même image, pour vérifier que le *feature matching* marchait bien, et on remarque que c'est bien le cas car tous les traits sont horizontaux. Ensuite, nous avons pris des photos avec le même pistolet, en miroir, en rotation et avec occlusion (une main qui cache une partie de la crosse). Lorsque l'on compare l'image de référence avec son miroir, on voit que le haut du pistolet et le bas de la gâchette sont bien identifiés, mais avec de légères erreurs. Pour la rotation, les résultats sont moins bons, l'algorithme associe la gâchette avec l'arrière du pistolet. Pour l'occlusion, il reconnaît bien le haut du pistolet, mais moins bien le reste.

4.4. Conclusion sur le feature matching

Nous voulions savoir si nous pouvions utiliser ces algorithmes pour détecter une arme sur une image de manière efficace.

L'avantage de ces techniques est qu'elles ne requièrent aucun entraînement ni de grands volumes de données. Une seule image de référence suffit. Cependant nous ne pouvons pas faire de détection automatique avec ces techniques, car elles ne sont pas assez précises au niveau de leurs correspondances.

5. Approche Deep Learning

5.1. Choix de l'algorithme

Nous avons décidé d'utiliser Tensorflow comme framework de Deep Learning. Ce choix est motivé par plusieurs critères. Premièrement, Tensorflow est performant dans la détection d'image, et a déjà plusieurs modèles pré-entraînés spécifiquement sur la détection d'images. De plus, on peut le faire tourner sur Google Colab, permettant



FIGURE 3. De haut en bas, correspondance des points caractéristiques sur le même pistolet, en miroir, en rotation et avec occlusion

l'accès à un GPU. Nous avons pensé à utiliser YOLO, fortement optimisé pour la détection d'objets sur flux vidéos. Malheureusement, nous n'avions pas accès à un GPU compatible et YOLO ne fonctionne pas sur Colab.

5.2. Choix de la technique

Nous avons décidé de faire du Transfer Learning sur des modèles pré-entraînés du module object-detection de Tensorflow. Nous avons considéré que si le modèle savait déjà reconnaître des petits objets du quotidien, nous pourrions l'entraîner à reconnaître des armes à feu.

Nous avons travaillé sur des architectures de type CNN, car celles-ci sont les plus adaptées à de la détection d'objet.

Le problème de la détection d'objet se pose comme ceci :

il y a d'abord une première étape de "proposition de régions" (il s'agit de repérer les différentes régions d'intérêt de l'image, correspondant typiquement à de potentiels objets). Ensuite, une étape de classification est réalisée (est-ce que tel objet de mon image correspond à la classe "handgun" ou non ?).

Nous verrons que la fonction de coût est composée d'une composante liée à l'erreur de localisation et d'une autre composante liée à l'erreur de classification.

5.3. Création du dataset

Les modèles de Deep Learning appliqués à la détection d'objets nécessitent un grand nombre de données d'entraînement, correspondant à des images contenant l'objet que l'on veut détecter, accompagnées d'un 'label' qui contient notamment l'information de la position de l'objet sur l'image.

La création du dataset consiste donc en 2 grandes parties : trouver un nombre suffisant d'images contenant l'objet à détecter (en l'occurrence une arme à feu) puis labelliser toutes les images.

Enfin, il y a une dernière étape à effectuer pour mettre les données dans un format compréhensible par Tensorflow.

5.3.1 Trouver les images

Nous avons d'abord fondé nos études sur le site www.imfdb.org (base de données d'images provenant de films, contenant des armes à feu). Nous avons récupéré 500 images de cette manière.

Nous avons ensuite eu la chance de trouver un dataset créé par des étudiants de Grenade qui ont travaillé sur un projet similaire : ce dataset est constitué de 3000 images d'armes à feu.

5.3.2 Labelliser les images

Une fois toutes les images récupérées, nous les avons labellisées. Nous avons utilisé LabelImg (<https://github.com/tzutalin/labelImg>), qui permet de labelliser les images aisément (nous créons à la main les 'boîtes enveloppantes' à l'endroit des armes à feu et LabelImg crée automatiquement un fichier xml compréhensible par les programmes que nous utilisons ensuite pour entraîner les modèles, voir Figure 4).

5.3.3 Conversion du format xml en format TFRecord

Pour entraîner les modèles pré-entraînés de Tensorflow, les données doivent être dans un format TFRecord. Il s'agit ici de simples conversions de

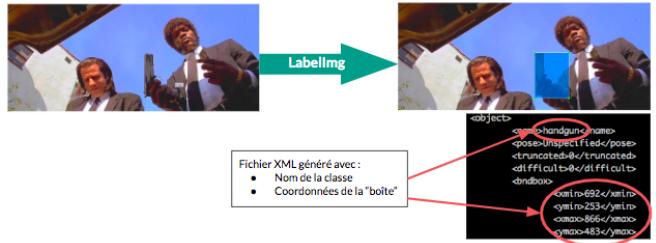


FIGURE 4. Labellisation d'image avec LabelImg

format que nous avons pu faire notamment grâce à un programme mis à disposition par Tensorflow.

Un détail important est à noter cependant : c'est à cette étape que nous avons créé nos ensembles d'entraînement et de validation. Nous avons séparé de manière aléatoire notre dataset selon une répartition 25% - 75% ($\frac{1}{4}$ des données pour le test, $\frac{3}{4}$ pour l'entraînement).

6. Expériences

Dans un premier temps, nous avons testé plusieurs modèles pré-entraînés pour les comparer. Ensuite nous en choisirons un et tenterons de l'améliorer en changeant les hyperparamètres.

Les modèles testés sont :

- ssd_mobilenet_v1
- ssd_mobilenet_v2
- faster_rcnn_inception_v2

Nous avons choisi de partir de ces modèles pré-entraînés car nous avons lu que l'architecture SSD (Single Shot Detection) était adaptée à une détection en temps réel donc nous nous sommes surtout concentrés dessus, et nous avons voulu tester une autre architecture comme faster RCNN, réputée plus précise. Nous avons appliqué la technique de transfer learning : il suffit d'utiliser le réseau de neurones déjà pré-entraîné sur un dataset connu (comme le dataset COCO) comme base pour entraîner le modèle à reconnaître des armes à feu. Cela est pertinent car si le modèle sait déjà reconnaître des objets du quotidien, nous pouvons lui faire reconnaître une arme à feu après entraînement sur des images d'armes à feu. Nous avons choisi ces modèles à partir de [8].

Le prototype fonctionne sur des images fixes mais aussi sur des flux vidéo (la webcam de l'ordinateur par exemple).

Pour le code, nous avons trouvé plus simple d'utiliser celui déjà fourni par tensorflow, avec le fichier `train.py` pour l'entraînement, le fichier `eval.py` pour l'évaluation et `export_inference_graph.py` pour l'export des graphes d'inférence.

6.1. Comparaison des modèles

L'ensemble de validation contient environ 800 images.

Modèle	mAP@.50IOU	Vitesse (FPS)
ssd-mobilenet-v1	0.79	33
ssd-mobilenet-v2	0.86	32
faster-rcnn-inception-v2	0.86	17

FIGURE 5. Performances des modèles

Avec le fichier eval, nous avons pu évaluer notre modèle. Nous avons choisi la métrique du Mean Average Precision au rang 0,5 IOU. Nous voulions utiliser Tensorboard, mais comme nous étions obligé d'exécuter notre code sur Google Colab, il n'était pas possible de faire tourner en parallèle le Tensorboard, qui ne marche qu'en local.

Nous avons aussi ajouté la vitesse de traitement, en nombre d'images par seconde (FPS), car nous voulons faire du temps réel à terme.

Le résultats sont en Figure 5.

6.2. Choix d'un modèle

Pour choisir notre solution nous avons décidé qu'il était plus important de détecter toutes les armes à feu, même si le modèle détecte des objets qui ne sont pas des armes, nous voulons donc minimiser le taux de faux négatifs. Ce choix est motivé par l'application de notre système : il est plus important bien sûr de détecter toutes les armes, quitte à ce que le système voie des armes là où il n'y en a pas (il faut quand même minimiser le nombre de faux positifs si possible). Un autre critère est le temps de traitement. En effet, nous avons testé les modèles RCNN sur notre webcam et le système ne traitait qu'une image toutes les deux secondes, alors que les modèles SSD fonctionnent parfaitement en temps réel. Le client souhaite utiliser la solution sur des systèmes en temps réels, cela nous semble donc plus pertinent de privilégier cet aspect, quitte à perdre en précision. Nous avons donc choisi le modèle Tensorflow ssd-mobilenet-v2, puis nous avons réfléchi aux différentes manières de l'optimiser.

6.3. Optimisation du modèle

Les modèles de Deep Learning comprennent des hyperparamètres influençant la performance finale du modèle. Nous avons donc tenté d'optimiser au mieux certains des paramètres importants du modèle pour maximiser la performance. Nous développons ci-dessous les aspects associés à 2 de ces paramètres.

6.3.1 Taux d'apprentissage (learning rate)

Le taux d'apprentissage est un paramètre important du modèle, il influence directement la capacité et la qualité de convergence de la descente de gradient pendant l'apprentissage du modèle. Le taux d'apprentissage peut être constant, ou diminuer dans le temps de manière linéaire,

ou il peut être optimisé en s'adaptant lui-même à la descente de gradient. Il y a plusieurs optimisations possibles, comme AdaGrad ou RMSProp, qui consistent en une mise à jour du taux d'apprentissage selon l'historique des valeurs de gradient (une grande dérivée partielle → diminution du taux, une petite dérivée partielle → augmentation du taux).

Nous avons testé plusieurs possibilités d'optimisation du taux d'apprentissage et avons choisi la meilleure selon la qualité de convergence de la fonction de coût. Nous avons effectué une optimisation RMSProp couplée à un momentum (technique d'accélération de la descente de gradient).

6.3.2 Options d'augmentation des données (data augmentation)

Tensorflow nous permet d'ajouter des options d'augmentation des données dans le fichier de configuration de l'entraînement. L'augmentation des données consiste en l'augmentation des données d'apprentissage par des transformations comme, par exemple :

- Des rotations
- Des transformations “miroir”
- Du floutage
- Du rognage

Ce type de transformation permet de créer de nouvelles images d'entraînement à partir de notre dataset de départ et d'améliorer l'erreur de généralisation que le modèle fait. Nous avons utilisé cette option pour entraîner notre modèle final.

6.3.3 Image resizer

Ce paramètre influence comment l'algorithme change les proportions de l'image.

6.4. Performances finales

6.4.1 Comparaison des différents hyperparamètres

Nous sommes partis du fichier de configuration donné initialement par le modèle ssd-mobilenet-v2. On suppose qu'il est déjà optimisé et on s'attend donc à ce que nos modifications ne feront que détériorer le modèle. Nous observons les métriques données par l'évaluation sur la Figure 1. En plus du Mean Average Precision, nous considérons la classification loss, qui est l'erreur de classification (c'est à dire donner le bon label à l'objet), et la localization loss, qui est l'erreur sur le placement de la boîte enveloppante.

On voit que le modèle initial est effectivement le meilleur pour ces trois métriques. Ainsi, même si nous avons essayé d'améliorer le modèle en changeant les hyperparamètres, nous n'avons pas réussi, probablement car nous avons pris un modèle déjà très connu et très optimisé. Ainsi même si nous l'avons entraîné sur un nouvel objet, comme c'était un

Modification	classification loss	localization loss	MAP@0.5IOU
Aucune (initial)	2.954661	0.64704	0.863571
image resizer	3.476352	0.699028	0.844879
data augmentation	3.843847	0.841447	0.764417
constant lr	3.293498	0.715733	0.837552

objet du même type que ceux de COCO, le modèle donné était déjà optimal.

6.4.2 Performance sur l'ensemble de test

Nous choisissons donc comme modèle celui avec les meilleures performances, qui est celui entraîné avec le fichier configuration de base.

Nous avons constitué un ensemble de test de 200 images, avec 100 images positives (avec des pistolets) et 100 images négatives, et nous avons testé le modèle sur ce dataset.

Nous avons obtenu les métriques en TABLE 2.

Les faux positifs sont des images ne contenant pas d'armes à feu mais sur lesquelles le modèle identifie une arme tout de même. À l'inverse, les faux négatifs sont les armes non détectées par le modèle. On a utilisé comme métrique de comparaison la vitesse et le score F1, qui est la moyenne harmonique entre la précision et le rappel. La précision quantifie la pertinence des détections, c'est le nombre de vrais positifs sur le nombre d'armes au total à détecter. Le rappel quantifie la capacité du modèle à trouver une arme quand il y en a une, c'est le nombre de vrais positifs sur le nombre d'images contenant une arme. Un modèle idéal a une précision et un rappel de 1, mais souvent il faut faire un compromis entre les deux.

Sur ce dataset en particulier, on voit que notre modèle n'est pas parfait, avec des pourcentages de faux positifs et de faux négatifs non négligeables. En particulier, un taux de faux négatif de 24% signifie que une situation sur 4 contenant une arme à feu n'est pas détectée. Si on revient à notre contexte, qui est d'utiliser la détection automatique et instantanée d'armes à feu sur les flux des caméras de surveillance de Paris, cela n'est pas suffisant.

6.4.3 Sur flux vidéos en temps réel

Dans le cadre de notre contexte, nous avons voulu pouvoir tester notre modèle sur un flux vidéo en temps réel. Nous avons trouvé un code permettant de faire tourner notre modèle entraîné par Tensorflow sur la webcam de l'ordinateur, c'est accessible sur notre repository : https://github.com/pellito1/osy_project_weapon_detection.



FIGURE 6. Exemples d'outputs du modèle : résultats corrects (les 4 images du haut), faux positifs et faux négatifs en bas

6.5. Amélioration possibles

Le modèle peut être amélioré surtout en augmentant la taille du dataset avec des images dans des contextes plus variés. Cela permettra de minimiser le taux de faux négatifs, pour de meilleures performances.

De plus, l'optimisation peut être affinée : nous n'avons fait varier que quelques uns des nombreux hyperparamètres pouvant être ajustés (ex : la taille des batchs, l'initialisation des poids, etc.). Avec le temps et les ressources nécessaires, nous pourrions par exemple appliquer un grid search pour

Modèle	FP	FN	Précision	Rappel	F1 Score
ssd-mobilenet-v2	34%	24%	69%	76%	72%

obtenir un set d'hyperparamètres nous offrant la meilleure convergence possible pour la fonction de coût.

7. Conclusion

L'enjeu de la détection de danger est complexe. La définition même de situation de danger est multiple et difficile à faire apprendre à un ordinateur. En commençant par se concentrer sur la détection d'armes à feu (armes de poing), nous avons compris la complexité d'un tel projet mais aussi ses nombreuses applications aux situations quotidiennes. Nous avons constitué un dataset d'armes de poing composé de 3500 images labellisées (avec une boîte enveloppante permettant d'identifier l'arme sur l'image). Nos résultats, sur un premier projet de Deep Learning, sont concluants et permettent d'identifier assez précisément des armes sur des images, après entraînement d'un modèle Tensorflow. Ce modèle peut être alors adapté à la reconnaissance sur flux vidéos, comme nous l'avons testé sur une webcam. L'étape suivante consiste à améliorer le modèle obtenu (beaucoup d'éléments jouent, comme la composition des datasets d'entraînement et de validation, ainsi que les paramètres du modèle) ainsi qu'à définir d'autres situations de danger et d'en identifier les caractéristiques. De plus, il faudrait, dans la continuation du projet, prendre en compte le fait que certains policiers et militaires patrouillent dans la rue avec leur arme de service, et qu'il faudrait que notre modèle le prenne en compte pour différencier une situation normale d'un danger.

Références

- [1] Page du site de la Préfecture de Police sur la Vidéoprotection
<https://www.prefecturedepolice.interieur.gouv.fr/Vous-aider/Actions-de-prevention/Dispositifs-ZSP-et-videoprotection/Videoprotection>
- [2] Article sur les caméras dans les transports parisiens
<http://www.lefigaro.fr/actualite-france/2018/08/17/01016-20180817ARTFIG00248-les-transports-parisiens-sous-haute-surveillance.php>
- [3] Article sur les interpellations dues aux caméras
<http://www.lefigaro.fr/actualite-france/2014/01/15/01016-20140115ARTFIG00628-paris-la-videosurveillance-a-permis-6800-interpellations.php>
- [4] Documentation détaillée d'openCV
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning
- [5] Techniques de seuillage
http://www.optique-ingenieur.org/fr/cours/OPI_fr_M04_C05/co/Grain_OPI_fr_M04_C05_3.html
- [6] Dataset d'images labellisées Pascal VOC
<http://host.robots.ox.ac.uk/pascal/VOC>
- [7] Repository Github utilisé pour notre prototype (Tensorflow framework)
<https://github.com/tensorflow/models>
- [8] Tensorflow detection model zoo
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
- [9] Cours sur les CNN (convolutional neural network) sur Coursera
<https://www.coursera.org/learn/convolutional-neural-networks>
- [10] Roberto Olmosa, Siham Tabika, Francisco Herrera,
Automatic handgun detection alarm in videos using Deep Learning
<https://www.sciencedirect.com/science/article/pii/S0925231217308196>
- [11] Mohammad Nakib, Rozin Tanvir Khan, Md. Sakibul Hasan, Jia Uddin,
Crime Scene Prediction by Detecting Threatening Objects Using Convolutional Neural Network
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8465583&tag=1>