Floutage de visage sur des caméras de surveillance dans des trains en temps réel
M2 IAAA 2022-2023

Mathieu Lalaque
Centrale Marseille
lalaquemathieu@gmail.com

Michiel Vandecappelle
Televic
m.vandecappelle@televic.com

Abstract

Afin de flouter les visages sur des vidéos de caméra de surveillance de train, nous avons implémenté le modèle compact et rapide Yunet, qui fait de la détection de visage frame par frame, puis afin d'améliorer ses performances, nous avons cherché à exploiter la dimension temporelle en rajoutant des algorithmes de post processing.

1 Présentation de la mission

1.1 Présentation de l'entreprise et de ses secteurs d'activité

Televic est une entreprise belge implantée à l'international spécialisée dans le développement et la fourniture de solutions de communication et de gestion de l'information principalement destinées aux secteurs des transports, de la santé et de l'éducation. Ces activités principales sont :

Communication ferroviaire et de transport : Televic propose des systèmes de communication destinés aux opérateurs ferroviaires et de transport en commun. Cela comprend des systèmes de communication passagers, des annonces en gare, des systèmes d'information en temps réel pour les passagers, etc.
Solutions de santé : Dans le domaine de la santé, Televic fournit des solutions de communication pour les hôpitaux et les établissements de soins de santé. Cela comprend des systèmes d'appel infirmière, des systèmes de communication patient-soignant, etc.
Éducation et conférences : Televic propose des systèmes de gestion de la parole et de la communication pour les salles de classe, les salles de réunion et les conférences. Ces systèmes permettent d'améliorer la qualité de la communication et de la collaboration dans ces environnements.
Systèmes de vote électronique : L'entreprise fabrique également des systèmes de vote électronique pour les assemblées législatives, les conseils d'administration, les assemblées générales, etc. Ces systèmes permettent de recueillir des votes de manière électronique, rapide et sécurisée.
Systèmes de traduction simultanée : Televic développe des systèmes de traduction simultanée pour les conférences internationales et les événements multilingues. Ces systèmes permettent aux participants de comprendre les discours dans leur langue maternelle.
Systèmes de contrôle et de gestion de l'information : Televic propose des solutions logicielles et matérielles pour la gestion de l'information dans divers domaines, notamment la gestion des ressources humaines, la gestion des réunions et la gestion des données.

1.2 Mission, contexte et enjeux

Televic s'intéresse actuellement à l'analyse des caméras de surveillance dans les trains. On peut en effet s'en servir en cas de problème dans un train ou plus généralement chercher à analyser et ajuster le flux des passagers. La division qui m'a accueillie, chargée de recherche en machine learning dans l'entreprise, développe notamment des solutions de reconnaissance faciale. Mais les lois sur la protection des données font monter le besoin d'anonymiser les vidéos, que ce soit pour quelqu'un qui regarde en live ou pour des ingénieurs souhaitant travailler sur les vidéos récoltées. Comme les trains sont pour la plupart, à l'instant, équipés uniquement de CPUs, il convient de trouver une solution compacte et avec une faible complexité pour flouter le visage des passagers en temps réel, si possible sur plusieurs caméras en même temps. Dans le domaine de la détection de visage, les modèles de deep learning ont fait leur preuve ce pourquoi nous allons nous pencher sur cette solution.

2 Introduction

La détection de visage, qui consiste généralement à encadrer les visages d'une image, à de nombreuses applications (point de départ de la traque de visage ou de la reconnaissance faciale, filtre sur les réseaux sociaux, autofocus...). Elle peut notamment et dans notre cas servir à flouter automatiquement les visages dans les vidéos prises par les caméras de surveillance dans les trains en temps réel.

Depuis l'algorithme de Viola-Jones avec des features prédéterminées par l'homme en 2001, les progrès en intelligence artificielle ont fait que les modèles de deep learning se sont imposés. Les meilleurs algorithmes obtiennent de très bons résultats sur la populaire banque de données Wider Face. Cependant, la plupart de ces modèles utilisent des stratégies coûteuses en ressources comme les FPN (Lin, 2016) (feature pyramid network) ou les TTA (Hubens, 2021) (test time augmentation). Cela peut augmenter la latence en inférence et pénaliser les applications réelles qui demandent souvent à être rapides et peu coûteuses en mémoire, avec des supports qui n'ont pas forcément de GPU . Une première réponse est apportée par le modèle Yunet (Wu et al., 2023), créé dans le but explicite d'avoir un modèle compact et rapide, un but réussi. Ce modèle, comme la plupart des modèles de détection des visages, fonctionne et est entraîné sur des images. Lorsqu'ils sont utilisés sur des vidéos, ils détectent donc les visages frame par frame. D'où la nécessité d'avoir une inférence rapide si on veut pouvoir avoir des FPS acceptables sur une solution en temps réel. Ceci étant dit, l'angle mort d'un modèle frame par frame est de ne pas utiliser l'information des frames précédentes (ou futur). En effet, on pose l'hypothèse qu'un modèle utilisant l'information des frames précédentes pourrait obtenir de meilleurs résultats. En particulier, nos données d'applications, des vidéos prises dans des trains, posent plusieurs défis pour la détection des visages : une personne peut brutalement tourner ou pencher la tête (modèles plus performants sur les visages de face), des visages de plusieurs tailles en fonction de la distance à la caméra ... On précise notre hypothèse en disant qu'en particulier, la connaissance préalable d'un visage au même endroit qu'un visage dur à détecter pour les raisons évoquées pourrait sensiblement améliorer la détection. Une piste explorable serait un modèle format RNN qui prendrait en entrée plusieurs frames. Bien que l'idée soit intéressante, elle ne convient pas à notre objectif, puisqu'elle serait très coûteuse en ressources.

Dans cet article nous proposons de prendre le modèle pré-entraîné Yunet, de l'utiliser sur les vidéos de caméra de surveillance des trains, et d'y ajouter une couche de post processing prenant en compte les résultats des frames précédentes pour améliorer les résultats. Aussi, pour répondre au problème, notre solution complète peut prendre le flux d'une vidéo et l'afficher en temps réel avec les visages détectés floutés. Afin d'accélérer l'inférence, nous avons également tenté d'utiliser une version quantifiée statiquement du modèle créée à l'aide la libraire onnx runtime (microsoft). Accessoirement, les données à notre disposition étant pour l'instant non labellisées, une pipeline de labellisation semi-automatique et de mesure de la précision a été créée.

Après analyse visuelle des données, on fait aussi l'hypothèse qu'un visage difficile à identifier reste souvent au même endroit (visage caché derrière un siège ou un plafond par exemple), notre solution vise donc à bien réagir à ce genre de situation. L'objectif étant de protéger la vie privée, on cherchera à détecter tous les visages reconnaissables quitte à augmenter le nombre de faux positifs.

3 État de l'art

3.1 Modèles de deep learning

Comme dit précédemment, de très bons modèles de détection des visages existent déjà. TinaFace (Zhu, 2020), le meilleur modèle, obtient une AP de 0.93 sur la banque de données Wider Face (Hard). Le backbone du modèle se base sur un ResNet 50, un Feature Pyramid Extractor comme neck à 6 niveaux, et une head anchor-based (Christiansen, 2022) avec des features enhancement modules (FEMs). Pour parvenir à ces résultats, plusieurs techniques sont utilisées, des Inception modules, de la group normalization, de l'augmentation de données, du TTA, des annotations supplémentaires, IoU aware branch, des deformation convolution network (Mishra, 2021) ... Mais le modèle qui nous intéresse et que nous allons utiliser est Yunet qui se distingue par plusieurs parti pris pour accélérer l'inférence. Comme ils le font judicieusement remarquer dans leur article, le nombre de paramètres, la com-

plexité et la contribution réelle d'une couche ne sont pas forcément corrélés. Ainsi les premières couches, qui capturent les petits visages durs à détecter prennent peu de paramètres et méritent donc plus d'attention. Le backbone emploie la depthwise separable convolution (Wang, 2018), qui "factorise" les couches de convolution classique et réduit le nombre de paramètres et la complexité. Pour le neck, comme TinaFace, ils utilisent une FPN mais réduite à 3 niveaux et en utilisant encore des depthwise separable convolutions. Comparé à TinaFace, il utilise une head anchor free avec du simple optimal transport assignment (simTOA) (Tuan, 2022) pour l'alignement des anchors positives. Pour l'entraînement, il utilise l'augmentation de taille des visages.

3.2 Méthodes de tracking

Notre couche de post-processing vise à utiliser les frames précédentes. Dans le domaine du tracking d'objet, deux algorithmes avec cette optique se démarquent. L'optical flow (aux projets Wikimedia, 2021) prend en compte l'entourage d'un point et suit l'évolution des pixels avec une méthode différentielle. La méthode CAM-shift (Vindimian, 2021) quant à elle prend l'histogramme des couleurs de l'objet traqué et cherche à le retrouver dans les frames suivantes en calculant un maximum de densité de probabilité. Malheureusement ces deux techniques manquent de robustesse et sont rarement utilisées de nos jours.

4 Methods and data

Les données brutes à notre disposition sont directement extraites de caméras de surveillance dans des trains. Pour chaque train on a une douzaine de vidéos synchronisées parfois sur plusieurs jours qu'on peut exporter en format classique à l'aide d'un logiciel spécialisé. Les vidéos n'étant pas labellisées et constituant plusieurs centaines de Go, nous avons commencé par manuellement sélectionner des plages intéressantes dans chaque train. En effet chaque train a des conditions différentes (couleur, luminosité, placement des caméras...) donc il est intéressant d'utiliser plusieurs configurations pour tester la robustesse de notre modèle. Les vidéos sont originellement de taille 800*600.

La première étape fut d'implémenter le modèle Yunet et d'y ajouter du floutage de visage, pour avoir un modèle baseline. A cette fin on utilise l'interface de la librairie opencv, prévue pour et utilisée par les créateurs du modèle. Cette interface permet notamment de régler un seuil de confiance minimum pour les détections ainsi qu'une Non Maximum Suppression (NMS). Quand un visage est détecté, pour le flouter, on prend la partie encadrée, on y applique un filtre moyennant (10*10) et on la réinsère dans l'image.

Concernant l'évaluation du modèle, nous nous sommes rapidement rendus compte qu'ils seraient très fastidieux de labelliser des vidéos entières. De plus, mesurer la précision sur des frames prises au hasard peut manquer de pertinence de par leur redondance (sur une vidéo : même cadre, mêmes personnes ...) et le manque de nuance (pourquoi le modèle ne détecte pas un visage ? Occlusion, taille, luminosité ?). Aussi, le but du projet étant de flouter un visage pour protéger la vie privée, il est considéré comme acceptable de ne pas détecter un visage non reconnaissable. Pour ces raisons et de par la rapidité du modèle, la méthode d'évaluation privilégiée est le visionnage de la vidéo après floutage appliquée. En effet, le modèle Yunet simple tourne aux alentours de 40 FPS sur un Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz (utilisé pour toutes les expériences) si on ne modifie pas la taille des vidéos, ce qui permet de voir rapidement sur un grand nombre de frames les résultats du modèle et où ils peinent à détecter les visages. Pour plus de clarté, on laisse parfois un cadre vert traditionnel plutôt que le floutage. Il peut toutefois être difficile de comparer précisément deux résultats avec cette méthode, ce pourquoi on réalisera une pipeline de labellisation et d'évaluation par la suite.

Après visionnage d'une vidéo avec le Yunet simple, on peut voir que si le modèle fonctionne très bien pour la plupart des visages bien visibles, il a du mal avec les visages partiellement cachés, sous un plafond ou un siège.

C'est donc là que commence notre réel apport. Nous avons développé 3 algorithmes différents, les derniers se voulant être l'amélioration des précédents.

Dans chacune de ces versions, le score accordé à chaque prédiction joue un rôle crucial. En effet, on définit 2 seuils. L'un, classiquement, décrit quel visage est détecté et est considéré comme une "bonne" détection. Les détections avec un score entre le premier et le deuxième seuil 'plus pas que le premier) sont considérées comme "ambiguës".

L'idée derrière tous les algorithmes est, lors d'une détection ambiguë, de vérifier localement si une bonne détection a été faite récemment. En effet, le cas le plus classique d'une détection ambiguë dans nos données correspond à un passager qui vient de s'asseoir et dont le visage est obstrué par un siège. Dans ce cas, son visage est quasiment toujours détecté avant l'occlusion et il ne bouge plus ensuite. Le moment critique est lorsque le visage est partiellement caché mais encore reconnaissable.

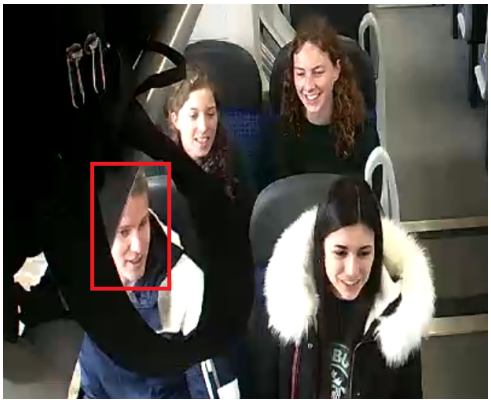

Figure 1: Exemple d'un visage partiellement caché

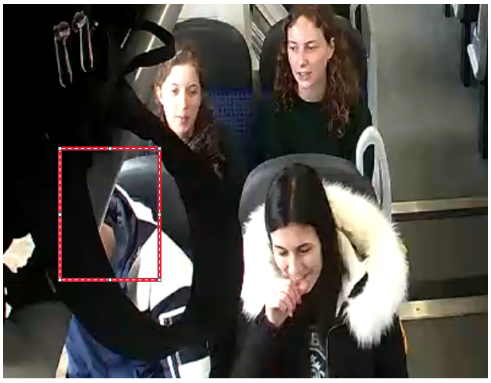

Figure 2: Même visage caché par le plafond

Le premier algorithme définit une partition de la vidéo. A chaque partie de cette partition est associé un compteur initialisé à zéro. Lorsqu'une bonne détection a lieu dans une partie de la vidéo, le compteur prend la valeur n_latence (hyper-paramètre). A chaque itération d'une frame, on soustrait 1 à tous les compteurs. Lorsqu'il y a une détection ambiguë, l'algorithme considère la détection valide si elle est localisée dans une partie avec un compteur positif.
Pour récapituler, les hyper paramètres sont les 2 seuils, la partition de la vidéo et la valeur d'initialisation du compteur lors d'une bonne détection.

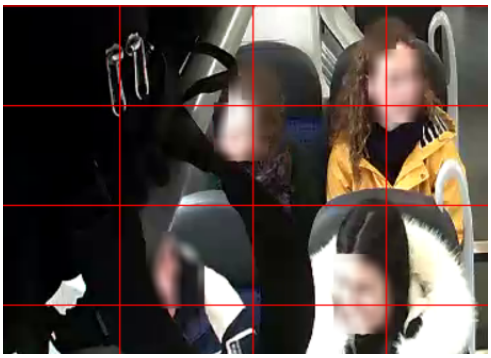

Figure 3: Exemple de partition

Le deuxième algorithme cherche à éviter l'effet de bord (visage non détecté car sur le bord d'une partie de la partition). Il crée un array de taille identique à l'image avec ses valeurs initialisées à zéro. Lors d'une bonne détection, tous les pixels autour prennent la valeur n_latence. Précisément, les pixels contenus dans le carré de taille size_latence/2 (hyper paramètre) avec pour milieu le coin haut-gauche de la bonne détection. Comme précédemment, on soustrait 1 à toutes les valeurs de l'array à chaque itération de frame et en cas de détection ambiguë, on regarde si le pixel correspondant dans l'array a une valeur positive.
Le troisième algorithme vise à limiter le nombre de faux positifs. En effet, même si ce n'est pas le plus gênant (la vidéo reste lisible même en cas de quelques floutages abusifs), le fait de baisser le seuil par zone entraîne rapidement des faux positifs en cas de déplacement d'une personne si le seuil d'ambiguïté est trop bas. De plus, la fluctuation rapide des floutages des détections à faible score, instables, sont désagréables à l'œil. Comme le deuxième algorithme, il crée un array dont il met à jour de la même manière les valeurs. La différence est la création d'une nouvelle valeur "max" qui recense le maximum de bonnes détections simultanées dans une frame. Elle vise à compter le nombre de personnes dans la pièce. Un tri est effectué sur les prédictions selon leur score. Toutes les bonnes détections sont floutées. On floute ensuite (max - le nombre de bonnes détections + une marge) de détections ambiguës en commençant par celles qui ont le meilleur score. Le max est remis à 0 toutes les 30s.
Pour pouvoir évaluer plus précisément nos algorithmes, nous avons créé une pipeline de labellisation semi-automatique. On prend 10 images uniformément réparties dans le temps par vidéo (en commençant 10s après le début pour qu'on puisse

prendre en compte les frames précédentes) et on enregistre les prédictions sous le format de Yolo5. Par facilité d'implémentation nous avons utilisé notre propre 3ème algorithme mais il aurait été judicieux d'utiliser un plus lourd et puissant comme TinaFace. On peut ensuite vérifier et annoter à la main à l'aide de YoloLabel. Pour calculer les précisions, on peut ensuite utiliser l'outil suivant Cependant nous n'avons pas annoté suffisamment d'images pour que ce soit utile pour mesurer la précision des modèles.
Une solution très simple pour accélérer l'inférence est de changer la taille des frames en entrée, solution qui a été implémentée. Les hyperparamètres sont définis de telle façon qu'ils sont robustes au changement de taille.
Une autre solution qui a été étudiée est la quantification du modèle Yunet. Une quantification statique a été réalisée à l'aide de la libraire onnx runtime.
Note : le premier algorithme envisagé enregistrait les prédictions et faisait une recherche par distance mais cette implémentation mais cette solution a une complexité exponentielle et peut difficilement marcher sur un grand nombre de frames sans faire exploser la mémoire.

5 Experiments and results

Le premier algorithme obtient de meilleurs résultats que le modèle Yunet simple quand il s'agit de détecter les visages partiellement cachés au prix d'un plus grand nombre de faux positifs. On remarque rarement un effet de bord. Comme pour les algorithmes suivant, la complexité additionnelle est négligeable, on perd 2-3 FPS et on arrive aux alentours de 37 FPS.

Le deuxième algorithme fait disparaître l'effet de bord et détectent encore mieux les visages difficiles avec les paramètres choisis (plus flexibles) même s' il y a encore plus de faux positifs.

Le troisième algorithme conserve la performance mais fait quasi entièrement disparaître les faux positifs. On en a encore quelques-uns qui se superposent avec les visages.

Le 3ème algorithme est relativement robuste au changement de taille de l'image en entrée et peut être grandement accélérée de cette façon : une division de la taille par 2 (de l'aire par 4) multiplie la vitesse par environ 4. Les petits visages finissent cependant par en pâtir, même si on pourrait argumenter qu'ils ne sont de toute façon pas toujours identifiables.

-	1er	2e	3e
seuil bonne détection	0.15	0.2	0.2
seuil ambiguité	0.8	0.6	0.6
n_latence	100	100	100
size latence	-	10	10
marge max	-	-	2
partition	8*8	-	-

Table 1: Valeur des hyperparamètres des algorithmes

Au sujet de la quantification, si on obtient effectivement un modèle plus compact, il est étonnamment plus lent que le modèle original. La piste privilégiée pour expliquer ce problème est une moindre optimisation au niveau du hardware.

Par ailleurs, les tests nous auront fait remarquer qu'un floutage plus élevé est nécessaire pour les visages de très grande taille.

6 Conclusions and future work

A partir du modèle compact et rapide Yunet, frame par frame, nous avons développé des scripts de post-processing en espérant améliorer nos résultats en prenant en compte la dimension temporelle. Malgré un manque déplorable d'analyse quantitative, nous estimons par une analyse qualitative cet objectif atteint avec notre 3ème algorithme qui baisse localement le seuil lorsque que de précédentes détections ont eu lieu. Si ce script a été réalisé avec une banque de données particulières en tête, il devrait pouvoir bénéficier à d'autres tâches de détection de visage sur vidéo, notamment en baissant la valeur d'initialisation du compteur après une bonne détection, pour parer à plus de mouvement.
Concernant les perspectives futures, plusieurs remarques. Si l'objectif initial était de faire fonctionner le modèle sur CPU, il pourrait aussi être intéressant de le tester sur GPU, et nous sommes en train de l'implémenter. Il serait aussi profitable de résoudre le problème de la quantification.
Évidemment, ce projet aurait pu grandement bénéficier d'une banque de données labellisées pour tester de façon plus fiable et précise les algo-

rithmes et finetune les hyper paramètres par grid-search et nous prévoyons également d'exploiter la pipeline de labellisation et d'évaluation.
Concernant le cœur du sujet même, on peut également envisager avec des données labellisées de traiter les prédictions de Yunet non pas avec un algorithme codé en dur mais avec une couche dense ou un réseau convolutionnel avec des channels correspondant aux frames précedentes. Au contraire, on pourrait tenter de trouver une solution exploitant des méthodes plus anciennes comme l'optical flow ou le CAM-shift.

7 Retour sur la mission

L'objectif d'avoir une solution acceptable mais pas parfaite pour flouter les visages en temps réel ayant rapidement été atteint par l'implémentation de Yunet, j'ai cherché à la demande et sous la supervision de mon tuteur a intégré la dimension temporelle pour améliorer les résultats. Aussi, si la solution fonctionne bien pour une caméra (40 FPS) nous avons cherché à aller plus loin dans l'objectif de pouvoir le faire fonctionner sur plusieurs caméras en même temps, d'où le désir de quantification et le besoin de tester la robustesse des modèles lors d'un changement de taille de l'entrée. Mes travaux devraient se concrétiser dans la présentation d'une demo à un forum et serviront potentiellement à ceux travaillant sur la reconnaissance faciale en étant adapté au problème.
Concernant les apports personnels, ce stage m'a donné l'occasion d'étudier les meilleures méthodes de détection de visage et plus généralement de détection des objets. Aussi contrairement aux projets réalisés en cours, la complexité du modèle devenant une réelle contrainte, ce qui m'a donné l'occasion de m'intéresser aux diverses techniques pour accélérer l'inférence, comme la quantification post ou pré entraînement, au pruning, mais aussi à des solutions toutes bêtes mais diablement efficaces qui ne m'étaient pas forcément évidentes comme changer la taille de l'image.
Aussi j'ai pu découvrir de forts outils de labellisation et d'évaluation.
De façon générale, j'ai aussi découvert et/ou mis en pratique des logiciels ou modules comme une interface pour un serveur à distance, le module os pour gérer efficacement les fichiers, cmake pour compiler à partir du code source openCV avec CUDA, pyinstaller pour obtenir un fichier .exe afin de livrer une démo facilement utilisable par un client . . . ce qui me permet d'être plus à l'aise et efficace quotidiennement.

References

Contributeurs aux projets Wikimedia. 2021. Méthode de Lucas–Kanade. *fr.wikipedia.org*.

Anders Christiansen. 2022. Anchor boxes — the key to quality object detection | towards data science.

Maurice Gross and Jean Senellart. 1998. Nouvelles bases statistiques pour les mots du français. In *Journée d'Analyse Statistique des Données Textuelles (JADT)*, pages 335–349, Nice.

Nathan Hubens. 2021. Test Time augmentation (TTA) and how to perform it with KERAS.

Tsung-Yi Lin. 2016. Feature pyramid networks for object detection.

microsoft. Onnxruntime/onnxruntime/python/tools/quantization/quantize.py at main · microsoft/onnxruntime.

Divyanshu Mishra. 2021. Deformable convolutions demystified - towards data science.

Anh Tuan. 2022. Paper review: "OTA: Optimal Transport Assignment for Object Detection".

Claudio Vindimian. 2021. Understanding and implementing the CAMShift Object Tracking Algorithm (Python).

Chi-Feng Wang. 2018. A basic introduction to separable convolutions - towards data science.

Wei Wu, Hanyang Peng, and Shiqi Yu. 2023. YuNet: a tiny millisecond-level face detector. *Machine Intelligence Research*.

Yanjia Zhu. 2020. TinaFace: Strong but simple baseline for face detection.

a

A Annexe

Avec le rapport devrait être fourni le fichier demo_live.exe qui permet de lancer l'algorithme 3.

Pour l'utiliser, exécuter la commande :

demo_live.exe -v videopath

Et être dans le même dossier que le fichier yunet.onnx

Aussi, le code source de l'algorithme 3 correspond aux fichiers tuto_tr.py et functions_tr.py

A.1 Architecture de Yunet et TinaFace

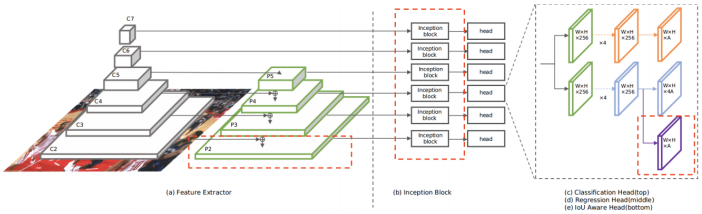

Figure 1: The model architecture of TinaFace. (a) Feature Extractor: ResNet-50 [11] and 6 level Feature Pyramid Network [18] to extract the multi-scale features of input image. (b) Inception block to enhance receptive field. (c) Classification Head: 5 layers FCN for classification of anchors. (d) Regression Head: 5 layers FCN for regression of anchors to ground-truth objects boxes. (e) IoU Aware Head: a single convolutional layer for IoU prediction.

Figure 4: Architecture de TinaFace

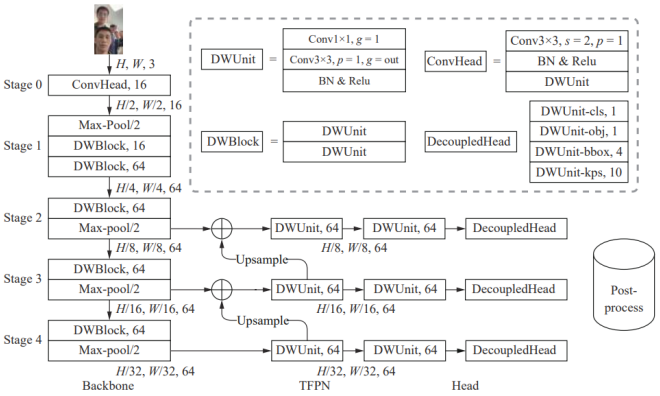

Figure 5: Architecture de Yunet