Projet de deep learning
Estimation des distances entre les individus dans une image.

Ilann Amiaud–Plachy[1], Gaétan Jacquemin[1]

[1]CentraleSupélec, Paris-Saclay University

Abstract

Nous nous sommes intéressés à l'estimation des distances entre les personnes dans une foule à partir d'une image monoculaire. Notre solution se concentre spécifiquement sur les distances entre les têtes, qui sont les seuls éléments clairement détectables dans ce contexte. Face à l'absence de modèles directement adaptés à cette tâche dans l'état de l'art, nous avons développé une approche hybride. Celle-ci repose sur deux modèles : un modèle de détection d'objets, affiné sur un jeu de données adapté pour identifier les têtes, et un modèle d'estimation de carte de profondeur. En combinant ces deux estimations, nous avons modélisé le fonctionnement de la caméra et utilisé les lois de l'optique géométrique pour, d'une part, corriger la carte de profondeur estimée, et d'autre part, calculer les distances entre les têtes. Nos résultats sont qualitativement prometteurs, mais une évaluation quantitative reste nécessaire pour les valider pleinement.

Introduction

L'analyse des foules à partir d'images constitue un domaine de recherche en pleine expansion, avec des applications variées allant de la gestion de la sécurité publique à l'optimisation des flux dans les espaces urbains. Parmi les défis inhérents à cette tâche, l'estimation précise des distances entre les individus dans une foule à partir d'une seule image monoculaire reste particulièrement complexe. En effet, l'absence de données tridimensionnelles directes et la variabilité des scènes rendent cette problématique difficile à résoudre avec les approches classiques.

Notre volonté pour réaliser cette tâche proviens de notre projet de mentions, qui vise à détecter les dangers dans une foules à partir des données de déplacement des individus. Cependant il n'existe pas vraiment de set de données contenant le déplacement des personnes au sein d'une foule, il est cependant facile de se procurer des vidéos de foules. Notre meta objectif est donc de pouvoir extraire les données de déplacements des individus d'une foule à partir d'une simple vidéo. Il était donc nécessaire dans la construction de notre solution de penser à l'adaptabilité future à un flux vidéo.

Dans ce contexte, notre projet s'est attaché à développer une méthode innovante pour estimer les distances interpersonnelles sur une image (dans un premier temps), en se concentrant spécifiquement sur les têtes, qui représentent les éléments les plus distinctement détectables dans une foule. Face à l'absence de solutions préexistantes dans la littérature, nous avons conçu une approche hybride combinant des techniques de vision par ordinateur et des principes d'optique géométrique. Ce rapport présente notre méthodologie, qui repose sur l'intégration d'un modèle de détection d'objets affiné et d'un modèle d'estimation de profondeur, ainsi que les premiers résultats obtenus. En explorant cette problématique, notre objectif est de poser les bases d'une solution robuste et généralisable, tout en identifiant les axes d'amélioration.

Détection des têtes

Etat de l'art de la détection des têtes

La détection des têtes est un sujet fondamental en vision par ordinateur, notamment pour des applications variées telles que la surveillance de foules, et les interactions homme-machine. Ce problème est souvent abordé avec des techniques similaires à la détection d'objets, en particulier les approches basées sur l'apprentissage profond.

Modèles basés sur les Réseaux de Neurones Convolutifs (CNNs) Les CNNs sont les architectures les plus populaires pour la détection d'objets grâce à leur capacité à extraire des caractéristiques discriminantes [1]. Les architectures telles que Faster R-CNN [2], SSD [3] et RetinaNet [4] ont été adaptées pour la détection des têtes grâce à leur capacité à gérer différentes tailles.

Modèles basés sur YOLO YOLO (You Only Look Once) est une approche de détection d'objets qui allie vitesse et précision [5]. Depuis YOLOv1 jusqu'à YOLOv11, les améliorations successives ont permis d'obtenir des performances toujours plus optimales en termes de précision et d'efficacité [6, 7]. YOLO est particulièrement adapté à la détection en temps réel, ce qui le rend pertinent pour des applications de surveillance de personnes [8, 9]. Récemment, YOLOv12 [10] a introduit une architecture centrée sur

les mécanismes d'attention tout en maintenant une vitesse d'inférence comparable aux modèles CNN classiques. Cette approche permet d'améliorer la qualité de la détection tout en conservant des performances adaptées aux besoins des applications en temps réel.

Présentation des datasets

Dans notre étude, nous recherchons des données contenant un grand nombre de personnes, car notre objectif est de nous intéresser spécifiquement aux foules denses. Pour faciliter les premières expérimentations, nous avons cependant commencé avec une densité plus faible. On portera une attention particulière à l'annotation des têtes, qu'on souhaite utiliser par la suite pour affiner notre estimation de la profondeur, la tête humaine ayant une taille relativement constante.

À cette fin, nous avons sélectionné le dataset CrowdHuman [11], qui contient des images de très bonnes qualités à faible densité et très bien annotées. Ensuite, nous avons utilisé JHU-Crowd++ [12], avec des images de moins bonnes qualités, aux annotations plus sommaires et mal définies au niveau des têtes. Par ailleurs, nous avons identifié deux autres datasets similaires que nous n'avons pas exploités dans ce projet : NWPU-Crowd [13], et UCF-QNRF [14] contenant 1 535 images affichant une densité moyenne de 815 personnes par image.

Dataset	Nombre d'images	Personnes par image (moyenne)
CrowdHuman [11]	15 000	22.64
JHU-Crowd++ [12]	4 372	346
NWPU-Crowd [13]	5 109	418
UCF-QNRF [14]	1 535	815

Table 1: *Tableau récapitulatif des datasets considérés.*

Finetuning des modèles YOLO

Pour la détection des têtes, nous avons choisi d'affiner des modèles de différentes tailles, YOLOv11 et YOLOv12, pré-entraînés sur le dataset COCO [15]. La bibliothèque développée par Ultralytics, qui contient ces modèles YOLO, intègre ses propres méthodes de fine-tuning adaptées. Dans un premier temps, nous avons utilisé le dataset CrowdHuman, puis nous avons essayé d'adapter notre modèle à une plus grande échelle avec JHU-Crowd++, en modifiant les annotations pour ne conserver que les têtes, conformément au format attendu par YOLO. Nous avons également enrichi l'entraînement avec des données synthétiques générées par le module Ultralytics appelé *Mosaic*, qui combine plusieurs images pour créer une nouvelle image d'entraînement, améliorant ainsi la robustesse du modèle. L'optimiseur Adam a été utilisé pour l'entraînement. Nous avons exploité un GPU A100, fourni par CentraleSupélec, avec 9.5GB de mémoire disponible. La taille du batch et celle des images ont été ajustées pour ne pas dépasser cette limite, selon la taille du modèle (en pratique, les versions *nano* et *large* ont été entraînées). Notre modèle n'a qu'une seule classe, celle des têtes, contrairement aux modèles pré-entraînés YOLO qui comprennent plusieurs classes. Les performances sont évaluées sur le dataset de validation en utilisant la *mean Average Precision* (mAP) et le *F1-score* comme principales métriques. Les résultats obtenus sont ensuite analysés pour comprendre les limites du modèle et identifier des pistes d'amélioration. Le dataset de test a été réalisé en prenant 20% des données de chaque dataset, et en les mélangeant pour obtenir un dataset de test équilibré, et qui généralise au mieux. Les résultats sont présentés dans la section suivante. De plus, nous gardons les poids du modèle qui a le meilleur mAP sur le dataset de validation, pour simuler un early stopping.

Résultats quantitatifs

Sur le dataset CrowdHuman, nous avons essayé plusieurs modèles YOLO de différentes tailles, pour comparer leurs performances. Les résultats obtenus sont présentés dans le tableau 3. Nous avons trouvé les meilleurs résultats avec le modèle large de YOLOv11, mais ce n'était pas significativement meilleur que les autres modèles. Nous avons donc décidé de sélectionner avec le modèle nano de YOLOv11 pour la suite de nos expérimentations, car il est plus rapide et moins gourmand en ressources.

Modèle	mAP	F1-score
YOLOv11 nano	0.67	0.69
YOLOv11 large	0.70	0.72
YOLOv12 nano	0.66	0.68
YOLOv12 large	0.67	0.69

Table 2: *Résultats quantitatifs de la détection des têtes selon la taille du modèle.*

Les résultats obtenus sur le dataset CrowdHuman sont très satisfaisants, avec un mAP de 0.65 et un f1-score de 0.68. Cependant, le modèle n'arrive pas à apprendre sur le dataset JHU-Crowd++, avec un mAP de 0.1 et un f1-score de 0.14. Ces résultats montrent que le modèle peine à généraliser à des densités plus élevées, dû sûrement à la mauvaise qualité du dataset, et au fait que, à une certaine densité, les têtes ne sont pas très visibles. On a donc essayé une approche alternative pour augmenter la densité Les résultats: nous avons fine-tuner une seconde fois, avec un learning rate plus bas, sur les images avec moins de 150 personnes et ne contenant pas de têtes

floues de JHU-crowd++, et nous avons réussi à augmenter grâce à cela les résultats de notre modèle.

Dataset	mAP	F1-score
CrowdHuman	0.65	0.68
JHU-Crowd++	0.1	0.14
CrowdHuman + JHU-Crowd++	0.66	0.7

Table 3: *Résultats quantitatifs de la détection des têtes selon le dataset.*

Résultats qualitatifs

Comme on peut l'observer sur l'exemple Figure 1, la détection des têtes est plutôt bonne, lorsque l'on est dans les champs proches et/ou que l'on voit bien la tête.

Cependant le modèle atteint ses limites lorsqu'il y a trop de têtes sur l'image, ou que celles-ci sont partiellement cachées ou en mauvaise résolution (car trop loin).

Figure 1: *Résultats de la détection des têtes.*

De même, on peut comparer notre modèle finetuné au modèle sans finetuning qui contient une classe "personne" et non "tête". On peut voir sur la Figure 4 que notre modèle finetuné est bien plus précis que le modèle sans finetuning, qui détecte moins d'1/3 des personnes dans une foule peu dense.

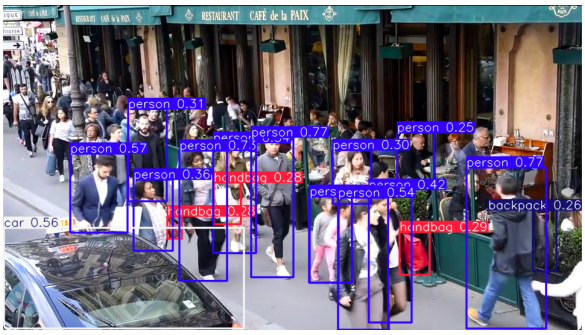

Figure 2: *Modèle YOLO non-finetuné (13 personnes détectées)*

Figure 3: *Modèle YOLO finetuné sur la détection de têtes (36 personnes détectées)*

Figure 4: *Comparaison de la détection de têtes entre un modèle YOLO non-finetuné et finetuné*

Estimation monoculaire de la profondeur

L'estimation monoculaire de la profondeur est un problème qui consiste à déterminer la distance à la caméra de chaques pixels d'une image en se basant uniquement sur l'image elle-même.

C'est un problème complexe d'actualité avec beaucoup d'application, comme la segmentation des objets ou la reconstruction 3D (pour le cinéma ou pour la modélisation). L'estimation de la profondeur est certe réalisable avec des technologies de prise de vue particulières (comme la stéréovision), mais avec une estimation monculaire transforme n'importe quelle caméra en un capteur de profondeur.

Etat de l'art

Comme expliqué précedemment, du fait de ses applications industrielles direct, ce problème est déjà largement adressé et des modèles de deep learning sont déjà incroyablement performants. Pour ceux proche de notre application on retiendra *DepthAnythingv2* [16], *Monodepth2* [17], ou encore *Depth Pro* [18] de apple qui vient tout juste de sortir.

On peut citer aussi *DenseDepth* [19] pour de la prédiction sur des images 360°.

Choix & résultat

Pour la réalisation de ce projet et après les avoir essayé notre choix s'est porté sur *DepthAnythingv2* pour plusieurs raisons. En premier lieu, il faut comprendre que la plupart des modèles d'estimation monoculaire marchent bien sur des champs proches, mais ont tendance à être limité sur des champs lointains. Or ce qui nous interresse, comme nous utilisons des images extérieures à ciel ouvert (souvent), il s'agit souvent de champs lointains et extérieur, ce qui ne correspond pas aux datasets d'entrainement de ces modèles.

Or justement *depthanythingv2* propose une version "outdoor" et en système métrique, ce qui convient précisément à notre application, notre choix s'est donc porté sur ce modèle et nous avons obtenu des résultats plutôt satisfaisant comme vous pouvez le voir Figure 5.

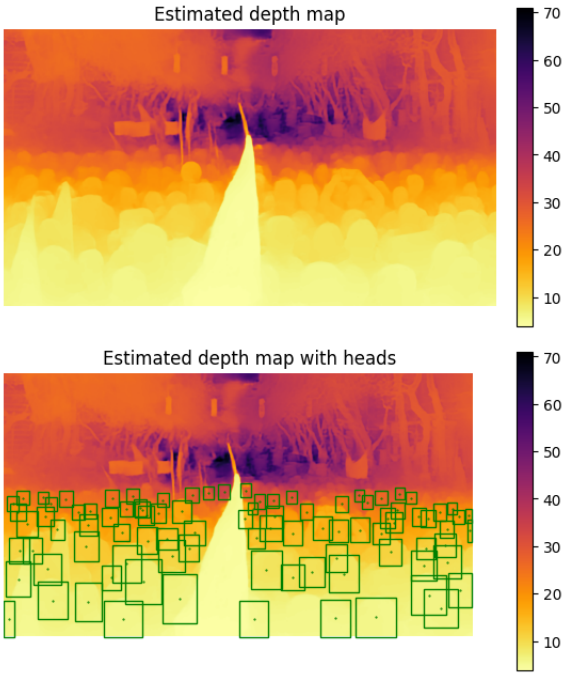

Figure 5: *Carte de profondeur estimée par DepthAnythingv2 outdoor (base) sur notre exemple. Sans les headboxes (haut) et avec les headboxes (bas).*

Correction de la carte de profondeur

Le problème est que le résultat est satisfaisant visuellement, mais après exploitation (expliquée ci-dessous) on s'est rendu compte que l'on avait pas des résultats très cohérent suivant si on se trouvait dans un champs proche ou lointain. par exemple une distance entre têtes dans un champs lointain parraissait cohérente tandis que une distance entre têtes dans un champs proche était beaucoup trop grande. On a donc décidé de corriger cette carte de profondeur en lui appliquant une transformation.

Pour se faire, et comme c'est les têtes qui nous intéressent, on s'est appuyé sur elles pour corriger la carte de profondeurs. En effet, avec un peut de recherche dans la litérature on comprend que les têtes font sensiblement toutes la même taille, en tout cas qu'il y a une faible variance dans la population.

Et justement grâce à notre modèle finetuné de *Yolov11*, pour les têtes detectés on a une boite englobante de la tête qui est assez précise. Par ailleur, en modélisant l'appareil photo (de manière simplifié) comme une lentille et un capteur placé à son point focale (voir Figure 6), on peut établir la relation suivante 1 .

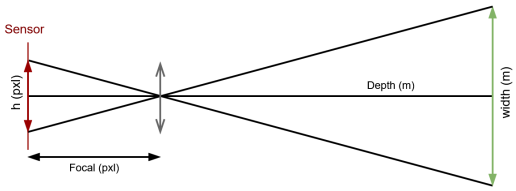

Figure 6: *Hyopthèse de modélisation (simplifiée) de l'optique d'un appareil photo.*

$$\frac{\text{width (pxl)}}{\text{focale (pxl)}} = \frac{\text{width (m)}}{\text{depth (m)}} \tag{1}$$

et comme notre focale d'appareil est une constante, alors le produit width $\times$ depth est proportionnel à la taille de la tête et doit donc être également répartie et ne pas dépendre de la prodonfeur, hors ce n'est pas ce que l'on observe, on voit en général nettement une correlation, comme sur notre exemple Figure 7 (gauche).

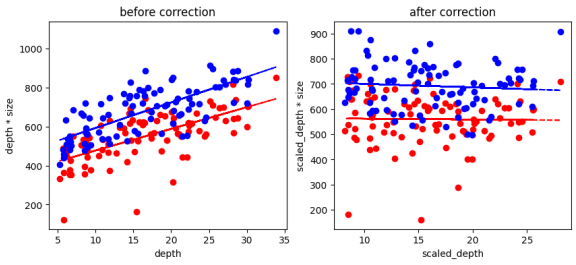

Figure 7: *(à gauche) avant la correction, on voit la corrélation. (à droite) la correction annule la correlation.*

On effectue donc une régression linéaire pour déterminer une correctione linéaire à la carte de profondeur qui permet d'annuler la correlation entre le produit expliqué et la taille des têtes, comme on le vois figure 7 (droite).

On corrige ainsi la carte des profondeurs pour notre image (voir l'exemple Figure 8).Sur notre exemple, on voit que l'échelle des profondeurs a été réduit,

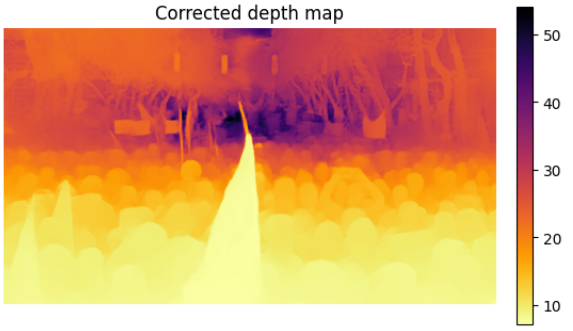

Figure 8: *Correction de la carte de profondeur sur notre exemple.*

et que la distance maximum est passé à 50m ce qui est peut cohérent. Cependant comme on s'est concentré uniquement sur la région où sont disposé les têtes pour la correction, sur cette région la correction est plutôt cohérente (testé sur beaucoup d'images)

Estimation de la focale

On détermine aussi la focale grâce à la taille des têtes. En effet, grâce à la relation 1 On estime une valeur de la focale pour chaques tête détecté, en supposant que une tête doit avoir des mensurations constantes(pour la largeur et pour la hauteur), on prend les mensurations suivantes (tiré d'études statistiques):

- hauteur : 22cm
- largeur : 19cm

Finalement, on prend comme focale le maximum de la distribution des focales estimées pour chaques têtes (voir Figure 9). On se base au finale seulement sur les focales estimés à partir de la hauteur de la tête après avoir empiriquement constaté une meilleur constance de la hauteur que de la largeur.

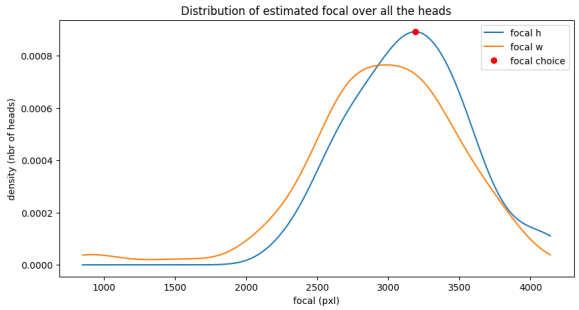

Figure 9: *Distribution des focales estimées pour chaque têtes, et choix de la valeur estimé de la focale (en pixels).*

Cette focale est très importante puisque c'est elle qui nous permet de calculer la ditance sur l'axe X en fonction de la profondeur (toujours grâce à la relationn 1).

Estimation des distances

Pour l'estimation des distance on doit décomposer l'espace en plusieurs dimensions. Pour simplifier on va diviser l'espace en 2 dimensions.

Pour 2 têtes, On prend comme référence le plan qui passe par ces 2 têtes et dont une des 2 dimensions est parrallêle au sol (donc au bord inférieur de l'image). Ainsi on a les 2 dimensions:

- la dimension X est la dimension parrallêle au bord inférieur de l'image
- la dimension Y est celle perpendiculaire à celle-ci, mais qui suit le plan passant par les 2 têtes.

Ainsi, en obtenant la projection de lu vecteur disctantce sur ces 2 dimensions, on calcul la distance totale avec le théorème de pythagore.

Par ailleurs on notera : (x_1, y_1) et (x_2, y_2)les coordonnées des 2 têtes (en pixel), (d_1, d_2) leur profondeur respective (en mêtre) et (dX, dY) le vecteur distance entre les 2 têtes (en mêtres). Et f la focale de la caméra (en pixel).

distance sur l'axe Y

Pour calculer la distance sur l'axe y, qui est en fait la différence de profondeur corrigé pour prendre en compte la position relative des personnes, on utilise le théorème d'alkachy.

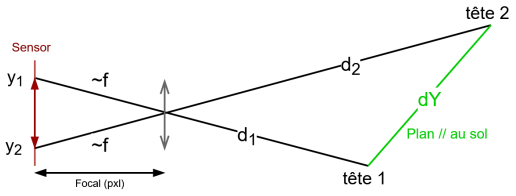

Figure 10: *Modélisation pour la distance sur l'axe Y.*

$$dY^2 = d_1^2 + d_2^2 - 2d_1d_2(1 - (\frac{|y_2 - y_1|^2}{2 * focal^2})) \qquad (2)$$

distance sur l'axe X

Pour la calculer la distance dans cette dimensions c'est assez facile, en utilisant la formume 1 on détermine la distance en mêtre au niveau de x1 avec d1, puis au niveau de x2 avec d2. On fait l'hypothèse que la perspecive est parfaitement plongeante ce qui fait qu'il faut enlever la moitié de la différence pour obtenir la distance entre les 2 têtes (avec $d_2 > d_1$).

$$dX1 = (x_2 - x_1) * d1/focal \quad (3)$$
$$dX2 = (x_2 - x_1) * d2/focal \quad (4)$$
$$dX = dX2 - (dX2 - dX1)/2 \quad (5)$$
$$= (dX2 + dX1)/2 \quad (6)$$

distance totale

Ainsi, on obtiens grâce au théorème de pythagore on peut calculer la distance inter-personnes en mètre.

Figure 11: *Exemple de distance estimé entre 2 têtes.*

Par exemple, sur notre exemple entre les 2 têtes de la Figure 11, on obtient les distances:

- $d_1 = 20.82m$ (point cyan)
- $d_2 = 10.9m$ (point bleu)
- $dY = 9.93m$ (vert)
- $dX = 1.12m$
- $d_{total} = 10.04m$ (rouge)

Pour aller plus loin

Pour aller plus loin sur la détection de têtes, on pourrait continuer de fine-tuner notre modèle sur des dataset plus variés, ou plus grand, pour améliorer la généralisation de notre modèle. On pourrait aussi essayer d'augmenter la résolution de notre modèle pour détecter des têtes plus petites, ou plus éloignées. On pourrait de même finetuner un modèle de prédiction de la profondeur en construisant un dataset de profondeur à partir de nos mesures, ou à partir d'une simulation informatique de foules, où on pourrait obtenir directement les distances. Grâce aux estimations de distances entre les têtes, on peut envisager plusieurs applications direct:

- Soit construire un graphe de distance estimé entre les personnes dans le plan du sol.
- Soit placer directement les tête dans un espace 3D en utilisant les vecteurs de distance estimé, ce qui sera certainement plus précis

Dans tous les cas ces mesures nous permettent d'estimer la densité de personnes dans une foule avec une simple caméra, et donc évaluer les risques associés ou simplement retracer les mouvements des personnes au cours du temps.

Finalement, on pourrait entraîner un dernier modèle d'intelligence artificielle prenant en entrée les distances entre les têtes ainsi que la profondeur donné par le modèle DepthAnything au niveau de la tête pour prédire la distance entre les personnes, et ainsi obtenir une estimation plus précise et adaptable à des situations plus complexes.

Conclusion

En conclusion, nous avons utilisé des modèles de deep learning avancés pour concevoir une application hybride efficace. Cette solution fusionne détection de têtes et estimation de profondeur, offrant des résultats satisfaisants.

Notre méthode présente des limites, notamment l'imprécision héritée des modèles combinés, qu'on a tenté de compenser. Une approche "end-to-end" aurait pu être plus précise, mais nous manquions du dataset nécessaire.

Ses atouts reposent sur l'utilisation de modèles pré-entraînés performants, demandant peu de ressources. Cela la rend idéale pour notre projet et facilement adaptable aux progrès futurs. De plus, sa construction la rend tout à fait adaptable à un flux vidéo (pour notre objectif final), en trackant les têtes détecté au cours du temps.

Enfin, il nous manque une évaluation chiffrée pour juger pleinement notre méthode. Une expérience réelle avec caméra ou une simulation via un moteur de jeu vidéo pourrait serait la prochaine étape pour valider nos performances.

References

[1] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[2] Shaoqing Ren et al. "Faster R-CNN: Towards real-time object detection with region proposal networks". In: (2015), pp. 91–99.

[3] Wei Liu et al. "SSD: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[4] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *IEEE transactions on pattern analysis and machine intelligence* 42.2 (2017), pp. 318–327.

[5] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[7] Glenn Jocher et al. "YOLO by Ultralytics". In: *Zenodo* (2023). DOI: 10.5281/zenodo.7466651.

[8] Weichao Ge et al. "Efficient and accurate head pose estimation using lightweight convolutional neural networks". In: *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2019, pp. 4694–4698.

[9] Deng-Ping Lian et al. "Locating heads in dense crowd scenes: A recognition-based approach". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (2021), pp. 3874–3888.

[10] Yunjie Tian, Qixiang Ye, and David Doermann. "YOLOv12: Attention-Centric Real-Time Object Detectors". In: (2025). arXiv: 2502.12524 [cs.CV]. URL: https://arxiv.org/abs/2502.12524.

[11] Shuai Shao et al. "CrowdHuman: A Benchmark for Detecting Human in a Crowd". In: *arXiv preprint arXiv:1805.00123* (2018).

[12] Vishwanath A Sindagi, Rajeev Yasarla, and Vishal M Patel. "JHU-CROWD++: Large-Scale Crowd Counting Dataset and A Benchmark Method". In: *Technical Report* (2020).

[13] Qi Wang et al. "NWPU-Crowd: A Large-Scale Benchmark for Crowd Counting and Localization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). DOI: 10.1109/TPAMI.2020.3013269.

[14] Haroon Idrees et al. "Composition Loss for Counting, Density Map Estimation and Localization in Dense Crowds: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part II". In: Sept. 2018, pp. 544–559. ISBN: 978-3-030-01215-1. DOI: 10.1007/978-3-030-01216-8_33.

[15] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: https://arxiv.org/abs/1405.0312.

[16] Lihe Yang et al. *Depth Anything V2*. 2024. arXiv: 2406.09414 [cs.CV]. URL: https://arxiv.org/abs/2406.09414.

[17] Clément Godard et al. *Digging Into Self-Supervised Monocular Depth Estimation*. 2019. arXiv: 1806.01260 [cs.CV]. URL: https://arxiv.org/abs/1806.01260.

[18] Aleksei Bochkovskii et al. *Depth Pro: Sharp Monocular Metric Depth in Less Than a Second*. 2024. arXiv: 2410.02073 [cs.CV]. URL: https://arxiv.org/abs/2410.02073.

[19] Seongyeop Yang, Kunhee Kim, and Yeejin Lee. "Dense depth estimation from multiple 360-degree images using virtual depth". In: *Applied Intelligence* 52.12 (Mar. 2022), pp. 14507–14517. ISSN: 1573-7497. DOI: 10.1007/s10489-022-03391-w. URL: http://dx.doi.org/10.1007/s10489-022-03391-w.