Distant Viewing : Tutoriel 2

Taylor Arnold
Distant Viewing Lab
tarnold2@richmond.edu

Modèles avancés

Distant Viewing Lab

Bienvenue

Ces diapositives accompagnent les tutoriels Python que nous avons conçus pour introduire les fondements théoriques et méthodologiques du distant viewing, ainsi qu'un aperçu des outils du Distant Viewing Toolkit.

Elles contiennent des notes, références et schémas supplémentaires que nous trouvons utiles lors des ateliers en présentiel. La plupart des notes techniques et du matériel se trouvent directement dans les notebooks.

Pour en savoir plus sur notre travail, rendez-vous sur le site du Distant Viewing Lab à l'adresse https://distantviewing.org, ou écrivez-nous directement à tarnold2@richmond.edu.

Un télescope surplombant Paris, en clin d'œil à la méthodologie du distant viewing.

Plan

Python et Colab

Ce tutoriel s'appuie sur l'environnement Google Colab, qui permet d'utiliser Python directement dans le navigateur, sans installer de logiciel sur sa machine. Colab est gratuit pour les petits projets, et des formules payantes donnent accès à davantage de ressources de calcul.

Nous utiliserons trois modules Python courants pour l'analyse de données : NumPy, Polars et plotnine.

Toutes les bibliothèques utilisées dans le tutoriel sont disponibles sous licence libre. Le code peut aussi être exécuté sur votre propre machine, moyennant une configuration un peu plus poussée.

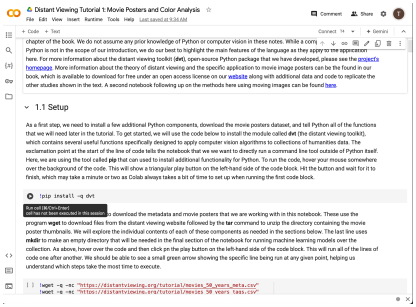
Logos de Google Colab, Python, Pandas, NumPy et Matplotlib.

Exécuter le notebook

Lorsqu'on ouvre le notebook du tutoriel dans Colab, une fenêtre similaire à celle de droite s'affiche dans le navigateur. Le notebook est consultable par tous ; pour exécuter le code, il faut se connecter à un compte Google.

Remarque : vous pouvez modifier le code et le texte localement dans votre navigateur. Ces modifications ne seront pas enregistrées pour les autres. N'hésitez donc pas à expérimenter avec le code à tout moment !

Une fois connecté à un compte Google, vous pourrez exécuter les blocs de code (les portions sur fond gris). Pour cela, passez la souris sur le fond gris et cliquez sur le bouton de lecture indiqué par la flèche rouge. Tout au long du tutoriel, il faut exécuter chaque bloc dans l'ordre.

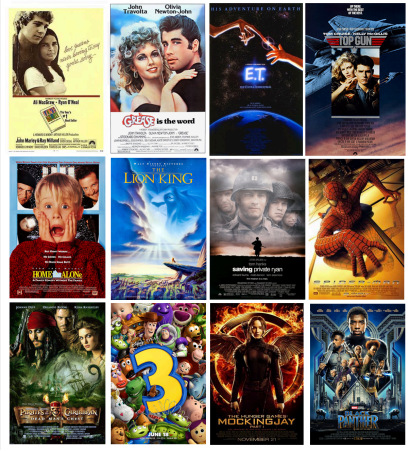


Capture d'écran de l'interface Colab exécutant le code de configuration du DVT.

Présentation du corpus

Dans ce tutoriel, nous allons explorer une collection d'affiches issues des 100 films les plus rentables de chaque année entre 1970 et 2019. Notre corpus comprend près de 5000 images (nous n'avons pas pu retrouver un petit nombre d'affiches anciennes).

Notre objectif est de comprendre comment la couleur et la composition des affiches (1) ont évolué dans le temps et (2) sont utilisées pour véhiculer (ou détourner) les conventions de genre. Nous voulons mobiliser des techniques computationnelles pour identifier et comprendre des motifs à travers les milliers d'images de notre collection.



Exemples d'affiches de films à travers les décennies et les genres du jeu de données.

Plan

DETR-ResNet

La détection d'objets, une tâche relativement ancienne en vision par ordinateur, consiste à identifier dans une image une ou plusieurs catégories d'objets définies à l'avance. Elle combine deux tâches initialement distinctes : (1) la classification et (2) la localisation. La plupart des modèles modernes pour classes fixes constatent qu'il vaut mieux les traiter conjointement.

Nous utiliserons le modèle DETR-RESNET-50, entraîné sur le jeu de données COCO 2017 qui compte 1000 catégories étiquetées sur plus de 100 000 images.[1] Comme la plupart des modèles de détection, il produit des scores de confiance et des boîtes englobantes rectangulaires censées contenir l'objet en question.

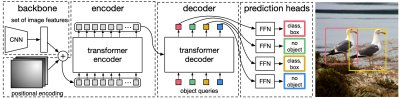


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

Architecture du modèle DETR-ResNet d'après l'article original.

[1] Carion et al. 2020.

Plan

Grounding DINO

Une version plus récente de la détection d'objets combine l'idée de détection avec un modèle de langage de type transformeur pour repérer n'importe quel objet décrit par du texte libre. Nous utiliserons Grounding DINO, un modèle disponible en poids ouverts et exécutable sans architecture ni matériel particuliers.[1]

Le modèle accepte n'importe quelle description textuelle et produit des résultats raisonnables dans les langues bien dotées avec des objets courants. Ses meilleures performances s'obtiennent toutefois en décrivant les objets par leur nom en anglais.

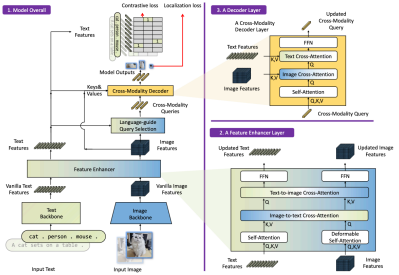


Fig. 3: The framework of Grounding DINO. We present the overall framework, a feature enhancer layer, and a decoder layer in block 1, block 2, and block 3, respectively.

Architecture du modèle Grounding DINO d'après l'article original.

[1] Liu et al. 2023.

Plan

Grounding DINO + TrOCR

Les affiches de films, comme beaucoup d'images historiques que l'on peut vouloir étudier, contiennent énormément de texte, et ce texte est essentiel pour comprendre leur message. On peut combiner Grounding DINO avec un modèle OCR pour localiser puis décoder le texte présent sur chaque affiche. Nous utiliserons TrOCR, accessible via la même API `transformers` que les autres modèles vus jusqu'ici.[1]

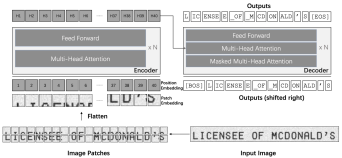


Figure 1: The architecture of TrOCR, where an encoder-decoder model is designed with a pre-trained image Transformer as the encoder and a pre-trained text Transformer as the decoder.

Architecture du modèle TrOCR.

[1] M. Li et al. 2022.

Plan

SAM : Segmentation à partir d'un point

Segment Anything (SAM) et ses successeurs ont fait beaucoup parler d'eux dans la communauté de l'apprentissage automatique.[1] SAM permet d'estimer la région, c'est-à-dire l'ensemble précis des pixels, correspondant à un objet sans avoir à savoir à l'avance de quel objet il s'agit. Les approches précédentes de la segmentation supposaient qu'il fallait connaître le nom de l'objet au préalable, mais à bien y réfléchir, cela rejoint notre façon de voir : on peut identifier la portion d'une image qui correspond à un nouvel animal même sans connaître son nom.

C'est un modèle très puissant, mais — comme on va le voir — un peu mal compris, car la plupart des gens le connaissent uniquement à travers l'image de droite, tirée de l'article original.

Figure 2: Example images with overlaid masks from our newly introduced dataset, **SA-1B**. SA-1B contains 11M diverse, high-resolution, licensed, and privacy protecting images and 1.1B high-quality segmentation masks. These masks were annotated *fully automatically* by SAM, and as we verify by human ratings and numerous experiments, are of high quality and diversity. We group images by number of masks per image for visualization (there are ~100 masks per image on average).

Exemple de SAM en action.

[1] Kirillov et al. 2023.

Plan

Segmentation guidée par texte

Comme pour le cas du texte, on peut combiner un modèle de grounding avec SAM pour produire les régions correspondant à n'importe quelle description textuelle d'un objet. Cette approche a été formalisée sous le nom de Grounded SAM, mais on peut l'implémenter soi-même directement à partir du code déjà utilisé.[1]

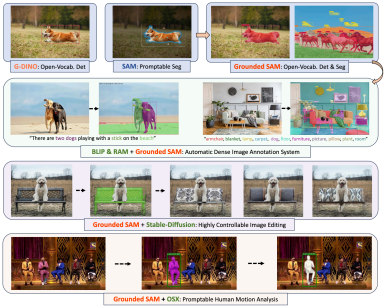


Figure 1: **Grounded SAM** can simultaneously detect and segment corresponding regions within images based on arbitrary text inputs provided by users. And it can seamlessly integrate with other Open-World models to accomplish more intricate visual tasks

Exemple de Grounded SAM.

[1] Ren et al. 2024.

Plan

Depth Anything

Le modèle suivant, Depth Anything, cherche à mesurer la distance entre chaque objet et la caméra.[1] Certaines variantes fournissent des mesures métriques précises, mais celles-ci ne sont pas très utiles pour nos affiches de films, qui ne sont pas une photographie unique prise sous un angle donné, mais plutôt un assemblage d'éléments composés dynamiquement dans un cadre. Nous resterons donc sur la version originale, plus rapide, du modèle Depth Anything.

Si la distance exacte à la caméra n'a pas de sens précis dans notre cas, nous verrons qu'elle permet tout de même de démêler l'arrière-plan et le premier plan sur de nombreuses affiches.

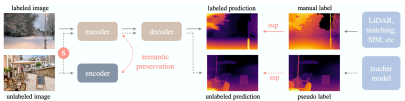


Figure 2. Our pipeline. Solid line: flow of labeled images, dotted line: unlabeled images. We especially highlight the value of large-scale unlabeled images. The S denotes adding strong perturbations (Section 3.2). To equip our depth estimation model with rich semantic priors, we enforce an auxiliary constraint between the online student model and a frozen encoder to preserve the semantic capability (Section 3.3).

Architecture du modèle Depth Anything, telle que décrite dans l'article original.

[1] L. Yang et al. 2024.

Plan

DINOv2

Toutes les annotations vues jusqu'ici sont *interprétables* : on peut dire « il y a deux personnes sur cette affiche » ou « 30 % de l'image est au premier plan ». Mais on peut aussi vouloir une représentation plus abstraite, qui résume l'image dans son ensemble en un vecteur de nombres — un *plongement* (embedding).

Les plongements sont utilisés depuis une quinzaine d'années pour travailler aussi bien sur du texte que sur des images. Bien avant les modèles de grounding vus plus haut, c'était la seule manière de travailler avec des catégories définies à l'avance et entraînées spécifiquement sur un grand jeu de données.

Nous utiliserons aujourd'hui le modèle DINOv2, qui sert également de base au modèle Grounding DINO vu précédemment.[1]

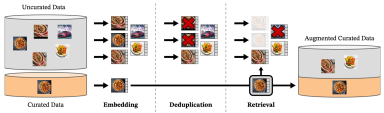


Figure 3: **Overview of our data processing pipeline.** Images from curated and uncurated data sources are first mapped to embeddings. Uncurated images are then deduplicated before being matched to curated images. The resulting combination augments the initial dataset through a self-supervised retrieval system.

Architecture du modèle DINOv2 telle que décrite dans l'article original.

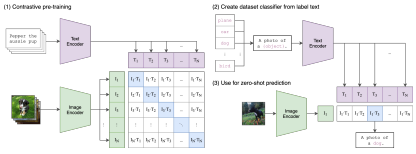
[1] Oquab et al. 2023.

Plan

SigLIP2

DINOv2 produit des plongements à partir d'images seules. Une autre famille de modèles, dits *contrastifs*, apprend conjointement à représenter images et textes dans un même espace. Le modèle le plus connu est CLIP ; SigLIP et sa version améliorée SigLIP2 en sont des évolutions plus performantes.[1]

Nous utiliserons aujourd'hui le modèle SigLIP2, qui nous permet de plonger images et textes dans un même espace de grande dimension.



Vue conceptuelle du modèle CLIP d'origine.

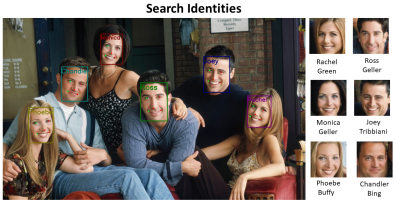
[1] Tschannen et al. 2025.

Plan

SigLIP2

Les visages sont au cœur de la grammaire visuelle des affiches de films. Nous allons ici aller un cran plus loin avec la bibliothèque InsightFace, qui détecte les visages, estime l'âge et le genre apparents, et produit un plongement permettant de reconnaître si deux visages sont ceux de la même personne.[1]

Quelques mots de précaution s'imposent. L'âge et le genre prédits par ces modèles sont des estimations probabilistes, fondées sur les données d'entraînement utilisées. Ces données présentent des biais connus (sous-représentation de certaines populations, étiquettes binaires pour le genre, etc.), et il est important d'en tenir compte dans toute analyse.



Visualisation de la détection et de la reconnaissance de visages, tirée du site d'InsightFace.

[1] Deng et al. 2021.

Plan

ViTPose

Au-delà de la simple détection d'une personne, on peut chercher à caractériser sa *pose* : position de la tête, des bras, des jambes. L'estimation de pose consiste à localiser un ensemble de points-clés anatomiques (nez, yeux, épaules, coudes, etc.) sur le corps humain.

Notre pipeline procède en deux étapes : on détecte d'abord les personnes avec DETR, puis on applique ViTPose, un modèle dédié à l'estimation de pose, sur chaque personne détectée.[1]

Figure 3: Visual pose estimation results of ViTPose on some test images from the MS COCO dataset.

Exemple de ViTPose en action, tiré de l'article original.

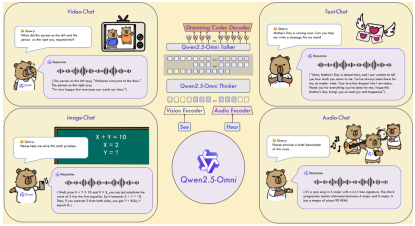
[1] Xu et al. 2022.

Plan

ViTPose

Les modèles vus jusqu'ici produisent des sorties structurées : des boîtes, des masques, des vecteurs. Les modèles vision-langage (VLM) font quelque chose de différent : on leur fournit une image et une question en langage naturel, et ils répondent en langage naturel.

On pourrait utiliser de nombreux modèles à poids ouverts, en local ou via un service comme OpenRouter. Par simplicité, nous utiliserons aujourd'hui l'API d'OpenAI. L'avantage est qu'on peut facilement l'adapter pour appeler des modèles locaux via des outils comme Ollama ou LMStudio.

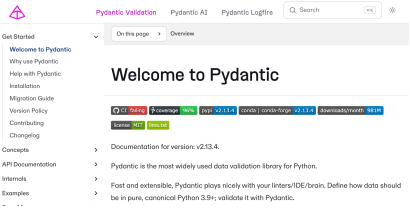


Le modèle Qwen2.5-Omni, l'un des modèles VLM généralistes à poids ouverts les plus populaires.

Plan

Pydantic

Les modèles VLM deviennent encore plus utiles lorsqu'on les force à produire des données structurées. La méthode la plus répandue, en tout cas en 2026, passe par Pydantic. Cet outil permet de décrire le schéma souhaité pour les données en sortie, puis de vérifier que le VLM les a bien structurées comme demandé.



Page d'accueil de l'outil de validation Pydantic.

Plan

Prochaines étapes

Nous avons vu toute une variété de modèles dans ce tutoriel. Ils couvrent une bonne partie des modèles de pointe disponibles mi-2026 pour travailler avec des données visuelles fixes.
Les modèles spécifiques — et, à un rythme plus lent, les tâches elles-mêmes — vont continuer à évoluer et à se renouveler dans les années qui viennent. Mais comme on l'a vu ici, les nouveaux modèles s'appuient sur les précédents et ne les remplacent que rarement totalement.
Il faut rester attentif aux évolutions du domaine, mais les outils présentés ici devraient rester utiles pendant plusieurs années.

Références

Carion, Nicolas et al. (2020). “End-to-End Object Detection with Transformers”. In: *CoRR* abs/2005.12872. arXiv: 2005.12872. URL: https://arxiv.org/abs/2005.12872.

Deng, Jiankang et al. (2021). “Masked face recognition challenge: The insightface track report”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1437–1444.

Kirillov, Alexander et al. (2023). “Segment anything”. In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4015–4026.

Li, Minghao et al. (2022). *TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models*. arXiv: 2109.10282 [cs.CL]. URL: https://arxiv.org/abs/2109.10282.

Liu, Shilong et al. (2023). “Grounding dino: Marrying dino with grounded pre-training for open-set object detection”. In: *arXiv preprint arXiv:2303.05499*.

Oquab, Maxime et al. (2023). *DINOv2: Learning Robust Visual Features without Supervision*.

Ren, Tianhe et al. (2024). “Grounded SAM: Assembling open-world models for diverse visual tasks”. In: *arXiv preprint arXiv:2401.14159*.

Tschannen, Michael et al. (2025). “Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features”. In: *arXiv preprint arXiv:2502.14786*.

Xu, Yufei et al. (2022). “ViTPose: Simple Vision Transformer Baselines for Human Pose Estimation”. In: *Advances in Neural Information Processing Systems*.

Yang, Lihe et al. (2024). “Depth anything: Unleashing the power of large-scale unlabeled data”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10371–10381.