Snacks Clustering

Alaeddine Cheniour - Simon Ali Alexandre Khan
Spécialité Informatique et Ingénierie Mathématique

Vendredi 28 mars 2025

Polytech Paris-Saclay, Maison de l'ingénieur, bâtiment 620,Orsay 91400

Table des matières

Introduction

L'objectif de ce travail est de clusteriser des images de snacks grâce à de l'apprentissage non supervisé. Nos données sont au format jpg, codées en RGB, et nous savons qu'il y a 20 différentes classes de snacks. Nous avons essayé différentes approches pour classifier au mieux les images. Nous avons commencé avec des méthodes vues en cours et en TP puis nous avons fait nos propres recherches dans le but de mieux clusteriser les images. Dans ce rapport, nous recensons ces différentes approches. À chaque fois, notre méthodologie est la même :

- préparation des images
- feature engineering
- apprentissage du modèle
- évaluation du modèle

Dans un premier temps, nous expliquerons nos approches pour chaque modèle. Puis nous justifierons nos choix de métriques. Et enfin nous présenterons nos résultats. Notre code est basé sur l'architecture du code du TP3.

1 Explication des approches choisies (descripteurs, modèles)

A) Préparation des images

Dans un premier temps nous avons exploré le jeu de données à la main. Nous nous sommes mis d'accord pour utiliser les images de l'ensemble de validation : 955 images. Entraîner notre modèle avec un ensemble plus grand (comme l'ensemble d'entraînement contenant ≃5000 images) n'aurait pas d'impact sur les résultats car l'apprentissage est non supervisé. L'ajout de données sans labels ne garantit pas une meilleure performance.

A-1) Collecte et Redimensionnement

Nous lisons nos images avec le module C2V de python pour les stocker dynamiquement. Chaque image est dans un premier temps redimensionnée en une image de 224*224 pixels. Cette méthode permet d'homogénéiser les données. Il est bien plus simple de créer des espaces vectoriels cohérents à partir d'images qui ont le même nombre de pixels. Nous avons volontairement choisi une taille d'image raisonnable afin de réduire la complexité de nos algorithmes de clustering. Nous avons dans un premier temps choisi de redimensionner les images plutôt que de les croper pour préserver leur intégralité.

Nous stockons les images de l'ensemble de validation ainsi que les labels associés (utilisés uniquement pour mesurer la performance de nos modèles) dans deux tableaux numpy. Le tableau d'image est ensuite shuffle pour ne pas biaiser certains des modèles que nous avons utilisés.

A-2) Détection et Recadrage des Aliments

Dans les images, on remarque souvent la présence de certaines impuretés ou de divers objets non pertinents. Par exemple, certaines images montrent des personnes mangeant un snack, d'autres présentent des fruits accrochés à un arbre. Afin d'améliorer la qualité des images utilisées pour l'entraînement du modèle, nous avons expérimenté avec une approche de prétraitement qui consiste à détecter et recadrer automatiquement les aliments dans chaque image. L'objectif est d'éliminer les éléments non pertinents, tels que l'arrière-plan, les objets étrangers ou les erreurs de cadrage, et de centrer l'objet principal de l'image.

Pour cela, nous avons utilisé Faster R-CNN, conformément à la méthode proposée par Ren et al (2015) [1] avec un backbone ResNet-50 et un Feature Pyramid Network (FPN). Faster R-CNN (Region-based Convolutional Neural Network) est un modèle qui détecte les objets en deux étapes :

1. Région Proposals (RPN) : Le modèle génère des propositions de régions où des objets sont susceptibles d'être présents.
2. Classification et Affinage : Chaque région proposée est analysée pour identifier la classe de l'objet et ajuster sa position.

Étapes de l'algorithme :

1. Détection des objets : Le modèle Faster R-CNN pré-entraîné est appliqué pour détecter les objets dans l'image.
2. Filtrage des détections :
   - Seuil de confiance à 80% : Ce choix permet d'éliminer les fausses détections.
   - Filtrage des objets trop petits : Seuls les cadrages ayant une longueur et une largeur supérieures à 30 pixels sont retenus, afin d'éviter les détections parasites.
   - Filtrage basé sur le rapport hauteur/largeur : Les objets ayant un rapport hauteur/largeur trop élevé sont exclus, afin d'éviter de détecter des personnes par erreur.
   - Priorité aux objets proches du centre : Cela garantit un recadrage optimal et réduit le risque de découper des parties importantes de l'image.
3. Sélection de la meilleure détection : Parmi les boîtes restantes, la plus pertinente est choisie en fonction de la confiance du modèle et de la proximité avec le centre de l'image.

A-3) Réduction du Bruit par Floutage

Pour améliorer la qualité des images avant leur utilisation dans notre modèle, nous avons expérimenté une approche en appliquant un flou gaussien afin de réduire le bruit et les détails inutiles.

B) Feature engineering

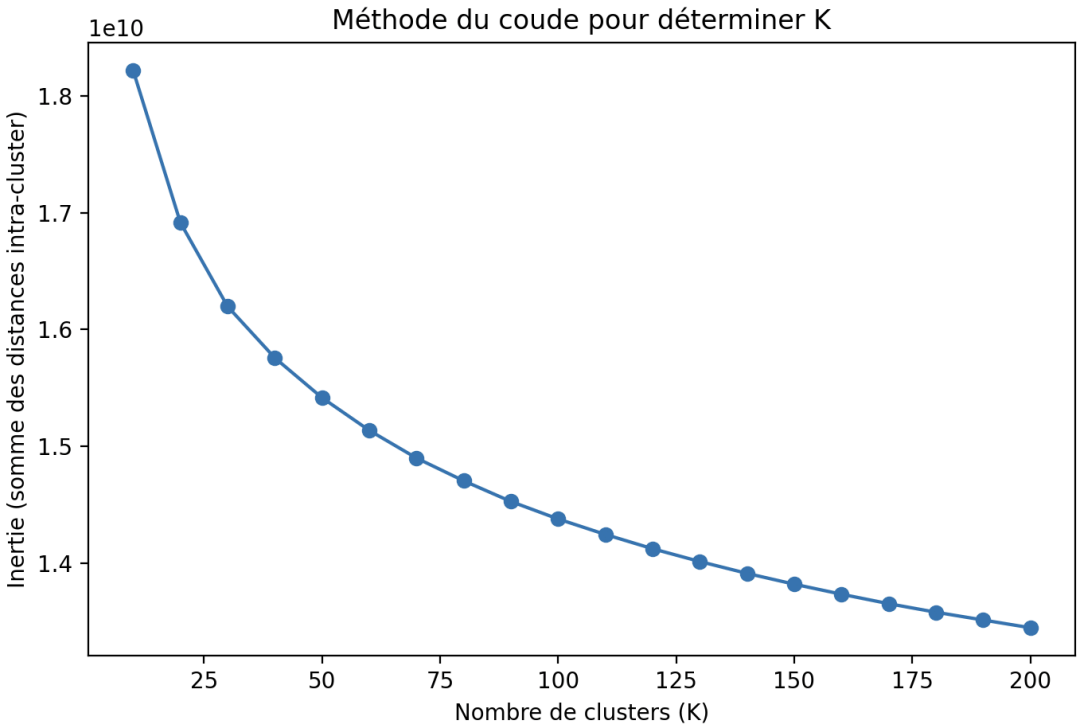
Nous exploitons plusieurs descripteurs de nos images afin d'entraîner nos modèles dessus. Ils sont listés ci-dessous :

HOG (Histogram of Oriented Gradients) : Extrait les gradients locaux de l'image pour capturer la forme et les contours. Chaque image est transformée en un vecteur décrivant les orientations de bord. Dalal et Triggs (2005) [2] ont montré que les Histogrammes de Gradients Orientés étaient très performants pour la détection d'objets. Nous l'avons choisi en pensant qu'il pourrait être utile pour différencier les structures visuelles caractéristiques des snacks.

Histogrammes de couleurs (RGB) : Pour chaque canal de couleur, un histogramme est calculé afin de représenter la distribution des couleurs dans l'image. Chaque image est transformée en un vecteur des trois histogrammes concaténés. Nous l'avons choisi en pensant que la couleur globale d'une image pourrait être utile pour différencier les snacks.

Moyenne SIFT (Scale-Invariant Feature Transform) : Détecte des points clés dans l'image, puis extrait des "descripteurs SIFT" propres à ces points clés. Chaque image a un nombre variable de "descripteurs SIFT". Un "descripteur SIFT" est un vecteur de taille 128. Pour que nos descripteurs finaux soient de la même dimension, chaque image est transformée en un vecteur qui est la moyenne de ses "descripteurs SIFT". Nous l'avons choisi car chaque descripteur final est invariant à un changement d'échelle uniforme, à la rotation et à l'illumination.[3]

Bag of Visual Words : Basé sur SIFT. Comme pour le descripteur précédent, on extrait de chaque image des points clés et les "descripteurs SIFT" propres à ces points clés. On applique un algorithme de clustering (ici KMeans) sur les "descripteurs SIFT" pour classer les descripteurs dans k clusters différents. On appelle ces clusters des mots visuels.[4] Ensuite, pour chaque "descripteur SIFT" de chaque image, on prédit à quel mot visuel il appartient. Pour chaque image, on compte combien de fois chaque mot visuel apparaît, ce qui nous donne un histogramme des fréquences. Chaque image est représentée par un histogramme de fréquence : un vecteur de dimension k. Nous l'avons choisi car il est plus fin que le descripteur précédent.
Nous avons d'abord choisi k = 169 car c'est le nombre moyen de "descripteurs SIFT" par image. Afin d'être plus rigoureux, nous avons ensuite choisi k en utilisant la méthode du coude.



On observe une diminution de la diminution de l'inertie vers k = 100. En fixant l'hyper paramètre k = 100, on observe en effet de meilleurs résultats (score AMI) qu'avec k = 169.

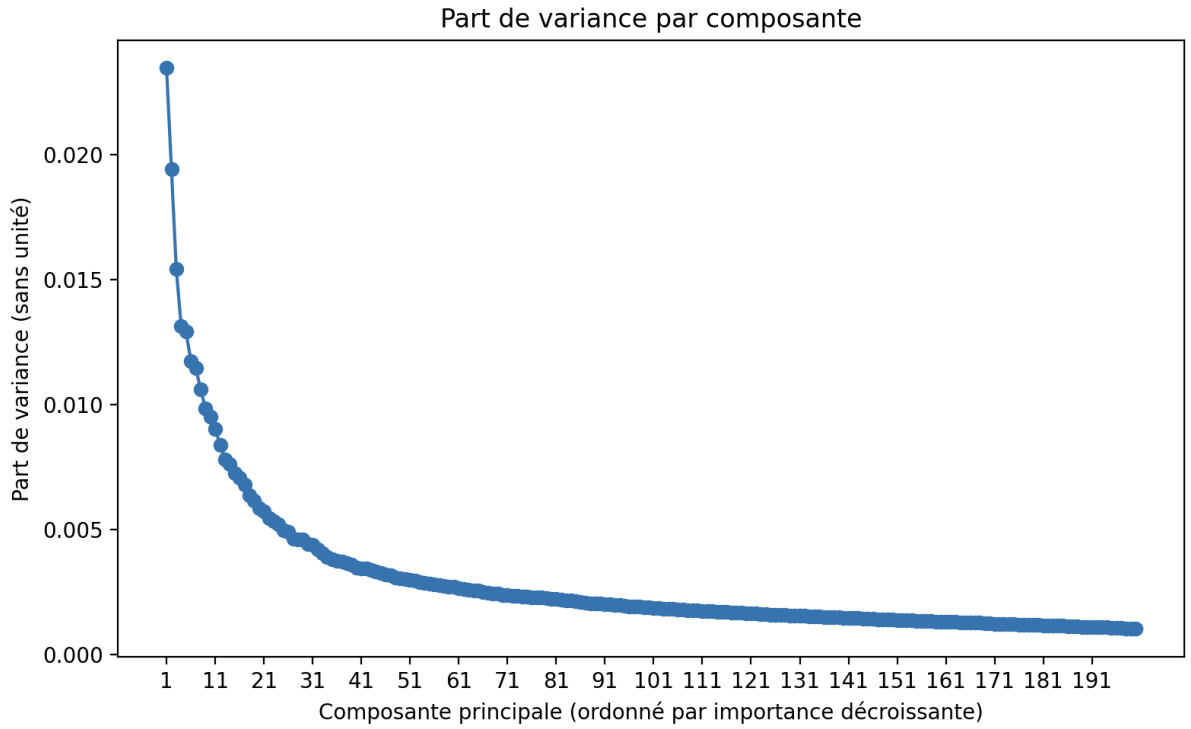
Weighted Combined Features :
Afin de pouvoir combiner plusieurs features pour notre modèle, nous avons implémenté une fonction dans laquelle nous pouvons paramétrer quelle feature choisir et quel pondération lui associer.

Comme vous pourrez le voir dans la partie 3, les descripteurs de machine learning classiques ne nous ont pas donné de bons résultats. Nous nous sommes tournés vers des extractions de descripteurs basées sur du deep learning. Nous avons dans un premier temps décidé d'extraire des représentations profondes de nos images grâce à des réseaux de neurones convolutifs pré-entraînés sur des millions d'images.

Encodage avec des deep features (ResNet50) :
Utiliser un réseau de neurone résiduel permet d'avoir un réseau avec une architecture profonde sans que cela ne risque une dégradation des performances. Nous avons choisi ResNet50 introduit par He *et al.* (2016) [5] car il est assez profond et que ResNet101 demandait trop de temps de calcul et de mémoire pour nos machines. ResNet étant un modèle de classification, nous excluons la dernière couche du modèle, qui correspond à la couche de classification. Chaque image est convertie en une feature map de dimension (4,4,2048). Pour convertir chaque feature à un vecteur de dimension réduite, nous avons utilisé deux méthodes.

- GlobalPooling : On fait la moyenne de chaque feature map sur ses 2 premières dimensions. Chaque image est convertie en un vecteur de taille 2048.
- Flatten + PCA : On flatten chaque feature map pour obtenir un vecteur de dimension 4*4*2048 = 32768. On applique l'analyse en composantes principales (PCA) pour transformer ce vecteur en un vecteur de taille n.

Nous remarquons de bien meilleurs résultats avec la deuxième méthode. Pour obtenir la valeur optimale de n, on applique la méthode du coude en plottant la part de variance en fonction de chaque composante. En effet, l'analyse en composantes principales réduit la dimension en conservant la variance. Les résultats sont sur la figure qui ci dessous :

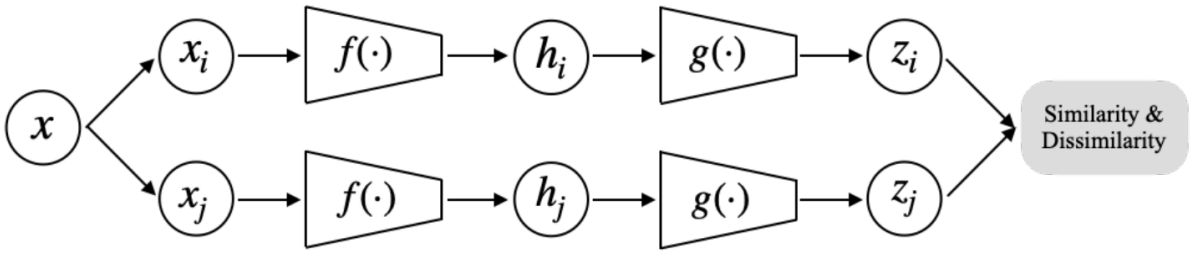


Lorsque la variance n'augmente plus significativement, l'ajout de nouvelles composantes devient inutile, ce qui indique un bon compromis entre réduction de dimension et conservation de l'information. Même si le graphique suggère un coude autour de 20 composantes, nous avons observé de meilleurs résultats de clustering en conservant 50 composantes principales. Ce choix permet de préserver davantage de variance (et donc d'information) dans les données extraites par ResNet50, tout en limitant la dimensionnalité par rapport au vecteur d'origine (32 768 dimensions).

Apprentissage Auto Supervisé :

Nous avons ensuite décidé d'explorer une autre méthode de feature engineering basée sur de l'apprentissage auto supervisé. Il existe 3 grandes catégories d'apprentissage auto-supervisé, ici nous nous penchons sur le "contrastive self supervised learning" et notamment le modèle SimCLR de Chen et al. (2020) [6] (simple framework for contrastive learning of visual representations).

Le schéma ci dessous et les explications qui suivent en résume les grandes étapes :



La principe est le suivant :

Notre jeu de données est divisé en des batchs de N images. Pour chaque image x on crée deux augmentations appelées “vue”, notées xi et xj. Une vue est une version transformée de l’image originale (recadrage, changement de couleur, rotation, flou, etc.). Chaque batch contient donc 2N vues.

Chaque vue est comparée aux 2N-2 autres vues (issues d’images différentes) de son batch dans le but de maximiser la distance entre elles (on les considère comme des paires négatives). Cela correspond à l’expression au dénominateur de la fonction coût ci dessous. En parallèle, chaque vue est également comparée à sa paire augmentée (la vue de la même image) pour minimiser la distance entre elles (paires positives). Cela correspond au numérateur de l’expression de la fonction coût ci-dessous. Cette fonction de coût appelée NT-Xent loss (*Normalized Temperature-scaled Cross Entropy*), basée sur la méthode du produit scalaire, pousse les représentations similaires à se rapprocher dans l’espace vectoriel.

$$\ell_{i,j} = -\log \frac{\exp\left(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau\right)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp\left(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau\right)} \quad \text{avec :} \quad \mathrm{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|},$$

z le descripteur de l’image x et ⍦ la température (hyperparamètre).

Le modèle utilise un encodeur pour créer une représentation vectorielle de chaque image notée h. Dans le schéma, l’encodeur est noté f(•), c’est souvent un réseau ResNet. Nous avons choisi ResNet50. Ensuite, les représentations vectorielles sont projetées dans un perceptron multi-couche, noté g(•), afin de réduire leur dimension. Les vecteurs réduits sont notés z. En effet, d’après les travaux de SimCLR, cette étape de projection améliore les performances de l’apprentissage contrastif. À chaque passe, les paramètres de f(•) et g(•) sont mis à jour pour minimiser la fonction coût. Cette mise à jour se fait par rétropropagation, comme en apprentissage supervisé, mais ici sans utiliser de labels explicites.

Une fois le modèle entraîné, on garde f(•). On peut maintenant appliquer la même pipeline que dans l’extraction de feature précédent, mais cette fois-ci avec un encodeur ré-entraîné par nos images.

Hyper paramètres :

- La taille du batch joue un rôle crucial, plus il est grand, plus il y aura de comparaisons de vues et plus le modèle pourra généraliser. En effet, on calcule les similarités entre tous les vecteurs du batch, soit une matrice (2N × 2N). Donc la complexité est quadratique en fonction de la taille du batch : doubler la taille du batch augmente d'un facteur 4 la complexité. Pour nos machines, une taille de batch convenable est N = 64. Cela n'est pas du tout optimisé : dans l'article de Chen et al, les batchs pour la même tâche sont de taille N = 256.
- Nous avons d'abord choisi un hyperparamètre τ= 0.07. Selon l'article original de SimCLR, c'est avec cette valeur que le modèle a obtenu les meilleurs résultats sur ImageNet. Cependant, notre jeu de données étant petit, nous n'excluons pas la possibilité d'augmenter τ.

Clustering avec CLIP

Pour extraire des informations utiles de nos images, nous avons utilisé CLIP (Contrastive Language-Image Pretraining), un modèle créé par OpenAI [7] qui relie les images et le texte de manière intelligente. CLIP a été entraîné sur une immense quantité d'images accompagnées de descriptions, ce qui lui permet de comprendre non seulement ce qu'il voit, mais aussi le contexte dans lequel se trouvent les objets. Contrairement aux méthodes classiques qui se basent uniquement sur des éléments visuels simples comme la couleur ou la forme, CLIP génère un vecteur numérique (embedding) qui représente l'image de façon plus complète en tenant compte de sa signification. Il fonctionne avec deux parties : un modèle de vision qui transforme l'image en un vecteur, et un modèle de langage qui fait de même avec du texte. Ces deux modèles sont entraînés ensemble avec une technique spéciale qui leur apprend à rapprocher les images et les descriptions qui correspondent et à éloigner celles qui ne vont pas ensemble. Grâce à cet entraînement, CLIP reconnaît des objets et leur contexte sans avoir besoin d'un entraînement spécifique sur notre dataset, ce qui nous permet de gagner du temps et d'éviter d'annoter manuellement nos données.

Dans notre projet, nous avons utilisé la version ViT-B/32. Ce modèle crée des vecteurs de taille raisonnable, ce qui accélère notre clustering, tout en capturant assez de détails pour bien différencier les types d'aliments.

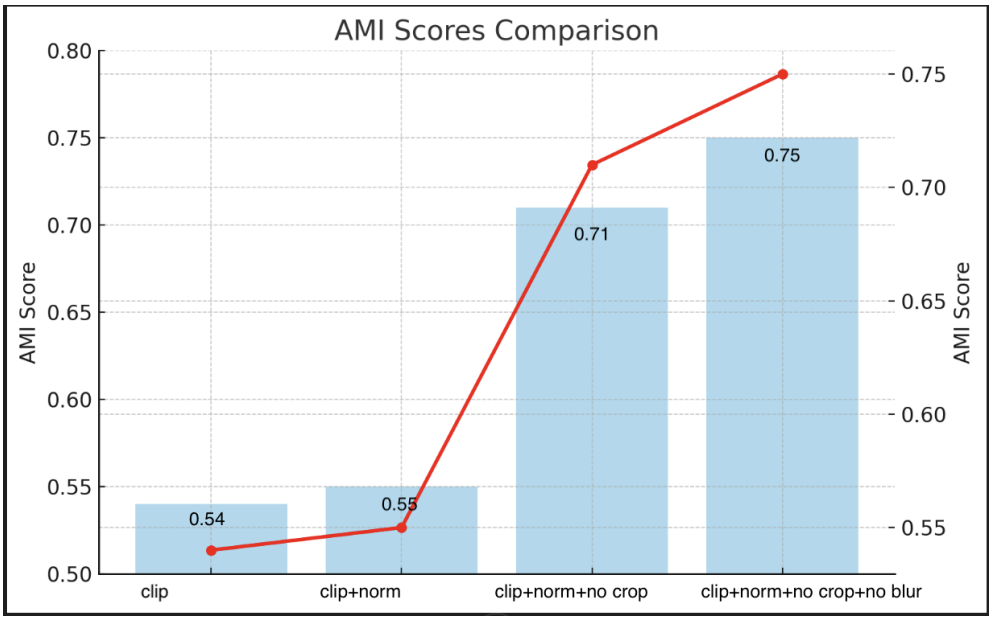
Effet de la Normalisation des Vecteurs CLIP

Nous avons étudié l'impact de la normalisation des vecteurs extraits par CLIP. Nos tests montrent une légère amélioration de l'AMI après normalisation.

Impact du Floutage et du Recadrage Intelligent

Nous avons testé le clustering avec et sans prétraitement des images, notamment le floutage (Gaussian Blur) et le recadrage intelligent (Faster R-CNN). Contrairement à nos attentes, ces transformations ont dégradé les performances du clustering. En effet, CLIP fonctionne mieux lorsqu'il analyse des images complètes, avec leur contexte, plutôt que des images trop modifiées. Le floutage supprime des détails importants, et le recadrage peut

retirer des éléments qui aident CLIP à bien comprendre l'image. Adjusted Mutual Information (AMI) est plus faible après ces prétraitements, montrant que le clustering est moins précis.



Évolution du score AMI en fonction de la préparation des données.

Au cours des exécutions de clustering pour ces comparaisons, HDBSCAN a été utilisé avec les paramètres suivants : `min_cluster_size=15` et `min samples=2`.

```
clusterer = HDBSCAN(min_cluster_size=15, min_samples=2, metric="euclidean")
```

Réduction de Dimension : PCA vs UMAP

Les embeddings CLIP ont une grande dimension (512 pour ViT-B/32), ce qui peut compliquer le clustering. Nous avons testé la réduction de dimension avec PCA (Analyse en Composantes Principales) et UMAP (Uniform Manifold Approximation and Projection) [8]. Les résultats montrent que réduire la dimension à 50 ou 100 améliore nettement les performances du clustering. PCA est plus rapide et donne des résultats corrects, mais UMAP offre des clusters plus nets et plus séparés.

Le graphique montre que UMAP surpasse systématiquement PCA sur toutes les métriques (ARI, AMI, Silhouette) et dans toutes les dimensions testées (3, 10, 20, 50, 100).

Performances optimales de UMAP

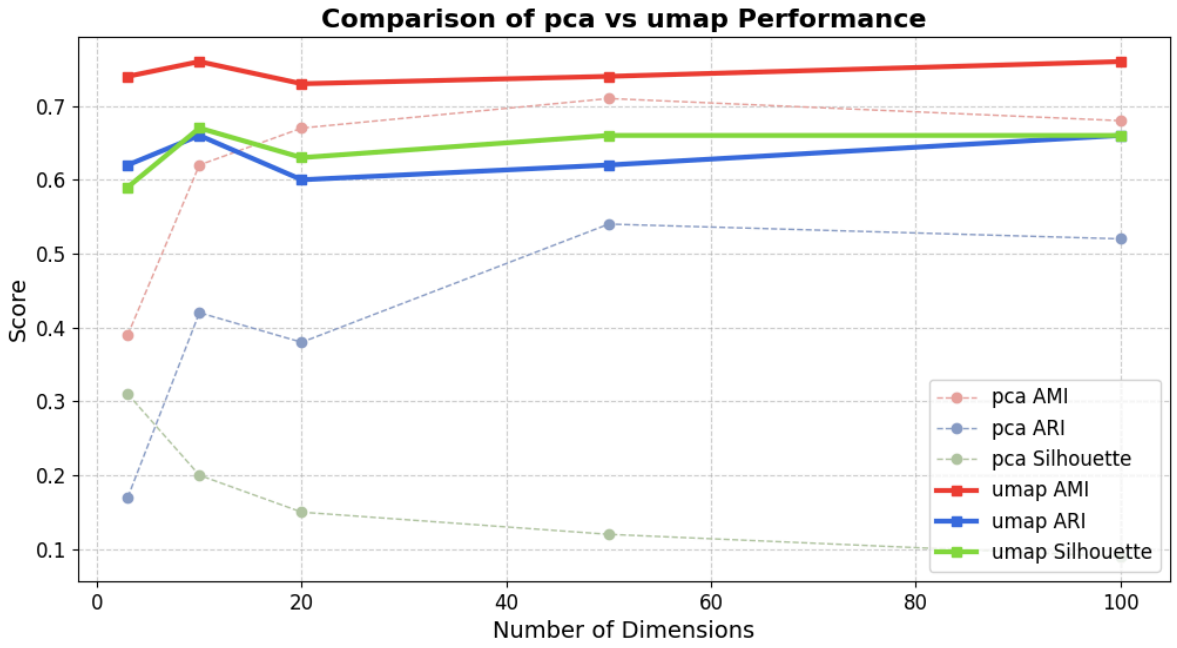
-Dimension 10 (pic de performance) :

{ARI : 0.66 , AMI : 0.76 *(meilleur score global)* , Silhouette : 0.67 *(meilleur score global)}*

-Dimension 100

{ARI : 0.66 *(meilleur score global)* , AMI : 0.76, Silhouette : 0.66*}*

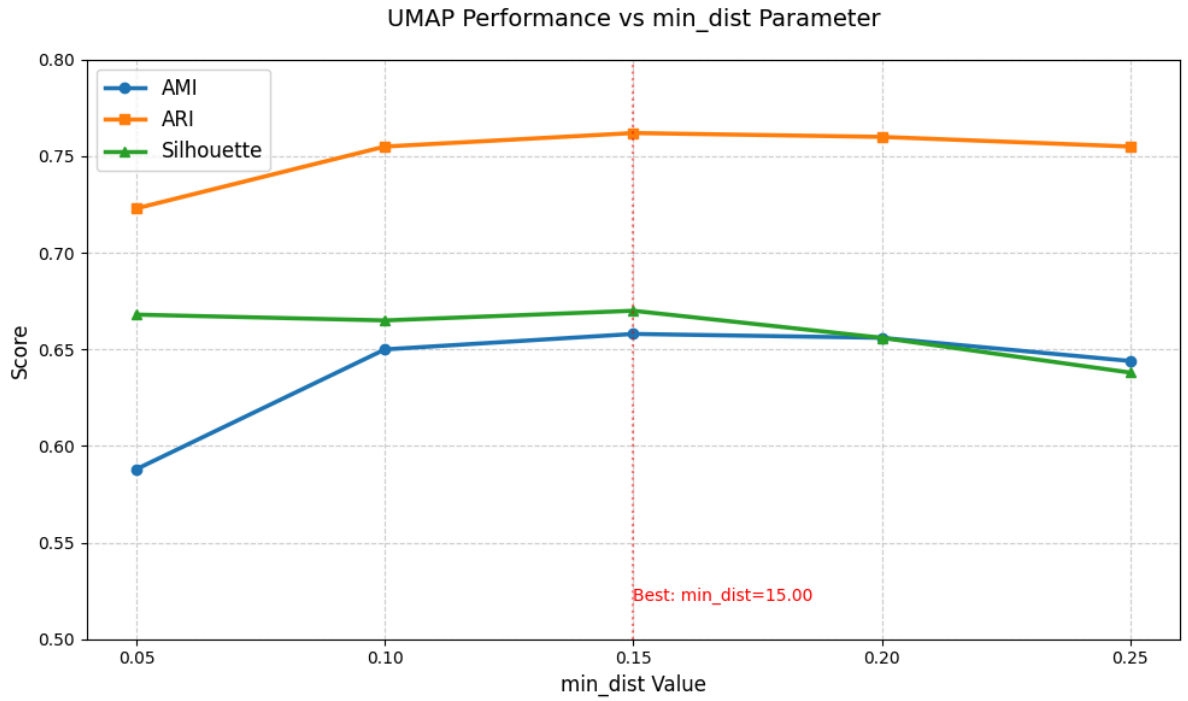
Une dimension plus basse (10 vs 100) réduit la complexité du modèle, accélère les calculs, et évite le bruit inutile présent dans les espaces de grande dimension. De plus, les performances de UMAP se stabilisent dès la dimension 10, avec des gains marginaux (voire une décroissance) pour des dimensions supérieures. Cela indique que 10 dimensions capturent déjà l'essentiel de la structure des données.



Pour UMAP, on a utilisé trois paramètres clés pour la réduction de dimension :

- min_dist=0.1 (défaut) détermine l'espacement entre clusters. On a choisi la valeur par défaut car nos clusters sont naturellement séparés.
- metric='cosine' (plutôt que la distance euclidienne par défaut) qui est mieux adaptée aux images.

-n_neighbors=15 (défaut) selon la figure ci-dessus, la valeur optimale est 15.



C Choix du modèle de clustering

Après avoir essayé plusieurs modèles, nous avons choisi de garder deux modèles complémentaires : K-Means++ et HDBSCAN

Comparaison entre K-Means++ et HDBSCAN

- K-Means++ est une variante améliorée de K-Means, qui choisit intelligemment les centres initiaux afin d'éviter une mauvaise convergence [9]. Cet algorithme fonctionne bien lorsque :
    - Le nombre de clusters est connu à l'avance,
    - Les clusters ont une forme convexe, taille similaire et sont bien séparés.

  Dans notre cas, K-Means++ s'est révélé très efficace sur la plupart de nos descripteurs. Toutefois, sa performance chute lorsque les clusters ont des formes irrégulières, des densités variables ou qu'ils se chevauchent légèrement. De plus, comme K-Means++ attribue toujours chaque point à un cluster, il peut forcer certains éléments mal positionnés à rejoindre un cluster inapproprié, ce qui nuit à la cohérence globale.

- HDBSCAN, qui détecte automatiquement le nombre de clusters et gère mieux les formes irrégulières, donne de meilleurs résultats sur notre dataset [10]. Il permet notamment d'identifier des groupes plus naturels sans imposer un nombre fixe de clusters. Pour les points non classés (qui appartiennent au cluster -1) on a fait le choix de le rejoindre au cluster le plus proche car ça nous donne de meilleurs métriques et dans la plupart de temps ça ne contredit pas le sens de cluster en question.

```
def reassign_noise_points(features, labels):
      non_noise = labels != -1
      knn = KNeighborsClassifier(n_neighbors=1)
      knn.fit(features[non_noise], labels[non_noise])
      for i, label in enumerate(labels):
            if label == -1:  # If it's a noise point
                  labels[i] = knn.predict([features[i]])[0]  # Assign
return labels
```

Pour HDBSCAN, nous avons sélectionné les paramètres min_cluster_size dans l'intervalle [12, 15] et min_samples = 2. Cette configuration nous donne entre 20 et 22 clusters, ce qui est bien avec notre cas.

```
clusterer = HDBSCAN(min_cluster_size=15, min_samples=2, metric="euclidean")
```

Pour notre clustering. Les algorithmes KMeans et HDBSCAN démontrent des performances très proches. KMeans obtient des scores légèrement supérieurs sur : L'AMI et L'accuracy. Par contre, HDBSCAN présente une meilleure homogénéité des clusters et possède des regroupements plus cohérents avec notre observation quand on redistribue les points non tenu en compte (points de cluster -1).

résultats et explication: voire Partie (3)

Combinaison avec d'autres features

En combinant CLIP avec d'autres features. Les performances dégradent peu importe les poids des features introduites.

2 Explication des choix de métriques

Le clustering étant une tâche non supervisée, il n'existe pas de métrique unique et universelle pour évaluer la qualité des résultats. Nous avons principalement utilisé l'AMI (Adjusted Mutual Information) et le Silhouette Score pour évaluer la qualité du clustering. L'AMI nous donne une mesure objective de la correspondance entre les clusters et les vraies classes, tandis que le Silhouette Score nous informe sur la cohésion et la séparation des clusters dans l'espace de représentation.
Ces deux métriques sont complémentaires : l'une dépend des labels, l'autre non. Malgré tout, nous avons également consulté des métriques secondaires et analysé les clusters de manière qualitative sur StreamLit.

3 Evaluations et explications des résultats

Résultats pour les modèles avec extraction de features basée sur du machine learning :

Extractions de Features	Clustering	AMI	Silouhette
HOG	Kmeans ++	0.03	0.25
Histogrammes de couleurs (RGB) :	Kmeans ++	0.08	0.37
Moyenne SIFT (Scale-Invariant Feature Transform)	Kmeans ++	0.07	0.35
Bag of words	Kmeans ++	0.07	0.3
Weighted Combined *{Hist:1, M_Sift:0.8, BOW:0.8, Hog: 0.1}*	Kmeans ++	0.11	0.31

-HOG donne les pires résultats et Histogrammes et SIFT moyen performent mieux, mais l'AMI reste faible.

-La combinaison pondérée améliore légèrement l'AMI (0.11) mais elle n'apporte pas de gain significatif.

=>Ces méthodes ne capturent pas la sémantique des images. Elles sont très sensibles aux variations et aux bruits introduits dans le contenus (lumière, rotation, etc). De ce fait, leur combinaison naïve n'ajoute pas d'information utile. Elle moyenne juste les performances.

Résultats pour les modèles avec extraction de features basée sur du deep learning :

Résultats pour le modèle Deep features (ResNet) + KMeans ++ :

AMI	Silhouette
0.3	0.37

Ces résultats sont nettement meilleurs que les résultats précédents.

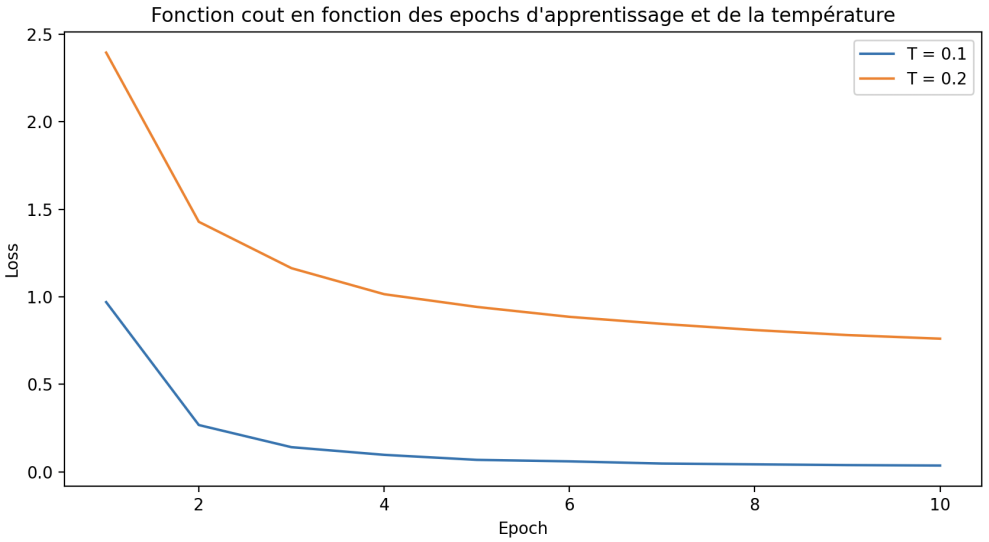
Cela s'explique par la capacité du modèle à extraire des features grâce à son entraînement sur un large ensemble d'images (ImageNet). Cela permet une meilleure séparation des clusters (AMI = 0.30 contre 0.03–0.11 pour les autres méthodes) et une plus grande cohérence interne (silhouette = 0.37).

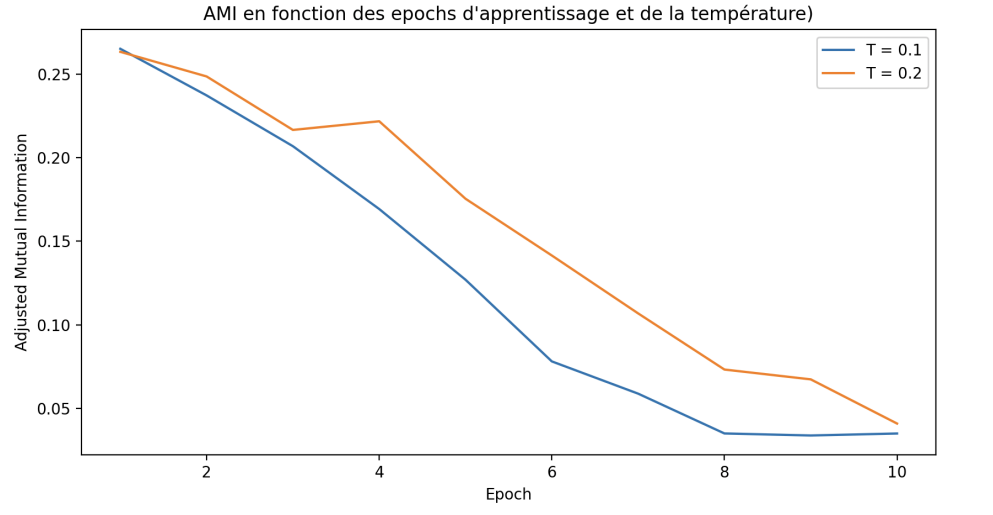
Résultats pour le modèle Apprentissage Auto Supervisé + KMeans ++ :

Comme vous pouvez le voir sur le tableau ci -dessous, les résultats pour ce modèle ne sont pas très bons.
Hyperparamètres : τ = 0.2 et N = 64.

AMI	Silhouette
0.05	0.25

Cela peut s'expliquer par du sur apprentissage. Les graphes ci-dessous en témoignent.





En observant ces graphes, on peut faire le constat d'un sur-apprentissage. La loss décroît très rapidement, mais la précision du clustering diminue également, alors qu'elle devrait augmenter au fur et à mesure que le modèle apprend. Le modèle triche en résolvant la tâche de rapprochement des paires positives et éloignement des paires négatives de manière opportuniste. Cela peut s'expliquer de plusieurs façons :

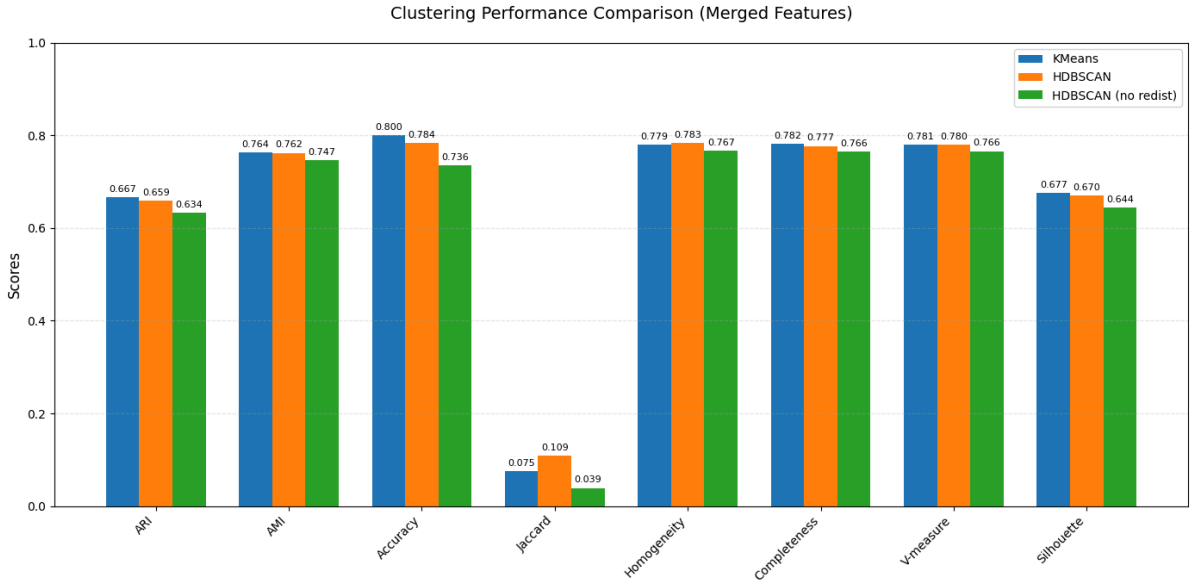
- le batch est trop petit, il n'y a pas assez de négatif pour entraîner le modèle.
- le dataset est trop petit, il n'y a pas assez de diversité pour que le modèle apprenne suffisamment.

Résultats pour le modèle Deep features (CLIP) + :

Modèle	AMI	Silhouette
Kmeans + +	0.764	0.677
HDBSCAN + (redistribution)	0.762	0.670

- Un score de 0.76 en AMI est considéré élevé, ce qui indique que les clusters obtenus sont bien séparés et cohérents entre les deux méthodes. Cela suggère que les deux algorithmes trouvent des structures de groupes similaires malgré leurs différences (K-means est centré sur les distances, HDBSCAN sur la densité).
- Un score de 0.67 en silhouette est très bon. Cela indique que les clusters sont bien définis et séparés et les points à l'intérieur d'un cluster sont proches les uns des autres.

Le fait que les deux méthodes donnent des résultats similaires (AMI élevé) suggère que les clusters dans à partir des données CLIP sont bien structurés. Aussi, cela suggère que CLIP fournit une bonne séparabilité des données.

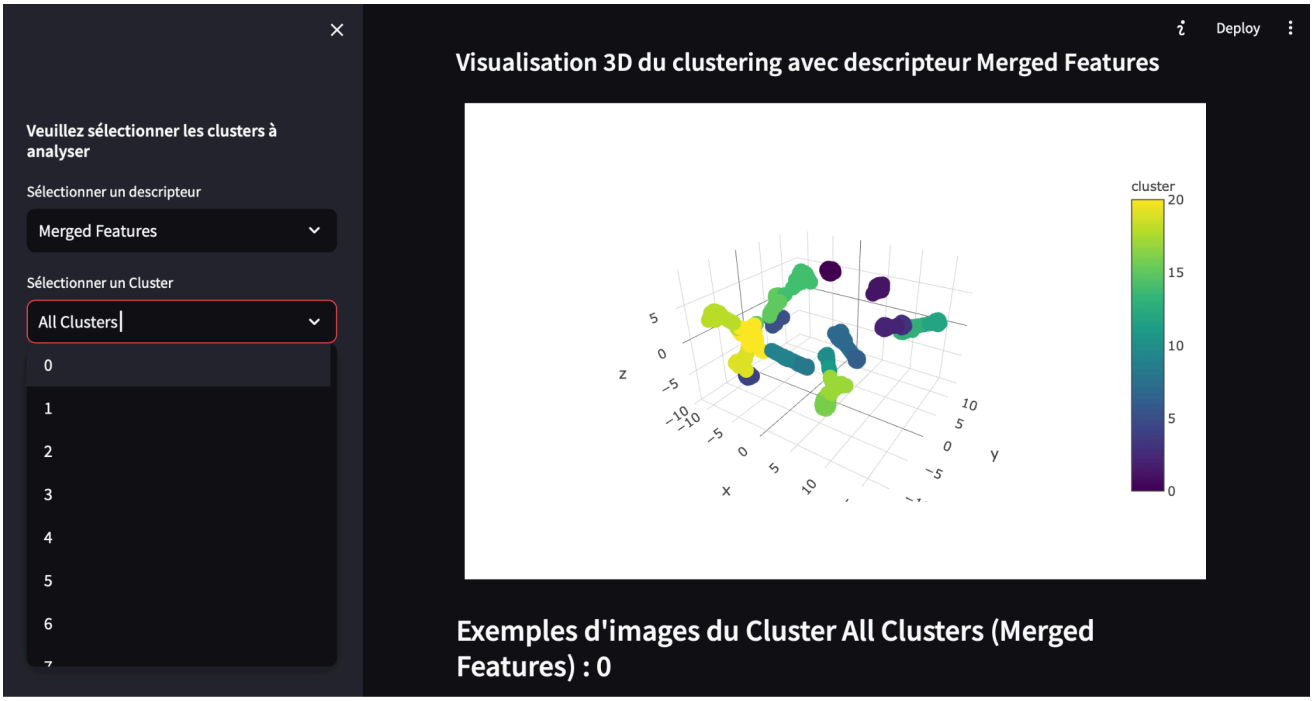


4 Visualisation des données

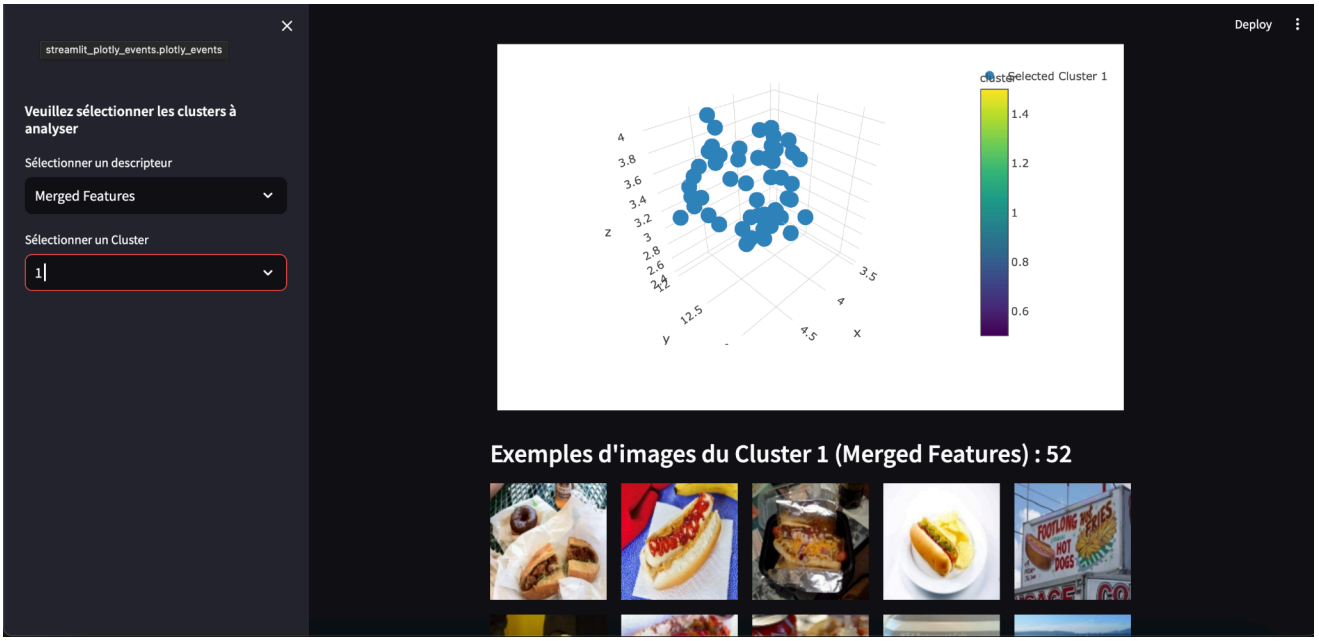
Dans notre visualisation,

- On peut visualiser chaque cluster indépendamment (avec ses images).

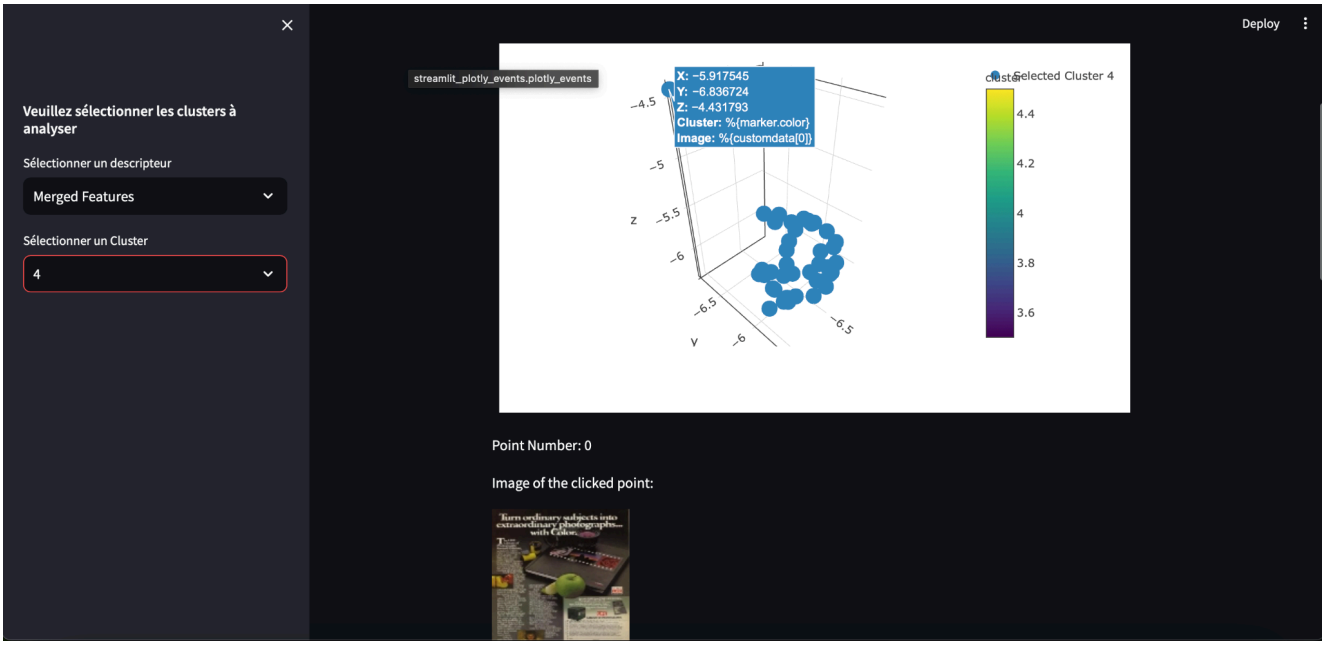
- Quand on clique sur un point. On récupère l' image représentée par ce point.



Vue d'ensemble de tous les clusters



Un cluster accompagné par ses images



Quand on clique sur un point, on récupère l'image correspondante

Conclusion (et pistes d'amélioration)

Ce projet nous a permis d'explorer en profondeur les différentes approches de clustering d'images dans un cadre non supervisé, en mettant particulièrement l'accent sur le feature engineering, la préparation des données, et l'impact des techniques avancées issues du deep learning. La partie la plus importante de ce projet n'était pas celle que l'on pensait. La préparation des images et le feature engineering ont un impact majeur sur nos résultats. Le choix du modèle de clustering quant à lui n'a eu qu'un léger impact.

Nos résultats montrent que les méthodes classiques de descripteurs (HOG, histogrammes de couleurs, SIFT, Bag of Words) sont limitées dans notre cas, malgré les tentatives de combinaison pondérées.

À l'inverse, l'utilisation de représentations profondes issues de réseaux convolutifs pré-entraînés (ResNet50) et du modèle CLIP a vraiment amélioré la qualité du clustering. CLIP, en particulier, se distingue en capturant à la fois les éléments visuels et le contexte sémantique de l'image, ce qui permet de générer des représentations robustes et bien séparables. Les meilleurs scores ont été obtenus avec CLIP + KMeans++ ou HDBSCAN, atteignant un AMI de 0.76 et un Silhouette score de 0.67.

Contrairement à nos attentes initiales, certaines techniques de nettoyage (floutage, recadrage par détection) ont parfois dégradé les performances, en particulier avec CLIP qui tire profit du contexte global de l'image. Les méthodes de réduction de dimension ont eu un impact significatif sur les performances, UMAP surpassant systématiquement PCA, en particulier dans des dimensions faibles.

Enfin, notre expérimentation sur l'apprentissage auto-supervisé avec SimCLR a montré des résultats mitigés, nous supposons que ces méthodes nécessitent des ressources importantes (taille de batch, taille de dataset, capacité du modèle) pour exprimer pleinement leur potentiel.

Pistes d'amélioration :

- Améliorer les méthodes de recadrage intelligent en s'appuyant sur des modèles spécialisés dans la détection alimentaire.
- Augmenter la taille du dataset et la diversité des images pour permettre un apprentissage auto-supervisé plus efficace.
- Explorer d'autres méthodes de clustering basées sur la structure du graphe ou des approches de clustering spectral.

Bibliographie

[1] Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* arxiv.org . *(Proceedings of NIPS 2015).*

[2] Dalal, N. & Triggs, B. (2005). *Histograms of Oriented Gradients for Human Detection* lear.inrialpes.fr *(Proc. CVPR 2005).*

[3] Lowe, D.G. (2004). *Distinctive Image Features from Scale-Invariant Keypoints* cs.ubc.ca *Int. J. Computer Vision, 60(2):91–110.*

[4] Csurka, G., Dance, C.R., Fan, L., Willamowski, J., & Bray, C. (2004). *Visual Categorization with Bags of Keypoints. (Workshop on Statistical Learning in Comp. Vision, ECCV 2004).* cs.cmu.edu

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition* arxiv.org *(Proc. CVPR 2016).*

[6] Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). *A Simple Framework for Contrastive Learning of Visual Representations (SimCLR)* arxiv.org *(Proc. ICML 2020).*

[7] https://openai.com/index/clip/

[8]McInnes, L., Healy, J., & Melville, J. (2018). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction* arxiv.org *(ArXiv preprint 1802.03426).*

[9]Arthur, D. & Vassilvitskii, S. (2007). *k-means++: The Advantages of Careful Seeding* dl.acm.org *(Proc. ACM-SIAM SODA 2007, pp. 1027–1035).*

[10] Campello, R.J.G.B., Moulavi, D., Zimek, A., & Sander, J. (2015). *Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection* theoj.org *ACM TKDD, 10(1), Article 5.*