POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Travail pratique I

Description et comparaison de régions d'intérêt

Élèves :
Slimane Boussafeur
Noémie Kpatenon

Enseignants :
G.A. Bilodeau
Khalil Sabri

7 février 2025

1 Présentation des approches

Dans le cadre de ce premier travail pratique, il nous a été demandé de comparer deux techniques de description d'image en évaluant leurs performances dans des situations et configurations spécifiques.

Pour ce faire, nous avons implémenté les deux méthodes suivantes :

- HOG (Histogrammes des orientation du gradient)
- CLIP (Pré-apprentissage contrastif de texte et d'image)

1.1 HOG

Comme son nom l'indique, HOG est une méthode où l'on calcule des histogrammes qui compilent les différentes orientations du gradient de la région d'intérêt d'une image.

En gros, on subdivise notre région en cellules dont on calcule l'orientation du gradient pour chacune d'entre elles grâce à des opérations de convolution. On comptabilise, ensuite, ces orientations dans un histogramme (1 par cellule) qu'on divise en intervalles (classes d'angles). Ces cellules forment, en réalité, la base de blocs qui se chevauchent sur l'image. Ainsi, on agrège les histogrammes des cellules pour obtenir les histogrammes des blocs. Par la suite, l'ensemble des histogrammes de blocs sont mis dans un vecteur pour obtenir le descripteur de l'image (voir figure 1) [1].

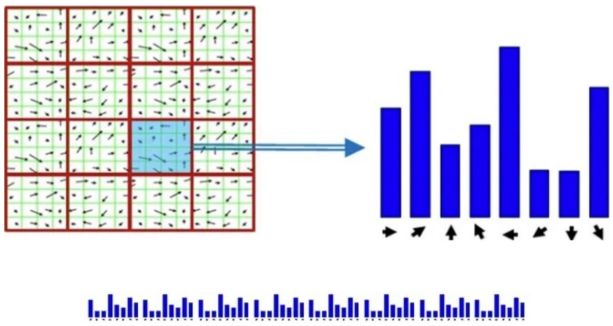


FIGURE 1 – Résumé de la méthode HOG [2]

1.2 CLIP

Concernant CLIP, c'est assez différent de l'approche précédente. En plus de générer un descripteur d'image, un descripteur textuel est également créé.

Concrètement, ce modèle créé par Open IA consiste en un encodeur d'image et un encodeur textuel qui sont pré-entraînés de manière conjointe. Chacun des encodeurs génère des descripteurs et ceux-ci sont comparés les uns aux autres en utilisant la similarité du cosinus pour pouvoir maximiser les ressemblances (les cases bleues de la figure 2) et

minimiser les différences (les cases grises de la figure 2) entre les concepts textuels et visuels [1][3].

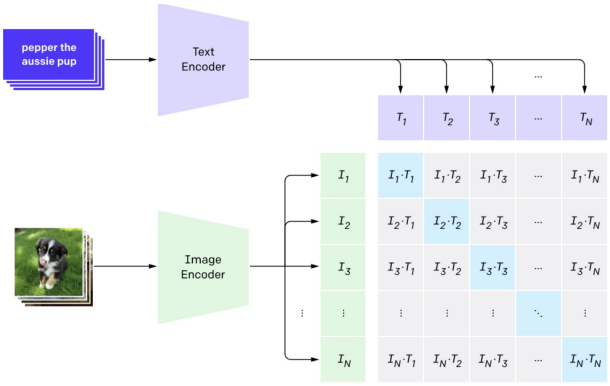


FIGURE 2 – Phase de pré-entraînement CLIP [4]

Suite à cela, on peut envoyer une description dans l'encodeur textuel et avec le descripteur généré, le faire correspondre à l'image la plus pertinente grâce à l'entraînement fait au préalable. L'inverse est aussi vrai. Ainsi, CLIP crée un espace où les concepts similaires sont rapprochés [1][3].

2 Hypothèses de performance dans les cas spécifiques

Premièrement, en termes de **vitesse d'exécution**, nous pensons que HOG prendra le dessus de par la nature de la méthode. En effet, HOG, en lui-même, n'est que la partie d'*embedding*, contrairement à CLIP qui lui est beaucoup plus complexe. De plus, il n'y a aucun entraînement nécessaire dans le cas de HOG. Ainsi, **HOG sera plus rapide**, car l'étape de classification se fait indépendamment en comparant les vecteurs de description [1].

Deuxièmement, en ce qui concerne le taux de succès pour la **reconnaissance d'images en général**, nous avons la conviction que CLIP gagnera haut la main. En plus d'avoir un descripteur d'images, il y a aussi la présence d'un descripteur textuel. Ces derniers ont tous deux été très bien entraînés en essayant de maximiser les ressemblances comme mentionné précédemment. Ainsi, **CLIP devrait être bien meilleur** en ce qui a trait à la reconnaissance en général vu la robustesse du modèle, sachant que les données comportent plusieurs problèmes (voir section 4) [1].

Troisièmement, on peut affirmer que **HOG devrait être supérieur pour ce qui de la reconnaissance de texture**. En effet, ce descripteur de texture cherche à trouver des gradients, c'est-à-dire des zones de fort contraste. Ainsi, si on a deux images, et chacune comportant un objet distinct mais partageant la même texture, alors on devrait voir des similarités au niveau des descripteurs. Contrairement à HOG, si on envoie, par exemple, un sac et une chemise avec la même texture à CLIP, ce dernier ne devrait pas y retrouver

de similitudes. De ce fait, pour de l'analyse de texture, HOG devrait être meilleur [1].

Finalement, CLIP devrait largement surclasser HOG concernant **les images ayant subi des rotations**. En effet, nous avons vu que HOG traite l'image en la divisant en blocs. Ainsi, si nous avons 2 images identiques A et B mais que l'image B s'est fait rotate de 45 degrés, alors les deux vecteurs d'histogrammes seront très différents. **Pour ce qui est de CLIP, on pense qu'il sera plus robuste aux rotations** dû à l'entraînement des encodeurs [1].

3 Hypothèses de performance sur les boîtes englobantes

Pour HOG, la présence de boîtes devrait aider à **augmenter la précision** des vecteurs de descriptions en termes de représentation puisque les blocs de l'image seront uniquement sur la région d'intérêt plutôt que sur l'image entière. Ainsi, on réduit la présence de bruit et la quantité d'informations visuelles inutiles.**Concernant CLIP**, on devrait probablement **voir des améliorations, mais dans une moindre mesure**, car c'est un modèle entraîné de manière extensive.

4 Description des expériences, des données et des critères d'évaluation

4.1 Expériences

En premier lieu, nous allons tester **la vitesse d'exécution**. La méthode d'évaluation est assez directe puisque nous allons chronométrer la vitesse d'exécution des modèles sans prendre en compte les étapes préliminaires (le prétraitement des images pour HOG et pour CLIP). Nous pourrons voir la rapidité des techniques en augmentant graduellement la quantité d'images à traiter : 1, 5, 10, 25 et 48 (le nombre d'images requêtes total).

Ensuite, pour ce qui est de **la reconnaissance d'image**, on va simplement faire rouler les deux modèles et voir à quel point ils arrivent à bien identifier les images requêtes. Nous allons comparer le descripteur de chaque image requête au descripteur de chacune des images dans la base de données et établir un classement basé sur la similitude des descripteurs. Nous pourrons ainsi voir à quel point ils arrivent à bien ordonner les images d'une même classe d'objet.

Dans un même ordre d'idées, nous allons nous pencher sur **la reconnaissance de texture**. Ici, la méthode d'évaluation est assez simple. Nous allons prendre une image de chemise contenant une texture quadrillée et l'évaluer face à deux catégories d'images (voir figure 3) :

- 3 images de chemise ayant des textures différentes

- 3 images d'objet qui ne sont pas des chemises (couverture, chapeau, sac) ayant la même texture quadrillée.

FIGURE 3 – Images utilisées pour le test de reconnaissance de texture [5]

On va chercher à voir si le modèle a plus tendance à aller trouver des similarités en se basant sur la texture ou bien sur l'objet en lui-même. Nous allons tout bonnement comparer les descripteurs des images et établir un classement de similarité.

Par la suite, les deux méthodes seront testées quant à leur **robustesse face à la rotation**. Dans un premier temps, nous allons analyser les résultats provenant du test de l'hypothèse sur le taux de réussite au niveau de la reconnaissance d'image pour voir comment chaque modèle a réagi face aux images de dauphins ayant déjà subi des rotations. Dans un second temps, nous allons prendre une image de visage dans la base de données et on va lui faire subir des rotations de : 0, 45, 90 et 180 degrés. Nous allons ensuite faire passer ces images dans nos modèles et voir l'effet des rotations au niveau des similitudes. Nous allons comparer l'image requête à nos images rotates et, encore une fois, obtenir les scores de similarité des distances .

Pour finir, **l'influence de la présence de boîtes englobantes** sera testée recadrant les zones d'intérêts des images de ballons (requête et base de données) et les distances entre les descripteurs seront calculées. Ainsi, nous pourrons les comparer aux distances obtenues par les images de ballons sans recadrage (celles calculées par le test de l'hypothèse sur le taux de succès). Les images de ballons ont été choisies car elles comportent des arrières-plans et du bruit pour certaines. Elles sont donc idéales pour un avant-après.

4.2 Données

Il est vrai que certaines images requêtes comportent quelques problèmes qui pourraient avoir une influence sur les résultats de nos expériences.

Voici une liste des problèmes présents :

- Les images ne sont pas toutes de même dimensions.
- Certaines images comme *ball_2.jpg* présentent des occlusions.
- Certaines images de dauphins on subi des rotations dans le plan de l'image.
- Une portion des images de voitures ainsi que des images de dauphins sont de faible contraste.
- Les images d'une même catégorie ne sont pas uniformes en ce qui concerne la luminosité (image prise de jour ou de nuit).
- On retrouve des images bruitées comme *dolphin_4.jpg*.
- Des images de lotus et de pots à cornichons ont subi des rotations hors du plan de l'image, ce qui fait qu'on a plusieurs angles du même objet.
- Les images ne sont pas toutes à la même distance de l'objectif.

4.3 Critères d'évaluation

En ce qui concerne les critères d'évaluations, quasiment identiques d'une expérience à une autre excepté pour la vitesse d'exécution. Pour cette dernière, c'est juste une question de **mesurer le temps** en augmentant graduellement le nombre d'images qu'on envoie au modèle. Dans ce cas, on enregistre le nombre de secondes pour avoir une idée de la rapidité par paliers.

Pour le reste des hypothèses, on en revient toujours au même principe, car indépendamment des cas, on recherche toujours à analyser les résultats de la comparaison des descripteurs d'une manière ou d'une autre. Pour ce qui est du **taux de succès général**, on regarde le classement final pour chaque image requête pour voir combien d'images de la même classe se retrouvent dans le **top 10** et le **top 5**. De plus, on va extraire l**a distance moyenne calculée** pour chacune des images d'une même classe (ex : dans le classement pour cat_query.jpg on a, en moyenne, un score de 0.3 pour les photos de chats). Concernant **les rotations**, les distances calculées nous permettent de déterminer **l'impact des différentes rotations** sur la similarité en comparaison toujours avec l'image de base inchangée. Idem pour le test en lien avec **les boîtes englobantes**, on évalue l'impact de la présence de ces dernières sur les distances calculées. En revanche, pour les résultats des **tests sur les images de textures**, nous allons nous rabattre sur **le classement** des scores de similarités pour voir quel modèle avantage quelle catégorie (même objet ou même texture).

Concernant la méthode utilisée pour **le calcul des distances** entre les descripteurs, nous avons opté pour la similarité du cosinus. L'*embedding* de nos images produit des vecteurs de description contenant des valeurs entre 0 et 1, ce qui réduit l'effet de la norme. Ainsi, la similarité du cosinus devient un outil assez efficace [6].

5 Description des implémentations utilisées

5.1 Implémentation de HOG

Bien évidemment, nous n'avons pas codé HOG à la main puisqu'on a profité de la fonction proposée par ***sickit-image*** nommée tout simplement ***hog***. Nous avons également utilisé d'autres petites fonctions de la même librairie telles que ***resize***, ***rotate***, ***imread*** et ***rgb2gray***. Pour le reste des fonctions externes, on a ***glob***, le module ***spatial*** de ***scipy*** et ***time***. Pour le reste, voici **les méthodes implémentés par notre équipe** pour l'accomplissement de tous nos tests :

- **pre_treatment (img)** : cette fonction **prend en paramètre une image**, change les dimensions de celle-ci avec ***resize*** et la transforme en tons de gris avec ***rgb2gray*** si nécessaire. La fonction **retourne l'image post-traitement**.
- **hog_wrapper(treated_img)** : cette fonction est un *wrapper* de la fonction *hog* provenant de *sickit-image* qui **prend en paramètre une image ayant déjà subi un traitement**. Elle applique le modèle HOG et **retourne le descripteur de l'image**.
- **get_embedding(img)** : cette fonction **prend en paramètre une image** et appelle les fonctions *pre_treatment* et *hog_wrapper* pour obtenir **le descripteur** de cette même image qui est ensuite **retourné par notre fonction**.
- **embed_image_list(path_list, image_list)** : cette fonction **prend en paramètres une liste de chemins de fichiers** et **une liste vide** qui a pour but de contenir les descripteurs de toutes les images. La méthode lit les images à partir de la liste de chemins avec *imread* et elles sont ensuite envoyées dans *get_embedding* pour obtenir les descripteurs. Par la suite, le descripteur et le nom du fichier (extrait à partir du chemin) sont associés sous forme de *tuple* et ce même *tuple* est ajouté à notre paramètre image_list. **Cette fonction ne retourne rien**.
- **get_scores(metric, query_list, db_list)** : cette fonction **prend en paramètres un *string*** représentant la méthode de calcul de distance (ex : similarité du cosinus) et **deux listes** contenant respectivement des *tuples* descripteur-nom de fichier des images requêtes et des images de la base de données. La méthode **compare chaque descripteur d'image requête avec ceux de toutes les images dans la base de données** à l'aide de la méthode de calcul de distance choisie. **Les distances sont ajoutées dans un *tuple* et sont associées au nom de leur fichier respectif** (ceux de la base données). Ces ***tuples* sont rassemblés dans une liste**. Cette **liste de *tuples* est ensuite ordonnée** de manière croissante en fonction de la distance calculée pour obtenir un classement. **À noter qu'on génère une liste par image requête**. La fonction retourne une liste de *tuples* contenant la liste du classement et le nom du fichier de l'image requête. **Ainsi, on a un classement pour chacune des images requêtes**. Le format de ce qui est retourné est le suivant : **[Tuple(nomFichier, [Tuple(nomFichier,distance)])]**.
- **get_time(img_arr)** : cette fonction **prend en paramètre une liste d'image**. Essentiellement, cette méthode calcule le temps nécessaire à HOG pour pouvoir

calculer les descripteurs de toutes les images contenues dans la liste. Le temps est calculé par *time* et *hog_wrapper* est utilisé pour pour produire le descripteur. **La fonction retourne le temps en seconde**.

- **do_all_pretreatment(path_arr)** : cette fonction **prend en paramètre une liste de chemins de fichier**. Elle lit toutes les images à partir des chemins et appelle *pre_treatment* pour faire le pré-traitement. **La fonction retourne une liste d'images traitées**.

Pour l'**implémentation de nos tests**, nous avons utilisé toutes les méthodes mentionnées ci-dessus. Après avoir lu tous les chemins de fichier avec glob, nous avons appelé *embed_image_list* pour obtenir l'*embedding* pour nos images requêtes et nos images dans la base de données. Nous avons ensuite récupéré nos classements à l'aide de *get_scores*. Après, on analyse le top 10 de chaque classement, on dénombre la quantité d'images de la même classe que celle de l'image requête, on calcule le pourcentage et on représente le tout sous forme de **diagramme à bandes** grâce à *matplotlib*. Ensuite, on a **calculé le temps d'exécution** de HOG lorsqu'il traite 1,5,10,25 et 48 images. Les images ont été piochées dans les listes de chemins de fichier déjà présentes avant d'être envoyées dans *do_all_pretreatment*. Notre liste d'images en main, *get_time* est invoquée et nous retourne le temps d'exécution. **Avec les différents temps collectés, nous avons généré un graphe**.

En ce qui concerne **les test de rotation**, les scores des images de dauphins ont été récupérés dans le classement de l'image requête dauphin pour en dériver leur positionnement dans ce classement et en faire une figure. Ensuite, nous avons pris une image de visage à laquelle nous avons appliqué des rotations de 45,90 et 180 degrés à l'aide de *rotate*. Ces images ainsi que l'image requête du visage ont été *embedded* pour, au final, générer un mini-classement avec *get_scores*. Similairement, on **génère un mini-classement pour les images texture** en comparant notre image de chemise aux autres images de chemise de différentes textures et aux images d'objets différents de même texture. Pour **les tests de boîtes englobantes**, on compare simplement les images avec et sans boîtes aux images requêtes pour voir l'impact de la présence des boîtes sur la distance entre les descripteurs calculés avec la similarité du cosinus.

5.2 Implémentation de CLIP

Pour CLIP, nous sommes allés chercher le *patch* 16 de *openAI*. Nous l'avons manipulé grâce à CLIPModel, CLIPProcessor et la librairie **transformers**. Comme pour HOG, nous avons utilisé ***resize***, ***rotate***, et ***imread*** de *skimage*. Pour le reste des fonctions externes, on a ***glob***, le module ***spatial*** de ***scipy*** et ***time***. Pour le reste, voici **les méthodes implémentés par notre équipe** pour l'accomplissement de tous nos tests :

- **pre_processing(classes, path_list)** : cette fonction **prend en paramètres un tableau contenant les classes des images** et **un tableau des chemins de fichiers**. La méthode lit les images à partir des chemins et crée des descriptions

pour chaque classe. Le tout est envoyé dans le processeur de CLIP pour générer ***des inputs* qui eux sont retournés**.

- **run_model (inputs)** : cette fonction **prend en paramètre les *inputs* générés par *pre_processing*** pour les insérer dans le modèle CLIP téléchargé. On **retourne ce qui a été produit par le modèle** (duquel on peut extraire les descripteurs d'ailleurs).
- **embed_image_list (classes, path_list)** : cette fonction **prend en paramètres un tableau contenant les classes des images** et **un tableau des chemins de fichiers**. Elle appelle *pre_processing* suivi de *run_model* pour pouvoir extraire les descripteurs des images. Ainsi, **des *tuple* descripteur-nom de fichier** pour chaque image sont conçus et sont **placés dans un tableau qu'on retourne**.
- **get_scores(metric, query_list, db_list)** : voir la description de *get_score* pour HOG plus haut.
- **get_time (inputs)** : cette fonction marche essentiellement comme celle de HOG. Cependant, au lieu de calculer le temps d'exécution de *hog_wrapper*, on le fait pour *run_model*.

Pour **l'implémentation de nos tests**, nous avons procédé **exactement de la même manière que pour HOG**. Pour plus de détails, voir la section 5.1.

5.3 Comparaison des différents patchs de CLIP

Pour comparer les performances des différentes tailles de patchs de CLIP, nous avons étendu l'implémentation initiale en incluant les modèles *openAI* suivants :

- 'clip-vit-base-patch16',
- 'clip-vit-base-patch32',
- 'clip-vit-large-patch14'

Chacun de ces modèles a été testé sur les mêmes jeux d'images et les mêmes expériences afin d'analyser leurs différences en termes de précision et de temps d'exécution. Comme pour HOG et CLIP Patch16, on a utilisé les bibliothèques transformers, skimage, scipy et time, ainsi que les méthodes suivantes :

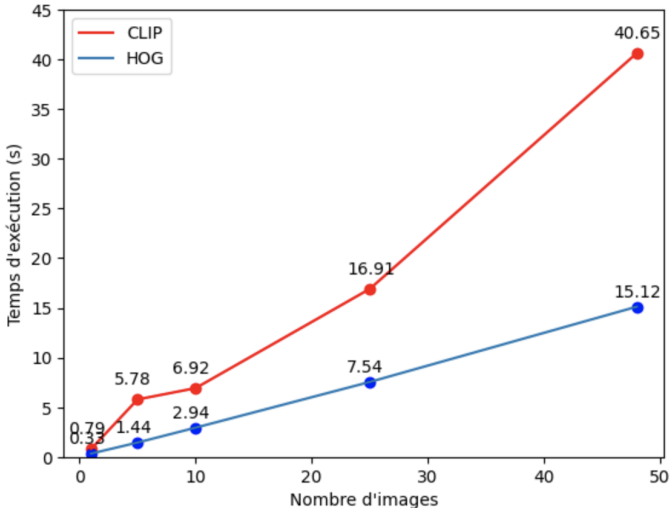
- pre_processing(classes, path_list, processor)
- run_model (inputs, model)
- embed_image_list (classes, path_list,model, processor)
- get_scores(metric, query_list, db_list)
- get_time (inputs, model)

La principale modification des fonctions présentées en section 5.2 a consisté à les adapter pour prendre en paramètre le **modèle** ou le **processeur**, ce qui nous a permis de tester les différents patchs sans dupliquer le code.

6 Résultats des expériences

Suite à nos expériences, nous avons pu collecter des résultats qui ont été mis sous forme de graphiques et de tables.

FIGURE 4 – Temps d'exécution des deux modèles en fonction du nombre d'images à traiter



Premièrement, on peut observer sur **la figure 4**, l'évolution du temps d'exécution de HOG et CLIP en fonction du nombre d'images. Très tôt, on constate une différence assez importante entre les deux droites puisque pour chaque nombre d'image, le temps pour CLIP est, en moyenne, **2.7 fois plus élevé** que pour HOG. De plus, les deux courbes semblent prendre des trajectoires linéaires.

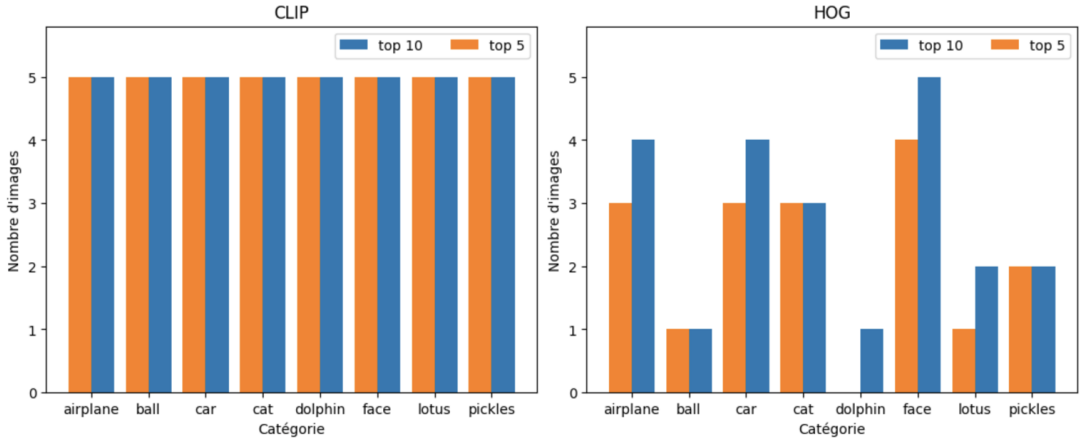


FIGURE 5 – Nombre d'images présentes dans le top 5 et top 10 du classement de leurs classes respective selon le modèle utilisé

Ensuite, en termes de taux de succès au niveau de la reconnaissance, la **table 1** nous montre les distances moyennes relatives à chaque classe. On remarque que CLIP performe

Catégorie	Distance moy. HOG	Distance moy. CLIP
airplane	0.367	0.168
ball	0.450	0.072
car	0.355	0.108
cat	0.335	0.127
dolphin	0.425	0.209
face	0.314	0.073
lotus	0.548	0.188
pickle	0.385	0.138

TABLE 1 – Mesures des distances moyennes pour chaque catégorie pour HOG et CLIP

bien mieux que HOG pour toutes les classes d'images. Sachant que HOG est assez bon pour la reconnaissance faciale, c'est assez impressionnant de voir que **la moyenne pour CLIP (0.073) est 4.3 fois petite que celle de HOG (0.314)** [1]. La plus grande différence se trouve au niveau des ballons où la distance moyenne pour HOG (0.450) est 6.25 fois plus élevée que pour CLIP (0.072). On peut en ressortir les mêmes conclusions en analysant **la figure 6**. Sans aucune exception, toutes les images de toutes les classes **se trouvent dans le top 5 de leurs classements respectifs**. En revanche, pour HOG, les classes 'dauphin' et 'ballon' ont seulement un représentant dans le top 10. Toujours pour HOG, on constate que si l'image se trouve dans le top 10, elle est forcément dans le top 5.

Angle (°)	Distance HOG	Distance CLIP
0	0.292	0.083
45	0.580	0.490
90	0.609	0.496
180	0.352	0.499

TABLE 2 – Mesures de distances entre descripteurs pour HOG et CLIP quant à l'effet de la rotation

Pour ce qui est des résultats des tests de rotation, on peut aller voir la **table 2** qui nous montre des résultats assez intéressants. En effet, comme prévu par l'hypothèse, les distances calculées par HOG augmentent lorsque l'image subit des rotations. Cependant, c'est également le cas pour CLIP. On voit un bond important entre **la distance pour l'image sans rotation (0,083) et les autres distances (≈ 0,5)**. À noter que peu importe l'angle de rotation, la distance semble rester la même dans le cas de CLIP. Malgré cela, si l'on retourne voir la **table 1** et la figure 5, on voit quand même que dans le cas de la classe 'dauphin' (qui contient des images rotates) CLIP performe mieux que HOG puisque toutes les images de dauphins se trouvent dans le top 5 et que la **distance moyenne est 2 fois moins grande** que pour HOG.

Maintenant pour les boîtes englobantes, la table 3 nous montre l'impact de ces dernières sur les valeurs de distances. On voit que les différences sont bien plus importantes chez HOG. De ce fait, la somme des valeurs de différences **pour HOG est de -0.246 alors celle de CLIP est de 0.018**. Ainsi, on observe une assez bonne réduction quant aux

HOG

Nom	Distance sans b.o.	Distance avec b.o.	Différence
ball_1	0.390	0.403	+0.013
ball_2	0.620	0.524	-0.096
ball_3	0.387	0.382	-0.005
ball_4	0.336	0.331	-0.005
ball_5	0.582	0.429	-0.153

CLIP

Nom	Distance sans b.o.	Distance avec b.o.	Différence
ball_1	0.086	0.080	-0.006
ball_2	0.048	0.043	-0.005
ball_3	0.076	0.082	+0.006
ball_4	0.052	0.082	+0.03
ball_5	0.099	0.034	-0.007

TABLE 3 – Distances avec et sans la présence de boîtes englobantes (b.o.) ainsi que la différence entre ces distances pour HOG (haut) et CLIP (bas)

distances calculées pour HOG alors qu'il y a une très légère augmentation des distances en général pour CLIP.

Nom	**Distance HOG**	**Distance CLIP**
bag.jpg	0.501	0.131
blanket.jpg	0.547	0.140
hat.jpg	0.558	0.175
shirt_flinstone.jpg	0.588	0.222
shirt_abstract.jpg	0.588	0.224
shirt_face.jpg	0.657	0.290

TABLE 4 – Mesures de distances entre descripteurs pour HOG et CLIP quant aux tests en lien avec la texture.

Concernant les tests pour la texture, la table 4 nous informe sur les distances calculées entre notre chemises et nos deux catégories d'objets (voir figure 3). Ainsi, on remarque que pour les 2 modèles, **les objets différents de notre chemise qui ont la même texture sont plus similaires** (en termes de distance) que les chemises de textures différentes. De plus, les classements sont exactement les mêmes.

Pour évaluer l'effet de la taille du patch sur les performances du CLIP, nous avons enregistré le temps d'exécution des trois variantes du modèle : *patch32, patch16* et *patch14* de openAI. La **figure 4** montre qu'il existe une relation directe entre la taille du patch et le temps nécessaire au traitement d'un ensemble d'images. Plus la taille du patch est petite, plus le modèle a besoin de ressources informatiques et de temps pour extraire les descripteurs.

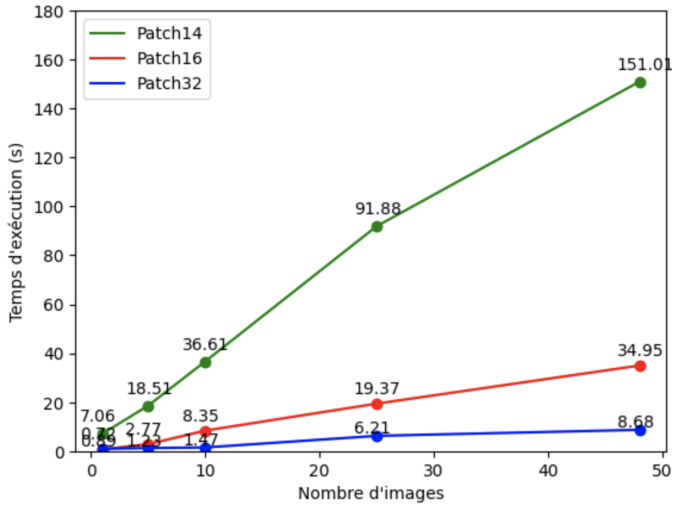


FIGURE 6 – Temps d'exécution en fonction du nombre d'image des différentes versions de CLIP

On peut observer sur **la figure 6**, l'évolution du temps d'exécution des différents patchs de CLIP (patch14, patch32 et patch16) en fonction du nombre d'images. On constate que le modèle *'clip-vit-large-patch14'* est le plus lent, avec un temps d'exécution bien plus élevé que les autres patchs. À titre d'exemple, pour un lot de 48 images, il prend environ 151.01 secondes, contre 34.95 secondes pour *'clip-vit-base-patch16'* et seulement 8.68 secondes pour *'clip-vit-base-patch32'*.

Nom	Dist patch14	Dist patch16	Dist patch32
airplane_1	0.176	0.189	0.135
airplane_2	0.256	0.151	0.161
airplane_3	0.168	0.134	0.146
airplane_4	0.286	0.205	0.245
airplane_5	0.250	0.162	0.183

TABLE 5 – Mesures de distances entre descripteurs entre les différentes version de CLIP sur les images d'avion

Le tableau ci-dessus (**Table 5**) met en évidence les distances cosinus calculées entre les descripteurs des images d'avions pour les trois variantes du modèle CLIP : *Patch14*, *Patch16* et *Patch32*.

En analysant les résultats, Patch16 montre des performances globalement meilleures. En effet, Patch16 obtient les distances les plus faibles sur 4 des 5 images testées (*airplane2, airplane3, airplane4* et *airplane5*). Cependant, pour *airplane1*, **Patch32** enregistre la plus faible distance (0.135 contre 0.176 pour Patch14 et 0.189 pour Patch16). Patch14, quant à lui, affiche systématiquement les distances les plus élevées. Par exemple, pour *airplane4*, sa distance cosinus est de 0.286, contre 0.205 pour Patch16 et 0.245 pour Patch32.

7 Discussion

Après avoir formulé des hypothèses, implémenté du code pour pouvoir tester celles-ci et produit et présenté des résultats, il est maintenant temps d'analyser tout ça et de voir si nos hypothèses ont tenu la route.

Commençons par le test de temps. Comme prévu par l'hypothèse, **CLIP est effectivement plus lent que HOG**, en moyenne, 2.7 fois comme mentionné précédemment. En effet, nous avions émis l'hypothèse que HOG serait plus rapide dû à la simplicité de la méthode (calcul d'orientation de gradients) et les résultats obtenus nous le confirment. On pourrait améliorer ce test en augmentant le nombre d'images qu'on utilise pour obtenir plus de points dans les courbes de temps. Ainsi, on pourrait mieux modéliser ces dernières et voir si elles sont réellement linéaires.

Quant à l'hypothèse de performance en lien avec la reconnaissance d'image, nous avions énoncé que CLIP serait bien meilleur que HOG dû à la présence des deux encodeurs (image et texte) qui ont été très bien entraînés au préalable. Les résultats vont le sens de l'hypothèse, car **pour toutes les classes d'objet CLIP éclipse son concurrent** et produit des descripteurs dont les distances se rapprochent le plus des images requêtes pour une classe donnée. De plus, au niveau des classements, on voit bien la supériorité que CLIP possède. Ceci est une preuve de la robustesse globale du modèle face aux nombreux problèmes présents dans nos images provenant de la base de données (voir section 4.2). Ainsi, on peut affirmer que **notre hypothèse a été confirmée pour ce qui est de la performance globale**.

Ensuite, nous avons suggéré que CLIP serait bien moins affecté par les effets de la rotation que HOG. Concernant ce dernier, on observe les résultats attendus puisque les distances calculées augmentent en présence de rotation. Là où on est un peu surpris, c'est pour ce qui est des résultats de CLIP. **Comme pour HOG, CLIP est assez bien affectée par les rotations effectuées**. Cependant, peu importe la rotation appliquée, on obtient toujours la même distance. Malgré de longues recherches sur internet, nous n'avons pu trouver d'explications concernant ce phénomène observé. En revanche, on peut voir aussi qu'au niveau du top 5, toutes les images de dauphins (pour lesquelles certaines sont rotates) y sont présentes pour CLIP alors qu'il n'y en a aucune pour HOG. Ainsi, **notre hypothèse n'est que partiellement confirmée** puisque nous avons été témoins d'un comportement inattendu de la part de CLIP. Pour améliorer ce test, on pourrait simplement tester tous les angles allant de 0° à 360° pour voir l'évolution graduelle. De plus, on pourrait essayer de nouvelles méthodes de calcul de distances.

Pour continuer, nous avons évoqué le fait que HOG serait meilleur pour reconnaître les textures. Suite à nos tests où les distances entre les descripteurs ont été calculées pour comparer notre image de chemise avec les images des deux catégories pour voir quel modèle pencherait vers quelle catégorie, nous obtenons les mêmes résultats pour les deux méthodes.

En effet, HOG et CLIP ont tendance à être plus influencés par la texture au lieu de l'être par le même type d'objet. **Similairement à la rotation, nous ne nous attendions pas à ce que ce CLIP se comporte comme HOG**, sachant que CLIP a été entraîné de façon importante. En plus de cela, le classement est exactement le même (*bag.png* en pôle position). On peut tenter d'expliquer cela par le fait que la phase d'entraînement a permis de produire un descripteur qui a été capable de prendre en compte des caractérisques telles que la texture. Ainsi, comme vu à la section 1.2, CLIP a possiblement rapproché les images ayant les mêmes textures [3]. En conclusion, **notre hypothèse est réfutée par nos résultats**.

Concernant l'hypothèse sur la présence des boîtes englobantes, nous avons affirmé qu'elles auraient **un impact beaucoup plus significatif sur les performances de HOG que sur celles de CLIP**. Nos résultats vont effectivement dans ce sens, car nous constatons une réduction évidente des valeurs de distance pour HOG, alors que pour CLIP, nous observons une très légère augmentation globale (que nous considérons comme négligeable). De ce fait, nous concluons que **notre hypothèse est confirmée**.

En ce qui concerne l'analyse des différents patch de CLIP, la **Figure 6**, nous a permis de constater que les temps d'exécution de *Patch16* et de *Patch14* sont nettement plus élevés que celui de *Patch32*. Cette variation de performance est due à **la plus grande densité des petits patchs, qui génèrent des descripteurs plus détaillés mais nécessitent plus de calculs** [7]. La **table 5**, quant à elle, nous a permis de constater que *Patch16* est le plus efficace pour produire des embeddings cohérents au sein de la classe "avion", il a en effet une **meilleure capacité à regrouper les images similaires avec une forte précision**. Cependant, *Patch32* reste compétitif, il est capable d'être efficace dans certains cas, bien que son comportement soit plus variable selon les images. *Patch14*, quant à lui, affiche systématiquement les distances les plus élevées, ce qui suggère une plus grande sensibilité aux variations intra-classe. Cela peut être dû à sa capacité à capturer davantage de détails. On remarque donc que *Patch32* offre **un bon compromis entre vitesse et précision**, ce qui le rend plus adapté pour des applications en temps réel où la rapidité est un facteur clé.

Ainsi, le choix du modèle dépend du contexte d'utilisation. Si l'on privilégie une grande rapidité d'exécution ; *Patch32* est à privilégier. En revanche, si l'on recherche une meilleure qualité des correspondances malgré un temps de traitement plus long, *Patch14* et *Patch16* seront plus performants.

En conclusion, ce travail pratique nous a donné l'occasion de comparer deux méthodes de description de régions d'intérêt : HOG et CLIP. À l'aide d'une panoplie de tests, nous avons pu analyser les forces ainsi que les faiblesses de chaque méthode. HOG est assez simple et produit des descripteurs de manière rapide, alors que CLIP est beaucoup plus performant, bien qu'il soit plus lent. De plus, notre analyse des différentes tailles de patchs de CLIP a mis en évidence un compromis entre rapidité et précision, où Patch16 s'est

révélé être le plus équilibré. Au final, HOG n'est plus trop utilisé de nos jours, étant donné la très grande efficacité des modèles de réseaux de neurones tels que CLIP.

8 Bibliographie

[1] G.A. Bilodeau, *Chapter 1 : Describing the content of an image*, Notes de cours, INF6804 : 2024.

[2] Dangyan. (2023) Comparison of HOG (Histogram of Oriented Gradients) and SIFT (Scale Invariant Feature Transform). [En ligne]. Disponible : https ://medium.com/@danyang95luck/comparison-of-hog-histogram-of-oriented-gradients-and-sift-scale-invariant-feature-transform-e2b17f61c9a3

[3] M. Dwyer & M. Brems. (2024) What is CLIP ? Contrastive Language-Image Pre-Processing Explained. [En ligne]. Disponible : https ://blog.roboflow.com/openai-clip/

[4] A. Radford & al., "Learning transferable visual models from natural language supervision" présenté à International Conference on Machine Learning, 2021. [En ligne]. Disponible : https ://arxiv.org/abs/2103.00020

[5] Amazon (2024). [En ligne]. Disponible : https ://www.amazon.ca

[6] Faried. (2023) Image Similarity using CNN feature embeddings. [En ligne]. Disponible : https ://medium.com/@f.a.reid/image-similarity-using-feature-embeddings-357dc01514f8

[7] PI. (2024) Evaluation of OpenAI's CLIP Model. [En ligne]. Disponible : https ://medium.com/neural-engineer/evaluation-of-openais-clip-model-956b3475a456