Optimisation des Prompts dans les Modèles Vision et Langage

Claire Tualle, Juliann Duchemin

Master TRIED

Mars 2025

Sommaire

1 Introduction

1.1 Contexte

Depuis l'avènement du Deep Learning en 2012, le domaine du machine learning a connu une évolution significative. Ce changement a profondément transformé la vision par ordinateur et le traitement du langage, grâce à des techniques qui permettent aujourd'hui d'exploiter de très grands jeux de données non annotés. Des modèles novateurs tels que CLIP [2] ont émergé, utilisant un apprentissage contrastif pour rapprocher les représentations d'images et de textes. Ces avancées offrent des performances impressionnantes dans la reconnaissance de concepts décrits en langage naturel.

1.2 Objectif

Le but de ce projet est d' optimiser les prompts qui génèrent les représentations dans l'espace conjoint image/texte, afin d'améliorer la classification Zero-shot tout en conservant une dimension d'explicabilité. Autrement dit, nous souhaitons affiner les descriptions textuelles associées aux classes pour obtenir des embeddings plus discriminants et permettre au modèle de mieux différencier les catégories, sans nécessiter un entraînement supervisé sur le jeu de données cible. L'enjeu étant d'améliorer les performances en classification tout en ayant des prompts explicables.

1.3 Méthodologie

Dans un premier temps, nous implémenterons un code permettant de générer des attributs pour chaque classe à partir d'un grand modèle de langage (LLM). Nous définirons ensuite une méthode pour sélectionner les meilleurs attributs, et évaluerons les performances de classification obtenues avec ces templates enrichis. Par la suite, nous testerons une approche alternative basée sur la proximité entre les embeddings image et texte pour chaque classe. Enfin, nous procéderons à un fine-tuning des deux dernières couches des encodeurs d'image et de texte afin d'améliorer encore les performances du modèle sur nos prompts optimisés.

1.4 Technologies utilisées

- **Modèle CLIP avec architecture ResNet-50 (RN50)** : Pour l'apprentissage des représentations visuelles à partir de descriptions textuelles.
- **Modèle de langage Mistral-7B-Instruct-v0.3** : Version fine-tunée de Mistral-7B, offrant une meilleure compréhension et génération du langage.
- **Environnement de développement** : Google Colab, qui permet un accès facile aux GPU pour l'entraînement et l'évaluation.
- **Principales bibliothèques Python** :
  - **torch** et **torchvision** : Pour la création, l'entraînement et le traitement des images.
  - **transformers** : Pour l'utilisation de modèles de langage pré-entraînés.
  - **numpy** : Pour les opérations mathématiques et la manipulation de tableaux.
  - **matplotlib** et **seaborn** : Pour la visualisation des données et des résultats.
  - **tqdm** : Pour le suivi des boucles d'entraînement via des barres de progression.

2 État de l'art

2.1 Modèles Vision-Langage

Les modèles de Vision-Langage sont des modèles d'apprentissage profond multi-modaux traitant conjointement du texte et des images. Ils représentent une avancée significative dans les domaines de l'apprentissage profond et de l'apprentissage automatique tant ils utilisent les grandes capacités du traitement automatiques d'image et de langage naturel. Ils sont utilisés dans de nombreux domaines, de la génération automatique de légendes d'images à la description de scène, en passant par la recherche d'images par texte.

Généralement, ce sont des modèles qui combinent des architectures de réseaux de neurones profond, pouvant extraire les caractéristiques des images (par exemple des modèles comme ResNet, VGG-16, ViT, etc.), à des transformers permettant le traitement du langage. Ce sont des modèles coûteux en données d'entraînement, mais dont l'efficacité et la précision peut-être très grande.

De nombreux modèles de Vision-Langage ont été développés avec les années, comme par exemple **DALL-E**, permettant la génération d'images par des prompts textuels, ou encore **VisualBERT** qui utilise l'encodeur de texte classique BERT et permet la génération de légendes ainsi que la description de scène.

2.2 CLIP

CLIP (Contrastive Language-Image Pretraining) est un modèle de Vision-Langage développé par OpenAI [2] conçu pour aligner les représentations d'images et de texte dans un espace commun.

2.2.1 Principe du modèle

Le principe fondamental de CLIP repose sur la comparaison directe des données par apprentissage contrastif, qui "rassemblent" les représentations des images et du texte correspondant par maximisation de la similarité, tout en "éloignant" les paires non-correspondantes en minimisant cette similarité.

L'architecture de CLIP consiste en deux encodeurs distincts : un encodeur d'images et un encodeur de texte. L'encodeur d'images peut être basé sur un réseau de neurones convolutifs (CNN) ou un transformer, tandis que l'encodeur de texte utilise généralement un transformer. Ces encodeurs sont entraînés conjointement pour produire des représentations compatibles dans un espace de représentation commun.

2.2.2 Utilisation de CLIP

Les applications de ce modèle sont très variées, notamment grâce à sa capacité à comprendre les similarités entre les images et les textes associés et à les alignés dans un espace. Ainsi, on peut l'utiliser pour certaines tâches telles que :

- La recherche d'images par texte : En donnant une requête textuelle à CLIP, grâce à la détection de similarités, le modèle peut renvoyer des images pertinentes et correspondant au mieux à l'image demandée.
- La génération de légendes d'images : Le modèle peut s'appuyer sur l'alignement des représentation pour générer des descriptions textuelles précises pour des images.
- La classification d'images : CLIP peut être utilisé pour classer des images en catégories prédéfinies, en utilisant des descriptions textuelles comme références.

2.2.3 Zero-shot Classification

L'une des grandes forces et spécificités de CLIP est sa capacité à réaliser une classification zero-shot des données qui lui sont soumises. Cela signifie que le modèle peut classifier des images dont les classes et donc les caractéristiques classiques n'ont pas été vues pendant l'entraînement, et ce tout en gardant une grande précision. Pour ce faire, CLIP utilise les descriptions textuelles des catégories et compare les représentations des images avec celles des descriptions textuelles.

Par exemple, pour classer une image d'un animal, CLIP peut comparer l'image avec des descriptions textuelles telles que "chat", "chien", "oiseau", etc. La catégorie dont la description textuelle est la plus similaire à l'image dans l'espace de représentation est alors attribuée à l'image. Cette capacité rend CLIP extrêmement flexible et adaptable à de nouvelles tâches sans nécessité de ré-entraînement.

2.3 Amélioration du prompting

Dans le cas des modèles Vision-Langage, la méthode standard consiste à utiliser un prompt fixe, par exemple en associant systématiquement la phrase "a photo of a <class>" à chaque classe. Toutefois, cette approche ne permet pas de saisir toute la variabilité des concepts visuels.

2.3.1 La méthode CoOp

La méthode *CoOp* (Context Optimization), présentée dans [4] propose d'automatiser l'optimisation des prompts en remplaçant les tokens fixes par des vecteurs continus appris. Concrètement, pour chaque classe, le prompt initial est étendu par un ensemble de tokens dont les représentations sont ajustées pendant l'entraînement. Au lieu d'utiliser systématiquement "a photo of a <class>", on introduit des vecteurs de contexte spécifiques qui vont être optimisés pour mieux capturer les caractéristiques visuelles de la classe considérée. L'entraînement se fait en mode few-shot, c'est-à-dire sur un nombre restreint d'exemples par classe. Pendant cette phase, seule la partie correspondant aux tokens de prompt est mise à jour via une descente de gradient, tandis que l'architecture du modèle (encodeur de texte, encodeur d'image, etc.) reste figée. Ainsi, le modèle apprend à ajuster le prompt de façon à maximiser la similarité entre l'image et sa représentation textuelle, ce qui conduit à une amélioration notable de la classification.

2.3.2 Limites de CoOp

Malgré l'amélioration en termes de performance, une limite majeure de CoOp réside dans le fait que les prompts optimisés sont des vecteurs continus dépourvus de signification sémantique explicite. Pour remédier à ce problème, une variante, appelée *CoCoOp*, a été proposée dans l'article [3] afin d'adapter plus finement le prompt aux variations spécifiques de chaque instance ou groupe d'instances. Cette variante permet de générer des prompts qui varient en fonction de l'image. En observant ces variations, il devient possible d'identifier quelles parties du prompt s'ajustent pour mettre en avant certaines caractéristiques visuelles. Cela offre une piste sur les attributs que le modèle considère comme importants pour la classification. Cependant, même cette approche reste limitée du point de vue de l'explicabilité, puisque les représentations optimisées demeurent sous forme de vecteurs opaques, sans correspondance directe avec un langage naturel.

2.3.3 Vers une optimisation de prompt plus explicable

Face à ces limites, notre projet propose d'explorer d'autres approches d'optimisation de prompt qui intègrent explicitement une dimension d'explicabilité. Inspirés par des méthodes récentes telles que la classification par description [1] – où l'on exploite les capacités des grands modèles de langue pour générer des descripteurs explicites – nous envisageons de construire des prompts sous forme de textes compréhensibles. L'objectif est double : améliorer la performance tout en fournissant une explication lisible des critères visuels utilisés pour la classification afin de mieux comprendre et expliquer le processus décisionnel du modèle.

3 Méthodes

3.1 Génération d'attributs

Pour améliorer les performances de la classification Zero-shot, nous avons utilisé un grand modèle de langage (LLM) afin de générer des attributs visuels pertinents pour chaque classe d'images. Nous avons utilisé le modèle `Mistral-7B-Instruct-v0.3` un LLM open-source développé par Mistral AI. Afin de le charger efficacement en mémoire, nous avons eu recours à une technique de quantification en 8 bits, dénommée **LLM.int8()**. Cette technique permet de réduire la précision des poids du modèle, diminuant ainsi son empreinte mémoire tout en conservant ses performances. **LLM.int8()** combine deux approches :

- La quantification **NF4** (NormalFloat 4-bit), qui représente les poids du modèle sur 4 bits au lieu de 32 bits (float32), réduisant considérablement la taille en mémoire tout en préservant la précision grâce à une normalisation adéquate.
- L'utilisation du type de données **bfloat16** pour les calculs afin d'accélérer les opérations tout en maintenant une précision suffisante.

Pour chaque classe d'images, nous avons demandé à Mistral de générer trois attributs visuels facilement identifiables sur une image correspondante avec le prompt suivant:

> "Please generate three visual attributes that are easy to see in an image of a classname. Each attribute should be a single word or a very short phrase (max 3 words). Provide the answer in strict JSON format with keys "feature1", "feature2", and "feature3", where the values are actual descriptions (for example, "brown", "round", or "metallic"). Do not include any additional text."

Nous sélectionnerons ensuite le meilleur de ses trois attributs avec la méthode `get_best_attributes` (détaillée dans la sous-partie suivante) pour élaborer nos prompts. Pour ajouter des attributs supplémentaires, nous n'allons pas simplement sélectionner un des autres attributs générés mais faire de nouveau appel au LLM pour sélectionner de nouveaux attributs en prenant en compte le meilleur attribut précédent. La recherche récursive de ce deuxième attributs se fait en demandant au LLM:

> "Please generate three visual attributes that are easy to see in an image of a classname other than 'best_attribute', which is already chosen. Each attribute should be a single word or a very short phrase (max 3 words). Provide the answer in strict JSON format with keys "feature1", "feature2", and "feature3", where the values are actual descriptions (for example, "brown", "round", or "metallic"). Do not include any additional text."

La méthode `get_best_attributes2` (également décrite en partie suivante) permettra de sélectionner le meilleur de ces trois nouveaux attributs générés. Nous emploierons une approche analogique pour la sélection du troisième meilleur attribut.

Afin de constituer une expérience témoin et de comparer les performances de la classification Zero-shot, nous avons mis en place des procédures pour générer des attributs aléatoires. Deux méthodes ont été envisagées :

1. **Génération de 3 attributs aléatoires** : Le prompt utilisé est le suivant :

   > "Please generate three random words describing an object. Each attribute should be a single word or a very short phrase (max 3 words). Provide the answer in strict JSON format with keys "feature1", "feature2", and "feature3", where the values are actual descriptions (for example, "brown", "round", or "metallic"). Do not include any additional text."

2. **Génération récursive d'attributs aléatoires** : Cette méthode, qui repose sur une procédure itérative permettant d'éviter les répétitions en précisant les attributs déjà sélectionnés, sera détaillée dans la sous-section **La méthode** `get_best_attributes`.

3.2 La méthode `get_best_attributes`

Pour améliorer la précision de la classification zero-shot, nous avons adopté la méthode `best_attributes` consistant à sélectionner les attributs les plus pertinents pour chaque classe d'images.

Après avoir demandé au modèle `Mistral-7B-Instruct-v0.3` de générer pour chaque classe trois attributs, on utilise la fonction `get_best_attribute`. Cette fonction effectue une boucle sur les classes et, pour chaque classe et pour chaque attribut (parmi les trois), construit des templates de prompts de la forme suivante:

- "a photo of a classe because of attribute", pour la classe correspondant à celle de la boucle.

- "a photo of a classe", pour les autres.

On calcule ensuite la Top-1 accuracy associée à ces templates pour chacun des 3 attributs. L'attribut engendrant le meilleur score sera sélectionné comme `best_attribute1`.

Pour sélectionner le deuxième meilleur attribut, on envoie au LLM le deuxième prompt décrit dans la partie précédente qui précise le premier attribut déjà retenu. Mistral renvoie alors trois nouveaux attributs, dont le meilleur (selon la performance du classificateur) est sélectionné comme `best_attribute1` avec la procédure `get_best_attribute2`. Cette fonction est similaire à `get_best_attribute1` si ce n'est les templates qui sont cette fois:

- "a photo of a classe because of best_attribute and attribute", pour la classe correspondant à celle de la boucle.
- "a photo of a classe", pour les autres.

Une fois le deuxième meilleur attribut sélectionné, on procède de manière analogique pour obtenir le troisième meilleur attributs. On demande au LLM de générer à nouveau 3 attributs pour chaque classe mais en prenant en compte cette fois-ci les deux meilleurs attributs sélectionnés, puis on utilise la procédure `get_best_attribute3`.

La méthode de **génération récursive d'attributs aléatoires** s'inspire de cette approche itérative mais en générant à chaque fois trois attributs aléatoire. On comparera également nos résultats avec ceux donnés par la génération directe de trois attributs aléatoires sans chercher à sélectionner les meilleurs.

3.3 Zero-shot avec `get_best_attributes`

Lorsque nous avons nos **get_best_attributes**, nous pouvons alors réaliser la classification zero-shot sur l'ensemble de nos images de test avec les meilleurs prompts possibles pour chaque classe. Pour ce faire, on crée une liste de prompts dont chacune correspond à chaque classe avec un ou plusieurs de ses meilleurs attributs, puis on calcule les représentations des textes pour chaque classe afin de les comparer aux représentations des images. Cela nous permet alors de déterminer avec quelle précision les images sont bien classées (précision dite top-1).

Pour réaliser le calcul des représentations, on utilise une fonction `zeroshot_classifier_with_attributes` qui, pour chaque classe, prend le prompt associé et l'encode pour extraire sa représentation dans l'espace commun. Ensuite, les représentations sont normalisées. La fonction renvoie alors la matrice des représentations pour chaque classe appelée `zeroshot_weights`.

Pour chaque image, on calcule alors les scores de l'image pour chaque classe en multipliant les représentations de l'image et les représentations textuelles par la formule suivante :

$$logits = 100 \times image\_features \cdot zeroshot\_weights \tag{1}$$

Ces *logits* permettent alors de déterminer les k classes les plus probables (scores les plus élevées), et on détermine grâce à la fonction `accuracy` la précision (*top-1 accuracy*) que chaque image aie le meilleur score dans la bonne classe.

Cette précision, on l'observe de deux manières différentes :

- Premièrement on observe les résultats de manière globale, c'est-à-dire qu'on récupère la précision correspondant à la classification de toutes les images dans leurs classes. Cela nous donne une idée de comment les résultats évoluent par rapport à des prompts identiques pour chaque classe.
- Dans un second temps, on observe les résultats par classe, avec les représentations des images et des textes alignés pour toutes les classes. Cela nous permet de voir dans quelle mesure l'ajout d'attributs est impactant sur certaines classes, négativement comme positivement, et empêche le discernement de certaines classes.

Cela nous permet donc de comprendre l'impact de ces attributs à différentes échelles : une échelle macroscopique nous donnant une idée de comment le modèle évolue, et une échelle microscopique montrant l'impact sur les données spécifiques que nous avons, permettant le discernement des classes qui voient leur précision augmenter et celles qui la voient diminuer.

On utilise l'exacte même méthode, mais en utilisant des attributs aléatoirement générés afin de comparer aussi l'impact de la génération d'attributs correspondant à la classe dans nos résultats. On regarde alors à nouveau les résultats pour chaque classe ainsi que les résultats globaux.

3.4 Autre méthode : Similarité cosinus

Afin de comparer nos résultats avec différentes méthodes de choix des "meilleurs" attributs, nous avons décidé d'adopter une autre stratégie en parallèle. Cette stratégie consistait à déterminer les attributs dont les représentations étaient les plus proches possible de la classe de l'image que nous cherchions à classifier.

Pour cela, nous avons donc utilisé la même fonction que CLIP dans sa recherche de similarité entre les images et les textes : la similarité cosinus. On la calcule selon la formule suivante :

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sum_{i=1}^{n} B_i^2} \tag{2}$$

On utilise alors la même méthode que précédemment pour générer une dizaine d'attributs et on compare leur similarité au mot [CLASS]. On choisit alors les 3 meilleurs en tant qu'attributs afin de tester la classification Zero-shot comme réalisé précédemment.

Pour aller plus loin, on regarde aussi les résultats obtenus après avoir déterminé dix fois l'attribut ayant la similarité cosinus la plus grande parmi dix attributs générés, et ce pour chaque classe. Ensuite, on détermine alors la précision de la classification zero-shot en ajoutant entre un et dix attributs aux prompts.

3.5 Fine-Tuning

Pour améliorer les performances de classification de notre modèle sur des invites optimisées, nous avons mis en œuvre deux stratégies distinctes de fine-tuning : l'une avec des templates fixes et l'autre avec des templates dynamiques. Ces approches visent à affiner les deux dernières couches des encodeurs d'image et de texte afin d'aligner plus étroitement les représentations des images avec celles des textes associés aux classes.

Préparation des données et des représentations : Soit $x \in R^{H \times W \times C}$ une image et t une invite textuelle associée à une classe. Pour chaque image, nous extrayons un vecteur de caractéristiques défini par :

$$\mathbf{z}_I = \frac{f_\theta(x)}{\|f_\theta(x)\| + \epsilon}, \tag{3}$$

où ϵ est une petite constante (par exemple, $\epsilon = 10^{-6}$) pour éviter la division par zéro.

De même, pour une invite textuelle t, nous calculons :

$$\mathbf{z}_T = \frac{g_\phi(t)}{\|g_\phi(t)\| + \epsilon}. \tag{4}$$

Construction des logits et fonction de perte : Pour une image x et un ensemble de $K = 10$ classes (comme dans CIFAR-10), nous construisons les logits par :

$$\ell = \alpha \, \mathbf{z}_I^\top \mathbf{Z}_T, \tag{5}$$

où $\mathbf{Z}_T = [\mathbf{z}_{T,1}, \mathbf{z}_{T,2}, \ldots, \mathbf{z}_{T,K}]$ représente les vecteurs de caractéristiques des invites pour chaque classe, et α est un facteur d'échelle (ici $\alpha = 50.0$).

La fonction de perte utilisée est l'entropie croisée (*Cross-Entropy Loss*) :

$$\mathcal{L} = -\sum_{i=1}^{K} y_i \log\left(\text{softmax}(\ell)_i\right), \tag{6}$$

où $y \in \{0, 1\}^K$ est le vecteur one-hot de la classe vraie.

Stratégies de fine-tuning :

- **Gélation du modèle pré-entraîné :** Tous les paramètres du modèle sont gelés à l'exception des deux dernières couches de l'encodeur d'images et de l'encodeur de texte.
- **Fine-tuning avec templates fixes :** Des invites textuelles fixes sont utilisées pour chaque classe pendant tout le processus de fine-tuning, ce qui permet de maintenir une cohérence dans les représentations textuelles associées à chaque classe.
- **Fine-tuning avec templates dynamiques :** Pour chaque époque, un ensemble d'invites est généré dynamiquement en sélectionnant aléatoirement un attribut parmi un ensemble d'attributs prédéfinis pour chaque classe, ce qui introduit de la variabilité et réduit le risque de surapprentissage sur des invites fixes.

- **Rétropropagation :** À chaque itération, les caractéristiques textuelles sont recalculées afin de créer un nouveau graphe de calcul, permettant une rétropropagation sans erreur.

4 Résultats

4.1 Génération

Comme décrit précédemment, nous avons utilisé le modèle de langage `Mistral-7B-Instruct-v0.3` pour générer des attributs visuels pour chaque classe d'images du jeu de données CIFAR-10. Lors de la génération, on fixe la température, paramètre qui contrôle la dispersion de la distribution de probabilité lors de la génération de texte. Mathématiquement, si l'on note z_i les logits du modèle, la probabilité d'obtenir le token i est donnée par :

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

Ainsi :

- **Température basse ($T < 1$) :** Les différences entre les logits (sorties brutes du modèle avant qu'elles ne soient transformées en probabilité) sont accentuées, ce qui conduit à une distribution plus concentrée sur les tokens les plus probables, rendant la sortie plus déterministe et répétitive.
- **Température égale à 1 ($T = 1$) :** La distribution reste inchangée par rapport aux logits originaux.
- **Température élevée ($T > 1$) :** Les différences entre les logits sont atténuées, aplatissant la distribution et rendant la génération plus aléatoire et diversifiée.

Dans notre cas, nous souhaitons des attributs qui caractérisent simplement nos classes sans originalité, nous avons donc choisi une température de 0.5.

Exemples d'attributs générés par Mistral avec la fonction `get_attributes` pour "apple" selon la température :

- **T=0.5:** "red", "spherical", "glossy".
- **T=1.4:** "Red color", "Oval shape", "Pippy surface"

On a également remarqué que, pour une température supérieure à 1, le LLM ne suit plus rigoureusement le prompt et ne renvoie pas toujours ses réponses dans le format json souhaité.

4.2 get_best_attributes

La méthode `get_best_attributes`, décrite au 3.2, consiste à sélectionner les attributs les plus pertinents pour chaque classe afin d'améliorer la précision de la classification zero-shot. Le tableau ci-dessous présente les trois attributs sélectionnés pour chaque classe, obtenus via les fonctions `get_best_attribute1`, `get_best_attribute2` et `get_best_attribute3` :

Classe	Attribut 1	Attribut 2	Attribut 3
airplane	long	tail-finned	fuselage-long
automobile	four-door	sporty design	four wheels
bird	beak shape	tail length	feather color
cat	fur_color	whiskers_length	ear_shape
deer	long tail	large ears	spotted coat
dog	fur_color	tail_curl	eye_shape
frog	webbed feet	large eyes	long tongue
horse	coat_color	mane_length	long_ears
ship	horizontal stripes	square sails	white deck
truck	rectangular shape	paint color	headlight shape

Table 1: Attributs sélectionnés pour chaque classe d'images.

Les attributs générés sont cohérents et permettent de caractériser les individus de chaque classe, avec des caractéristiques simples comme les formes et couleurs.

4.3 Zero-shot avec `best_attributes`

Comme décrit précédemment, on utilise donc les différents attributs obtenus pour réaliser la classification Zero-shot. On crée alors des prompts textuels comme montré ci-dessous :

- Avec 1 attribut : "This is a photo of a [CLASS] because of [Attribut1]"
- Avec 2 attributs : "This is a photo of a [CLASS] because of [Attribut1] and [Attribut2]"
- Avec 3 attributs : "This is a photo of a [CLASS] because of [Attribut1] and [Attribut2] and [Attribut3]"

Par exemple, on aura donc les trois templates suivants pour la classe ***'airplane'*** : 'This a photo of a airplane because of long', 'This a photo of a airplane because of long and tail-finned' et 'This a photo of a airplane because of long and fuselage-long'.

Après avoir construit trois dictionnaires avec les différents prompts pour un, deux et trois attributs, on peut donc les utiliser pour réaliser la classification Zero-shot. On observe donc dans un premier temps les résultats globaux pour ces différents nombres d'attributs :

Nombre d'attributs	0	1	2	3
Précision	70.35%	72.91%	70.75%	69.67%

Table 2: Précision globale en fonction du nombre d'attributs orientés-classe ajoutés au prompts

On remarque alors que les résultats sont meilleurs que ceux obtenus avant l'utilisation d'attributs, au moins lorsque l'on en ajoute un ou deux. On remarque cependant un pic d'amélioration à un attribut. Cela s'explique par la sensibilité du modèle à la génération d'attributs. En effet, par notre méthode, nous générons 3 attributs par classe avant de choisir le meilleur, et ce pour le premier, le second et le troisième attribut. Cependant, la génération étant probabiliste, les résultats obtenus dépendent des attributs que le large language model a choisi. Alors les résultats sont logiquement plus simples à voir augmenter par le choix du meilleur premier attribut, que lorsque l'on en ajoute plus.

Cependant, il était aussi possible que l'ajout de ce premier attribut améliore les résultats de la classification totalement indépendamment de la nature du texte ajouté. C'est pourquoi nous avons décidé de comparer nos résultats à ceux obtenus après l'ajout de mots générés aléatoirement par le même LLM. Après avoir ajouté ces mots aléatoires et créés des templates comme précédemment, on obtient les résultats suivants :

Nombre d'attributs	0	1	2	3
Attributs orientés-classe	70.35%	72.91%	70.75%	69.67%
Attributs aléatoires	70.35%	65.27%	63.83%	63.92%

Table 3: Précision globale en fonction du nombre d'attributs orientés-classe ou aléatoires ajoutés au prompts

Dans ce tableau on remarque alors clairement que ce n'est pas l'ajout simple de texte qui améliore les résultats mais bien la recherche d'un prompt textuel décrivant mieux la classe.

Regardons alors les résultats obtenus pour chaque classe dans le cas où les meilleurs attributs sont choisis :

Nombre d'attributs	0	1	2	3
Airplaine	74.6%	82.3%	90.1%	**95.2**%
Automobile	**98.2**%	68.4%	81.9%	86.5%
bird	**87.0**%	77.8%	71.7%	67.8%
cat	62.1%	73.1%	**78.3**%	74.4%
deer	56.9%	**66.5**%	52.9%	49.5%
dog	**73.0**%	55.7%	53.8%	58.4%
frog	29.9%	**34.7**%	29.4%	32.3%
horse	82.3%	81.4%	87.7%	**91.8**%
ship	70.7%	**85.7**%	77.9%	65.7%
truck	52.9%	**92.5**%	83.8%	75.1%

Table 4: Précision par classe en fonction du nombre d'attributs orientés-classe ajoutés au prompts

On peut donc dire que certaines classes sont très sensibles à l'ajout de texte dans les prompts, comme la classe `airplane` et la classe `truck` qui voient leurs résultats de classification augmenter grandement, mais aussi comme les classes `automobile` et `dog` dont les résultats chutent avec l'ajout même d'un seul attribut. Cela nous montre que même pour l'ajout d'un seul attribut qui augmente les résultats globalement, certaines classes voient leurs résultats diminuer.

On observe alors les résultats obtenus par la même méthode, mais en ajoutant des attributs aléatoires :

Nombre d'attributs	0	1	2	3
Airplaine	74.6%	73.4%	**81.5**%	72.6%
Automobile	**98.2**%	83.3%	85.7%	81.9%
bird	87.0%	**87.6**%	71.2%	72.6%
cat	62.1%	38.4%	**67.4**%	49.4%
deer	**56.9**%	32.8%	37.0%	27.1%
dog	73.0%	59.8%	70.2%	**73.9**%
frog	29.9%	**44.6**%	19.8%	32.5%
horse	82.3%	**86.1**%	79.7%	80.9%
ship	70.7%	67.5%	61.0%	**71.0**%
truck	52.9%	**79.2**%	64.8%	77.3%

Table 5: Précision par classe en fonction du nombre d'attributs aléatoires ajoutés aux prompts

À nouveau, on remarque que la sensibilité de certaines classes, comme par exemple `truck` ou `frog`, dépend de l'ajout de texte mais pas nécessairement de l'ajout de texte pertinent, tandis que des classes comme `airplane`, `deer` ou encore `horse` sont sensibles à des choix d'attributs descriptifs clairs. Cependant, au vu des résultats généraux et des résultats par classes, on peut observer que l'amélioration de la précision reste meilleure lorsque l'on choisi des attributs spécifiques à chaque classe.

4.4 Zero-shot avec Similarité Cosinus

On peut à présent observer les résultats obtenus lors de l'exploration de la seconde stratégie, consistant à choisir des attributs dont les similarités avec la classe sont les plus hautes. Regardons par exemples la similarité obtenue pour 10 attributs générés avec la classe airplane :

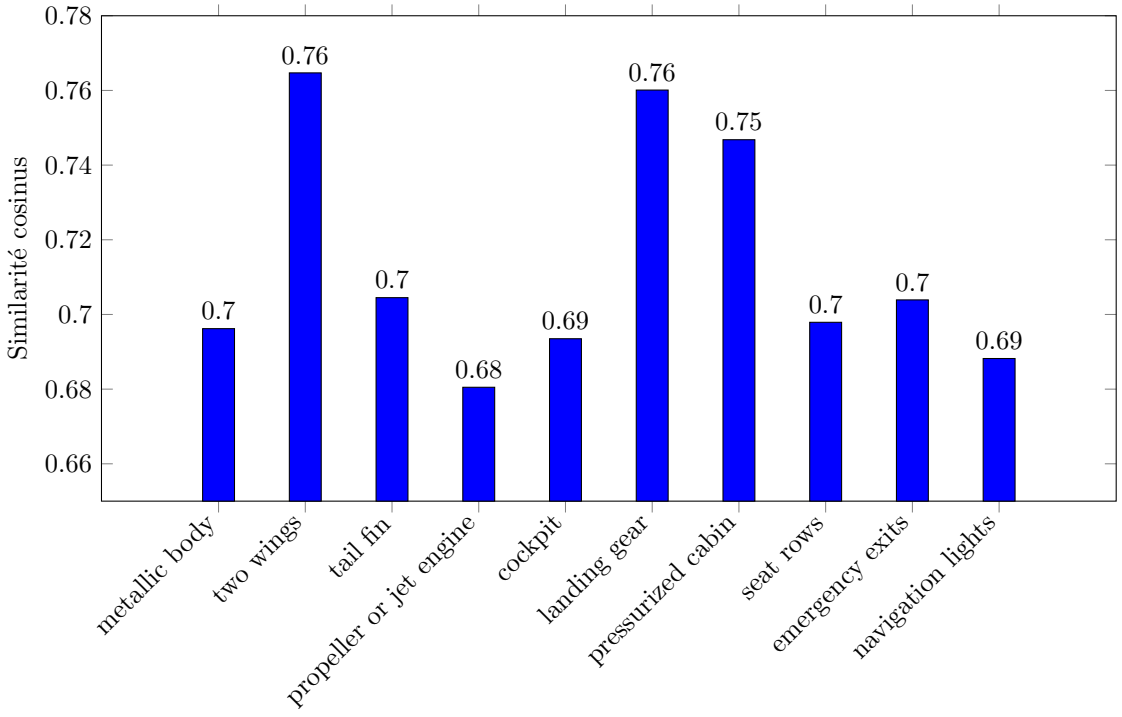


Figure 1: Similarité cosinus des attributs spécifiques à l'avion

On peut alors choisir les attributs `two wings`, `landing gear` et `pressurized cabin` afin de les ajouter à notre prompt initial *"a photo of a airplane"*.

Cela nous donne alors les résultats suivants, qu'on compare avec nos résultats précédents :

Nombre d'attributs	0	1	2	3
Attributs orientés-classe	70.35%	72.91%	70.75%	69.67%
Attributs similarité cosinus	70.35%	62.39%	66.02%	66.48%

Table 6: Précision globale en fonction du nombre de `best_attributes` ou d'attributs par similarité cosinus ajoutés au prompts

On remarque que les résultats augmentent avec le nombre d'attributs, alors on décide d'aller plus loin, cette fois en générant 10 nouveaux attributs par classe et en prenant le meilleur, et ce jusqu'à 10 fois.

Après avoir créé les différents templates en ajoutant donc jusqu'à dix attributs, on obtient les résultats de classification suivants :

Nombre d'attributs	0	1	2	3	4	5	6	7	8	9	10
Précision (en %)	70.35	70.59	73.00	67.60	68.69	69.53	65.76	62.37	55.35	64.23	62.93

Table 7: Précision globale en fonction du nombre d'attributs choisis par similarité cosinus ajoutés aux prompts

On remarque que cette méthode permet d'obtenir de bons résultats pour un faible nombre d'attributs, avec des valeurs de précision avoisinant celles obtenues par l'autre stratégie, et ce en étant moins coûteuse en ressource informatique, mais s'avère ne pas évoluer comme on l'espérait en continuant d'augmenter en précision avec le nombre d'attributs. Elle est aussi bien plus dépendante de la génération d'attributs comme le montre l'écart des résultats obtenus lors de notre première expérience et celle-ci (Table 6 et Table 7).

4.5 Fine-Tuning

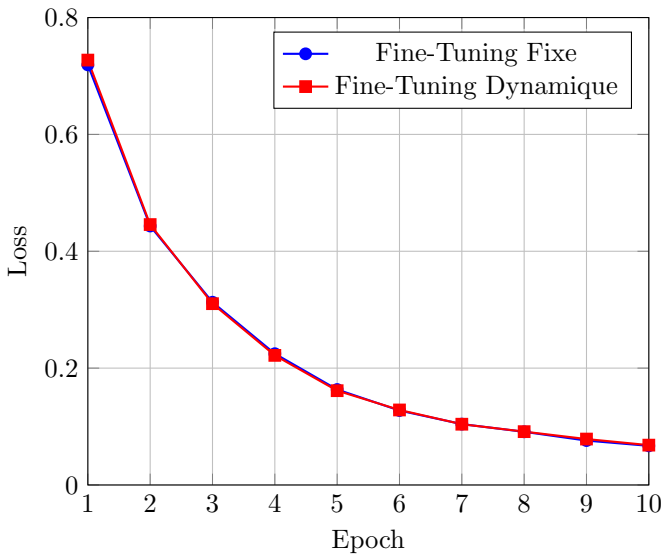
Après fine-tuning sur CIFAR-10 durant 10 époques, nous avons observé une amélioration notable des performances par rapport à la configuration zero-shot initiale. Les résultats moyens sur l'ensemble de test sont les suivants :

Métrique	Précision Top-1	Précision Top-5
Fine-Tuning Dynamique	85.77%	99.58%
Fine-Tuning avec Prompts Fixes	86.84%	99.44%

Table 8: Performances obtenues après fine-tuning.

Les courbes d'apprentissage ci-dessous indiquent une convergence stable et presque similaire dans les deux cas, avec une diminution progressive de la perte.

Courbes d'apprentissage des approches de Fine-Tuning



5 Conclusion

Au cours de ce projet, nous avons pu utiliser les différentes technologies citées en introduction et explorer les différentes stratégies mises en place. Voici alors les résultats et les limites que nous avons pu extraire de notre travail.

5.1 Résultats

Tout d'abord, le projet visait à explorer la possibilité d'utiliser de nouvelles méthodes pour créer des prompts automatiquement dans l'utilisation de CLIP tout en gardant des prompts intelligibles, ce qui était un problème rencontré par CoOp. Nous avons décidé de comparer les résultats que nous avons obtenus avec le résultat obtenu par CLIP pour des prompts simples dont on connaissait l'efficacité : les prompts "This is a photo of a [CLASS]". Cela nous a permis d'évaluer l'évolution des résultats de CLIP sans chercher à comparer l'efficacité de nos stratégies avec CoOp qui présentait de très bons résultats, mais sans garder des prompts intelligibles.

La première partie de notre travail était de réussir à trouver un LLM qui nous proposait des attributs pour une classe données qui faisait sens et qui était utilisable dans notre travail ultérieur. En utilisant `Mistrial-7B-Instruct`, nous avons réussi à obtenir des résultats tout à fait pertinents et qui variait d'un attribut à l'autre ce qui apportait de la diversité à nos données textuelles. Cela s'observe lorsque l'on regarde les meilleurs attributs détectés par la suite pour chaque classe qui représentait tout autant les caractéristiques géométriques des classes que des aspects de couleurs ou physiologiques (pour les animaux).

Ensuite, nous avons donc pu obtenir les résultats que nous cherchions quant à la performance du modèle en classification zero-shot avec l'ajout de nos attributs. Lorsque l'on a observé les résultats généraux, il est alors apparu que la stratégie que nous avions mise en place choisissant les attributs correspondant aux classes était tout à fait pertinente puisqu'elle améliorait les résultats de CLIP. De plus, ces résultats dépendait alors des attributs montrant la capacité du modèle à extraire les caractéristiques de chaque classe même en zero-shot et à les rapprocher des textes correspondant dans l'espace de représentation qu'il crée.
Cependant, lorsque l'on observe les résultats obtenus pour chaque classe, on remarque que chaque classe réagit différemment à CLIP. En effet, en zero-shot, certaines classes voient leurs images être très bien classées tandis que d'autres ont des performances très basses, et ce, sans l'ajout d'attributs (`airplane vs frog` par exemple). L'ajout d'attribut aide alors certaines classes tandis qu'elle en pénalise d'autres, et on remarque que le nombre optimal d'attributs à ajouter pour chaque classe n'est pas vraiment clair si on regarde ces résultats spécifiques. On peut cependant être sûr que les résultats pour des attributs générés pour chaque classe sont meilleurs que pour des attributs aléatoires en général.

La seconde stratégie observée, celle utilisant comme critère de meilleur attribut la similarité cosinus, admet des résultats différents. En effet, ceux-ci sont très dépendants des attributs générés, tant le meilleur attribut selon le critère peut parfois être d'une grande aide dans la classification, ou parfois être un fardeau. On peut cependant voir qu'avec une génération correcte, on obtient de bons résultats, mais que ceux-ci restent assez instables.

Enfin, nous avons décidé de quitter la classification zero-shot pour essayer de voir l'impact de l'utilisation de nos prompts textuels et des attributs dans l'entraînement des quelques dernières couches du modèle. Nous avons donc fine-tune les deux dernières couches de chaque encodeur suivant deux méthodes : en utilisant des prompts textuels fixe ou en utilisant des prompts textuels variables changeant d'attributs pour chaque batch. Après entraînement, on remarque alors, comme attendu, de meilleurs résultats en classification pour les deux méthodes en donnant des résultats plus ou moins équivalents. Cela montre que le modèle n'a pas nécessairement besoin d'une variété d'attributs textuels pour mieux s'entraîner.

5.2 Limites

Il est important tout de même de citer les différentes limites de notre projet qui impacte les résultats que nous avons pu observer.

Premièrement, l'utilisation de Google Colab comme espace de développement sans abonnement a limité les ressources de calcul que nous pouvions utiliser. Cela a donc impliqué de nombreuses limitations, notamment dans le choix du LLM utilisé. Bien que nous utilisions un modèle très réputé, il aurait été intéressant d'essayer des modèles plus performants avec plus de paramètres et sans forcément devoir quantifier en 8 bits les modèles utilisés. L'autre limitation était dans l'utilisation de nos méthodes. Le choix des attributs étant très coûteux, nous n'avons pas pu aller plus loin que 3 attributs générés pour 1 choisi, alors même que la génération avait un impact clair sur les résultats. Ainsi, nos résultats étaient limités et ils devraient être encore meilleurs si l'on pouvait toujours choisir le meilleur attribut parmi

100 attributs générés.

Enfin, le temps du projet ayant été limité nous n'avons pas pu aller plus loin, notamment dans l'affinement des paramètres du LLM (température, prompts) et dans l'étude des attributs dans leur rapport sémantique à chaque classe. De même, nous n'avons pas pu nous essayer à d'autres datasets plus gros, pour cause de temps et de capacités de calcul.

5.3 Conclusion globale

Pour conclure globalement, on a pu explorer différentes stratégies et différents aspects du projet qui ont montré que la stratégie de détection du meilleur attribut était pertinente même si elle ne propose qu'une amélioration faible, et qu'elle pouvait être une solution relativement robuste dans l'amélioration des résultats de CLIP en gardant une intelligibilité des prompts textuels générés et utilisés.
Il serait intéressant pour aller plus loin, et dans le cas où des ressources de calcul plus puissantes pourraient être utilisées, de chercher à augmenter les nombres d'attributs générés avant d'être filtrés pour l'ensemble des stratégies, et ce, jusqu'au fine-tuning, et de même de poursuivre ce fine-tuning qui n'a pas encore montré de signe de convergence. De même, il serait intéressant d'observer les résultats obtenus de nos méthodes sur d'autres jeux de données comme CIFAR100 qui propose un plus grand nombre de classes et donc de diversité des données.

6 Annexes

References

[1] Sachit Menon and Carl Vondrick. Visual classification via description from large language models, 2022.

[2] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pam Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[3] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Conditional prompt learning for vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16816–16825, 2022.

[4] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 130(9):2337–2348, Sept. 2022.