

Note technique – Proof of Concept : amélioration d'une classification d'images

LE GAL Jérôme

Openclassrooms – Projet 08

*Note de l'auteur*

Cette note technique est réalisée dans le cadre d'un projet de formation « Data Scientist ».

*Introduction :*

Ce projet vise à améliorer un modèle de classification d'articles pour une marketplace, où la précision est importante. ProtoViT est exploré pour dépasser les performances de VGG16 grâce à sa robustesse et son interprétabilité.

## 1. Dataset retenu

Le dataset retenu est une banque de 1050 images d'articles en vente sur un site de type Marketplace et composés de 7 catégories :

- Home Furnishing (qty : 150)
- Baby Care (qty : 150)
- Watches (qty : 150)
- Home Decor & Festive Needs (qty : 150)
- Kitchen & Dining (qty : 150)
- Beauty and Personal Care (qty : 150)
- Computers (qty : 150)

Ces images nous ont été fournies par **Linda**, Lead Data Scientist chez « Place de marché », l'entreprise pour laquelle un premier modèle de classification automatique des articles a été réalisé avec un algorithme de type VGG-16. Linda nous certifie avoir vérifié qu'il n'y avait aucune contrainte de propriété intellectuelle sur les images.

### Préparation des données :

Afin d'éviter toute fuite d'information entre les jeux d'entraînement, de test et de validation, les images ont été séparées avec les quantités suivantes :

Train: 630, Val: 210, Test: 210

Pour les besoins spécifiques de l'entraînement et test du nouvel algorithme, les images ont été classées dans des dossiers et sous-dossiers de cette façon :

```
datasets/  
├── train/  
│   ├── category1/  
│   ├── category2/  
│   └── ...  
├── val/  
│   ├── category1/  
│   ├── category2/  
│   └── ...  
└── test/  
    ├── category1/  
    ├── category2/  
    └── ...
```

## 2. Concepts de la méthode récente :

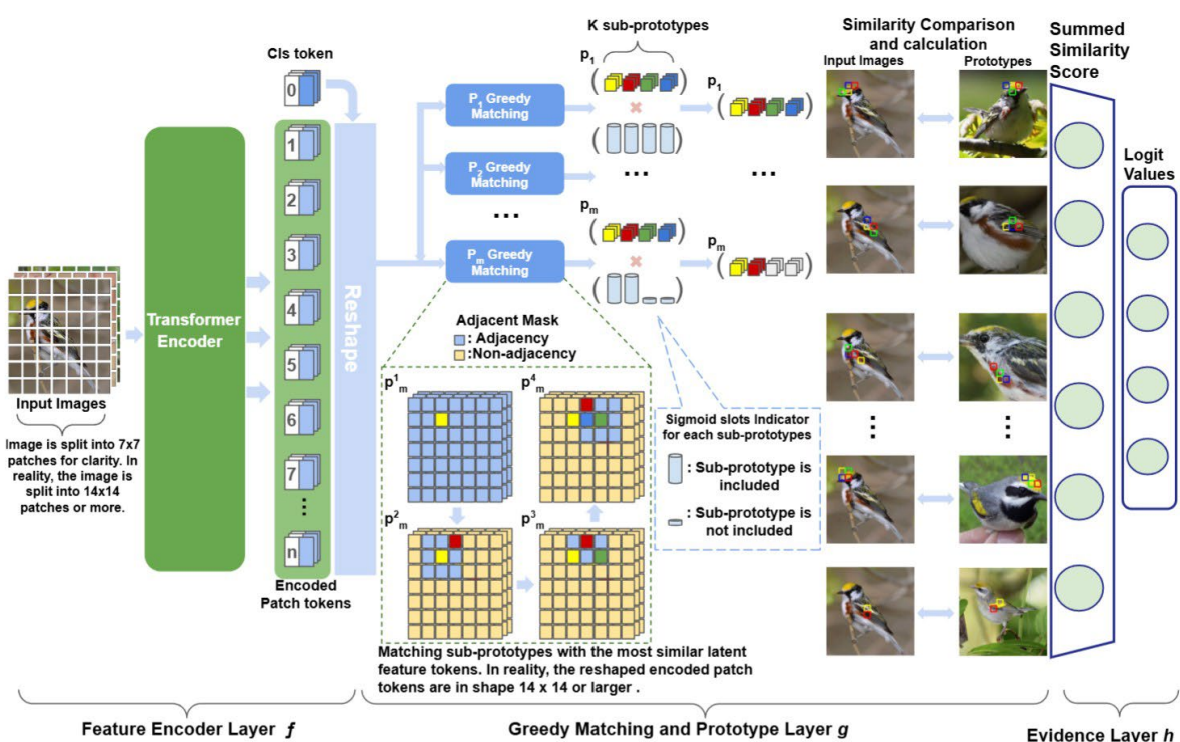
La méthode récente retenue pour tester des améliorations de classification d'images se nomme : ***ProtoViT*** – (Prototype-based Vision Transformers).

Le principe de base est la combinaison du *Deep Learning* en utilisant la technique des ***Transformers ViT*** et le raisonnement basé sur des cas avec une approche interprétable dont l'idée centrale de classer une image en la comparant à des ***Prototypes*** appris pendant l'entraînement.

### 2.1 Architecture de *ProtoViT* :

*ProtoViT* est architecturé en 3 grandes parties :

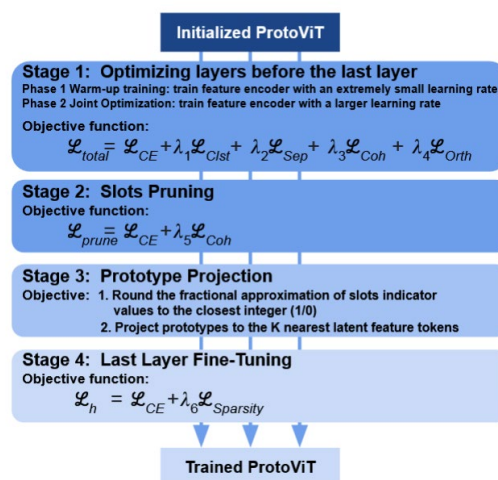
- Feature Encoder Layer
  - L'image est découpée en patches, et chaque patch est projeté dans un espace latent grâce à un ViT préentraîné (backbone).
  - Les « tokens » obtenus décrivent les relations entre les patches.
- Greedy Matching Layer
  - Cette couche compare les **tokens latents** de l'image aux prototypes en utilisant une mesure de similarité cosinus :
$$\text{similarité cosinus} = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$
- Evidence Layer
  - Les similarités sont agrégées pour déterminer la classe finale en combinant les scores des prototypes.



## 2.2 Entraînement du modèle :

ProtoViT est entraîné en plusieurs étapes :

- Initialisation des prototypes : chaque prototype est formé pour capturer des features spécifiques à une classe.
- Former les couches d'encodage de caractéristiques avant la dernière couche :
  - Perte d'entropie croisée : classique dans la classification
  - Perte de cluster : encourage les prototypes à être proches des images de leur classe
  - Perte de séparation : empêche les prototypes de capturer des features de classes incorrectes.
  - Perte de cohérence : garantit que les sous-parties d'un prototype représentent des concepts homogènes.
  - Perte d'orthogonalité : tend à rendre les prototypes indépendants les uns des autres.
- Elagage des slots : supprimer les slots qui n'apportent pas grand-chose (réduction de la complexité)
- Projection des prototypes : les prototypes sont ajustés pour correspondre aux tokens d'image les plus proches dans l'espace latent.
- Finetuning de la dernière couche : finaliser le modèle en équilibrant précision et efficacité.



## 2.3 Innovations et avantages de ProtoViT :

- **Interprétabilité** : il est possible de visualiser les prototypes et d'expliquer pourquoi une décision a été prise.
- **Flexibilité** : les prototypes peuvent s'adapter aux variations géométriques, comme des objets en rotation ou déformés.
- **Performance accrue** : ProtoViT atteint des performances supérieures à celles des méthodes basées sur CNN et autres modèles prototypes.

### 3. La modélisation :

Etapes de la modélisation :

1. Préprocessing des données
2. Data Augmentation
3. Entraînement et fine-tuning du modèle
4. Évaluation et optimisation

#### 3.1 – Preprocessing :

Les images doivent être redimensionner afin de pouvoir être exploitées par ProtoViT, elles sont ici redimensionnées en 224x224 pixels.

Les valeurs d'intensité des pixels sont également normalisées afin d'être dans la plage [0, 1].

#### 3.2 – Augmentation du dataset d'entraînement :

Afin d'améliorer l'entraînement du modèle, la création de nouvelles images intégrant des transformations est réalisée, ce qui permet également d'optimiser la généralisation.

Pour chaque image de chaque catégorie du dataset *Train* est créé une image pour chaque transformation suivante :

- Rotation / Inclinaison (modifie la géométrie d'une image en la déformant) / Cisaillement / Brilliance (modification aléatoire) / Distorsion (modification aléatoire) / Contraste (modification aléatoire) / Zoom (modification aléatoire)

#### 3.3 – Méthodologie de modélisation :

Le modèle a proprement parlé est basé sur une architecture PPNet qui combine une représentation explicable avec la classification basée sur des prototypes.

Le transformateur pour extraire les *Features* choisi ici est : *cait\_xxs24\_224.pth* (ViT backbone) qui est une variante légère du modèle **CaiT** (*Class-Attention in Image Transformers*), optimisé pour la classification d'images. C'est un modèle pré-entraîné pour la classification d'images, prêt pour du fine-tuning sur des tâches spécifiques.

La modélisation complète est réalisée en 3 phases :

1. Phase préliminaire – **Warm** :

Le but est d'ajuster les couches d'extraction de *Features* (le backbone *CaiT*) pour qu'elles apprennent des représentations utiles pour les prototypes.

2. Phase conjointe – **Joint** :

Entraîner les prototypes et les couches finales conjointement, le modèle apprend à associer ces prototypes aux classes correspondantes.

3. Phase finale - **Push & Fine-Tune** :

**Push** : Chaque prototype est "**poussé**" vers les échantillons d'entraînement qu'il représente le mieux, afin de maximiser leur similarité. Cela les rend plus spécifiques et explicables.

**Fine-tune** : Une fois les prototypes ajustés, les couches finales sont réentraînées pour affiner la classification globale du modèle.

À ce stade, les prototypes sont finalisés et deviennent les vecteurs représentatifs des motifs discriminants pour chaque classe.

### 3.4 – Evaluation et optimisation :

Critères d'évaluation : ProtoViT utilise plusieurs termes de pertes (L) pour optimiser ses performances et son interprétabilité :

$$L_{\text{total}} = L_{\text{CE}} + \lambda_1 L_{\text{Clst}} + \lambda_2 L_{\text{Sep}} + \lambda_3 L_{\text{Coh}} + \lambda_4 L_{\text{Orth}}$$

$L_{\text{CE}}$  : perte d'entropie croisée pour la classification

$L_{\text{Clst}}$  (Cluster Loss) : encourage les prototypes à être proches des tokens de la bonne classe

$L_{\text{Sep}}$  (Separation Loss) : sépare les prototypes des mauvaises classes

$L_{\text{Coh}}$  (Coherence Loss) : renforce la similarité entre les sous-prototypes d'un même prototype

$L_{\text{Orth}}$  (Orthogonality Loss) : encourage chaque prototype à capturer une caractéristique distincte.

Hyperparamètres clés et processus d'optimisation :

- Apprentissage prototypes :
  - $K = 1$  : représente le nombre de sous-prototypes utilisés pour chaque prototype
  - $\text{Sig\_temp} = 100$  : régularisation des sous-prototypes
- Optimisation des pertes :
  - $\lambda_1, \lambda_2, \lambda_3, \lambda_4$  : coefficients qui pondèrent l'importance des différentes composantes de la perte totale
- Taux d'apprentissage : il existe différents paramètres d'apprentissage pour chaque phase de l'architecture *PPNet*, ils influencent directement la convergence de l'optimisation :

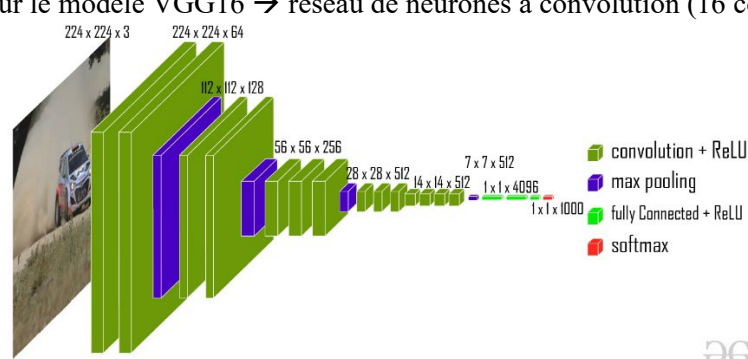
Composant	Taux d'apprentissage initial	Phase
Features (Backbone)	2e-5 (joint), 1e-6 (warm)	Warm et Joint
Prototype_vectors	3e-3	Joint et Slots
Patch_select (Slots)	5e-5	Stage 2 (Slots)
Last_layer (dernière couche)	5e-4	Fine-Tune (Last layer)

- Durée de l'entraînement : le nombre d'époques pour l'entraînement (*Warm* et *Joint*)
  - $\text{num\_joint\_epochs} = 12$
  - $\text{num\_warm\_epochs} = 10$

#### 4. Synthèse des résultats :

Le but principal de l'exercice est d'améliorer la précision de classification également de réduire le temps d'entraînement.

Afin de pouvoir comparer les résultats de la modélisation précédente avec celle-ci, un rappel des techniques :

- Technique 1 : [VGG16](#)
    - Basée sur le modèle VGG16 → réseau de neurones à convolution (16 couches)
- 
- Data Augmentation : oui
  - Couche Dense (7, activation='softmax') couche de classification
  - Datasets : Train: 630, Val: 210, Test: 210
- Technique 2 : [ProtoViT](#)
  - Basée sur une architecture PPNet, utilisation d'un backbone transformateur pré-entraîné CaiT-XXS24-224 et combinaison avec une approche de classification par prototypes.
  - Data Augmentation : oui
  - Datasets : Train: 630, Val: 210, Test: 210

#### Comparaison des résultats :

VGG16

Accuracy: 0.8333				
Classification Report:				
	precision	recall	f1-score	support
Baby Care	0.67	0.81	0.73	27
Beauty and Personal Care	0.70	0.90	0.79	21
Computers	0.92	0.92	0.92	38
Home Decor & Festive Needs	0.75	0.70	0.72	30
Home Furnishing	0.87	0.74	0.80	35
Kitchen & Dining	1.00	0.81	0.89	26
Watches	0.94	0.94	0.94	33
accuracy			0.83	210
macro avg	0.84	0.83	0.83	210
weighted avg	0.85	0.83	0.84	210

ProtoViT

Accuracy: 0.8810				
Classification Report:				
	precision	recall	f1-score	support
Baby Care	0.92	0.77	0.84	30
Beauty and Personal Care	0.96	0.80	0.87	30
Computers	0.85	0.93	0.89	30
Home Decor & Festive Needs	0.76	0.87	0.81	30
Home Furnishing	0.82	0.90	0.86	30
Kitchen & Dining	0.93	0.93	0.93	30
Watches	0.97	0.97	0.97	30
accuracy			0.88	210
macro avg	0.89	0.88	0.88	210
weighted avg	0.89	0.88	0.88	210

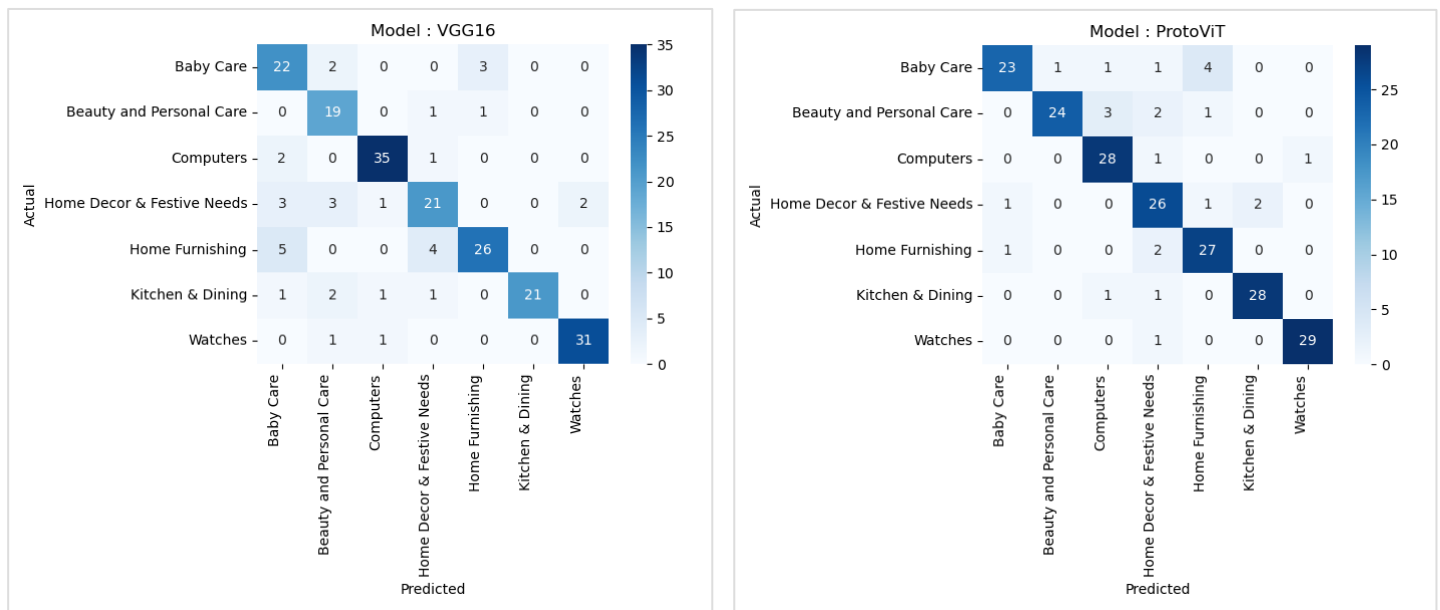
On trouve des F1-scores relativement bas sur les classes « Baby Care » et « Home Decor & Festive Needs » du côté *VGG16*, là où *ProtoViT* améliore significativement les confusions. Les classes « Kitchen & Dining » et « Watches » affichent des scores élevés côté *VGG16* avec pourtant trop d'images « Watches » et pas assez de « Kitchen & Dining », *ProtoViT* améliore encore ici le F1-score.

Modèle	Accuracy	Macro F1-score	Classes améliorées
VGG16	83.33%	0.83	Watches, Kitchen&Dining
ProtoViT	88.10%	0.88	Baby Care, Beauty&Personal

*ProtoViT* améliore significativement la performance globale (+4.77% d'accuracy) tout en offrant une meilleure gestion des confusions entre classes proches, comme « Home Decor » et « Baby Care ».

### Analyse qualitative des résultats :

*Rappel : 30 images par catégories*



En observant les matrices de confusion, on peut souligner les points suivants :

- Avec *VGG16*, les erreurs sont plus visibles dans des classes comme « Home Decor & Festive Needs » et « Baby Care », où il y a plus de confusions avec des classes voisines.
- **ProtoViT** réduit significativement ces confusions, comme pour « Home Decor & Festive Needs » (moins d'erreurs avec Baby Care et Home Furnishing).

### Conclusion d'analyse :

*ProtoViT* surpasse clairement *VGG16* en termes de performances globales (accuracy +4.77%, macro F1-score +5 points) et de robustesse pour les classes plus complexes.

Ses avantages incluent :

- Une meilleure gestion des confusions entre classes proches.
- Une généralisation plus robuste, aidée par la puissance des Transformers.

**Recommandation :** *ProtoViT* démontre son potentiel pour les applications en production grâce à son interprétabilité et sa performance. Cependant, ses limites, notamment la complexité de son entraînement, nécessitent des ajustements avant un déploiement à grande échelle.



## 5. Analyse de la Feature Importance Globale et locale :

Le modèle intègre des scripts afin de réaliser la Feature Importance avec une interprétabilité facilitée en affichant graphiquement les patches d'analyse sur les prototypes et les images testées afin d'expliquer ce qu'interprète le modèle avec la méthode : *ceci ressemble à cela*.

### *Scripts :*

- `analysis_settings.py` : contient emplacements, nom du modèle, variables, ...
- `global_analysis.py` : script de l'analyse de Feature Importance Globale
- `local_analysis.py` : script de l'analyse de Feature Importance Locale

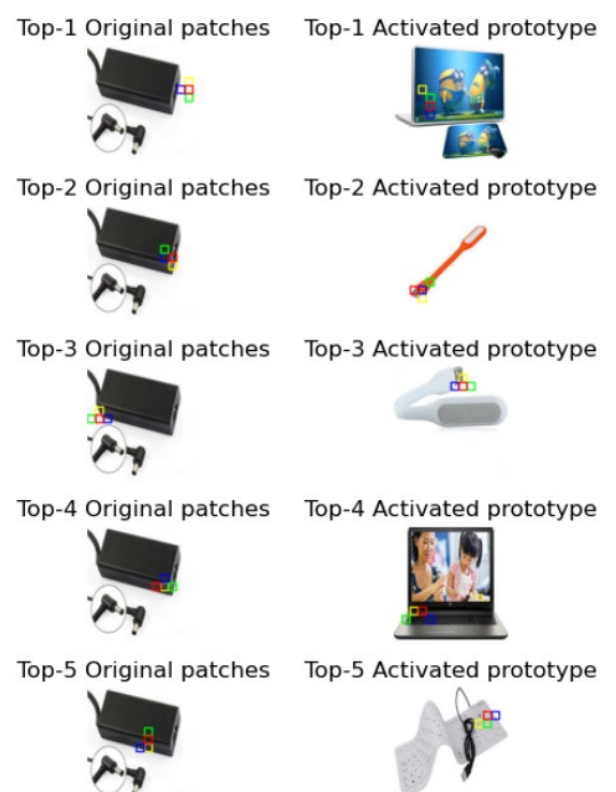
### • Feature Importance globale :

Ici le prototype n°85 (qui fait partie de la classe 6) et les 5 images qui lui ressemblent le plus :



### • Feature Importance locale :

En lançant le script d'analyse de la Feature Importance locale, après avoir sélectionné une image test (ici un « transformateur PC »), on nous montre d'un côté (à gauche) les patches qui ont influencés le plus la classification et à côté (à droite) le prototype qui a été le plus activé et qui participera fortement à la classification.



## 6. Limites et améliorations possibles :

### Limites du modèle :

Interprétation : Le modèle offre une interprétation visuelle des résultats via des scripts, mais certaines similarités restent difficiles à verbaliser.

Location misalignment : La méthode travaille avec des ViT qui utilisent des mécanismes d'attention qui captent des informations sur toute l'image ce qui peut conduire à des activations incorrectement alignées, surtout si les informations locales sont insuffisantes.

Prototypes : Malgré une optimisation réalisée pendant l'entraînement, on trouve au final des prototypes redondants ou inutilisés.

Complexité de la méthode : La méthode de ProtoViT est certes performante, mais complexe à adapter à un nouveau dataset. Le nombre d'étapes et le nombre d'hyperparamètres à optimiser complexifie énormément la mise en œuvre de la méthode et la rend très chronophage.

### Optimisation de l'entraînement :

- Réaliser un GridSearchCv sur les hyperparamètres les plus importants aux différentes étapes de l'entraînement (nombre de prototypes, nombre de patches, learning\_rate, coefs des loss, epochs,...)

### Optimisation de la performance du modèle :

- Supprimer ou fusionner les prototypes inutilisés ou redondants.
- Réentraîner le modèle pour réajuster les connexions après suppression ou fusion.
- Ajouter un Cross Validation à l'entraînement du modèle.
- Ajouter des données au dataset d'origine.

## References

1. Chiyu Ma and Jon Donnelly and Wenjun Liu and Soroush Vosoughi and Cynthia Rudin and Chaofan Chen. (2024). Interpretable Image Classification with Adaptive Prototype-based Vision Transformers. *Arxiv*, 20722(1). URL : <https://arxiv.org/abs/2410.20722>
2. chiyum609/ProtoViT · Hugging Face. (n.d.). Retrieved January 6, 2025, from <https://huggingface.co/chiyum609/ProtoViT>
3. facebookresearch. (n.d.). deit/README\_cait.md at main · facebookresearch/deit. GitHub. Retrieved January 6, 2025, from [https://github.com/facebookresearch/deit/blob/main/README\\_cait.md](https://github.com/facebookresearch/deit/blob/main/README_cait.md)