Note méthodologique

Vision Transformer (ViT)

Introduction

Cette note méthodologique a pour objectif d'aborder et de tester les **Vision Transformers** (**ViT**), une architecture introduite en **2020** qui a révolutionné les approches traditionnelles en vision par ordinateur. Après une présentation de cette architecture, nous nous concentrerons à l'adapter à notre preuve de concept (**POC**) en précisant les choix de modélisation et l'utilisation de notre dataset spécifique.

Nous proposerons ensuite une synthèse des résultats obtenus, accompagnée d'une analyse des performances du modèle.

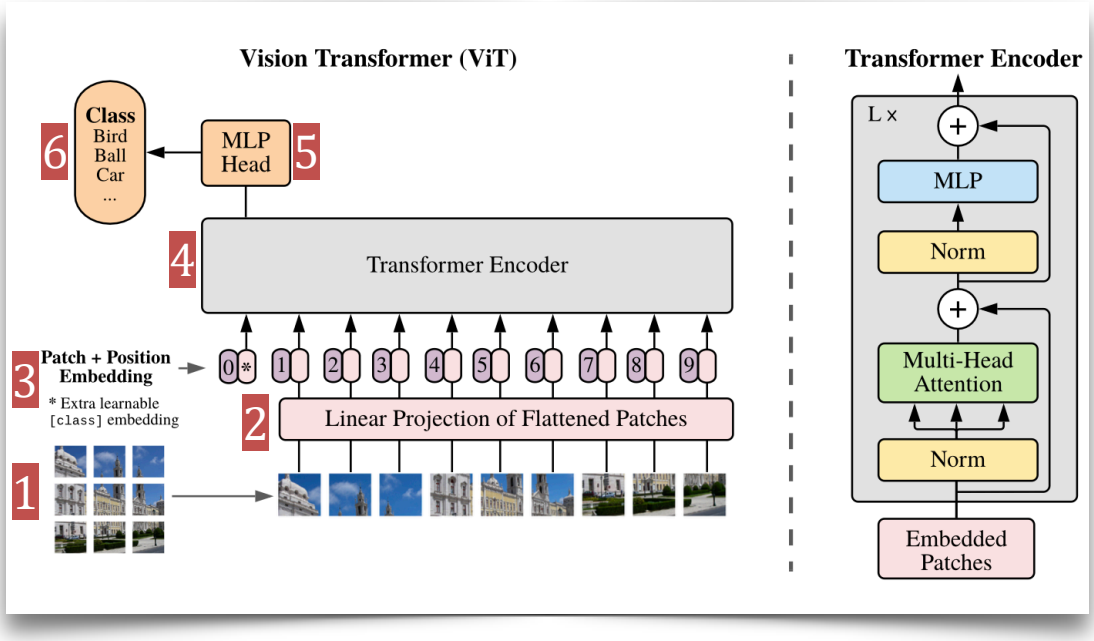
Enfin, nous aborderons les limites que nous avons rencontrées pour ce POC, ainsi que les axes d'amélioration possibles, en prenant en compte les contraintes propres aux données utilisées et au modèle ViT.

Concept du Vision Transformer (ViT)

Contrairement aux architectures classiques comme les réseaux convolutifs (CNN), qui reposent sur des convolutions pour extraire des caractéristiques locales des images, les ViT implémentent l'architecture des **Transformers**.

Les Transformers étant initialement développés pour les tâches de traitement du langage naturel (**NLP**), ils ont été introduits et adaptés au traitement d'image tout en conservant le principal aspect de cette architecture, à savoir l'**auto-attention**.

Processus de classification par le ViT



1 Entrée de l'image : l'image est d'abord divisée en petits carrés ou **patchs** de taille fixe (ex : 16x16). Chaque patch devient un élément unique qui sera traité indépendamment comme un "token" à la manière d'un mot dans le NLP.

2 Projection linéaire des patchs : chaque patch est aplati en un vecteur (vecteur d'**embedding**) pour être traité par les couches du transformer.

3 Ajout du Positional Encoding et du token de classification CLS : on ajoute un encodage de position à chaque vecteur d'embedding pour indiquer la position de chaque patch dans l'image d'origine. Ceci permet au modèle de comprendre la structure spatiale de l'image, c'est-à-dire, comment les patchs sont arrangés les uns par rapport aux autres. Le jeton CLS est utilisé pour agréger les informations globales de l'entrée et permettre au modèle de faire une prédiction de classification.

4 Passage à travers le Transformer Encoder : les vecteurs embedding ainsi que le jeton CLS, passent ensuite à travers plusieurs couches du Transformer Encoder qui est un

composant similaire au transformer utilisé dans le NLP, avec des couches de **Multi-Head Self-Attention** suivies de **normalisation** et de **Feed-Forward**. Durant cette phase, le token CLS est mis à jour de manière à résumer la représentation globale de l'image.

5 Passage à travers d'un MLP (Multi-Layer Perceptron) : après transformation par le transformer, le token de classification est utilisé pour faire une prédiction via un **perceptron multi-couche**.

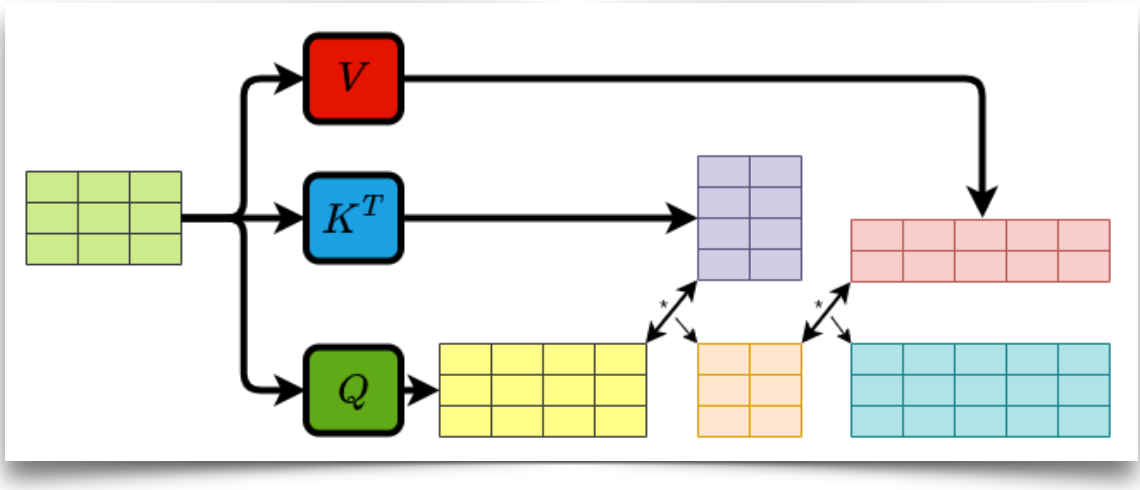
6 Prédiction finale : les **logits** produits par le MLP sont ensuite normalisés avec une fonction **softmax** pour obtenir des probabilités pour chaque classe. La classe avec la probabilité la plus élevée est choisie comme la classe prédite.

Principe fondateur du ViT : l'attention

L'**attention** permet de capturer les relations entre les différentes parties d'une image en calculant des scores qui indiquent l'importance de chacune des parties de l'image par rapport aux autres. Sa formule de calcul est la suivante :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Le calcul de l'attention s'effectue à partir de trois matrices : ***Q*** (queries), ***K*** (keys), et ***V*** (values), qui sont générées par trois transformations linéaires distinctes appliquées aux embeddings d'entrée, à l'aide des matrices de poids respectives W_q , W_k , et W_v. Les scores d'attention sont obtenus en calculant le produit QK^T, normalisé par la racine carré de dk (la dimension des keys), puis en appliquant une fonction softmax. Ces scores sont ensuite utilisés pour pondérer les valeurs V, donnant ainsi la sortie finale de l'attention.

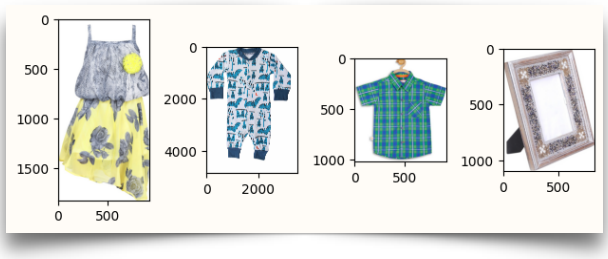


Processus de calcul de l'attention

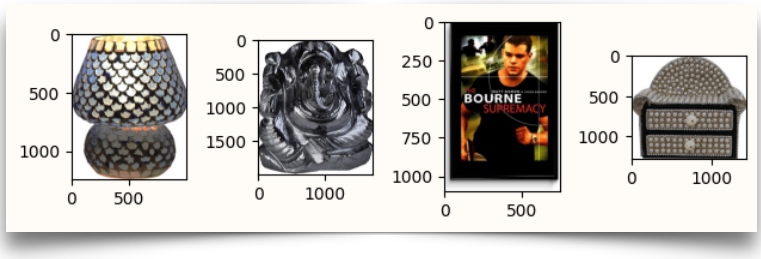
Dataset retenu pour le PoC

Pour essayer le modèle Vision Transformer nous nous basons sur un jeu de données sur lequel nous avons déjà effectué des tests à l'aide du CNN (modèle VGG16). L'utilisation du même dataset vise à comparer équitablement les performances du CNN et du ViT.

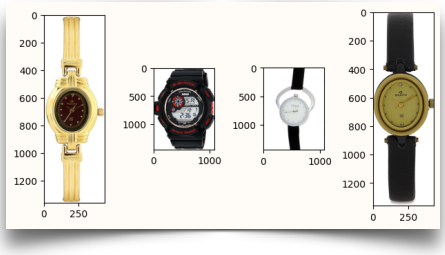
BABY CARE



HOME DECOR & FESTIVE NEEDS



WATCHES



Ces données sont des images de produits pouvant provenir d'une plateforme e-commerce. Elles sont au nombre de 1050 et sont réparties en 7 catégories de produits.

Lors des essais avec le CNN, nous avions mis les images au format carré (1:1) pour optimiser les performances du modèle. Nous appliquons la même approche pour les tests avec le ViT.

Modélisation

Pour la modélisation, les librairies fournies par **Hugging Face** ont été choisies pour plusieurs raisons.
Tout d'abord, la plateforme Hugging Face propose un vaste catalogue de modèles pré-entraînés, notamment pour les transformers comme ViT. De plus, Hugging Face bénéficie d'une intégration fluide avec des frameworks comme TensorFlow et PyTorch ce qui simplifie un peu plus la modélisation.
Enfin, l'interface de la plateforme est conviviale et est bien documentée.

Concernant le modèle ViT, le choix s'est porté sur ***google/vit-base-patch16-224*** fourni par Google à travers la plateforme Hugging Face. Ce modèle utilise des patchs de 16x16 pixels et produit des images de taille 224x224 en entrée.

Préparation des datasets d'entraînement et de test

Les images disponibles sont toutes stockées dans un seul dossier. Nous utilisons un dataframe, qui contient les labels ainsi que les chemins d'accès aux images, pour effectuer la séparation des données. En utilisant la fonction **train_test_split**, nous obtenons deux ensembles distincts : un ensemble d'entraînement et un ensemble de test.

Pour rendre ces données compatibles avec le modèle ***google/vit-base-patch16-224***, il est essentiel de préparer les inputs en les adaptant au format attendu, notamment en redimensionnant les images à 224x224 pixels grâce à la classe **ViTImageProcessor** de la librairie **transformers** de Hugging Face.
Aussi, la mise en place de techniques de **Data Augmentation** est nécessaire afin de diversifier les échantillons d'entraînement, ce qui permet de réduire le surapprentissage et d'améliorer la robustesse du modèle. Cela inclut des transformations telles que la rotation, le recadrage, le flip horizontal, ou encore les ajustements de luminosité et de contraste, afin de mieux généraliser le modèle aux variations des images réelles.

```python
augmentations = A.Compose([
    A.RandomRotate90(),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.1, rotate_limit=15, p=0.5),
    A.RandomBrightnessContrast(p=0.5)
])
```

Objet pour la Data Augmentation avec la librairie **albumentations**

Entraînement et évaluation

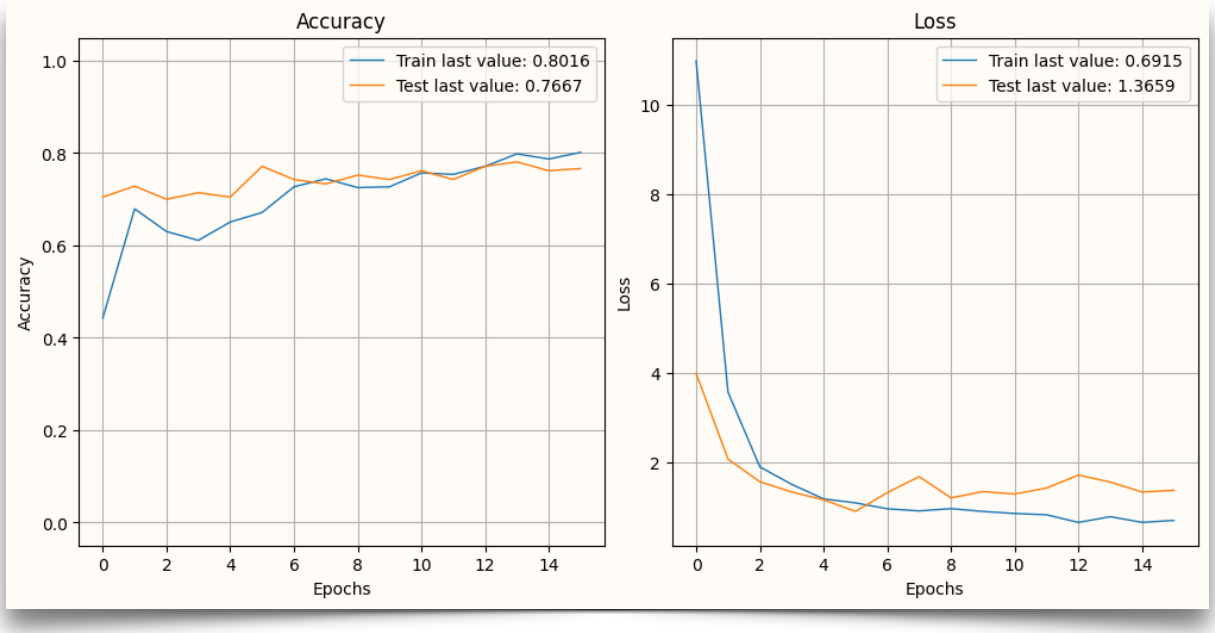
Avec Hugging Face, il est relativement simple d'entraîner et d'évaluer le modèle ViT.
Grâce à la classe **TrainingArguments**, nous pouvons facilement paramétrer les aspects essentiels de l'entraînement, tels que le répertoire pour les résultats (output_dir), la stratégie d'évaluation et de sauvegarde (eval_strategy, save_strategy), ainsi que la taille des batchs et le nombre d'epochs.
De plus, il est possible de définir des métriques personnalisées, telle que l'**accuracy** à travers le paramètre **compute_metrics** du trainer. Ceci nous permet de suivre d'autres indicateurs que la perte (**loss**) pendant l'entraînement. En utilisant ces arguments et ces paramètres nous pouvons configurer l'entraînement en quelques lignes seulement, tout en ayant accès à des informations clés dans le **log_history** du trainer.

```python
# Arguments d'entraînement
training_args = TrainingArguments(
    output_dir='./train_results',              # Répertoire pour sauvegarder le meilleur modèle
    logging_dir='./train_logs',                # Répertoire pour les logs de performance
    eval_strategy="epoch",                     # Évaluer à chaque époque
    save_strategy="epoch",                     # Sauvegarder à chaque époque
    save_total_limit=1,                        # Ne conserver que le meilleur modèle
    metric_for_best_model="eval_loss",         # Utiliser la perte d'évaluation pour déterminer le meilleur modèle
    load_best_model_at_end=True,               # Charger le meilleur modèle à la fin de l'entraînement
    per_device_train_batch_size=16,            # Taille des batches pour l'entraînement
    per_device_eval_batch_size=16,             # Taille des batches pour l'évaluation
    num_train_epochs=10,                       # Nombre d'époques d'entraînement
    logging_steps=10,                          # Fréquence d'enregistrement des logs
    learning_rate=5e-5,
    warmup_steps=500                           # Nombre d'étapes de "warmup" pour ajuster progressivement le learning rate
)

# Fonction pour récupérer l'accuracy lors de la phase d'entraînement
def compute_metrics(prediction):
    preds = np.argmax(prediction.predictions, axis=1)
    accuracy, _, _, _ = precision_recall_fscore_support(prediction.label_ids, preds, average='weighted')
    acc = accuracy_score(accuracy.label_ids, preds)
    return {"accuracy": acc}

# Objet trainer pour entraîner le modèle en tenant compte des paramètres d'apprentissage et de test
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics
)

# Lancer l'entraînement
best_model = trainer.train()
```

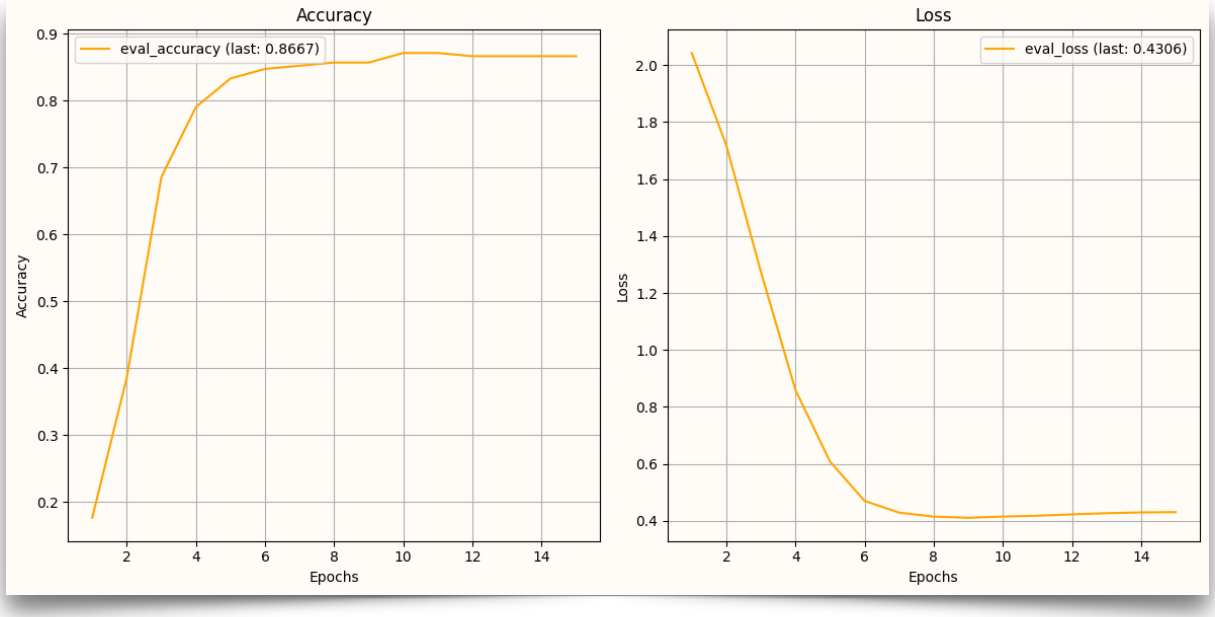
Trainer **paramétré**, **entraîné** et **évalué**

Synthèse des résultats

Le modèle Vision Transformer (**ViT-B/16**) que nous avons utilisé pour le PoC présente de meilleures performances que le modèle CNN (**VGG16**) utilisé dans un précédent usage, ce en suivant des processus d'entraînement et d'évaluation équivalents. Cela montre son potentiel supérieur pour des tâches de classification d'images dans notre contexte.



Modèle **VGG16** : courbe d'**accuracy** et de **loss** durant l'**entraînement** et la **validation**

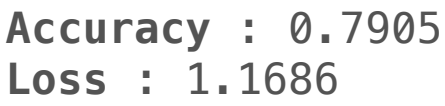


Modèle **ViT-B/16** : courbe d'**accuracy** et de **loss** durant la **validation**

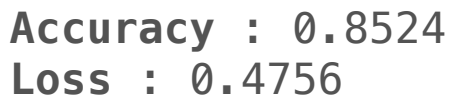
Les performances du ViT surpassent celles du CNN tant en validation qu'en évaluation sur le dataset de test. En validation, l'accuracy du ViT atteint 0.8667, contre 0.7667 pour le

CNN. Concernant le loss, le ViT affiche une valeur de 0.4306, soit plus de deux fois inférieure à celle du CNN, qui reste autour de 1.4.

Les résultats d'évaluation sur le dataset de test montrent également une tendance similaire au profit du modèle basé sur les Transformers.



VGG16 : évaluation sur le dataset de test



ViT-B/16 : évaluation sur le dataset de test

Limites et améliorations possibles

Bien que notre modèle ViT ait montré de bonnes performances, il reste évidemment des possibilités d'amélioration en intervenant sur certains aspects tels que l'optimisation des hyperparamètres, la qualité des données et l'exploration d'autres architectures s'appuyant sur les Transformers.

Optimisation des hyperparamètres

Une optimisation des hyperparamètres, notamment via des méthodes de recherche comme GridSearch ou RandomSearch, pourrait améliorer les performances du modèle. Cela permettrait d'explorer des valeurs optimales pour des paramètres clés comme le learning rate et la taille des batchs.

Amélioration des données

☞ Taille limitée du jeu de données

Le dataset utilisé est relativement petit, avec seulement 1050 images, ce qui réduit encore davantage la quantité de données réellement utilisée pour l'entraînement après le découpage en ensembles d'entraînement et de test. Un jeu de données plus volumineux pourrait permettre au modèle d'apprendre mieux et de mieux généraliser.

☞ Variabilité des catégories

Certains produits peuvent visuellement se ressembler et être placés dans la même catégorie en raison de caractéristiques d'image similaires alors que leur classes sont différentes. Cela pourrait limiter la capacité du modèle à différencier correctement ces produits. En ajustant les catégories pour mieux refléter la diversité visuelle des produits, il serait possible d'améliorer la performance de la classification.

☞ Amélioration la data augmentation

L'augmentation des données (data augmentation) pourrait être renforcée pour simuler une plus grande variété d'images d'entraînement.

Exploration d'architectures plus avancées

Au-delà de l'optimisation citées précédemment, d'autres variantes de Vision Transformers ou des approches hybrides combinant des CNN avec des Transformers pourraient être envisagées. Cela permettrait de bénéficier des avantages des deux architectures, surtout dans le cas de jeux de données relativement petits où un modèle uniquement basé sur les Transformers peut avoir du mal à converger efficacement.