Classification Multi-tâches de Cartes Magic: The Gathering

Chaumont Nicolas, Richard Clément, Trebern Ewan

30 janvier 2026

Résumé

Ce projet vise à automatiser la classification de cartes Magic : The Gathering via l'apprentissage profond. Nous avons constitué un jeu de données équilibré d'images via l'API Scryfall pour entraîner des réseaux de neurones afin de comparer leurs performances. L'architecture a été modifiée en Multi-Têtes pour prédire simultanément le Type, la Rareté et la Couleur d'une carte à partir d'une seule photo. Les résultats valident l'efficacité de cette approche pour la reconnaissance d'objets complexes.

1 Introduction

Magic : The Gathering, aussi appelé plus couramment "Magic", est un jeu de carte à jouer et à collectionner inventé par Richard Garfield en 1994. Plus de 31 000 cartes différentes existent à l'heure actuelle avec lesquelles les joueurs doivent construire des decks pour s'affronter. Avec autant de cartes, une sortie continue d'extension et donc encore des nouvelles cartes à ajouter à sa collection, on peut être vite perdu. Pour simplifier l'aspect de la collection (ou de l'organisation du stockage), on veut aider les joueurs à trier leurs cartes. Pour cela, on veut créer un réseau de neurones qui va venir détecter des caractéristiques de chaque photos de cartes qu'on lui donne afin d'éviter le tri manuel de la collection.

Pour l'instant, on cherche à créer un réseau de neurones réussissant à trouver simultanément 3 attributs différents, à savoir :

— Le type de la carte
— La rareté de la carte
— L'identité de couleur de la carte

2 Les données

2.1 Les attributs

On va ici détaillé comment retrouver les attributs recherché.

2.1.1 Le type

Le type de la carte est simplement marqué dessus, au milieu à gauche. Ainsi, nous allons traité les terrains, les créatures, les artefacts, les enchantements, les planeswalkers, les éphémères et les rituels (voir exemple plus bas).
Magic possède 2 types de cartes supplémentaire (les batailles et les tribaux) mais ces cartes n'existent pas en assez grandes quantité pour pouvoir être traité (36 cartes bataille et 62 cartes tribal).

(a) Créature

(b) Terrain

(c) Planeswalker

(a) Artefact

(b) Rituel

(c) Enchantement

(d) Éphémère

2.1.2 La rareté

Comme dans tous les jeux à collectionner, chaque carte à une rareté qui lui est attribué. Elles sont reconnaissable grâce à la couleur du signe de l'extension au milieu à droite de la carte.
En ordre de rareté nous avons : commun en noir, non-commun en argenté (uncommon ou unco), rare en doré puis mythique rare en orange (aussi appelé mythique).

FIGURE 3 – De commun à mythique.

Il existe encore d'autres cartes à la rareté encore plus grande (Timeshifted, Masterpiece, Special) mais elles ne sont pas assez représenté pour les inclure dans le data set (moins de 400 cartes).

2.1.3 L'identité de couleur

Dans Magic : The Gathering, la Couleur d'une carte est définie strictement par son coût de mana (les symboles en haut à droite) ou sa pastille de couleur, ce qui détermine ses interactions immédiates en jeu, comme la protection ou les sorts de destruction. En revanche, l'identité de couleur est un concept élargi utilisé principalement pour la construction de decks (notamment en format Commander) : elle englobe la couleur de la carte ainsi que tous les symboles de mana apparaissant dans son texte de règles. Ainsi, une créature peut être considérée comme blanche sur le champ de bataille, mais posséder une identité blanche et rouge si son texte contient un symbole de feu, ce qui restreint les decks dans lesquels elle peut être incluse.

2.1.4 Le nom

Comme dans beaucoup d'autres jeux de cartes (Pokémon et Yu-Gi-Oh), les cartes ont un nom indiqué en haut de la carte. Cela est utile pour identifier une carte. Nous nous sommes attaqué à cet attribut après les autres car il nécessitait d'utiliser une autre technologie que CNN, c'est pourquoi le nom est traité à part dans le projet.

Dans notre cas, on demande l'identité de couleur plutôt que simplement la couleur car elle est plus dure à trouver et cela en fait un objectif plus intéressant (il faut donc analyser toute la carte et pas seulement son coût en haut à droite).

2.2 API Scryfall

Pour constituer notre base de données d'apprentissage, nous avons choisi d'utiliser Scryfall, qui est actuellement la base de données la plus complète et la mieux maintenue par la communauté Magic : The Gathering. Contrairement à une approche de web scraping classique qui aurait été lente et instable, Scryfall met à disposition des fichiers "Bulk Data" contenant les métadonnées de plus de 20000 cartes uniques.

L'intérêt principal de cette source est la richesse des informations structurées : chaque entrée JSON fournit non seulement l'URL de l'image en haute qualité, mais aussi des étiquettes précises (Vérité Terrain) pour nos trois objectifs de classification :

— La ligne de type complète (type_line) pour extraire le type principal.
— L'identité couleur (color_identity) pour la classification multi-label.
— La rareté (rarity).

Nous avons développé un script Python capable de parser ce fichier volumineux et de télécharger sélectivement les images nécessaires.

2.3 Nettoyage des données

Les données brutes issues de Scryfall contiennent beaucoup de "bruit" qui pourrait nuire à l'apprentissage du modèle. Une phase rigoureuse de nettoyage a été nécessaire pour ne conserver que les cartes pertinentes.

Nous avons appliqué les filtres suivants :

1. **Exclusion des éléments non-jouables** : Le jeu contient des jetons (tokens), des emblèmes ou des cartes promotionnelles (art series) qui n'ont pas le format standard d'une carte Magic. Nous avons filtré ces éléments via le champ layout pour éviter que le modèle n'apprenne des caractéristiques visuelles incohérentes.
2. **Filtrage des Types** : Nous nous sommes concentrés sur les 7 types principaux du jeu (Creature, Instant, Sorcery, Enchantment, Artifact, Planeswalker, Land). Les types exotiques ou obsolètes ont été écartés.

Cette étape nous a permis de passer d'un fichier brut hétérogène à un ensemble de candidats propres et standardisés.

2.4 Équilibrage des données

L'un des défis majeurs du Deep Learning est d'éviter le biais d'apprentissage lié aux classes majoritaires. Dans Magic, certaines couleurs (comme le Vert ou le Blanc) et certains types (comme les Créatures) sont sur-représentés par rapport aux cartes Incolores ou aux Planeswalkers. Si nous avions sélectionné des cartes aléatoirement, le modèle aurait simplement appris à prédire la classe majoritaire pour minimiser son erreur, sans analyser l'image.

Pour contrer ce problème, nous avons implémenté une stratégie de quotas stricts lors de la sélection des images :

- **Distribution des couleurs** : Nous avons imposé une répartition équilibrée : 15% pour chaque couleur (Blanc, Bleu, Noir, Rouge, Vert), 20% pour les cartes multicolores et 5% pour les incolores.
- **Sélection aléatoire** : Au sein de chaque catégorie de quota, les cartes sont mélangées aléatoirement avant sélection pour garantir la variété des éditions et des styles artistiques. On réparti ainsi les couleurs des cartes afin que les modèles soit préparés à tous les

Le data set final de 7000 images a ensuite été divisé en trois sous-ensembles distincts pour garantir une évaluation honnête des performances : un ensemble d'entraînement (5000 images), un ensemble de validation pour ajuster les hyperparamètres (1000 images), et un ensemble de test pour l'évaluation finale (1000 images).

3 EfficientNet-B0 (CNN)

Pour établir notre première ligne de performance, nous avons opté pour une architecture de type CNN (Convolutional Neural Network). Les CNN sont historiquement les modèles de référence pour la vision par ordinateur, excellant dans l'extraction de caractéristiques locales (textures, bords, formes) grâce à leurs filtres convolutifs. On commence un essaie avec le modèle resnet18 qui n'a pas été concluant avec de très mauvais résultats. Cela est dû au fait que c'est un modèle rapide pour petit data set. on change donc de modèle :

3.1 Choix du Backbone : EfficientNet-B0

Plutôt que d'entraîner un réseau à partir de zéro (ce qui nécessiterait des millions d'images), nous utilisons le Transfer Learning. Nous avons choisi EfficientNet-B0 comme extracteur de caractéristiques (backbone).

Ce modèle se distingue des architectures classiques (comme ResNet) par sa méthode de "Compound Scaling" : il optimise simultanément la profondeur, la largeur et la résolution du réseau pour offrir un excellent compromis entre précision et coût de calcul.

3.2 Architecture Multi-Têtes (Multi-Head)

La particularité de notre implémentation réside dans la modification de la dernière couche du réseau. Un classifieur standard ne renvoie qu'une seule sortie. Pour notre problème multidimensionnel, nous avons conçu une architecture personnalisée à trois têtes parallèles :

- **L'entrée** : L'image de la carte (256×256 pixels).
- **Le Backbone** : EfficientNet extrait un vecteur de caractéristiques de taille 1280.
- **Tête 1 (Type)** : Une couche linéaire (Fully Connected) vers 7 neurones(classification multi-classe).
- **Tête 2 (Rareté)** : Une couche linéaire vers 5 neurones (classification multiclasse).
- **Tête 3 (Couleurs)** : Une couche linéaire vers 6 neurones (classification multi-label).

Voici l'implémentation PyTorch correspondante utilisée :

```python
class MultiOutputModel(nn.Module):
    def __init__(self, num_types, num_rarities, num_colors):
        super().__init__()
        # Chargement du backbone EfficientNet-B0 pré-entraîné
        self.backbone = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.IMAGENET1K_V1)

        # Récupération de la taille des features (1280 pour B0)
        in_features = self.backbone.classifier[1].in_features

        # On remplace le classifier par l'identité pour garder les features brutes
        self.backbone.classifier = nn.Identity()

        # Création des 3 têtes (Heads) distinctes
        self.fc_type = nn.Linear(in_features, num_types)       # Classification Multi-classe
        self.fc_rarity = nn.Linear(in_features, num_rarities)  # Classification Multi-classe
        self.fc_colors = nn.Linear(in_features, num_colors)    # Classification Multi-label

    def forward(self, x):
        features = self.backbone(x)
        # Retourne 3 sorties. Sigmoid est appliqué sur les couleurs pour le multi-label.
        return self.fc_type(features), self.fc_rarity(features), torch.sigmoid(self.fc_colors(features))
```

Listing 1 – Implémentation PyTorch de l'architecture Multi-Têtes avec EfficientNet-B0

3.3 Stratégie d'Entraînement

L'entraînement du modèle repose sur la combinaison de plusieurs fonctions de coût (Loss Functions) adaptées à la nature de chaque tâche :

CrossEntropyLoss pour le Type et la Rareté : Ces attributs sont mutuellement exclusifs (une carte ne peut pas être à la fois "Commune" et "Rare").

BCELoss (Binary Cross Entropy) pour les Couleurs : Il s'agit d'un problème multi-label. Nous traitons chaque couleur (Blanc, Bleu, etc.) comme une probabilité binaire indépendante (p>0.3 ⇒ Présent).

La Loss totale rétro-propagée est la somme simple des trois pertes :

$$\mathcal{L}_{total} = \mathcal{L}_{CE}(y_{type}, \hat{y}_{type}) + \mathcal{L}_{CE}(y_{rar}, \hat{y}_{rar}) + \mathcal{L}_{BCE}(y_{col}, \hat{y}_{col}) \tag{1}$$

Où :

— y représente la vérité terrain (ground truth) et $\hat{y}$ la prédiction du modèle.

— $\mathcal{L}_{CE}$ est la **Cross-Entropy Loss** (utilisée pour le Type et la Rareté), définie pour C classes par :

$$\mathcal{L}_{CE} = -\sum_{c=1}^{C} y_c \log(\hat{y}_c) \tag{2}$$

— $\mathcal{L}_{BCE}$ est la **Binary Cross-Entropy Loss** (utilisée pour les Couleurs), traitée comme une classification multi-label sur K couleurs possibles :

$$\mathcal{L}_{BCE} = -\frac{1}{K}\sum_{k=1}^{K} [y_k \log(\hat{y}_k) + (1 - y_k) \log(1 - \hat{y}_k)] \tag{3}$$

Hyperparamètres et Optimisation : Nous avons utilisé l'optimiseur Adam avec un taux d'apprentissage initial de 10^{-3}. Pour éviter que le modèle ne stagne, nous avons intégré un Scheduler (ReduceLROnPlateau) : si la perte sur le jeu de validation ne diminue pas pendant 3 époques (patience=3), le taux d'apprentissage est divisé par 5. Cela permet au modèle d'affiner ses poids ("Fine-tuning") dans les dernières phases de l'entraînement.

4 Vision Transformer (ViT)

Alors que les réseaux de neurones convolutifs (CNN) dominent la vision par ordinateur depuis 2012, une nouvelle architecture issue du traitement du langage naturel (NLP) a récemment bouleversé le domaine : le Vision Transformer (ViT). Dans cette section, nous explorons cette architecture "État de l'art" (SOTA) pour voir si le mécanisme d'attention offre de meilleures performances sur les cartes Magic que les convolutions classiques.

4.1 Principe de fonctionnement

Contrairement au CNN qui analyse l'image par glissement de filtres (traitement local), le Vision Transformer traite l'image comme une séquence, de la même manière qu'un Transformer (comme BERT ou GPT) traite une phrase.

Notre implémentation repose sur le modèle pré-entraîné `google/vit-base-patch16-224` issu de la bibliothèque Hugging Face Transformers. Le processus de traitement est le suivant :

1. **Patching :** L'image d'entrée $x \in \mathbb{R}^{224\times224\times3}$ est découpée en une grille de patches fixes de 16×16 pixels. Cela génère une séquence de 196 "mots visuels" (14×14 patches).
2. **Embedding linéaire :** Chaque patch est aplati et projeté linéairement dans un espace de dimension constante (le *hidden size*, ici 768).
3. **Token CLS :** Un vecteur spécial apprenable, appelé `[CLS]` token, est ajouté au début de la séquence. C'est ce vecteur unique qui servira, après avoir traversé tout le réseau, à synthétiser l'information globale de l'image pour la classification.
4. **Self-Attention :** La séquence passe à travers 12 couches d'encodeurs Transformer. Grâce au mécanisme de *Self-Attention*, chaque patch peut "regarder" n'importe quel autre patch de l'image, quelle que soit sa distance. Cela permet au modèle de capturer des relations à longue portée (ex : le lien entre la couleur de la bordure et un symbole de mana situé à l'opposé de la carte).

4.2 Adaptation Multi-Têtes du Token [CLS]

Dans la littérature classique, le ViT est utilisé pour une seule tâche de classification. Pour notre problème multi-attributs, nous avons exploité la sortie du token `[CLS]` issue de la dernière couche cachée (`last_hidden_state`).

Ce vecteur de taille 768 contient la représentation sémantique globale de la carte. Nous l'avons connecté à trois projections linéaires indépendantes :

```python
class ViTMTG(nn.Module):
    def __init__(self, num_types, num_rarities):
        super().__init__()
        # Chargement du modèle pré-entraîné Google ViT Base
        self.vit = ViTModel.from_pretrained('google/vit-base-patch16-224')
        hidden_size = self.vit.config.hidden_size # 768 dimensions

        # Têtes de classification connectées à la sortie du Transformer
        self.type_head = nn.Linear(hidden_size, num_types)
        self.rarity_head = nn.Linear(hidden_size, num_rarities)
        self.color_head = nn.Linear(hidden_size, 6)

    def forward(self, x):
        # Passage dans le backbone ViT
        outputs = self.vit(pixel_values=x)

        # Extraction du token [CLS] (premier token de la séquence : index 0)
        # Utilisation de last_hidden_state
        cls_token = outputs.last_hidden_state[:, 0, :]

        # Prédictions parallèles
        return self.type_head(cls_token), self.rarity_head(cls_token), self.color_head(cls_token)
```

Listing 2 – Adaptation du Vision Transformer pour la classification multi-têtes

4.3 Spécificités de l'entraînement (Fine-Tuning)

L'entraînement d'un Transformer est notoirement plus instable que celui d'un CNN. Nous avons dû adapter nos hyperparamètres par rapport à la partie précédente :

- **Optimiseur :** Utilisation de AdamW (Adam with Weight Decay) au lieu d'Adam standard. C'est l'optimiseur recommandé pour les Transformers car il gère mieux la régularisation des poids.
- **Taux d'apprentissage (Learning Rate) :** Réduit drastiquement à $5 \cdot 10^{-5}$ (contre 10^{-3} pour le CNN). Un taux trop élevé détruirait les représentations pré-apprises du modèle sur ImageNet ("Catastrophic Forgetting").
- **Fonction de Coût :** Nous utilisons `BCEWithLogitsLoss` pour les couleurs, qui intègre numériquement la fonction Sigmoid pour une meilleure stabilité numérique lors de la rétropropagation.

5 Résultats et Analyse

5.1 Analyse des Résultats - EfficientNet-B0

Les performances du modèle ont été évaluées sur le jeu de test (1000 images jamais vues) après avoir restauré les poids du "Meilleur Modèle" sauvegardé à l'époque 9.

5.1.1 Performances Quantitatives

Le modèle atteint une bonne précision globale, avec 89.69% des cartes parfaitement identifiées sur les trois critères simultanément.

- **Type (99.00%)** : C'est le score le plus élevé. Le réseau a parfaitement appris à distinguer les structures globales des cartes (la bordure d'un Planeswalker, le fond étoilé d'un Enchantement ou la texture d'un Land). Les erreurs résiduelles concernent probablement des confusions visuelles classiques entre Éphémère et Rituel qui partagent le même gabarit.
- **Couleur (95.50%)** : La gestion multi-label fonctionne efficacement. Le modèle détecte bien les bordures colorées et les teintes dominantes de l'illustration. Les 5% d'erreurs proviennent souvent des cartes des cartes dorées (multicolores).

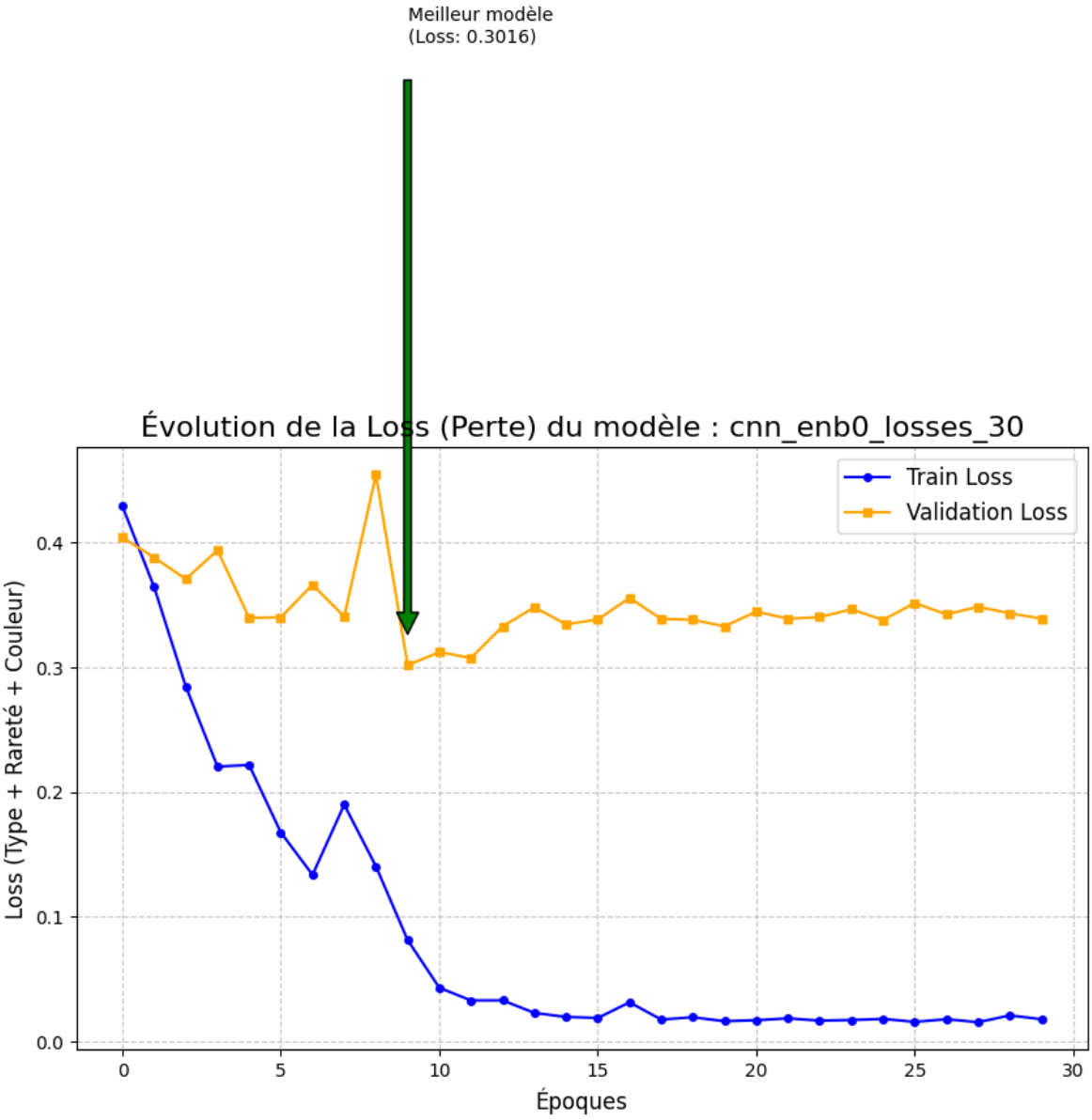

FIGURE 4 – Courbe de la losses du modèle EfficientNet-B0 sur 30 epoch

FIGURE 5 – HeatMap du modèle EfficientNet-B0

— **Rareté (94.69%)** : C'est une surprise très positive. Le symbole de rareté est minuscule (quelques pixels sur une image 256×256). Le fait que le CNN atteigne presque 95% prouve que l'architecture EfficientNet conserve une excellente résolution spatiale dans ses features maps profondes, lui permettant de "zoomer" sur ce détail.

5.1.2 Analyse de la Convergence et du Sur-apprentissage

La courbe d'apprentissage (Figure ci-dessus) nous offre des informations précieuses sur le comportement du modèle durant les 30 époques :

1. **Apprentissage Rapide** : Dès les 5 premières époques, la Loss (perte) d'entraînement chute drastiquement, et la validation suit bien. Le modèle apprend les concepts de base très vite.
2. **Phénomène de Sur-apprentissage (Overfitting)** : À partir de l'époque 10, nous observons une divergence claire.
   — La courbe bleue (*Train Loss*) continue de descendre jusqu'à frôler 0, indiquant que le modèle commence à apprendre le bruit du jeu d'entraînement au lieu de généraliser.
   — La courbe orange (*Validation Loss*) stagne, voire remonte légèrement autour de 0.35.
   Cela indique que le modèle a atteint sa capacité maximale de généralisation avec ce volume de données.
3. **Efficacité de l'arrêt Précoce (Early Stopping)** : La flèche verte indique le point de sauvegarde optimal (Époque 9, Loss Validation : 0.3016). Sans cette stratégie de sauvegarde basée sur la validation, nous aurions utilisé le modèle final (époque 30) qui est moins performant sur des nouvelles données malgré une erreur d'entraînement quasi nulle.

Conclusion sur le CNN : EfficientNet-B0 s'avère être une architecture robuste et performante pour cette tâche. Cependant, l'écart significatif entre le Train et la Validation suggère que le modèle bénéficierait grandement d'une augmentation plus agressive des données (Data Augmentation) ou d'un jeu de données plus vaste pour combler ce fossé de généralisation.

5.2 Analyse des Résultats - Vision Transformer (ViT)

Les performances du modèle ViT ont été évaluées sur le même jeu de test que le CNN. Le modèle retenu est celui de l'époque 15, correspondant à la perte de validation la plus faible (0.9742).

5.2.1 Performances Quantitatives

Contrairement au CNN qui obtenait des scores homogènes, le ViT présente des disparités très marquées selon la tâche :

— Rareté (89.50%) : C'est le point fort inattendu de cette architecture. Le mécanisme d'attention semble capable de focaliser efficacement sur le symbole d'extension, malgré le découpage de l'image en patchs de 16×16 pixels. Le score est proche de celui du CNN (94%).
— Type (83.90%) : Le résultat est correct mais en net retrait par rapport au CNN (98%). Le modèle capture la structure globale mais semble peiner à généraliser sur des mises en pages (layouts) variées.

— Couleur (65.40%) : C'est l'échec principal de cette architecture sur notre jeu de données. Une chute de 30 points par rapport au CNN suggère que le modèle n'arrive pas à corréler la teinte de la fine bordure extérieure avec l'identité couleur.

5.2.2 Analyse de la Dynamique d'Apprentissage

La courbe d'apprentissage (Figure 6) révèle un comportement typique d'un Transformer entraîné sur peu de données :

1. Sur-apprentissage Massif (Overfitting) : Nous observons un écart énorme entre la courbe d'entraînement (bleue), qui descend rapidement vers 0, et la courbe de validation (orange) qui stagne autour de 1.0. Cela signifie que le modèle a mémorisé par cœur les images d'entraînement mais échoue à généraliser.
2. Absence de Biais Inductif : Contrairement aux CNN qui possèdent une invariance par translation "innée" (ils savent qu'un objet est le même s'il bouge un peu), le ViT doit apprendre ces règles géométriques à partir de zéro. Avec seulement 5000 images, il n'a pas assez d'exemples pour apprendre ces règles, d'où la mauvaise performance sur les couleurs (qui dépendent de la géométrie de la bordure).

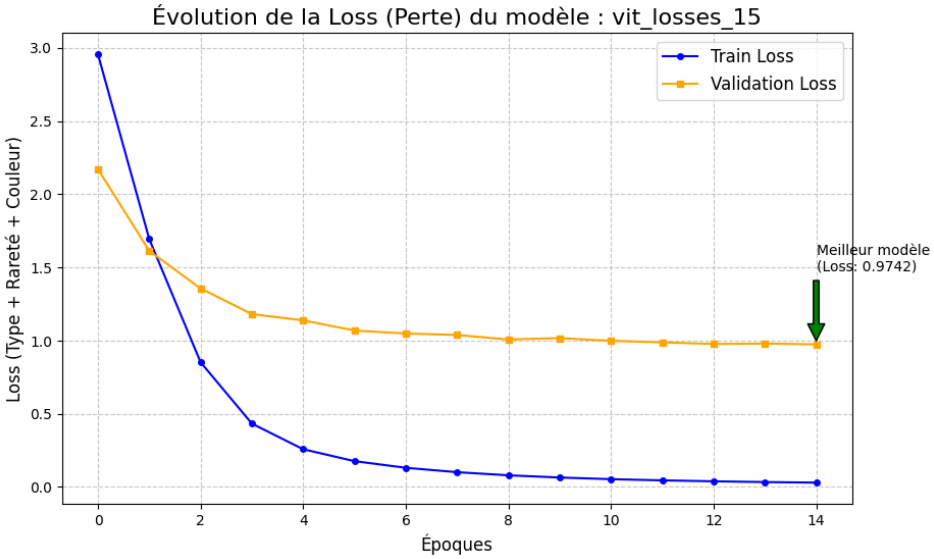

FIGURE 6 – Courbes de perte du ViT : Notez l'écart important entre Train et Val, signe d'un manque de données pour cette architecture.

5.2.3 Conclusion sur le ViT

Bien que le Vision Transformer soit l'état de l'art sur des bases de données massives (JFT-300M, ImageNet-21k), nos résultats confirment qu'il est moins adapté que les CNN pour les petits jeux de données ($<$ 5000 images). L'absence de convolutions locales l'empêche de détecter efficacement les textures fines (comme les bordures colorées) sans un volume de données beaucoup plus conséquent pour compenser.

6 Perspectives et Conclusion

Nous sommes conscient que en l'état, notre projet n'a pas vraiment d'utilité. Pour aller plus loin, nous aurions pu avoir un système de reconnaissance de carte en temps réel devant une webcam/caméra de téléphone (interface avec Streamlit ou OpenCV). On aurait aussi pu essayer de retrouver la carte en trouvant son nom sur la carte. C'est ce que l'on a tenté de faire avec un OCR. Nous avons aussi essayer d'utiliser le ConvNext, un autre CNN censé concurrencer les ViT mais par manque de temps nous l'avons mis de côté. Ce serait tout de même une autre piste d'amélioration.

6.1 Perspectives : Optical Character Detection (OCR)

1. **Un concept prometteur :** Comme nous avons été capables de coder un ViT pour reconnaître des caractéristiques dans l'image (type, rareté et identité de couleur), nous avons voulu faire de même pour la reconnaissance du nom de la carte. En effet, la reconnaissance de caractères se fait également par vision transformer. La principale différence entre les deux est que les transformeur pour image utilisent un encodeur uniquement, alors que ceux pour reconnaître des textes utilisent en plus un décodeur. C'est grâce au décodeur que l'interprétation caractère par caractère est possible.
2. **Ô Rage ! Ô désespoir ! Ô Overfitting ennemi !** Nous avons donc implémenté un OCR, cependant nous avons obtenus des résultats très mitigés. En effet, sur un data set de taille 100, seuls 2 mots, "Plains" et "Island" ont été correctement prédis, mais comme ces mots sont courts et souvent répétés, cela pointe vers une situation d'overfitting. Nous avons remarqués que des textes longs risquaient d'être tronqués. Il y a également des problèmes avec la ponctuation, qui semble faire paniquer le modèle.

```
VRAI: Consuming Sinkhole | PRÉDIT: Coningible
VRAI: Ghitu Fire-Eater | PRÉDIT: Obite Fist
VRAI: Plated Crusher | PRÉDIT: Plate Crush
VRAI: Island | PRÉDIT: Island
VRAI: Desdemona, Freedom's Edge | PRÉDIT: Desperate,' Es
VRAI: Butcher of Malakir | PRÉDIT: Butcher Ma
VRAI: Plains | PRÉDIT: Plains
VRAI: Sedgemoor Witch | PRÉDIT: Smoror
VRAI: Raze the Effigy | PRÉDIT: Rage Effy
VRAI: Seedship Broodtender | PRÉDIT: Seedbrounderunder Petty
VRAI: Dark Intimations | PRÉDIT: Darkstion
VRAI: Akiri, Fearless Voyager | PRÉDIT: A,' Fer,' Vasrag
VRAI: Jeleva, Nephalia's Scourge | PRÉDIT: Te,'' Si-eds
VRAI: Rohirrim Lancer | PRÉDIT: Roiian
VRAI: Steeple Creeper | PRÉDIT: Stone
VRAI: Sacrifice | PRÉDIT: Recruit
VRAI: Mind Sludge | PRÉDIT: Mindudge
VRAI: Psychotic Episode | PRÉDIT: Psychic Horde
VRAI: Battlefield Forge | PRÉDIT: Battle Forge
VRAI: Incriminate | PRÉDIT: Incinate
VRAI: Fanatic of Mogis | PRÉDIT: Fan ofis
VRAI: Epic Confrontation | PRÉDIT: E Confrontation
VRAI: Momentary Blink | PRÉDIT: Moment Hulk
VRAI: Infinite Guideline Station | PRÉDIT: Inite Guide the
VRAI: Hyrax Tower Scout | PRÉDIT: Hydra Sor Sprout
VRAI: Remove Soul | PRÉDIT: Repage
VRAI: Elves of Deep Shadow | PRÉDIT: Five Spell
VRAI: Dragonscale Boon | PRÉDIT: Dragon Doom
VRAI: Despark | PRÉDIT: Desperate
```

Listing 3 – Adaptation du Vision Transformer pour la classification multi-têtes

3. **Comment en est-on arrivé là ?** Cela peut s'expliquer par diverses raisons. Premièrement, nous avons réalisé tardivement qu'un data set optimisé pour de la reconnaissance d'image n'était pas nécessairement optimisé pour une OCR. Un data set avec de légères rotations de textes, une meilleure résolution aurait aidé à mieux entraîner le modèle. Deuxièmement, il n'y a pas assez de gestion d'erreur associé à notre OCR, ce qui nous force à accorder une confiance disproportionnée en la capacité du modèle à reconnaître tout seul les mots. Troisièmement, le data set pour ces tests était petit (taille 1000).
4. **Le verre à moitié plein :** Dans les points positifs, on note que le modèle arrive généralement à prédire le début des noms et qu'il parvient souvent à repérer la structure du nom (s'il est

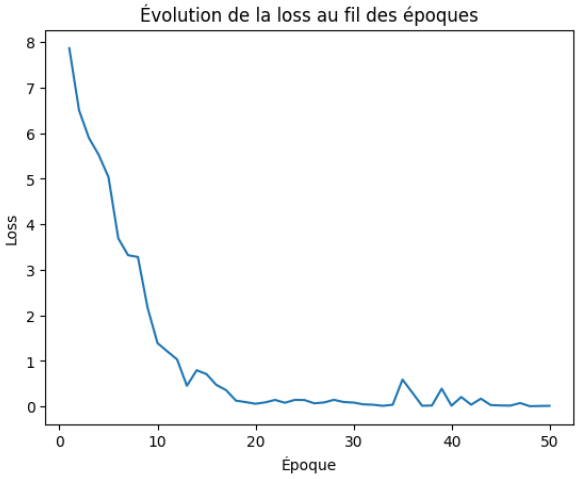

FIGURE 7 – Evolution de la losses de l'OCR

composé ou non).

La suite logique de ce projet serait donc de corriger l'OCR en appliquant les correctifs proposés.

6.2 Conclusion Générale du Projet

L'objectif de ce projet était de concevoir un système d'intelligence artificielle capable d'automatiser l'analyse de cartes de collection Magic : The Gathering, en identifiant simultanément leur Type, leur Rareté et leur Identité Couleur à partir d'une simple image. Pour répondre à ce défi multi-tâches, nous avons constitué un jeu de données équilibré de 7000 cartes et mis en compétition deux paradigmes d'apprentissage profond : les Réseaux de Neurones Convolutifs (CNN) et les Vision Transformers (ViT).

6.2.1 Bilan Comparatif : La victoire du Biais Inductif

Les résultats obtenus permettent de tirer une conclusion claire quant au choix de l'architecture dans un contexte de données limitées ("Small Data") :

- L'efficacité du CNN (EfficientNet-B0) : Ce modèle s'est imposé comme la solution la plus robuste, atteignant une précision de 98% sur les Types et 95% sur les Couleurs. Avec plus de 90% de cartes "parfaitement prédites" (les 3 attributs corrects simultanément), il démontre que les convolutions restent l'outil de référence pour extraire des caractéristiques locales (textures, bordures, symboles) sur des images de résolution standard.
- **Les limites du ViT sur petit dataset :** Bien que le Vision Transformer représente l'état de l'art actuel en vision par ordinateur, notre expérience a illustré sa "faim de données" (Data Hunger). Avec seulement 5 000 images d'entraînement, le modèle a rapidement sur-appris (Overfitting), incapable de généraliser correctement les règles géométriques des couleurs (65% de précision contre 95% pour le CNN). Il a cependant montré une capacité surprenante à détecter la Rareté (89.5%), prouvant que le mécanisme d'attention peut isoler des détails fins comme le symbole d'extension.

6.2.2 Enseignements Scientifiques

Ce projet met en lumière l'importance du Biais Inductif. Le CNN possède une "connaissance innée" de la localité des pixels (un pixel dépend de ses voisins), ce qui est crucial pour repérer une bordure colorée ou un cadre de texte. Le ViT, qui traite l'image comme une séquence globale, doit apprendre ces relations spatiales à partir de zéro, ce qui nécessite des dizaines de milliers d'images que nous ne possédions pas.

Si le prototype actuel basé sur EfficientNet est déjà fonctionnel pour une application réelle, plusieurs pistes pourraient le rendre plus performant :

1. Augmentation du Dataset : Pour rendre le ViT compétitif, il faudrait passer à l'échelle supérieure (ex : 30 000 images) en utilisant l'intégralité de la base Scryfall.
2. Reconnaissance de Texte (OCR) : Coupler notre classifieur visuel avec un moteur OCR (comme Tesseract) permettrait de lire le nom exact de la carte, offrant ainsi une identification à 100% unique pour relier la carte à sa cote financière en temps réel.
3. Déploiement Mobile : La légèreté du modèle EfficientNet (ou ConvNeXt Tiny) permettrait de l'embarquer facilement dans une application smartphone pour scanner une collection en temps réel via la caméra.

En conclusion, ce projet a permis de valider la faisabilité d'un trieur automatique de cartes Magic performant, tout en offrant une étude de cas pertinente sur les limites des Transformers lorsqu'ils sont privés de "Big Data".