

# Comprendre et Appliquer le Mécanisme d'Attention dans les Transformers pour la Restauration d'Images

COTTIER Alexandre, VU Anh Duy

février 2025

## Résumé

Ce rapport de projet porte sur l'adaptation de l'architecture Transformer, initialement conçue pour le traitement du langage naturel, à la restauration d'images. Nous abordons la technique des Vision Transformers (ViT) qui segmente l'image en petits patches, puis applique le mécanisme d'auto-attention pour extraire les corrélations spatiales. Après avoir testé une première approche de classification, nous nous sommes orientés vers une stratégie de reconstruction locale, où un patch masqué de taille  $2\times 2$  doit être prédit. Nous détaillons la mise à jour de l'architecture pour intégrer cette tâche de restauration, ainsi que la génération de données et l'entraînement. Enfin, nous analysons les performances obtenues, qui montrent un taux de reconstruction correct dans la plupart des cas.

*Mots-clés :* Transformers, Attention, Self-Attention, Patch Embedding, Reconstruction d'images, Vision Transformer, Deep Learning.

## Table des matières

<b>1. Introduction</b>	<b>1</b>
<b>2. Approche Technique</b>	<b>2</b>
<b>3. Mécanisme de Self-Attention</b>	<b>2</b>
<b>4. Multi-Head Attention</b>	<b>4</b>
<b>5. Gestion des Paramètres</b>	<b>4</b>
<b>6. Importance du Masquage</b>	<b>5</b>
<b>7. Extraction des Patches et Encodage Positionnel dans un Transformeur Visuel</b>	<b>6</b>
<b>8. Problématique de Restauration d'Image</b>	<b>7</b>
<b>9. Mise à Jour du Modèle Vision Transformer</b>	<b>8</b>
<b>10. Entraînement du Modèle Mis à Jour</b>	<b>9</b>
<b>11. Résultats et Analyse des Performances</b>	<b>9</b>
<b>12. Conclusion</b>	<b>10</b>
<b>13. Références</b>	<b>11</b>

## 1. Introduction

Ce rapport présente le travail réalisé dans le cadre de l'UE d'Ouverture à la recherche de Master Informatique au sein de l'Université UCBL Lyon 1. Ce projet s'inscrit dans un programme de recherche piloté par un binôme souhaitant se spécialiser l'un en IA et l'autre en image (d'où le sujet très intéressant pour les 2 parcours envisagés), sous la direction de l'enseignante CHAINE RAPHAËLLE.

Ce travail vise à adapter l'architecture Transformer initialement développée pour le traitement du langage naturel aux problématiques de restauration d'images. Pour ce faire, nous avons exploré des techniques de découpage d'images en patches, l'application du mécanisme d'auto-attention et l'utilisation de stratégies de masquage pour reconstruire des zones manquantes.

## 2. Approche Technique

Nous avons d'abord choisi de nous intéresser aux mécanismes qui permettent de transformer une séquence de texte en une représentation numérique exploitable. L'objectif fondamental du modèle est de prédire le mot suivant dans une séquence, en s'appuyant principalement sur deux piliers : les embeddings et le mécanisme d'attention.

### 2.1 Représentation des Tokens par Embeddings

Chaque mot ou fragment de mot (token) est d'abord converti en un vecteur dans un espace à haute dimension, connu sous le nom d'« embedding ». Ces vecteurs servent à capturer la signification sémantique du token, en permettant d'identifier des relations linguistiques (par exemple, la transformation d'un nom masculin vers un nom féminin) au sein de l'espace vectoriel. La richesse de ces représentations est cruciale pour la compréhension globale du texte et pour la prédiction du mot suivant.

### 2.2 Le Mécanisme d'Attention

Au-delà des embeddings, le modèle exploite le mécanisme d'attention, qui constitue l'une des innovations majeures des architectures de type Transformer. Ce mécanisme permet à chaque token d'interroger l'ensemble des tokens de la séquence afin d'en extraire les informations contextuelles pertinentes. Par exemple, le mot « banc » peut désigner un siège, un groupe de poissons ou une formation géologique. Le mécanisme d'attention adapte ainsi l'embedding du token en fonction des indices présents dans le contexte, garantissant ainsi une interprétation adaptée (cf. Figure 1).

### 2.3 Passage à des Espaces de Plongement de Grande Dimension

La dimension de l'espace dans lequel évoluent ces embeddings peut varier selon le modèle utilisé. Des modèles de type BERT utilisent par exemple des vecteurs de 768 dimensions, tandis que des architectures comme GPT-3 déplient des espaces de 1024 dimensions ou plus. Cette augmentation du nombre de dimensions permet de capter des nuances contextuelles plus fines, au prix d'une complexité de calcul accrue.

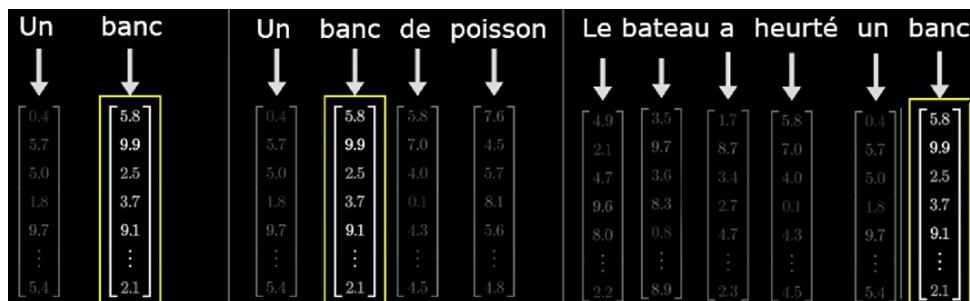


Figure 1 : Illustration du mécanisme d'attention appliquée au token « banc » dans différents contextes.

## 3. Mécanisme de Self-Attention

Le self-attention constitue l'élément central de l'architecture Transformer, car il permet de mettre à jour les embeddings de chaque token en fonction du contexte offert par les autres tokens de la séquence. Ce mécanisme repose sur plusieurs étapes clés :

### 3.1 Génération des Matrices Q, K et V

#### 1. Matrices de transformation

- Chaque embedding initial  $E_i$  (celui d'un token donné) est multiplié par trois matrices de poids distinctes :  $W_Q$ ,  $W_K$  et  $W_V$ .
- Ces matrices sont des paramètres appris du modèle et leur taille dépend de la dimension d'entrée (celle de l'embedding initial) et de la dimension choisie pour l'espace de la self-attention.

- Le résultat de ces multiplications produit trois vecteurs :  $Q_i = E_i \times W_Q$ ,  $K_i = E_i \times W_K$ ,  $V_i = E_i \times W_V$ .
- Q (Query) : interroge le contexte pour trouver les tokens pertinents.
- K (Key) : représente la clé permettant de comparer la similarité avec la query.
- V (Value) : contient l'information à récupérer ou à pondérer en fonction du score d'attention.

## 2. Objectif

- Plonger chaque embedding  $E_i$  dans un espace adapté au calcul des scores d'attention (pour Q et K) et au transport de l'information contextuelle (pour V).

## 3.2 Calcul des Scores d'Attention

### 1. Produit scalaire

- La similarité entre chaque Query  $Q_i$  et chaque Key  $K_j$  est mesurée par un produit scalaire, noté :  $\text{score}(i,j) = Q_i \times K_j$ .
- Ce score reflète la pertinence du token j pour le token i (cf. figure 2).

### 2. Mise à l'échelle (Scaling)

- Pour stabiliser les gradients et éviter des valeurs trop grandes, le score est divisé par la racine carrée de la dimension des vecteurs K :  $\text{score\_scaled}(i,j) = \frac{Q_i \times K_j}{d_k}$ , où  $d_k$  est la dimension de K (et de Q).

### 3. Matrice de Scores

- Les différents scores calculés pour tous les couples (i,j) forment une matrice dont chaque ligne correspond à un token (Query) et chaque colonne à un token (Key).

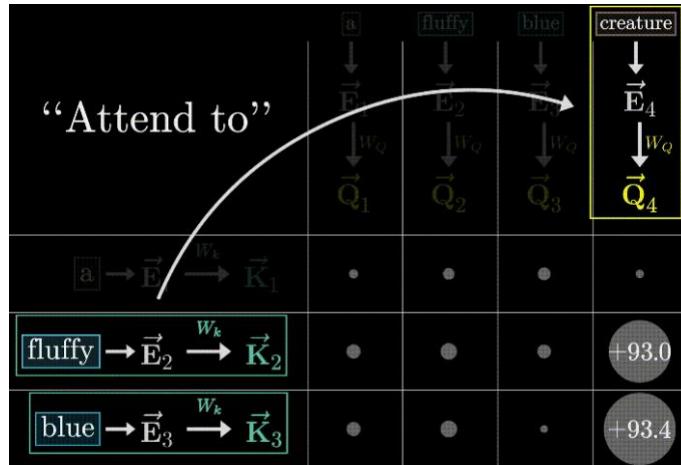


Figure 2 : Illustration du calcul du score d'attention

## 3.3 Application du Softmax

### 1. Transformation en probabilités

- Pour chaque ligne (token i), on applique la fonction Softmax sur les scores mis à l'échelle. On obtient alors un ensemble de poids d'attention compris entre 0 et 1, dont la somme est égale à 1 pour chaque ligne.
- Ces poids reflètent la « quantité d'attention » que le token i doit porter à chaque token j (cf. figure 3).
- Formule :  $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^n e^{z_k}}$

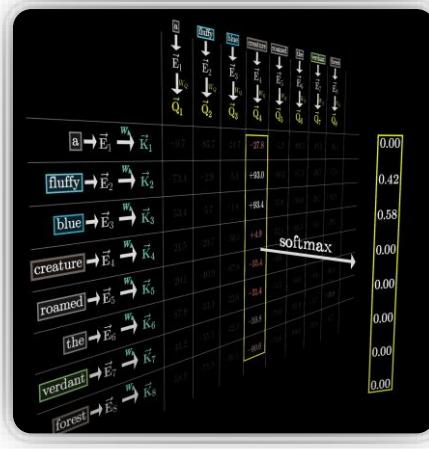


Figure 3 : Illustration de l’application du softmax

### 3.4 Mise à Jour des Embeddings

1. Combinaison Pondérée des Values
  - o Les vecteurs  $V_j$  (Values) de tous les tokens sont pondérés par les poids d’attention calculés.
  - o Pour le token i, le nouvel embedding contextuel  $V_i^{new}$  est obtenu par la somme pondérée :
 
$$V_i^{new} = \sum_j (\text{Softmax}(\text{score\_scaled}(i, j)) \times V_j)$$
2. Résultat
  - o Chaque embedding initial  $E_i$  est ainsi enrichi par l’information pertinente provenant des autres tokens, aboutissant à un vecteur contextuel plus précis pour la tâche de prédiction du mot suivant (ex : Une peluche auquel on va ajouter comme information qu’elle est verte et touffue).

## 4. Multi-Head Attention

Le mécanisme de multi-head attention étend l’idée de la self-attention en multipliant les têtes d’attention travaillant en parallèle. Chaque tête possède ses propres matrices  $W_Q$ ,  $W_K$ ,  $W_V$  et apprend à capturer différents types de relations contextuelles :

- Parallélisation : Plusieurs têtes d’attention fonctionnent simultanément, permettant de traiter diverses nuances (sémantiques, syntaxiques, etc.) en une seule passe.
- Spécialisation : Chaque tête peut se spécialiser sur une relation linguistique particulière (par exemple, l’accord sujet-verbe, la relation entre un adjectif et un nom, etc.).
- Exemple : Les modèles GPT-3 peuvent avoir jusqu’à 96 têtes d’attention dans un même bloc, démontrant la puissance de cette approche pour modéliser des dépendances complexes.

Une fois les pondérations calculées par chaque tête, les résultats (Values pondérées) sont concaténés puis projetés à nouveau dans l’espace des embeddings pour former la sortie finale de l’étage d’attention.

## 5. Gestion des Paramètres

Dans l’architecture Transformer, on manipule plusieurs types de matrices :

### 1. Matrices Q, K, V (Query, Key, Value)

- Résultent de la multiplication de l'embedding initial de chaque token par les matrices de transformation  $W_Q$ ,  $W_K$ ,  $W_V$ .
- Leurs dimensions dépendent du nombre de tokens (en lignes) et de la dimension choisie pour l'attention (en colonnes).

### 2. Matrices de Transformation $W_Q$ , $W_K$ , $W_V$

- Paramètres appris durant l'entraînement.
- Chaque tête d'attention dispose de ses propres matrices, ce qui explique la croissance du nombre total de paramètres pour des modèles à nombreuses têtes.

### 3. Résultats de l'Attention

- Les valeurs combinées (après multiplication par les poids de l'attention) doivent conserver la forme adaptée à la taille de l'embedding en sortie.
- Le nombre de colonnes (dimension) et de lignes (tokens) varie selon l'architecture et la taille du batch.

Cette gestion fine des paramètres assure la cohérence des dimensions à chaque étape (Q, K, V, puis la concaténation des têtes), garantissant la bonne propagation des informations contextuelles à travers le modèle.

## 6. Importance du masquage

Le masquage est un procédé essentiel, surtout dans les tâches de génération de texte :

- Principe : Empêcher les tokens futurs d'influencer les prédictions actuelles.
- Technique : Les positions non autorisées reçoivent une valeur de score d'attention extrêmement négative (comme des  $-\infty$  par exemple), de sorte que le Softmax leur attribue un poids nul.
- Utilisation :
  - Masquage de causalité : Dans un modèle génératif, on masque les tokens à venir pour éviter de « voir » le futur.
  - Masquage de remplissage (padding) : On ignore les tokens vides (padding) qui servent uniquement à atteindre une taille de séquence uniforme.

En appliquant le masquage, on s'assure que la génération se fasse de manière strictement séquentielle, respectant ainsi le principe de prédiction pas à pas sans fuite d'information. Cette propriété est particulièrement cruciale pour les modèles de langage où la cohérence temporelle est essentielle (par exemple, lors de la génération de phrases ou de textes longs).

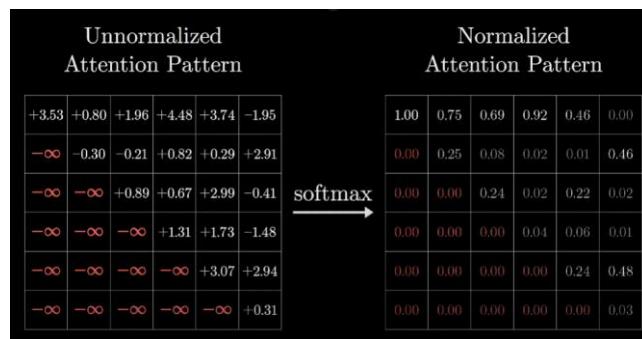


Figure 4 : Illustration de l'application du softmax

## 7. Extraction des Patches et Encodage Positionnel dans un Transformeur Visuel

Dans le cadre des Vision Transformers (ViT), l'idée centrale consiste à appliquer la même logique d'auto-attention que pour le texte, mais cette fois-ci à des images.

### 7.1 Extraction des Patches

#### 1. Principe

- L'image d'entrée (par exemple une matrice de taille  $4 \times 4$  cf. figure 5) est découpée en petits blocs, ou « patches », de taille fixe (ex.  $2 \times 2$ ).
- Chaque patch est ensuite « aplati » (flatten) pour être converti en un vecteur.
- Dans l'exemple de la figure 5, on obtient quatre patches  $P_1, P_2, P_3, P_4$ .

#### 2. Intérêt

- Cette étape permet de traiter l'image comme une séquence de tokens (un patch  $\approx$  un token), facilitant l'utilisation du mécanisme de self-attention.

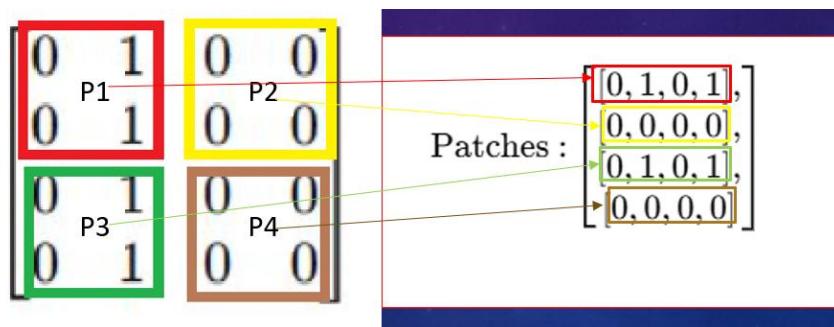


Figure 5 : Illustration de l'extraction des patches

### 7.2 Embedding des Patches

#### 1. Transformation Linéaire

- Pour chaque patch aplati, on applique une couche linéaire (ou un réseau de neurones simple) qui mappe le vecteur d'entrée vers un espace d'embedding de dimension souhaitée.
- La formule est :  $e = W \times x + b$ , où  $x$  est le vecteur du patch aplati,  $W$  la matrice de poids, et  $b$  le biais.

#### 2. Résultat

- On obtient un ensemble de vecteurs  $\{e_1, e_2, \dots\}$  représentant chaque patch dans un espace vectoriel de taille plus élevée que la dimension brute du patch.

### 7.3 Pourquoi Ajouter un Encodage Positionnel ?

#### 1. Problématique

- En reconnaissance de texte, si l'ordre des caractères n'était pas pris en compte, un mot comme 'CHAT' pourrait être interprété comme 'TACH'.
- Sans encodage positionnel, si une image est divisée en patchs, le modèle ne sait pas quel patch appartient à quelle partie de l'image
- Dans une tâche de reconstruction (inpainting), il pourrait reconstruire une image avec les patchs dans le désordre ! (cf. figure 6)



Figure 6 : Illustration d'une image reconstruis sans et avec un encodage

## 2. Méthodes Principales

- Encodage Appris : On apprend un vecteur de position pour chaque patch (similaire à l'approche utilisée dans certains Transformers pour le texte).
- Encodage Sinusoïdal : Défini mathématiquement, il associe à chaque position ( $p_x, p_y$ ) un ensemble de fréquences sinusoïdales qui indiquent la place du patch dans l'image.

## 3. L'encodage positionnel dépend uniquement de la position du patch

- Indépendant de la valeur du patch.
- Deux patches identiques, mais à des positions différentes, auront des encodages positionnels distincts (cf. figure 7).

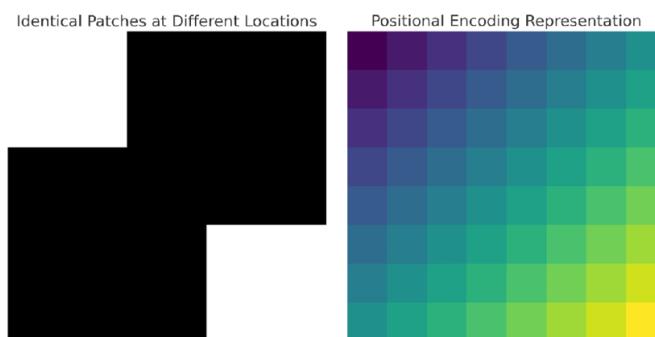


Figure 7 : Illustration de deux patches à des positions différentes

## 8. Problématique de Restauration d'Image

Nous nous sommes heurtés à une limite du modèle initial : il pouvait classer une image entière, mais ne savait pas reconstruire des pixels manquants. Nous avions donc besoin d'un modèle capable de prédire des pixels manquants dans une image.

### 8.1 Classification vs. Restauration

- Classification : Le modèle prédit une classe globale pour une image entière.
- Restauration : Le modèle doit reconstruire des pixels manquants en s'appuyant sur le contexte (cf. figure 8).

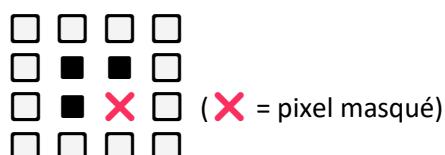


Figure 8 : → L'objectif est de reconstruire **X** en fonction des pixels autour.

Première tentative de fine-tuning du modèle

- Première approche : Masquer un seul pixel et entraîner le modèle à le prédire.
- Problème : Le modèle prédisait toujours la même valeur pour les pixels masqués (ex : toujours blanc).
- Hypothèse : Un seul pixel isolé ne contient pas assez d'informations contextuelles pour être reconstruit correctement.

## 8.2 Nouvelle Stratégie : Prédiction d'un Patch Entier (2x2)

Pour remédier à cette limite, une nouvelle stratégie a été proposée :

- Masquer un patch entier (2x2) au lieu d'un seul pixel (cf. Figure 9).
- Objectif : Reconstruire la totalité des 4 pixels manquants, ce qui force le modèle à apprendre des informations contextuelles plus riches.

Cette approche permet de fournir au modèle une unité de contexte plus large (le patch) qu'un simple pixel, ce qui améliore les capacités de reconstruction.

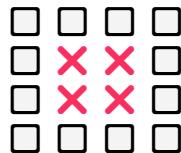


Figure 9 : → L'objectif est de reconstruire les 4 pixels masqués (X).

## 8.3 Mise en Place de l'Entraînement avec Patch Masking

- Génération des données : On crée des images d'entraînement dans lesquelles un patch 2x2 est aléatoirement masqué.
- Apprentissage : Le modèle est entraîné à prédire simultanément les 4 pixels manquants du patch.
- Résultat attendu : En contrignant le modèle à régénérer un bloc de pixels plutôt qu'un seul, on améliore la qualité de la restauration et on évite la prédiction d'un pixel moyen.

Cette stratégie s'avère particulièrement utile pour des tâches telles que l'inpainting, où l'on souhaite combler des zones manquantes dans une image. En étendant le principe du masquage à des blocs plus grands, on favorise un apprentissage plus riche des corrélations spatiales.

## 9. Mise à Jour du Modèle Vision Transformer

Pour passer d'une tâche de classification à une tâche de restauration d'image, il a fallu adapter l'architecture initiale du Vision Transformer :

1. Objectif Modifié
  - Avant : Le modèle sortait une classe prédite (via une couche de classification Softmax).
  - Maintenant : Le modèle doit régénérer quatre valeurs de pixels (2x2), correspondant au patch masqué.
2. Patch Embedding
  - Les *patch embeddings* restent inchangés mais au lieu de se concentrer sur la classification, le réseau est désormais entraîné pour la reconstruction locale de pixels.

## 10. Entraînement du Modèle Mis à Jour

1. Génération du Dataset
  - Création de 5000 images  $4 \times 4$  aléatoires.
  - Masquage aléatoire d'un patch  $2 \times 2$  par image.
  - Le modèle doit prédire les 4 pixels du patch masqué.
2. Fonction de Perte et Optimisation
  - Fonction de perte : Mean Squared Error (MSE), qui mesure la différence entre les pixels reconstruits et les pixels originaux.
  - Optimiseur : Adam, avec un taux d'apprentissage adapté (dans notre projet le taux est de  $5 \times 10^{-4}$ ).
  - Nombre d'époques : 30, permettant d'observer une convergence plus stable (cf. figure 10).
3. Déroulement de l'Entraînement
  - À chaque itération, le réseau reçoit une image avec un patch masqué et apprend à régénérer les valeurs manquantes.
  - Les poids du Vision Transformer sont mis à jour pour minimiser l'erreur de reconstruction.

```
Epoch 1, Loss: 12.9917
Epoch 2, Loss: 12.9026
Epoch 3, Loss: 12.8974
Epoch 4, Loss: 12.8938
Epoch 5, Loss: 12.8974
Epoch 6, Loss: 12.8936
Epoch 7, Loss: 12.8672
```

Figure 10 : échantillon de résultats obtenus

## 11. Résultats et Analyse des Performances

1. Taux de Réussite
  - 35 % des reconstructions sont considérées comme « échouées » (forte différence visuelle par rapport à l'original).
  - 50 % des valeurs prédites se rapprochent suffisamment de l'image originale pour être jugées « acceptables ».
  - 15 % des reconstructions sont « parfaites » ou quasiment identiques à l'original.
2. Courbe d'Évolution de la Perte
  - La figure 11 montre la perte MSE en fonction des itérations ou des époques.
  - Une tendance à la baisse globale indique que le modèle apprend progressivement à prédire correctement les pixels masqués.
3. Exemple visuel (cf. figure 12).
  - On observe que, même lorsque la reconstruction n'est pas parfaite, les patches prédis conservent souvent les grandes lignes de la structure originale.

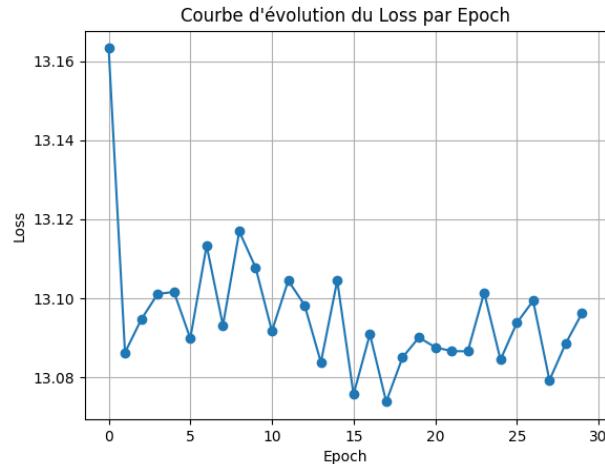
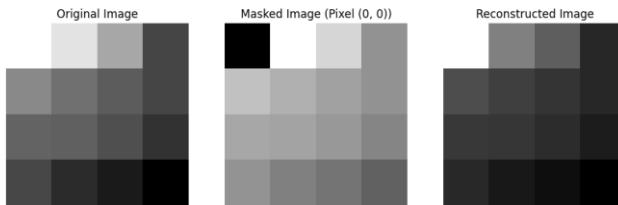
Figure 11 : Pertes par 30 epochs  $\sim 13\%$ 

Figure 12 : Un exemple de résultat, entrée par une image, image masque et image reconstruire

## Conclusion :

Ce travail a permis d'adapter avec succès une architecture Transformer, initialement conçue pour la classification, à une tâche de restauration d'images par le biais du masquage de patchs. Nous avons analysé en profondeur le mécanisme d'attention – notamment via la self-attention et la multi-head attention – et démontré comment ces concepts pouvaient être exploités pour reconstruire des zones manquantes dans une image.

Les résultats obtenus, quoique mitigés, confirment la faisabilité de cette approche : si une part significative des reconstructions reste perfectible, le modèle parvient néanmoins à produire des prédictions convaincantes dans certains cas. Parmi les avantages, on note la capacité du modèle à intégrer des informations contextuelles complexes, tandis que les limites résident principalement dans la variabilité des reconstructions et la sensibilité aux hyperparamètres.

Ces constats ouvrent la voie à plusieurs perspectives d'amélioration, telles que l'optimisation du processus de fine-tuning, l'exploration de stratégies de masquage alternatives ou encore l'extension de l'approche à des images de plus grande dimension.

Par ailleurs, cette expérience a enrichi notre compréhension des architectures Transformer et renforcé nos compétences en modélisation et en évaluation de systèmes de deep learning et de tout ce qui est lié de près ou de loin aux nouveautés liées à l'IA comme Chat GPT et les programmes de génération d'image via l'IA.

## IMPORTANT :

Dans le contexte du projet, l'un des buts était de pouvoir vulgariser et éduquer sur le domaine des transformer, nous avons donc fait une vidéo d'une vingtaine de minutes qui permet de présenter avec un peu plus de détail ce domaine et ses applications sur ce lien (<https://youtu.be/SsqCISrSwyM>).

### 13. Références :

- [1] Attention is All You Need (Vaswani et al., 2017)
- [2] Deep Learning: Contenus d'Andrej Karpathy (Karpathy, 2016)
- [3] Neural Network Interpretability: Vidéos de Chris Olah (Olah, 2017)
- [4] Neural Insights: Contributions de Vivek et Britt Cruz (Vivek & Cruz, 2018)
- [5] Visualizing Concepts in Mathematics: Vidéos de 3Blue1Brown (3Blue1Brown, 2018)
- [6] Transformers (how LLMs work) explained visually | DL5: Vidéos de 3Blue1Brown (3Blue1Brown, 2018)
- [7] Attention in transformers, step-by-step | DL6: Vidéos de 3Blue1Brown (3Blue1Brown, 2018)
- [8] How might LLMs store facts | DL7: Vidéos de 3Blue1Brown (3Blue1Brown, 2018)
- [9] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., 2018)
- [10] GPT-3: Language Models are Few-Shot Learners (Brown et al., 2020)
- [11] Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context (Dai et al., 2019)