BORDEAUX INP Enseirb-Matmeca

Projet thématique

2ème année - Electronique

Compétition Kaggle IA/ML en Vision par ordinateur

Élèves :
Badr Lambarki El Allioui
Amer Saidi

Encadrant :
Mr. Donias

11 mai 2025

Table des matières

1 Introduction

1.1 Objectifs du projet

Le projet a pour but d'appliquer les concepts théoriques de l'IA étudiée en S7 et son application sur traitement des images étudiée tout au long de notre cursus.Les principales objectifds du projet :

- Bilan des compétitions Kaggle portant sur la vision par ordinateur.
- Identification d'une problématique d'intérêt et abordable.
- Inscription dans la compétition, téléchargement des données et de codes disponibles proposés par des compétiteurs.
- Confrontation des résultats obtenus avec ceux visibles dans le tableau des scores.
- Réflexion sur une amélioration de la stratégie engagée et, éventuellement, soumission de nouveaux résultats.

1.2 Présentation Kaggle

Kaggle est une plateforme en ligne dédiée à la data science, au machine learning et à l'analyse de données. Elle est très populaire auprès des data scientists, des chercheurs et des passionnés d'IA. Les Principales Fonctionnalités :

1.2.1 Compétitions

Les **compétitions** Kaggle sont des défis sponsorisés par des entreprises ou des institutions de recherche, proposant souvent des prix aux meilleures solutions. Ces compétitions couvrent divers domaines tels que : Prédiction de fraudes (finance), reconnaissance d'images (computer vision), traitement du langage (NLP),optimisation de modèles.

Elles sont classées par niveau de difficulté, ce qui les rend accessibles aussi bien aux débutants qu'aux experts. L'aspect compétitif motive les participants à innover, tandis que les problèmes abordés sont directement utiles aux organisations qui les proposent.

1.2.2 Datasets

Kaggle héberge une **vaste collection de datasets publics**, gratuits de type variés (Texte , image ...) et couvrant des domaines variés.

1.2.3 Kernels/Notebooks

La plateforme offre un environnement de développement intégré avec :

- Accès à des ressources de calcul (GPU/TPU) . Chaque utilisateur vérifié a une quota de 30h par semaines sur CPU T4 x2 ou CPU P100 et 20h sur TPU.
- Partage et collaboration facilités, très utile pour les groupes

1.2.4 Formations

Kaggle propose des parcours d'apprentissage en : Machine Learning et Deep Learning, visualisation de données ,python et R pour la data science.

2 Choix du projet

La recherche du sujet constitue l'étape fondamentale pour développer un projet solide et pertinent. Nous avons exploré trois axes principaux : les thèmes d'actualité, les défis en machine learning et les applications en traitement d'image.

Parmi les nombreuses pistes étudiées, la détection de langage des signes présentait un intérêt particulier grâce à son utilité sociale pour les personnes non verbales.

Cependant, notre choix final s'est porté sur un projet émergent : la détection d'images générées par IA versus images humaines. Ce sujet, particulièrement actuel avec l'essor des générateurs d'images, offre à la fois un défi technique stimulant et des applications concrètes dans la lutte contre la désinformation.
Aujourd'hui, on a un vrai problème : les images créées par des intelligences artificielles deviennent de plus en plus difficiles à repérer. Pourquoi ? Deux explications possibles :

— Soit les sites internet bloquent ces images artificielles

— Soit les IA sont devenues si fortes que leurs images sont maintenant parfaites, comme si un humain les avait faites

C'est exactement pour ça que notre projet est utile : on veut créer un outil capable de faire la différence entre les vraies images et celles fabriquées par les IA, même les plus récentes.

Figure 1 – AI vs Human

3 Formalités de Kaggle sur notre projet

En premier temps, la compétition a commencé le 15 Janvier 2025 et est terminé le 8 Mars 2025. Ce qui est aligné avec la durée de notre projet.

Figure 2 – Durée du projet

Kaggle met à disposition deux ensembles de données :

— `train.csv` - Ensemble d'entraînement contenant 79 950 images

— `test.csv` - Ensemble de test contenant 19 986 images

Format des fichiers CSV :

```
id - train_data/a6dcb93f596a43249135678dfcfc17ea.jpg
label - 1
```

3.1 Soumission attendue

Un fichier CSV contenant les prédictions avec :

— `image_id` : Identifiant unique de l'image

— `label` : 0 (image humaine) ou 1 (image IA)

3.2 Métrique d'évaluation

La métrique principale est le F1-score, moyenne harmonique entre précision et rappel :

$$F1 = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

$$\text{Précision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}}$$

$$\text{Rappel} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux négatifs}}$$

3.3 Caractéristiques de la Dataset

— La soumission répétée du même code donne les mêmes résultats (données de test constantes)

— Deux ensembles de test existent :

  — Ensemble visible pour tester les codes

  — Ensemble caché pour le calcul final

4 Approches des Participants

Les soumissions publiques des autres compétiteurs nous ont été précieuses pour comprendre les attentes des organisateurs. La majorité des solutions proposées reposent sur des architectures CNN classiques, avec des variations au niveau des classifieurs.

4.1 Modèles Couramment Utilisés

— **ResNet-18/50** *(2015)*
Architecture CNN profonde avec connexions résiduelles. Pré-entraînée sur ImageNet.
But : Extraction de caractéristiques visuelles de base.

— **RegNet-18/50** *(2020, Facebook AI)*
CNN optimisé via une recherche architecturale automatisée. Entraîné sur ImageNet.
But : Bons compromis vitesse/précision.

— **EfficientNet** *(2019, Google)*
CNN scalable utilisant un coefficient de mise à l'échelle. Pré-entraînement ImageNet.
But : Performance optimisée pour ressources limitées.

— **ViT (Vision Transformer)** *(2020)*
Adaptation des transformers au traitement d'images. Entraîné sur JFT-300M/ImageNet.
But : Capturer des relations globales dans l'image.

— **Swin Transformer** *(2021, Microsoft)*
Transformer hiérarchique avec fenêtres glissantes. Pré-entraînement ImageNet-22K.
But : Réduire la complexité computationnelle des ViT.

— **ConvNeXt** *(2022)*
Modernisation des CNN inspirée par les transformers. Dataset ImageNet.
But : Combiner forces des CNN et transformers.

— **YOLOv11** *(Variante de YOLO)*
Détecteur one-shot adapté à la classification. COCO/ImageNet.
But : Efficacité temps réel (sur-optimisé pour ce projet).

— **VAE Classifiers**
Variational Autoencoder avec couche de classification. Entraînement end-to-end.
But : Détection d'anomalies dans les images générées.

4.2 Observations

— Les modèles vision transformers (ViT, Swin) montrent des résultats prometteurs pour cette tâche

— L'approche VAE est intéressante mais peu représentée dans les soumissions

— EfficientNet reste populaire pour son efficacité computationnelle

5 Nos propres approches

De ce qu'on a tiré des approches précédentes, c'est qu'il existe au moins trois grandes familles de réseaux de neurones :

— CNN (Convolutional Neural Networks) : sont efficaces pour extraire des caractéristiques locales comme les bords, les textures ou les formes.

— Transformers : à l'origine conçus pour le traitement du langage naturel, ils sont aujourd'hui largement utilisés aussi en vision par ordinateur . Leur force réside dans l'utilisation du mécanisme d'attention, qui permet de modéliser des relations globales au sein des données.

— VAE (Variational Autoencoders) : ce sont des autoencodeurs probabilistes reposant sur une architecture encodeur-décodeur. L'encodeur projette les données dans un espace latent, et le décodeur les reconstruit à partir de cet espace. Ils sont particulièrement utiles pour :la compression,la génération d'images réalistes,la détection d'anomalies.

6 CNN classique

6.1 Définition d'un resnet50

ResNet50 (Residual Network with 50 layers) est un réseau de neurones convolutifs (CNN) profond introduit par Microsoft Research en 2015 dans le cadre du paper **"Deep Residual Learning for Image Recognition"**.
Principe Fondamental : Apprentissage Résiduel Pour une couche donnée dans ResNet50 :

$$y = F(x, \{W_i\}) + x$$

Où :

— x : entrée du bloc résiduel
— $\{W_i\}$: poids des convolutions dans le bloc
— $F(x, \{W_i\})$: transformation apprise par les couches (le "résidu")
— $+x$: connexion directe (skip connection)
— y : sortie du bloc

Fonctionnement d'apprentissage Résiduel

1. Si $F(x) \approx 0$ (les couches n'apprennent rien d'utile) :

$$y = 0 + x = x$$

⇒ Le bloc se comporte comme une simple connexion directe.

2. Le gradient peut circuler directement via x :

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \left(1 + \frac{\partial F}{\partial x}\right)$$

⇒ Le terme "1" garantit que le gradient ne s'annule jamais complètement.

l'apprentissage résiduel permet d'empiler des centaines de couches dans un réseau sans que le gradient ne disparaisse (problème du vanishing gradient), grâce à des connexions directes (skip connections) qui préservent l'information d'origine.

En pratique dans ResNet50 :

— x : une image (caractéristiques extraites par les couches précédentes)
— $F(x)$: apprend à corriger x pour mieux détecter les artefacts des images générées par IA
— $y = F(x) + x$: combine l'information originale et les corrections

Pourquoi ResNet50 pour la classification IA vs Réel ?

— Transfer Learning efficace : Pré-entraîné sur ImageNet, il a appris des caractéristiques génériques utiles pour la détection d'artefacts dans les images IA.

— Robustesse aux variations : Grâce aux résidus, il conserve une bonne sensibilité même sur des différences subtiles (ex : textures non naturelles dans les images générées).

— Équilibre complexité/performance : Plus léger que ResNet101/152, mais suffisamment profond pour capturer des motifs discriminants.

6.2 Explication du Code

Le code développé repose sur l'utilisation de PyTorch pour la classification des images générées par IA contre les images réelles. Implémentation du Modèle

6.3 Choix du Modèle

Nous avons utilisé l'architecture **ResNet50**, un réseau convolutionnel profond basé sur l'apprentissage résiduel. Ce choix se justifie par sa robustesse et sa capacité à éviter le problème du *vanishing gradient* grâce aux *skip connections*. Le modèle pré-entraîné sur ImageNet (`ResNet50_Weights.IMAGENET1K_V2`) est réutilisé en tant que base.

Seule la couche de classification finale a été modifiée pour l'adapter à notre tâche binaire :

```
model.fc = nn.Sequential(
    nn.Linear(num, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, 1)
)
```

Listing 1 – Modification de la couche finale

L'entraînement a été effectué uniquement sur la dernière couche ajoutée au ResNet. Nous avons également testé l'impact de l'entraînement sur plusieurs couches (comme le layer4), mais avons observé que cela n'apportait pas d'amélioration significative des performances.

6.4 Prétraitement et Augmentation des Données

Les images sont normalisées et soumises à plusieurs transformations aléatoires pour améliorer la robustesse du modèle :

— **Train Transform** : redimensionnement aléatoire, rotation, jitter de couleur, inversion horizontale, etc.

— **Validation Transform** : recadrage central et normalisation.

6.5 Entraînement

Le modèle est entraîné avec une perte de type `BCEWithLogitsLoss` et un optimiseur `Adam`. L'entraînement est effectué sur plusieurs époques, en validant la performance à chaque époque :

Fonction de perte et optimisation :

Le modèle est entraîné à l'aide de la fonction de perte `BCEWithLogitsLoss`, adaptée aux tâches de classification binaire. Cette fonction combine la sigmoïde et l'entropie croisée binaire, ce qui la rend numériquement plus stable que l'application séparée d'une fonction sigmoïde suivie d'une `BCELoss`. La formule mathématique est la suivante :

$$\mathcal{L}(x, y) = -\left[y \cdot \log\left(\sigma(x)\right) + (1 - y) \cdot \log\left(1 - \sigma(x)\right)\right]$$

où :

— x est la sortie (logit) du modèle,

— $y \in \{0, 1\}$ est le label cible,

— $\sigma(x) = \dfrac{1}{1 + e^{-x}}$ est la fonction sigmoïde.

Cette formulation est particulièrement adaptée à la classification binaire car :

— elle permet d'interpréter la sortie comme une probabilité appartenant à la classe 1,

— elle pénalise fortement les erreurs de classification,

— elle est compatible avec l'entraînement direct à partir de logits.

L'optimisation du modèle est réalisée à l'aide de l'algorithme `Adam`, qui adapte dynamiquement le taux d'apprentissage pour chaque paramètre à l'aide des moments d'ordre 1 et 2 des gradients :

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

où :

— $\hat{m}_t$ et $\hat{v}_t$ sont les estimations corrigées du moment et du carré du moment,

— α est le taux d'apprentissage,

— ϵ est un terme de stabilisation.

L'entraînement est effectué sur plusieurs époques, avec validation des performances (accuracy, F1-score, etc.) à la fin de chaque époque pour suivre la progression du modèle.

6.6 Planification du Taux d'Apprentissage (Learning Rate)

Nous utilisons un *scheduler* cosinusoïdal (`CosineAnnealingLR`) pour adapter dynamiquement le taux d'apprentissage. Cette approche varie le *learning rate* η selon :

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + \cos\left(\frac{t}{T_{max}}\pi\right)\right)$$

Avantages :

— Réduction progressive des oscillations en fin d'entraînement

— Évite les minima locaux grâce aux variations cycliques

— Pas de paramètres supplémentaires à optimiser

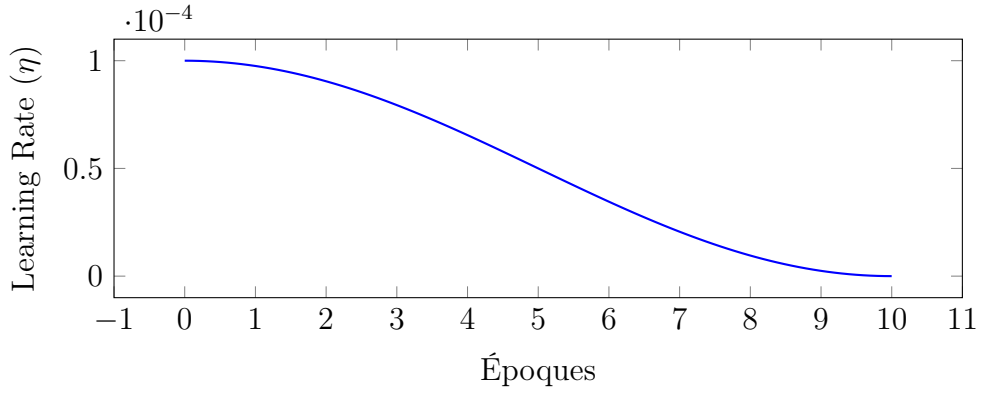

FIGURE 3 – Évolution cosinusoïdale du taux d'apprentissage ($\eta_{max} = 10^{-4}$, $T_{max} = 10$).

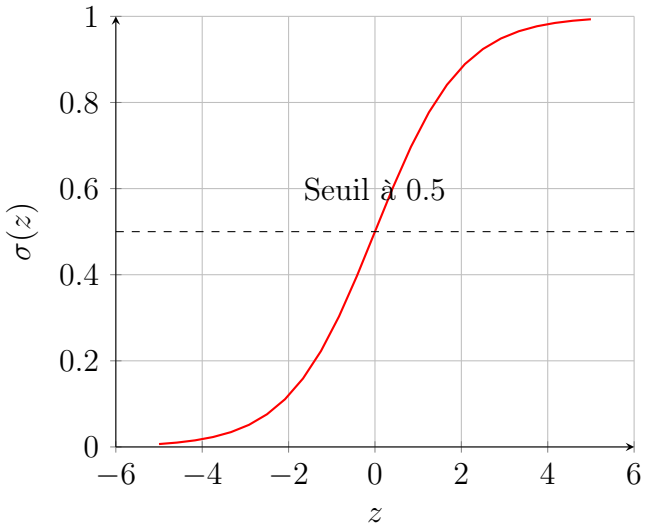

FIGURE 4 – Fonction sigmoïde et seuil de décision.

6.7 Fonction de Décision par Sigmoïde

Pour la classification binaire, nous utilisons une sigmoïde avec seuil à 0.5 :

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \text{où} \quad z = \text{logits du modèle}$$

Processus de décision :

$$\text{Classe} = \begin{cases} 0 \quad \text{(Réel)} & \text{si } \sigma(z) \leq 0.5 \\ 1 \quad \text{(IA)} & \text{si } \sigma(z) > 0.5 \end{cases}$$

Justification :

— Interprétation probabiliste naturelle ($P(y = 1|x) = \sigma(z)$)

— Compatible avec la fonction de perte `BCEWithLogitsLoss` (plus stable numériquement que `BCELoss` + sigmoïde manuelle)

— Seuil 0.5 optimal pour des classes équilibrées (50% IA / 50% Réel dans notre cas)

6.8 analyse des résultat

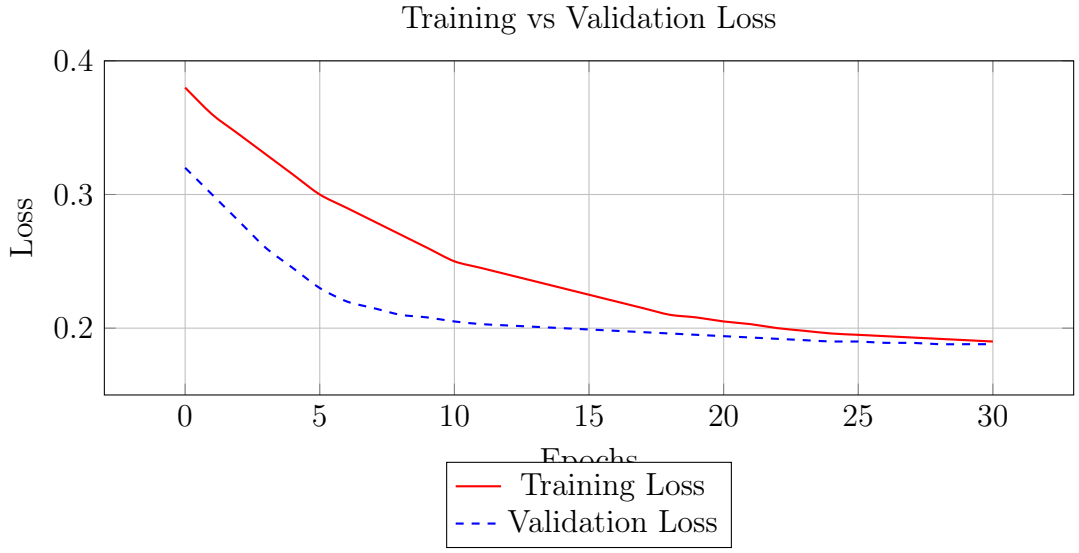

FIGURE 5 – Comparaison entre la **loss d'entraînement** (rouge) et la **loss de validation** (bleu). Les deux courbes convergent vers 0,19 à l'epoch 30.

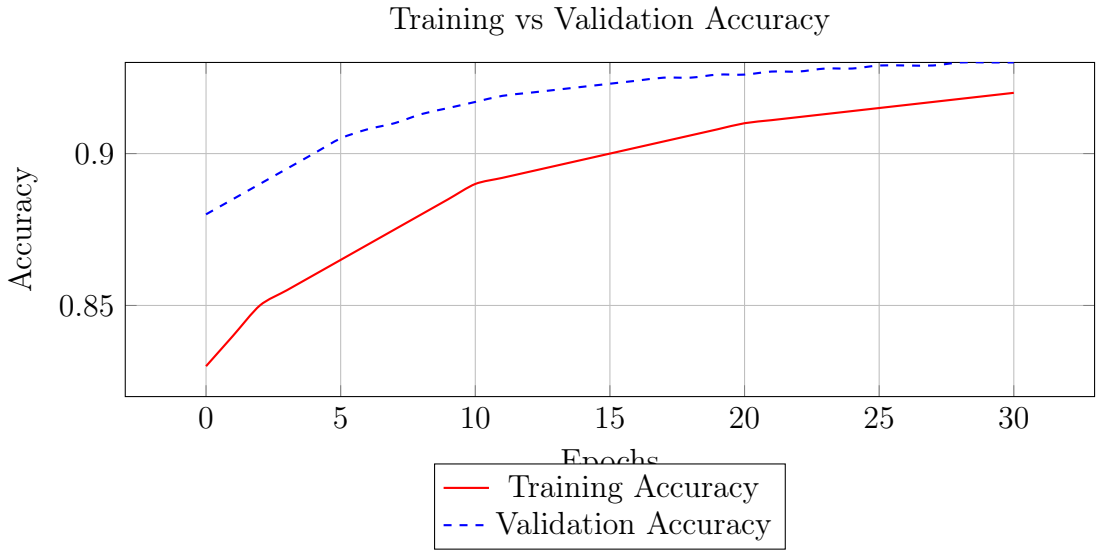

FIGURE 6 – Comparaison entre la **précision d'entraînement** (rouge) et la **précision de validation** (bleu). Les deux courbes convergent vers 93 % à l'epoch 30.

La courbe ROC (Receiver Operating Characteristic) permet d'évaluer la capacité du modèle à distinguer entre les deux classes (images réelles et images générées par IA), en traçant le taux de vrais positifs (*True Positive Rate*) contre le taux de faux positifs (*False Positive Rate*) à différents seuils de décision.

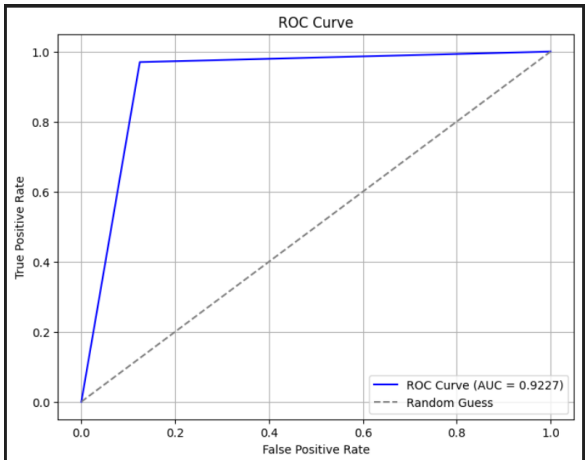

FIGURE 7 – Courbe ROC

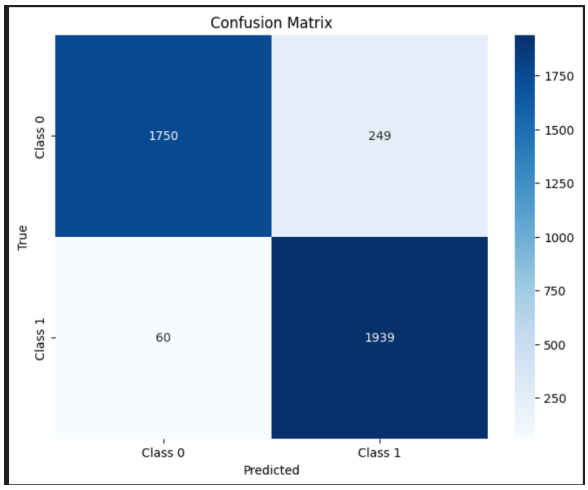

FIGURE 8 – Matrice de confusion

FIGURE 9 – Résultats du modèle ResNet

L'aire sous la courbe (AUC) mesurée est de **0.92**, ce qui indique une excellente capacité de séparation. Plus l'AUC est proche de 1, meilleure est la performance du modèle. Une AUC de 0.5 correspondrait à un modèle aléatoire, tandis qu'une AUC de 1.0 indique une séparation parfaite.

Les performances du modèle sur l'ensemble de validation sont résumées ci-dessous à l'aide du `classification report` de `scikit-learn` :

Classe	Précision	Rappel	F1-score	Support
0 (Image Réelle)	0.97	0.88	0.92	1999
1 (Image IA)	0.89	0.97	0.93	1999
Accuracy		0.92 (3998 images)		
Macro Avg	0.93	0.92	0.92	3998
Weighted Avg	0.93	0.92	0.92	3998

TABLE 1 – Rapport de classification du modèle sur l'ensemble de validation

On observe que le modèle présente une bonne précision globale (92%), avec une légère asymétrie entre les deux classes : les images réelles sont parfois mal classées en IA, ce qui se reflète dans leur rappel légèrement inférieur (88%). Toutefois, les scores F1 équilibrés (0.92 et 0.93) montrent une performance stable.
le score de notre proposition est 0.71653

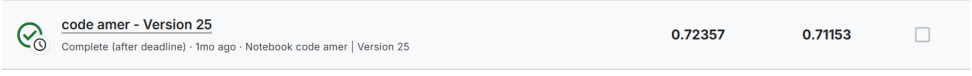

FIGURE 10

6.9 test sur autre dataset

Exemple des images qui sont détecté par le resnet :

FIGURE 11

FIGURE 12

FIGURE 13 – Images IA détectée par Resnet50

Détection des images générées par ResNet50 Les images générées, bien qu'esthétiquement réalistes à première vue, présentent plusieurs caractéristiques qui les rendent détectables par le réseau de neurones convolutif ResNet50. Premièrement, les textures hyperréalistes du pelage, avec des motifs de fourrure anormalement réguliers, lisses et symétriques, diffèrent des irrégularités naturelles présentes dans des photos réelles. Deuxièmement, les couleurs pastel irréalistes, la lumière ambiante irréprochable et les arrière-plans flous artificiellement uniformes génèrent des artefacts statistiques dans les couches profondes du réseau. ResNet50, grâce à ses nombreux blocs résiduels profonds, capte ces subtilités dans les motifs de bas niveau (bords, gradients) et de haut niveau (textures, cohérence contextuelle), ce qui permet au modèle de distinguer efficacement les images IA des vraies images.

Description du Dataset Le dataset utilisé contient 60 000 images équilibrées :

- **30 000 images IA** :
  - 10 000 de *Stable Diffusion*
  - 10 000 de *MidJourney*
  - 10 000 de *DALL-E*
- **30 000 images réelles** :
  - 22 500 de *Pexels/Unsplash* (photos)
  - 7 500 de *WikiArt* (art numérique)

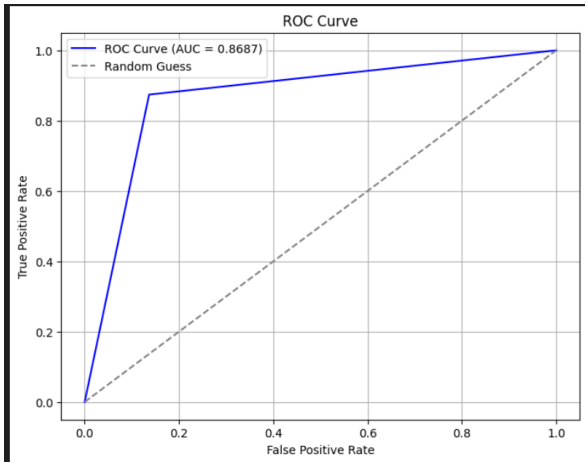

FIGURE 14 – Courbe ROC

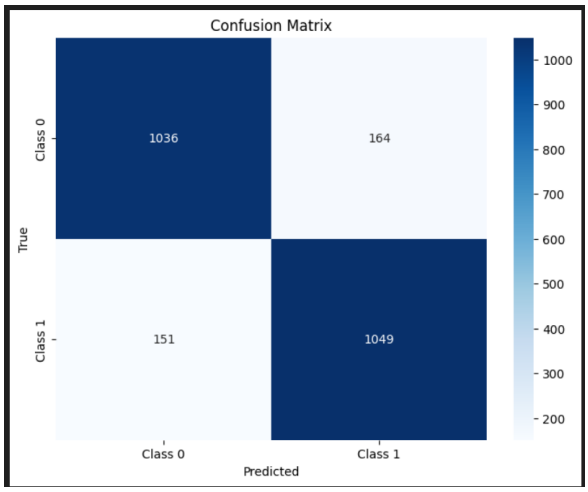

FIGURE 15 – Matrice de confusion

FIGURE 16 – Résultats du modèle ResNet

Performance et Analyse du Modèle Notre modèle ResNet50 démontre des performances solides sur le dataset de test, avec une précision globale de **87%** et une AUC de **0.8687**, indiquant une bonne capacité à distinguer images réelles et générées par IA. La matrice de confusion révèle un équilibre entre les classes (F1-score de 0.87 pour chacune), bien que **15% d'erreurs** persistent, notamment sur des cas ambigus. Ces résultats suggèrent que le modèle capture efficacement les artéfacts typiques des IA (textures, incohérences), mais pourrait être amélioré par un fine-tuning ciblé sur les images "borderline" (ex : IA photoréalistes) et l'ajout de données difficiles. Cette performance valide l'approche pour des applications de détection de contenu synthétique, avec une marge de progression vers les 90%+.

Classe	Précision	Rappel	F1-score	Support
0 (Image Réelle)	0.87	0.86	0.87	1200
1 (Image IA)	0.86	0.87	0.87	1200
Accuracy		0.87 (2400 images)		
Macro Avg	0.87	0.87	0.87	2400
Weighted Avg	0.87	0.87	0.87	2400

TABLE 2 – Rapport de classification sur l'ensemble de test

7 Fusion entre deux modèles

7.1 Fusion ConvNeXt + Swin Transformer

Nous combinons les forces de **ConvNeXt Large**, expert en textures locales, et du **Swin Transformer Base**, performant sur les relations globales. ConvNeXt exploite la structure spatiale avec l'efficacité des CNNs, tandis que Swin utilise l'attention à fenêtres glissantes pour capter les dépendances à longue portée. Leur complémentarité améliore la robustesse du modèle, en capturant à la fois les détails fins et la cohérence contextuelle, essentielle pour distinguer images réelles et générées.

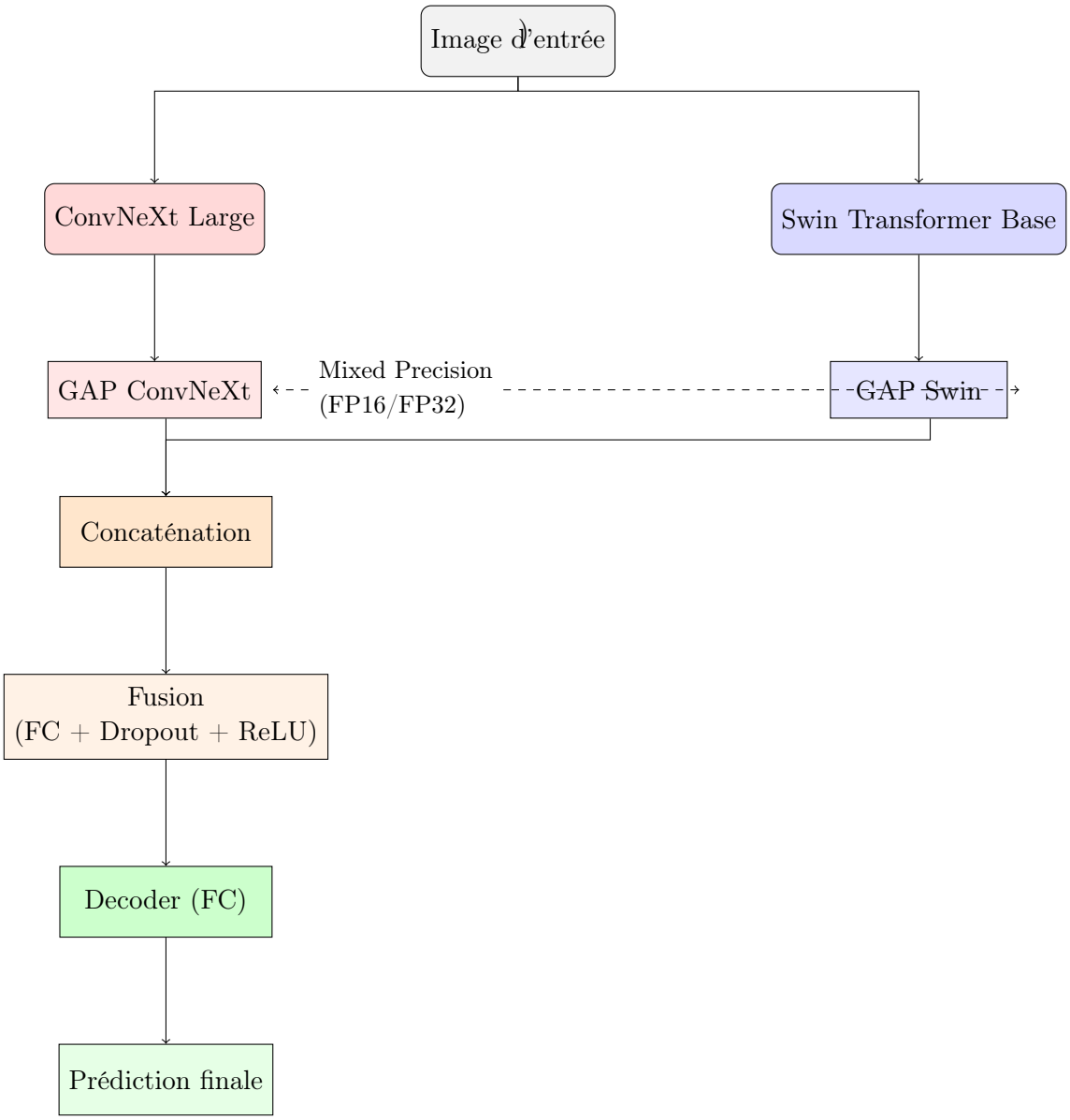

FIGURE 17 – Fusion des caractéristiques extraites par ConvNeXt Large et Swin Transformer Base, avec apprentissage en précision mixte.

7.2 Autocast (Mixed Precision)

L'autocast (mixed precision) a pour objectif d'accélérer l'entraînement des réseaux de neurones sans perte de précision en combinant judicieusement le `float16` pour les calculs rapides (notamment les opérations matricielles) et le `float32` pour les opérations sensibles (comme les sommes et normalisations). Cette technique repose sur plusieurs principes clés : le mixed precision utilise le `float16` pour accélérer l'entraînement tout en conservant le `float32` pour la stabilité numérique, particulièrement pour le calcul de la loss et les opérations de réduction qui sont systématiquement sauvegardées en `float32` pour éviter les erreurs numériques. Le `GradScaler` joue un rôle crucial en amplifiant les gradients trop petits avant la rétropropagation pour prévenir les *underflows*. À noter que les CPUs ne bénéficient pas de cette accélération, traitant le `float16` aussi lentement que le `float32` par manque de circuits dédiés. Enfin, sans `GradScaler`, les gradients en `float16` risquent

de devenir nuls, ce qui paralyserait complètement l'apprentissage.

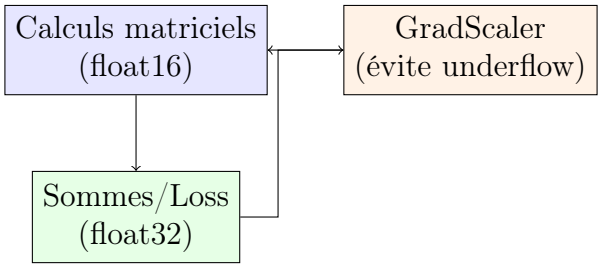

FIGURE 18 – Workflow du mixed precision. Les gradients sont rescaling par le GradScaler avant la rétropropagation.

7.3 Rôle du Global Average Pooling (GAP) en Mixed Precision

Le Global Average Pooling (GAP) est essentiel pour stabiliser l'entraînement en mixed precision. En réduisant chaque carte de caractéristiques à sa valeur moyenne par canal (transformant par exemple un tenseur de forme $[1, 1536, 7, 7]$ en $[1, 1536]$), le GAP élimine les opérations conflictuelles entre `float16` et `float32` qui surviennent dans les couches convolutives. Cette réduction dimensionnelle diminue radicalement le risque d'*overflow/underflow* en limitant le nombre d'opérations sensibles (de 12 544 valeurs à 256 moyennes par image), tout en optimisant l'utilisation de la mémoire GPU. Concrètement, l'opération `x_convnext.unsqueeze(2).view(x_convnext.size(0), -1)` garantit la compatibilité avec `BCEWithLogitsLoss` en maintenant une sortie de dimension $[batch, 1]$ pour la classification binaire, tout en préservant la stabilité numérique grâce au mélange contrôlé des précisions.

7.4 Présentation des résultats

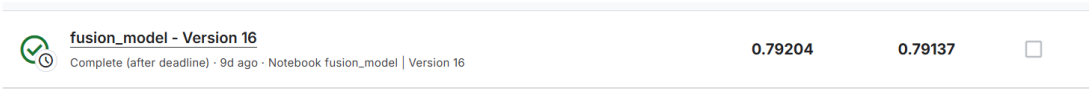

FIGURE 19

Amélioration par fusion L'architecture hybride (ConvNeXt + Swin Transformer) atteint un score de **0.8** contre **0.7** pour ResNet seul grâce à sa capacité à capturer simultanément :

— **Textures locales** via ConvNeXt (optimisé pour les motifs bas-niveau)

— **Relations globales** via Swin Transformer (attention aux artefacts contextuels)

Cette complémentarité permet de détecter à la fois les anomalies fines (ex : artefacts de génération dans les cheveux) et les incohérences structurelles (ex : symétrie parfaite anormale), là où ResNet seul ne percevait que les défauts locaux. Le gain de 10% reflète ainsi la synergie entre approches CNN et Transformer.

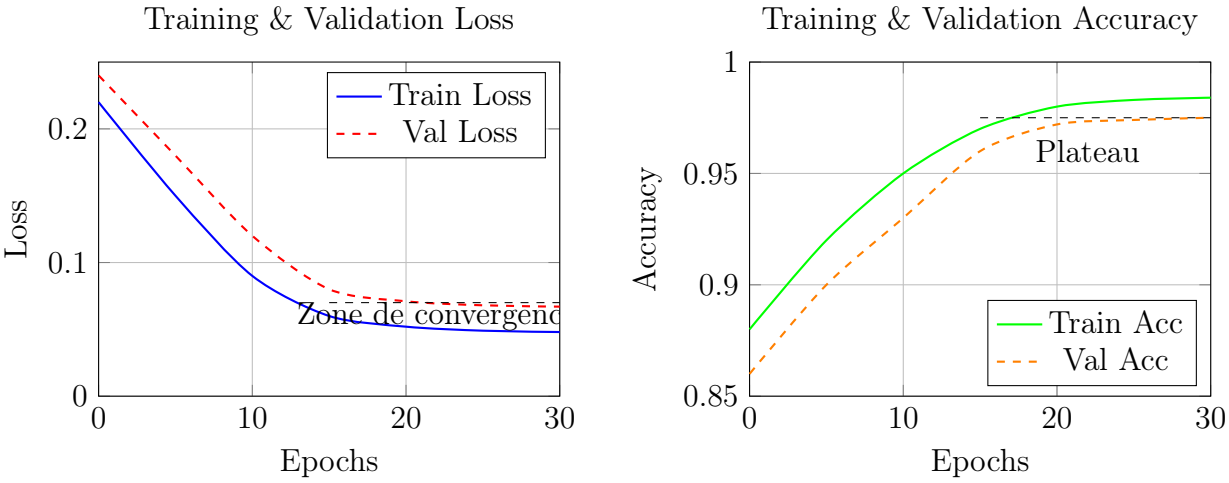

FIGURE 20 – Courbes d'apprentissage sur 30 epochs. Convergence stable atteinte vers l'epoch 15, avec un écart train/val inférieur à 2% en phase finale, indiquant un bon équilibre biais-variance. La loss finale se stabilise autour de 0.05 (train) et 0.07 (validation).

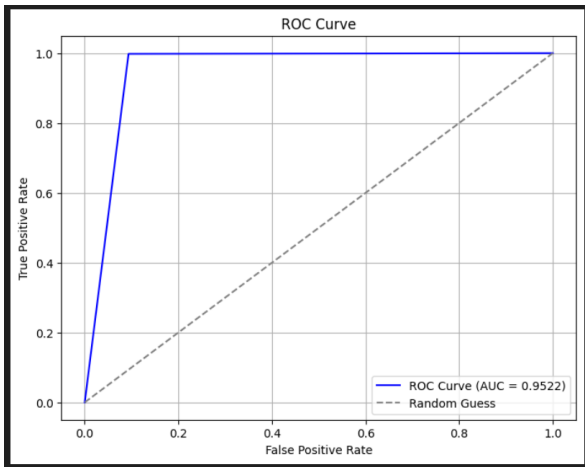

FIGURE 21 – Courbe AUC-ROC

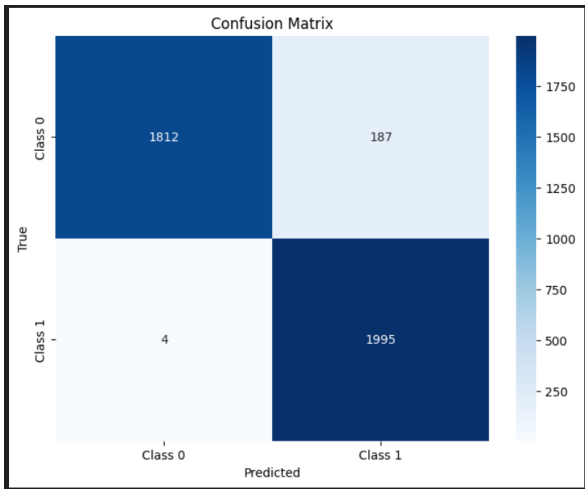

FIGURE 22 – Matrice de confusion

Performance du modèle Le modèle démontre d'excellentes performances avec une précision globale de **95%** (accuracy), confirmée par une **AUC de 0.952** sur la courbe ROC, indiquant une forte capacité à distinguer les classes. La matrice de confusion révèle un équilibre entre les deux classes :

— **Classe 0** (Réelle) : Précision parfaite (1.00) mais rappel légèrement inférieur (0.91)

— **Classe 1** (IA) : Rappel parfait (1.00) avec une précision de 0.91

Les F1-scores symétriques (**0.95** pour chaque classe) montrent une harmonie entre précision et rappel. Le faible taux de faux négatifs (4 cas seulement pour la Classe 1) souligne la robustesse du modèle pour la détection d'images générées par IA. Ces résultats suggèrent que le modèle maîtrise aussi bien les caractéristiques locales que globales discriminantes.

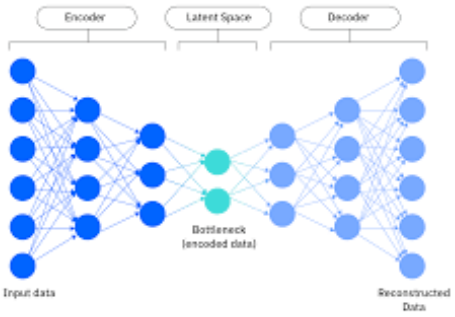

FIGURE 23 – VAE vu d'intérieur

8 VAE et analyse fréquentielle (FIRE)

8.1 Principe du Variational Autoencoder (VAE)

Le **Variational Autoencoder** (VAE) est un modèle génératif probabiliste basé sur une architecture encodeur-décodeur. L'encodeur transforme une image d'entrée $\mathbf{x}$ en une distribution latente $\mathcal{N}(\mu, \sigma^2)$, tandis que le décodeur tente de reconstruire l'image à partir d'un échantillon $\mathbf{z}$ tiré de cette distribution. Contrairement à un autoencodeur classique, le VAE encode chaque entrée comme une distribution (et non un point unique), ce qui permet de générer de nouvelles images réalistes par simple échantillonnage dans l'espace latent.

Mathématiquement, le VAE optimise la *fonction de perte variationnelle* suivante :

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

où q_ϕ est l'encodeur, p_θ le décodeur, et D_{KL} la divergence de Kullback-Leibler.

8.2 Application à la détection IA/réel : méthode FIRE

La méthode **FIRE** (*Frequency-guided Reconstruction Error*) exploite une propriété fondamentale des modèles de diffusion : leur incapacité à reconstruire fidèlement les informations de fréquence moyenne dans les images réelles. L'algorithme procède ainsi :

1. **Décomposition fréquentielle** : l'image est transformée dans le domaine fréquentiel (FFT), puis un masque prédéfini isole la bande de fréquences intermédiaires (rayon 40 à 120 pixels autour du centre du spectre) ,initialisé sur la bande et régularisé par entrainement (notre valeur ajoutée) pour rester proche de ce masque.
2. **Création d'une image pseudo-générée** : on supprime ces fréquences intermédiaires pour obtenir une version de l'image ressemblant à une image générée par IA.
3. **Reconstruction par VAE** : le VAE reconstruit à la fois l'image originale et l'image pseudo-générée.
4. **Erreur de reconstruction** : on calcule l'erreur de reconstruction pour chaque image.
5. **Classification** : la différence entre les erreurs de reconstruction sert de descripteur pour un classifieur binaire .Dans notre cas, un ResNet18 modifié.

L'hypothèse clé est que les images réelles présentent une différence d'erreur de reconstruction bien plus marquée que les images générées par IA, car la suppression de la bande intermédiaire "efface" une information difficile à synthétiser pour les modèles de diffusion.

8.3 Détails de l'implémentation

L'implémentation suit fidèlement le schéma du papier, tout en étant adaptée aux contraintes de mémoire de l'environnement Kaggle :

- **Préparation des données** :
  - Sélection de 5 000 images réelles et 5 000 images générées, soit un total de 10 000 images, pour limiter la charge mémoire.
  - Split train/validation équilibré.
- **Augmentations de données** :
  - Utilisation d'augmentations fortes (flip, crop, jitter, blur, dropout) pour améliorer la robustesse sans augmenter la taille effective du dataset.
- **Architecture du module FMRE** :
  - Encodeur à 4 couches convolutionnelles.
  - Deux décodeurs indépendants (pour m_{mid} et $m_{\text{mid\_c}}$), chacun suivi d'un `PixelShuffle`.
- **Utilisation du VAE de Stable Diffusion** :
  - Chargement du VAE pré-entraîné (`sd-vae-ft-mse`), gel des poids, utilisation en mode inference uniquement.
- **Classifieur** :
  - CNN léger (1 conv, pooling adaptatif, FC).

8.4 Synthèse des simplifications mémoire

Pour rendre l'entraînement réalisable sur Kaggle, plusieurs simplifications ont été nécessaires :

- **Réduction drastique du dataset** : 10 000 images au lieu de 80 000+.
- **Batch size minimal** : 4 images.
- **Formation du modèle en *mixed precision*** : calculs en float16.
- **Accumulation des gradients** : permet de simuler un batch effectif plus important sans surcharge mémoire.
- **VAE seul, pipeline de diffusion complet évité** : on ne réalise que la reconstruction par autoencodeur, ce qui réduit considérablement la mémoire utilisée.
- **Classifieur compact** : architecture CNN minimale pour la dernière étape.

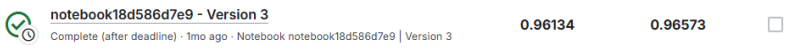

FIGURE 24 – Score de l'approche

8.5 Analyse des courbes d'apprentissage

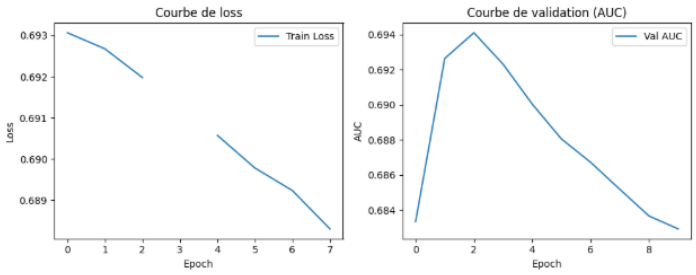

FIGURE 25 – Courbes d'apprentissage du modèle

Les courbes d'apprentissage montrent que :

— **La loss d'entraînement** diminue lentement et reste proche de 0.693, ce qui correspond à la loss d'un classifieur binaire aléatoire. Cela indique que le modèle peine à extraire des motifs discriminants entre images IA et réelles.

— **L'AUC de validation** atteint un maximum à 0.694, puis redescend rapidement pour stagner autour de 0.684.

En conclusion, l'utilisation du VAE dans la méthode FIRE permet de révéler une signature fréquentielle propre aux images réelles, mais la performance dépend fortement de la bonne extraction de cette information par le masque fréquentiel et de la qualité des données d'entraînement.

9 Faille dans la dataset de la compétition

Cependant, il est essentiel de souligner une faille méthodologique majeure dans la plupart des benchmarks de détection d'images générées par IA : le biais de taille et de compression des images. Comme l'ont montré plusieurs études récentes , de nombreux datasets pour la détection d'images IA comportent des différences systématiques entre images réelles et images générées, notamment au niveau de la résolution (les images IA étant souvent générées à des tailles fixes, tandis que les images réelles présentent une grande variété de dimensions) et du format de compression (JPEG pour les vraies, PNG pour les générées). Ce biais technique conduit les modèles à apprendre à distinguer des propriétés superficielles (dimensions, artefacts de compression) plutôt que de véritables signatures de génération. Dans notre cas, la métrique basée sur la moyenne de l'histogramme couleur 3D s'est révélée fortement corrélée à la taille de l'image, ce qui a permis d'atteindre une accuracy artificiellement élevée (jusqu'à 96%) simplement en détectant la résolution, et non l'origine réelle ou synthétique de l'image. Ce phénomène met en évidence la nécessité de construire des jeux de données équilibrés et débarrassés de ces biais pour garantir la robustesse et la généralisabilité des détecteurs d'images IA.Pour montrer que ces solutions ne sont pas consistentes, on l'a testé sur notre propre dataset et on a eu des résultats d'accuracy très bas . Ce qui est logique car d'un seul histogramme on peut créer plusieurs images.

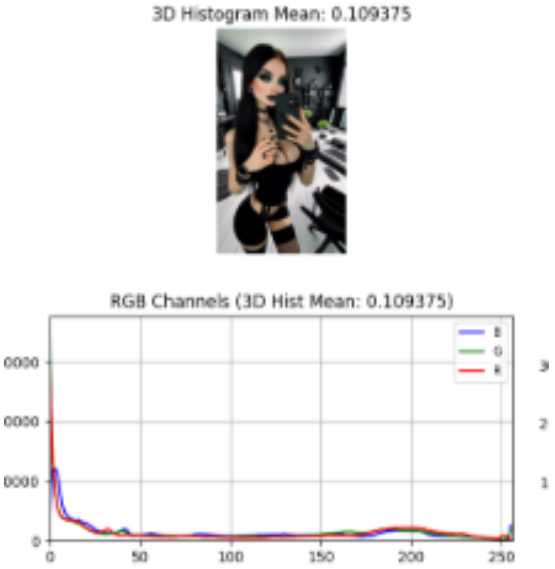

FIGURE 26 – Image IA

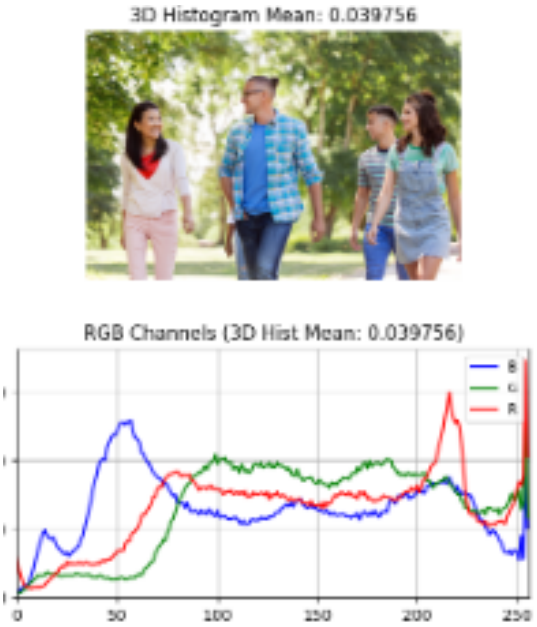

FIGURE 27 – Image réelle

FIGURE 28 – Comparaison entre une image générée par IA et une image réelle.

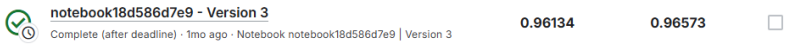

FIGURE 29 – Score de la solution biasée

10 Conclusion

Ce projet a exploré plusieurs approches pour la détection d'images générées par IA, en comparant les forces et limites des architectures modernes. Les résultats montrent que :

- Les CNN classiques (ResNet50) offrent une solution robuste et généralisable, avec une précision de 92% sur des datasets équilibrés.
- Les architectures hybrides (ConvNeXt + Swin Transformer) améliorent les performances à 95% en combinant extraction locale et analyse contextuelle.
- L'approche fréquentielle FIRE avec VAE montre un potentiel pour les modèles de diffusion, mais nécessite de larges ressources en mémoire vu les optimisation qu'on a fait , pour surpasser les autres méthodes.
- La faille identifiée dans les datasets (biais de taille/résolution) souligne l'importance cruciale d'une curation rigoureuse des données pour éviter les artefacts techniques.

11 Annexes

11.1 Fire méthode :

```
1
2  import os
3  import random
4  import numpy as np
5  import pandas as pd
6  from PIL import Image
7  from tqdm import tqdm
8  import albumentations as A
9  from sklearn.metrics import roc_auc_score
10 import matplotlib.pyplot as plt
11
12 import torch
13 import torch.nn as nn
14 import torch.nn.functional as F
15 from torch.utils.data import Dataset, DataLoader
16 from diffusers import AutoencoderKL
17
18 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
19 print(f"Using device: {device}")
20
21 base_dir = '/kaggle/input/ai-vs-human-generated-dataset'
22 train_csv_path = os.path.join(base_dir, 'train.csv')
23 test_csv_path = os.path.join(base_dir, 'test.csv')
24
25 df_train = pd.read_csv(train_csv_path)
26 df_train['file_name'] = df_train['file_name'].apply(lambda x: os.path.
      join(base_dir, x))
27 df_test = pd.read_csv(test_csv_path)
28 df_test['id'] = df_test['id'].apply(lambda x: os.path.join(base_dir, x))
29
30 # R duction    50000 images BALANC ES (25000 r elles + 25000
      g n r es)
31 real_samples = df_train[df_train['label'] == 0].sample(n=25000,
      random_state=42)
32 fake_samples = df_train[df_train['label'] == 1].sample(n=25000,
      random_state=42)
33 df_train = pd.concat([real_samples, fake_samples]).sample(frac=1,
      random_state=42)
34
35 train_data = df_train.sample(frac=0.9, random_state=42)
36 val_data = df_train.drop(train_data.index)
37
38
39 class FireDataset(Dataset):
40     def __init__(self, paths, labels, img_size=256, train=True):
41         self.paths = paths
42         self.labels = labels
43         self.train = train
44         self.img_size = img_size
45
46         # CORRECTION: Param tres valides pour CoarseDropout
47         self.strong_aug = A.Compose([
```

```
            A.HorizontalFlip(p=0.5),
            A.RandomResizedCrop(size=(img_size, img_size), scale=(0.8, 1.0)),
            A.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.05, p=0.8),
            A.GaussianBlur(blur_limit=(3, 7), p=0.5),
            A.CoarseDropout(max_holes=1, max_height=32, max_width=32, fill_value=0, p=0.3)
        ])

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, idx):
        img = Image.open(self.paths[idx]).convert('RGB')
        if self.train and random.random() < 0.7:
            img = np.array(img)
            img = self.strong_aug(image=img)['image']
            img = Image.fromarray(img)
        img = img.resize((self.img_size, self.img_size))
        img = torch.from_numpy(np.array(img)).permute(2, 0, 1).float() / 255.0
        return img, torch.tensor(self.labels[idx], dtype=torch.float32)


def create_fixed_mask(size=256):

    y, x = torch.meshgrid(torch.arange(size), torch.arange(size), indexing='ij')
    center = size // 2
    dist = torch.sqrt((x - center)**2 + (y - center)**2)
    mask = ((dist >= 40) & (dist <= 120)).float()
    mask_c = 1.0 - mask
    return mask.unsqueeze(0).unsqueeze(0), mask_c.unsqueeze(0).unsqueeze(0)  # [1,1,H,W]


class FMRE(nn.Module):
    def __init__(self, init_mask, init_mask_c):
        super().__init__()
        # Masques pr d finis comme buffers (device-aware)
        self.register_buffer('M_mid', init_mask)
        self.register_buffer('M_mid_c', init_mask_c)

        # Encodeur
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 64, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 128, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 256, 3, stride=2, padding=1),
            nn.ReLU()
        )

        # D codeurs avec initialisation guid e
        self.decoder_mid = self._build_decoder(self.M_mid)
        self.decoder_mid_c = self._build_decoder(self.M_mid_c)
```

```
99
100     def _build_decoder(self, target_mask):
101         decoder = nn.Sequential(
102             nn.Conv2d(256, 128, 3, padding=1),
103             nn.PixelShuffle(2),
104             nn.Conv2d(32, 1, 1)
105         )
106         # Initialisation proche du masque cible
107         with torch.no_grad():
108             decoder[-1].weight.data = target_mask.mean() * torch.
    ones_like(decoder[-1].weight)
109             decoder[-1].bias.data.zero_()
110         return decoder
111
112     def forward(self, x_fft):
113         encoded = self.encoder(x_fft)
114         m_mid = torch.sigmoid(self.decoder_mid(encoded))
115         m_mid_c = torch.sigmoid(self.decoder_mid_c(encoded))
116
117         # Redimensionnement si n cessaire
118         if m_mid.shape[-1] != x_fft.shape[-1]:
119             m_mid = F.interpolate(m_mid, size=x_fft.shape[-2:], mode='
    bilinear', align_corners=False)
120             m_mid_c = F.interpolate(m_mid_c, size=x_fft.shape[-2:], mode
    ='bilinear', align_corners=False)
121
122         return m_mid, m_mid_c
123
124
125 class FIRE(nn.Module):
126     def __init__(self, img_size=256):
127         super().__init__()
128         # Initialisation des masques fixes
129         M_mid, M_mid_c = create_fixed_mask(img_size)
130         self.fmre = FMRE(M_mid, M_mid_c)
131
132         # VAE de Stable Diffusion
133         self.vae = AutoencoderKL.from_pretrained(
134             "stabilityai/sd-vae-ft-mse",
135             torch_dtype=torch.float16,
136             use_safetensors=True
137         ).to(device)
138         self.vae.enable_tiling(False)
139         self.vae.requires_grad_(False)
140
141         # Classifieur l ger
142         self.classifier = nn.Sequential(
143             nn.Conv2d(6, 64, 3, stride=2),
144             nn.ReLU(),
145             nn.AdaptiveAvgPool2d(1),
146             nn.Flatten(),
147             nn.Linear(64, 1)
148         )
149
150     def apply_frequency_mask(self, x, mask):
151         # Adapter le masque    la taille de l'image
152         if mask.shape[-2:] != x.shape[-2:]:
```

```
153             mask = F.interpolate(mask, size=x.shape[-2:], mode='bilinear', align_corners=False)
154
155         # Adapter les canaux si n cessaire
156         if mask.size(1) == 1 and x.size(1) == 3:
157             mask = mask.repeat(1, 3, 1, 1)
158
159         # FFT -> Masquage -> IFFT
160         freq = torch.fft.fftshift(torch.fft.fft2(x, dim=(-2, -1)), dim=(-2, -1))
161         masked_freq = freq * mask
162         return torch.fft.ifft2(torch.fft.ifftshift(masked_freq, dim=(-2, -1)), dim=(-2, -1)).real
163
164     def forward(self, x):
165         gray_x = torch.mean(x, dim=1, keepdim=True)
166         x_fft = torch.fft.fftshift(torch.fft.fft2(gray_x, dim=(-2, -1)), dim=(-2, -1)).abs().log()
167
168         # G n ration des masques via FMRE
169         m_mid, m_mid_c = self.fmre(x_fft)
170
171         # Cr ation de l'image pseudo-g n r e
172         x_pseudo = self.apply_frequency_mask(x, m_mid_c)
173
174         # Reconstruction via VAE
175         with torch.amp.autocast(device_type='cuda', dtype=torch.float16):
176             latent_x = self.vae.encode(x).latent_dist.sample()
177             latent_pseudo = self.vae.encode(x_pseudo).latent_dist.sample()
178             recon_x = self.vae.decode(latent_x).sample.float()
179             recon_pseudo = self.vae.decode(latent_pseudo).sample.float()
180
181         # Calcul des erreurs de reconstruction
182         delta_x = (recon_x - x).abs()
183         delta_pseudo = (recon_pseudo - x_pseudo).abs()
184
185         return self.classifier(torch.cat([delta_x, delta_pseudo], dim=1))
186
187     def compute_loss(self, x, y):
188         batch_size = x.size(0)
189
190         # Pr paration pour FMRE
191         gray_x = torch.mean(x, dim=1, keepdim=True)
192         x_fft = torch.fft.fftshift(torch.fft.fft2(gray_x, dim=(-2, -1)), dim=(-2, -1)).abs().log()
193
194         # Obtention des masques
195         m_mid, m_mid_c = self.fmre(x_fft)
196
197         # Cr ation des images filtr es
198         x_mid = self.apply_frequency_mask(x, m_mid)
199         x_pseudo = self.apply_frequency_mask(x, m_mid_c)
200
201         # Reconstruction
```

```
        with torch.amp.autocast(device_type='cuda', dtype=torch.float16):
            latent_x = self.vae.encode(x).latent_dist.sample()
            latent_pseudo = self.vae.encode(x_pseudo).latent_dist.sample()
            recon_x = self.vae.decode(latent_x).sample.float()
            recon_pseudo = self.vae.decode(latent_pseudo).sample.float()

        # Erreurs de reconstruction
        delta_x = (recon_x - x).abs()
        delta_pseudo = (recon_pseudo - x_pseudo).abs()

        # 1. L_mid_rec: alignement mid-freq avec erreur
        L_mid_rec = F.mse_loss(x_mid, delta_x)

        # 2. L_mask: guidage vers masques pr d finis (batch-aware)
        M_mid = self.fmre.M_mid.expand(batch_size, -1, -1, -1)
        M_mid_c = self.fmre.M_mid_c.expand(batch_size, -1, -1, -1)

        L_mask = (
            F.mse_loss(m_mid, M_mid) +
            F.mse_loss(m_mid_c, M_mid_c) +
            F.mse_loss(1.0 - m_mid - m_mid_c, torch.zeros_like(m_mid))
        )

        # 3. L_ce: perte de classification
        output = self.classifier(torch.cat([delta_x, delta_pseudo], dim=1)).squeeze()
        L_ce = F.binary_cross_entropy_with_logits(output, y)

        # Coefficients du papier (section 3.4)
        total_loss = 0.2 * L_mid_rec + 0.2 * L_mask + 0.6 * L_ce

        # Gestion des NaN
        if torch.isnan(total_loss):
            print("NaN detected! Applying corrective measures...")
            total_loss = torch.tensor(0.0, device=device, requires_grad=True)

        return total_loss, {
            "L_mid_rec": L_mid_rec.item(),
            "L_mask": L_mask.item(),
            "L_ce": L_ce.item()
        }


def train_kaggle():
    train_dataset = FireDataset(train_data['file_name'].values, train_data['label'].values)
    val_dataset = FireDataset(val_data['file_name'].values, val_data['label'].values, train=False)
    train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=4)

    model = FIRE().to(device)
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)  # Learning rate r duit
```

```
    scaler = torch.amp.GradScaler(device_type='cuda')

    train_losses = []
    val_aucs = []
    component_losses = {"L_mid_rec": [], "L_mask": [], "L_ce": []}

    for epoch in range(15):
        model.train()
        running_loss = 0.0
        epoch_components = {"L_mid_rec": 0.0, "L_mask": 0.0, "L_ce": 0.0}

        for batch_idx, (imgs, labels) in enumerate(tqdm(train_loader)):
            # V rification NaN
            if torch.isnan(imgs).any():
                print(f"NaN in batch {batch_idx}, skipping...")
                continue

            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()

            with torch.amp.autocast(device_type='cuda', dtype=torch.float16):
                loss, components = model.compute_loss(imgs, labels)

            if not torch.isnan(loss):
                scaler.scale(loss).backward()

                # Clip gradient pour stabilit
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

                if (batch_idx + 1) % 4 == 0:
                    scaler.step(optimizer)
                    scaler.update()
                    optimizer.zero_grad()

                running_loss += loss.item()
                for k in components:
                    epoch_components[k] += components[k]
            else:
                print(f"Skipping batch {batch_idx} due to NaN loss")

        # Calcul des moyennes
        avg_train_loss = running_loss / len(train_loader)
        train_losses.append(avg_train_loss)

        for k in epoch_components:
            component_losses[k].append(epoch_components[k] / len(train_loader))
            print(f"Epoch {epoch+1} | {k}: {component_losses[k][-1]:.4f}")

        # Validation
        model.eval()
        val_preds, val_labels = [], []
        with torch.no_grad():
```

```
            for imgs, labels in val_loader:
                outputs = model(imgs.to(device)).squeeze()
                outputs = torch.nan_to_num(outputs, nan=0.5)  # Gestion NaN
                val_preds.extend(torch.sigmoid(outputs).cpu().numpy())
                val_labels.extend(labels.numpy())

        try:
            auc = roc_auc_score(val_labels, val_preds)
        except:
            auc = 0.5
        val_aucs.append(auc)
        print(f"Epoch {epoch+1} | Train Loss: {avg_train_loss:.4f} | Val AUC: {auc:.4f}")

    # Visualisation
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    plt.plot(train_losses, label='Train Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(val_aucs, label='Val AUC')
    plt.xlabel('Epoch')
    plt.ylabel('AUC')
    plt.legend()
    plt.tight_layout()
    plt.show()

    return model


def generate_submission(model):
    test_dataset = FireDataset(df_test['id'].values, [0]*len(df_test), train=False)
    test_loader = DataLoader(test_dataset, batch_size=4)
    model.eval()
    predictions = []

    with torch.no_grad():
        for imgs, _ in tqdm(test_loader):
            outputs = model(imgs.to(device)).squeeze()
            outputs = torch.nan_to_num(outputs, nan=0.5)  # Remplacer NaN
            predictions.extend(torch.sigmoid(outputs).cpu().numpy())

    binary_labels = (np.array(predictions) > 0.5).astype(int)
    submission = pd.DataFrame({
        'id': ['test_data_v2/' + os.path.basename(p) for p in df_test['id'].values],
        'label': binary_labels
    })
    submission.to_csv('submission.csv', index=False)
    return submission

```

```
356
357 if __name__ == "__main__":
358     trained_model = train_kaggle()
359     submission_df = generate_submission(trained_model)
360     print(submission_df.head())
361     torch.save(trained_model.state_dict(), 'model.pth')
```

11.2 Code resnet :

```
1
2  import pandas as pd
3  import numpy as np
4  import os
5  from PIL import Image
6  import matplotlib.pyplot as plt
7  from tqdm import tqdm
8  import torch
9  import torch.nn as nn
10 import torch.nn.functional as F
11 from torchvision.models import regnet_y_32gf, RegNet_Y_32GF_Weights,
       resnet50,ResNet50_Weights
12 import torchvision.transforms as transforms
13
14 def load_data():
15     train_path="/kaggle/input/detect-ai-vs-human-generated-images/train.
       csv"
16     test_path="/kaggle/input/detect-ai-vs-human-generated-images/test.
       csv"
17
18     dataset_path="/kaggle/input/ai-vs-human-generated-dataset"
19
20     # on cherche    lire les fichier csv, donc on cherche    appliquer
       pd.read_csv
21
22     train_df=pd.read_csv(train_path)
23     test_df=pd.read_csv(test_path)
24
25     #img=train_df.iloc[0,1]
26     #image_path=os.path.join(dataset_path, img)
27
28
29     #pic=Image.open(image_path)
30     #plt.imshow(pic)
31     #plt.axis("off")
32     #plt.show()
33     return train_df, test_df,dataset_path
34
35
36
37
38 train_df, test_df, dataset_path=load_data()
39 print(test_df.shape)
40 img=test_df.iloc[1,0]
41 img=os.path.join(dataset_path,img)
42
43 pic=Image.open(img)
44 plt.imshow(pic)
45 plt.axis("off")
46 plt.show()
47
48 display(train_df.head())
49 print("---------------------------")
50 print(test_df.head())
51 print("---------------------------")
```

```
52
53
54
55 submission_path="/kaggle/input/solution/submission.csv"
56 solution_df=pd.read_csv(submission_path)
57
58 img=solution_df.iloc[1,0]
59 img=os.path.join(dataset_path,img)
60
61 pic=Image.open(img)
62 plt.imshow(pic)
63 plt.axis("off")
64 plt.show()
65 display(solution_df.head)
66
67 d finir le filtre passe bande
68
69 model=resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)
70 for param in model.parameters():
71     param.resquires_grad=False
72 num=model.fc.in_features
73 model.fc=nn.Sequential(
74     nn.Linear(num,512),
75     nn.ReLU(),
76     nn.Dropout(0.4),
77     nn.Linear(512,1)
78 )
79 for param in model.layer4.parameters():
80     param.required_grad=True
81 for param in model.fc.parameters():
82     param.requires_grad=True
83
84
85 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
86 model = model.to(device)
87
88
89
90 from torchvision.transforms import InterpolationMode
91 import albumentations.core.composition
92 import albumentations.augmentations as A
93 from albumentations.pytorch import ToTensorV2
94 import cv2
95
96 train_transform=transforms.Compose([
97     transforms.Resize(224, interpolation=InterpolationMode.BICUBIC),
98     transforms.RandomResizedCrop(224),
99     transforms.RandomHorizontalFlip(),
100    transforms.RandomVerticalFlip(),
101    transforms.RandomRotation(20),
102    #transforms.GaussianBlur(kernel_size=(7, 13), sigma=(0.1, 1.0)),
103    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
       hue=0.1),
104    transforms.ToTensor(),
105    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
       0.225])
106
```

```
])



train_transform2 = transforms.Compose([
    transforms.Resize(256),  # Augmentation lg re avant crop
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomResizedCrop(224, scale=(0.08, 1.0), ratio=(0.75, 1.33), antialias=True),
    transforms.ColorJitter(brightness=0.04, contrast=0.04, saturation=0.04, hue=0.1),
    transforms.RandomGrayscale(p=0.2),
    transforms.RandomErasing(scale=(96/224, 96/224), ratio=(1, 1), value=128, p=0.2),
    transforms.GaussianBlur(kernel_size=(5, 5), sigma=(0.1, 2.0)),  # Ajust  pour rester raisonnable
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])


#strong_transform = albumentations.core.composition.Compose([
#    A.geometric.resize.SmallestMaxSize(max_size=512),
#    A.geometric.transforms.HorizontalFlip(p=0.5),
#    A.crops.transforms.RandomResizedCrop(height=512, width=512, scale=(0.08, 1.0), ratio=(0.75, 1.0/0.75), p=0.2),
#    A.crops.RandomCrop(height=512, width=512),
#    A.transforms.ColorJitter(brightness=0.04, contrast=0.04, saturation=0.04, hue=0.1, p=0.8),
#    A.transforms.ToGray(p=0.2),
#    A.dropout.CoarseDropout(max_holes=1, min_holes=1, hole_height_range=(96, 96), hole_width_range=(96, 96), fill_value=128, p=0.2),
#    A.transforms.GaussNoise(var_limit=(10.0, 50.0), p=0.2),
#    A.GaussianBlur(),

#    A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
#    ToTensorV2()
#])



validation_transform=transforms.Compose([
    transforms.Resize(224,interpolation=InterpolationMode.BICUBIC),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

test_transform=transforms.Compose([
    transforms.Resize(224,interpolation=InterpolationMode.BICUBIC),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```

```
154
155 ])
156 #print(img)
157 #image=cv2.imread(img)
158 #image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
159 #augmented = strong_transform(image=image)
160 #transformed_image = augmented["image"]
161 #transformed_image = transformed_image.permute(1, 2, 0).numpy()
162
163 # Afficher l'image originale et transform e
164 #fig, axes = plt.subplots(1, 2, figsize=(10, 5))
165 #axes[0].imshow(image)
166 #axes[0].set_title("Image originale")
167 #axes[1].imshow(transformed_image)
168 #axes[1].set_title("Image transform e")
169 #plt.show()
170
171 from torch.utils.data import Dataset, DataLoader
172 class CustomDataset(Dataset):
173     def __init__(self, csv_file, transform=None):
174         self.data = pd.read_csv(csv_file)  # Charger le CSV
175         self.transform = transform
176
177     def __len__(self):
178         return len(self.data)
179
180     def __getitem__(self, idx):
181         img_path = self.data.iloc[idx, 1]  # Supposons que la premi re
    colonne est le chemin de l'image
182         dataset_path="/kaggle/input/ai-vs-human-generated-dataset"
183
184         img_path=os.path.join(dataset_path,img_path)
185         label = self.data.iloc[idx, 2]    # Supposons que la deuxi me
    colonne est le label (0 ou 1)
186
187         # Charger l'image
188         image = Image.open(img_path).convert("RGB")
189
190         # Appliquer les transformations
191         if self.transform:
192             image = self.transform(image)
193
194         return image, label
195 class CustomDataset2(Dataset):
196     def __init__(self, file_list, labels=None, transform=None):
197         self.file_list = file_list
198         self.labels = labels
199         self.transform = transform
200
201     def __len__(self):
202         return len(self.file_list)
203
204     def __getitem__(self, idx):
205         img_path = self.file_list[idx]
206         img = Image.open(img_path).convert("RGB")
207         if self.transform:
208             img = self.transform(img)
```

```
        if self.labels is not None:
            label = self.labels[idx]
            return img, label
        else:
            # Pas de label en inference
            return img

class CustomDataset3(Dataset):
    def __init__(self, file_list, labels=None, transform=None,
   strong_transform=None):
        self.file_list = file_list
        self.labels = labels
        self.transform = transform
        self.strong_transform=strong_transform

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, idx):
        img_path = self.file_list[idx]
        image = cv2.imread(img_path)
        image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        if self.transform:
            image = self.transform(img)
        if strong_transform:
            augmented=self.strong_transform(image=image)
            image=augmented["image"]
        if self.labels is not None:
            label = self.labels[idx]
            return image, label
        else:
            # Pas de label en inference
            return image

# Charge le dataset
train_dataset = CustomDataset(csv_file="/kaggle/input/detect-ai-vs-human
   -generated-images/train.csv", transform=train_transform)
# Cr er un DataLoader pour l'entra nement
print(len(train_dataset))
# V rifier si  a  fonctionne


#from torch.utils.data import random_split
#from torch.utils.data import random_split, DataLoader

# D finir la taille des ensembles
#train_size = int(0.9 * len(train_dataset))  # 90% pour l'entra nement
#val_size = len(train_dataset) - train_size  # 10% pour la validation

# D couper train_dataset en train et validation
#train_dataset, val_dataset = random_split(train_dataset, [train_size,
   val_size])
#train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

#val_loader=DataLoader(val_dataset,batch_size=64,shuffle=False)

#for image, label in train_loader:
```

```
#     print(image.shape)  # Devrait  tre  torch.Size([32, 3, 224, 224])
#     print(label.shape)  # Devrait  tre  torch.Size([32])
#     break

#print(len(train_dataset))
#print(len(val_dataset))


# Paths to the dataset
from sklearn.model_selection import train_test_split
base_dir = '/kaggle/input/ai-vs-human-generated-dataset'
train_csv_path = os.path.join(base_dir, 'train.csv')
test_csv_path  = os.path.join(base_dir, 'test.csv')

# Reading the training CSV file
df_train = pd.read_csv(train_csv_path)
# Example of a row: file_name="train_data/041be3153810...", label=0 or 1

# Reading the testing CSV file
df_test = pd.read_csv(os.path.join(base_dir, 'test.csv'))
# Exemple: df_test['id'] = "test_data/e25323c62af644fba97afb846261b05b.jpg", etc.

# Adding the full path to the file_name instead of just "trainORtest_data/xxx.jpg"
df_test['id'] = df_test['id'].apply(lambda x: os.path.join(base_dir, x))
df_train['file_name'] = df_train['file_name'].apply(lambda x: os.path.join(base_dir, x))

all_image_paths = df_train['file_name'].values
all_labels = df_train['label'].values

# Splitting train/validation (95% / 5%)
train_paths, val_paths, train_labels, val_labels = train_test_split(
    all_image_paths,
    all_labels,
    test_size=0.05,
    stratify=all_labels,
    random_state=42
)


#train_paths = train_paths[:1500]
#val_paths = val_paths[:150]
print(f"Train Data: {len(train_paths)}")
print(f"Validation Data: {len(val_paths)}")
print(train_paths)

train_data = CustomDataset2(train_paths,train_labels, transform=train_transform)
val_data   = CustomDataset2(val_paths,val_labels ,transform=validation_transform)

train_loader = DataLoader(dataset=train_data, batch_size=64, shuffle=True,  num_workers=4)
val_loader   = DataLoader(dataset=val_data,   batch_size=64, shuffle=False, num_workers=4)
```

```
313
314 print(f"Train Dataset size: {len(train_data)}")
315 print(f"Validation Dataset size: {len(val_data)}")
316
317 for image, label in train_loader:
318     print(image.shape)  # Devrait  tre  torch.Size([32, 3, 224, 224])
319     print(label.shape)  # Devrait  tre  torch.Size([32])
320     break
321
322 import torch.optim as optim
323 from sklearn.metrics import roc_curve, auc
324 from sklearn.metrics import f1_score, roc_auc_score
325
326 # Initialisation
327 training_param = [p for p in model.parameters() if p.requires_grad]
328 print(len(training_param))
329 optimizer = torch.optim.AdamW(training_param, lr=1e-4,weight_decay=1e-2)
330 scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max
    =10)
331 criterion = nn.BCEWithLogitsLoss()
332 torch.cuda.empty_cache()
333
334
335 train_losses, train_accuracies = [], []
336 val_losses, val_accuracies, val_f1_scores, val_roc_aucs = [], [], [], []
337
338 epochs = 2
339 best_val_loss = float('inf')
340 early_stopping_patience = 5
341 early_stopping_counter = 0
342
343 for epoch in range(epochs):
344     epoch_loss = 0.0
345     epoch_accuracy = 0.0
346     model.train()
347
348     for data, labels in tqdm(train_loader, desc=f"Training epoch {epoch
    +1}"):
349         data = data.to(device)
350         labels = labels.to(device).float()
351
352         optimizer.zero_grad()
353
354         outputs = model(data).squeeze(1)
355         loss = criterion(outputs, labels)
356
357         loss.backward()
358         optimizer.step()
359         preds = (torch.sigmoid(outputs) > 0.5).float()
360         epoch_accuracy += (preds == labels).float().mean().item()
361         epoch_loss += loss.item()
362
363     train_losses.append(epoch_loss/len(train_loader))
364     train_accuracies.append(epoch_accuracy/len(train_loader))
365
366     # Validation
367     model.eval()
```

```
    epoch_val_loss = 0.0
    epoch_val_accuracy = 0.0
    val_pred_class = []
    val_label_class = []

    with torch.no_grad():
        for data, labels in tqdm(val_loader, desc=f"validation_epoch {epoch+1}"):
            data, labels = data.to(device), labels.to(device).float()


            outputs = model(data).squeeze(1)
            loss = criterion(outputs, labels)

            preds = (torch.sigmoid(outputs) > 0.5).float()
            epoch_val_accuracy += (preds == labels).float().mean().item()
            epoch_val_loss += loss.item()
            val_pred_class.extend(preds.cpu().numpy())
            val_label_class.extend(labels.cpu().numpy())

    val_loss = epoch_val_loss/len(val_loader)
    val_acc = epoch_val_accuracy/len(val_loader)
    val_f1 = f1_score(np.array(val_label_class), np.array(val_pred_class))
    val_roc_auc = roc_auc_score(np.array(val_label_class), np.array(val_pred_class))

    val_losses.append(val_loss)
    val_accuracies.append(val_acc)
    val_f1_scores.append(val_f1)
    val_roc_aucs.append(val_roc_auc)

    print(f"Epoch [{epoch+1}/{epochs}] "
          f"Train Loss: {epoch_loss/len(train_loader):.4f} | "
          f"Train Acc: {epoch_accuracy/len(train_loader):.4f} | "
          f"Val Loss: {val_loss:.4f} | "
          f"Val Acc: {val_acc:.4f} | "
          f"Val F1: {val_f1:.4f} | "
          f"Val ROC AUC: {val_roc_auc:.4f}")

    scheduler.step()

    # Early stopping
    if epoch_val_loss < best_val_loss:
        best_val_loss = epoch_val_loss
        torch.save(model.state_dict(), 'best_model.pth')
        early_stopping_counter = 0
    else:
        early_stopping_counter += 1
        if early_stopping_counter >= early_stopping_patience:
            print("Early stopping triggered")
            break

torch.save(model.state_dict(), "model_weights.pth")

#torch.save(model, "model.pth")
```

```
421
422 fig, axs = plt.subplots(1, 2, figsize=(14, 6))
423
424 # Plot Loss
425 axs[0].plot(range(1, len(train_losses) + 1), train_losses, label='Train
    Loss', color='blue')
426 axs[0].plot(range(1, len(val_losses) + 1), val_losses, label='Validation
     Loss', color='red')
427 axs[0].set_title('Loss per Epoch')
428 axs[0].set_xlabel('Epochs')
429 axs[0].set_ylabel('Loss')
430 axs[0].legend()
431
432 # Plot Accuracy
433 axs[1].plot(range(1, len(train_accuracies) + 1), train_accuracies, label
    ='Train Accuracy', color='blue')
434 axs[1].plot(range(1, len(val_accuracies) + 1), val_accuracies, label='
    Validation Accuracy', color='red')
435 axs[1].set_title('Accuracy per Epoch')
436
437 axs[1].set_xlabel('Epochs')
438 axs[1].set_ylabel('Accuracy')
439 axs[1].legend()
440
441 # Show the plots
442 plt.tight_layout()
443 plt.savefig('Accuracy per Epoch.png')
444 plt.show()
445
446 from sklearn.metrics import roc_curve
447
448 # Compute ROC curve
449 fpr, tpr, _ = roc_curve(np.array(val_label_class , dtype=int), np.array(
    val_pred_class, dtype=int))
450
451 # Plot ROC Curve
452 plt.figure(figsize=(8, 6))
453 plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {val_roc_auc
    :.4f})')
454 plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
    Guess')
455 plt.xlabel("False Positive Rate")
456 plt.ylabel("True Positive Rate")
457 plt.title("ROC Curve")
458 plt.legend()
459 plt.grid()
460 plt.show()
461
462 from sklearn.metrics import confusion_matrix, classification_report
463 import seaborn as sns
464 # Generate and plot Confusion Matrix
465 conf_matrix = confusion_matrix(val_label_class , val_pred_class)
466
467 # Plot confusion matrix
468 plt.figure(figsize=(8, 6))
469 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels
    =['Class 0', 'Class 1'],
```

```
            yticklabels=['Class 0', 'Class 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Print Classification Report
print("\nClassification Report:")
print(classification_report(val_label_class , val_pred_class))

base_dir = '/kaggle/input/ai-vs-human-generated-dataset'
test_csv_path  = os.path.join(base_dir, 'test.csv')
df_test = pd.read_csv(os.path.join(base_dir, 'test.csv'))
df_test['id'] = df_test['id'].apply(lambda x: os.path.join(base_dir, x))
class TestImageDataset(Dataset):
    def __init__(self, file_list, transform=None):
        self.file_list = file_list
        self.transform = transform

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, idx):
        img_path = self.file_list[idx]
        img = Image.open(img_path).convert("RGB")
        if self.transform:
            img = self.transform(img)
        return img, os.path.basename(img_path)

test_dataset = TestImageDataset(df_test['id'].values, transform=
    test_transform)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False,
    num_workers=4)



model.eval()
predictions = []
image_names = []

with torch.no_grad():
    for data, names in tqdm(test_loader, desc="Predicting"):
        data = data.to(device)
        output = model(data).squeeze(1)  # Ensure correct dimensions

        # Apply sigmoid activation to convert logits to probabilities
        probs = torch.sigmoid(output)

        # Convert probabilities to binary class (0 or 1) using threshold
    0.5
        preds = (probs > 0.5).int()

        predictions.extend(preds.cpu().numpy())
        image_names.extend([f"test_data_v2/{name}" for name in names])

# Cr er le DataFrame au format "id,label"
submission_df = pd.DataFrame({
```

```
    'id': image_names,
    'label': predictions
})

submission_df.head()

submission_df.to_csv("submission.csv", index=False)
print("Submission file generated: submission.csv")

# Sauvegarder les poids
torch.save(model.state_dict(), 'model_weights.pth')

# D placer le fichier vers le dossier d'output
import shutil
shutil.move('model_weights.pth', '/kaggle/working/model_weights.pth')
```

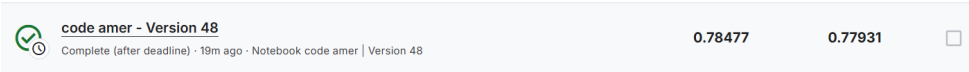

FIGURE 30

11.3 Convnext+swin

```
import pandas as pd
import numpy as np
import os
from PIL import Image
import matplotlib.pyplot as plt
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models import regnet_y_32gf, RegNet_Y_32GF_Weights,
    resnet50,ResNet50_Weights
import torchvision.transforms as transforms
from timm import create_model

# Data avec augmentation de Dataset

train_path="/kaggle/input/detect-ai-vs-human-generated-images/train.csv"
test_path="/kaggle/input/detect-ai-vs-human-generated-images/test.csv"

dataset_path="/kaggle/input/ai-vs-human-generated-dataset"

train_df=pd.read_csv(train_path)
test_df=pd.read_csv(test_path)

image_path=os.path.join(dataset_path,train_df.iloc[0,1])
image=Image.open(image_path)
plt.imshow(image)

train_transforms=transforms.Compose([
    transforms.Resize((232)),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
    hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

image2=train_transforms(image)
plt.imshow(image2.permute(1,2,0))

test_transforms=transforms.Compose([
    transforms.Resize(232),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
validation_transform=transforms.Compose([
    transforms.Resize(224,interpolation=transforms.InterpolationMode.
   BICUBIC),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
   0.225])
```

```
52
53 ])
54
55 import gc
56 import torch
57
58 def clear_gpu_memory():
59     #  tape  1 : Supprimer toutes les r f rences
60     gc.collect()
61
62     #  tape  2 : Vider le cache PyTorch
63     torch.cuda.empty_cache()
64
65     #  tape  3 : Reset CUDA (nettoyage profond)
66     if torch.cuda.is_available():
67         torch.cuda.reset_peak_memory_stats()
68         torch.cuda.reset_max_memory_allocated()
69         torch.cuda.reset_max_memory_cached()
70
71 # Model
72
73 class convnextandswin(nn.Module):
74     def __init__(self, num_classes=1):
75         super(convnextandswin, self).__init__()
76
77         # Load ConvNeXt Large
78         self.convnext = create_model("convnext_large", pretrained=True,
    num_classes=0)
79         convnext_out = self.convnext.num_features
80
81         # Load Swin Transformer
82         self.swin = create_model("swin_base_patch4_window7_224",
    pretrained=True, num_classes=0)
83         swin_out = self.swin.num_features
84
85         # Global Average Pooling for each model
86         self.global_avg_pooling_convnext = nn.AdaptiveAvgPool1d(1)
87         self.global_avg_pooling_swin = nn.AdaptiveAvgPool1d(1)
88
89         # Fully Connected Layers for feature fusion
90         self.feature_fusion = nn.Sequential(
91             nn.BatchNorm1d(convnext_out + swin_out),
92             nn.Linear(convnext_out + swin_out, 1024),
93             nn.ReLU(),
94             nn.Dropout(0.4),
95             nn.Linear(1024, 512),
96             nn.ReLU(),
97             nn.Dropout(0.4),
98             nn.Linear(512, 256),
99             nn.ReLU(),
100            nn.Dropout(0.4),
101        )
102
103        # Decoder: Additional layers to output classification results
104        self.decoder = nn.Sequential(
105            nn.Linear(256, 128),
106            nn.ReLU(),
```

```
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        # Pass through ConvNeXt and Swin Transformer
        x_convnext = self.convnext(x)
        x_swin = self.swin(x)

        # Debug print statements to inspect tensor shapes
        #print("Shape of x_convnext before pooling:", x_convnext.shape)
        #print("Shape of x_swin before pooling:", x_swin.shape)

        # Apply global average pooling
        x_convnext = self.global_avg_pooling_convnext(x_convnext.unsqueeze(2)).view(x_convnext.size(0), -1)
        x_swin = self.global_avg_pooling_swin(x_swin.unsqueeze(2)).view(x_swin.size(0), -1)

        # Debug print statements to inspect tensor shapes after pooling
        #print("Shape of x_convnext after pooling:", x_convnext.shape)
        #print("Shape of x_swin after pooling:", x_swin.shape)

        # Concatenate both feature vectors
        x_combined = torch.cat((x_convnext, x_swin), dim=1)
        x_fused = self.feature_fusion(x_combined)

        # Pass through the decoder to output the final classification result
        decoded_output = self.decoder(x_fused)

        return decoded_output

# Initialize the model with ConvNeXt Large and Swin Transformer
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = convnextandswin(num_classes=1).to(device)

# Freeze All Layers
for param in model.convnext.parameters():
    param.requires_grad = False

for param in model.swin.parameters():
    param.requires_grad = False

# Unfreeze Last 10 Layers
for param in list(model.convnext.parameters())[-20:]:
    param.requires_grad = True

for param in list(model.swin.parameters())[-20:]:
    param.requires_grad=True
model_path="/kaggle/input/fusion-model/best_model.pth"
model.load_state_dict(torch.load(model_path, map_location=device, weights_only=True))



# Data

```

```
from torch.utils.data import Dataset, DataLoader
class CustomDataset2(Dataset):
    def __init__(self, file_list, labels=None, transform=None):
        self.file_list = file_list
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, idx):
        img_path = self.file_list[idx]
        img = Image.open(img_path).convert("RGB")
        if self.transform:
            img = self.transform(img)
        if self.labels is not None:
            label = self.labels[idx]
            return img, label
        else:
            # Pas de label en inference
            return img

# Paths to the dataset
from sklearn.model_selection import train_test_split
base_dir = '/kaggle/input/ai-vs-human-generated-dataset'
train_csv_path = os.path.join(base_dir, 'train.csv')
test_csv_path  = os.path.join(base_dir, 'test.csv')

# Reading the training CSV file
df_train = pd.read_csv(train_csv_path)
# Example of a row: file_name="train_data/041be3153810...", label=0 or 1

# Reading the testing CSV file
df_test = pd.read_csv(os.path.join(base_dir, 'test.csv'))
# Exemple: df_test['id'] = "test_data/e25323c62af644fba97afb846261b05b.
    jpg", etc.

# Adding the full path to the file_name instead of just "
    trainORtest_data/xxx.jpg"
df_test['id'] = df_test['id'].apply(lambda x: os.path.join(base_dir, x))
df_train['file_name'] = df_train['file_name'].apply(lambda x: os.path.
    join(base_dir, x))

all_image_paths = df_train['file_name'].values
all_labels = df_train['label'].values

# Splitting train/validation (95% / 5%)
train_paths, val_paths, train_labels, val_labels = train_test_split(
    all_image_paths,
    all_labels,
    test_size=0.05,
    stratify=all_labels,
    random_state=42
)


#train_paths = train_paths[:1500]
```

```
#val_paths = val_paths[:150]
print(f"Train Data: {len(train_paths)}")
print(f"Validation Data: {len(val_paths)}")
print(train_paths)

train_data = CustomDataset2(train_paths,train_labels, transform=
    train_transforms)
val_data   = CustomDataset2(val_paths,val_labels ,transform=
    validation_transform)

train_loader = DataLoader(dataset=train_data, batch_size=8, shuffle=True
    ,  num_workers=4,pin_memory=True)
val_loader   = DataLoader(dataset=val_data,   batch_size=8, shuffle=
    False, num_workers=4, pin_memory=True)

print(f"Train Dataset size: {len(train_data)}")
print(f"Validation Dataset size: {len(val_data)}")

for image, label in train_loader:
    print(image.shape)  # Devrait  tre  torch.Size([32, 3, 224, 224])
    print(label.shape)  # Devrait  tre  torch.Size([32])
    break

device='cuda' if torch.cuda.is_available() else 'cpu'
print(device)

import torch.optim as optim
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import f1_score, roc_auc_score

# Initialisation
training_param = [p for p in model.parameters() if p.requires_grad]
print(len(training_param))
optimizer = torch.optim.AdamW(training_param, lr=1e-4,weight_decay=1e-2)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max
    =10)
criterion = nn.BCEWithLogitsLoss()
torch.cuda.empty_cache()
scaler = torch.GradScaler(device='cuda')
print(scaler)

train_losses, train_accuracies = [], []
val_losses, val_accuracies, val_f1_scores, val_roc_aucs = [], [], [], []

epochs = 5
best_val_loss = float('inf')
early_stopping_patience = 7
early_stopping_counter = 0

for epoch in range(epochs):
    epoch_loss = 0.0
    epoch_accuracy = 0.0
    model.train()

    for data, labels in tqdm(train_loader, desc=f"Training epoch {epoch
    +1}"):
        data = data.to(device)
```

```
        labels = labels.to(device).float()

        optimizer.zero_grad()
        with torch.autocast(device_type='cuda'):
            outputs = model(data).squeeze(1)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        preds = (torch.sigmoid(outputs) > 0.5).float()
        epoch_accuracy += (preds == labels).float().mean().item()
        epoch_loss += loss.item()

    train_losses.append(epoch_loss/len(train_loader))
    train_accuracies.append(epoch_accuracy/len(train_loader))

    # Validation
    model.eval()
    epoch_val_loss = 0.0
    epoch_val_accuracy = 0.0
    val_pred_class = []
    val_label_class = []

    with torch.no_grad():
        for data, labels in tqdm(val_loader, desc=f"validation_epoch {epoch+1}"):
            data, labels = data.to(device), labels.to(device).float()

            with torch.autocast(device_type='cuda'):
                outputs = model(data).squeeze(1)
                loss = criterion(outputs, labels)

            preds = (torch.sigmoid(outputs) > 0.5).float()
            epoch_val_accuracy += (preds == labels).float().mean().item()
            epoch_val_loss += loss.item()
            val_pred_class.extend(preds.cpu().numpy())
            val_label_class.extend(labels.cpu().numpy())

    val_loss = epoch_val_loss/len(val_loader)
    val_acc = epoch_val_accuracy/len(val_loader)
    val_f1 = f1_score(np.array(val_label_class), np.array(val_pred_class))
    val_roc_auc = roc_auc_score(np.array(val_label_class), np.array(val_pred_class))

    val_losses.append(val_loss)
    val_accuracies.append(val_acc)
    val_f1_scores.append(val_f1)
    val_roc_aucs.append(val_roc_auc)

    print(f"Epoch [{epoch+1}/{epochs}] "
          f"Train Loss: {epoch_loss/len(train_loader):.4f} | "
          f"Train Acc: {epoch_accuracy/len(train_loader):.4f} | "
          f"Val Loss: {val_loss:.4f} | "
```

```
            f"Val Acc: {val_acc:.4f} | "
            f"Val F1: {val_f1:.4f} | "
            f"Val ROC AUC: {val_roc_auc:.4f}")

    scheduler.step()

    # Early stopping
    if epoch_val_loss < best_val_loss:
        best_val_loss = epoch_val_loss
        torch.save(model.state_dict(), 'best_model.pth')
        early_stopping_counter = 0
    else:
        early_stopping_counter += 1
        if early_stopping_counter >= early_stopping_patience:
            print("Early stopping triggered")
            break

# Plot Loss and Accuracy

fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plot Loss
axs[0].plot(range(1, len(train_losses) + 1), train_losses, label='Train Loss', color='blue')
axs[0].plot(range(1, len(val_losses) + 1), val_losses, label='Validation Loss', color='red')
axs[0].set_title('Loss per Epoch')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend()

# Plot Accuracy
axs[1].plot(range(1, len(train_accuracies) + 1), train_accuracies, label='Train Accuracy', color='blue')
axs[1].plot(range(1, len(val_accuracies) + 1), val_accuracies, label='Validation Accuracy', color='red')
axs[1].set_title('Accuracy per Epoch')

axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
axs[1].legend()

# Show the plots
plt.tight_layout()
plt.savefig('Accuracy per Epoch.png')
plt.show()

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

# Compute ROC curve
fpr, tpr, _ = roc_curve(np.array(val_label_class , dtype=int), np.array(val_pred_class, dtype=int))

# Plot ROC Curve
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {val_roc_auc:.4f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guess')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid()
plt.show()


from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
# Generate and plot Confusion Matrix
conf_matrix = confusion_matrix(val_label_class , val_pred_class)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Print Classification Report
print("\nClassification Report:")
print(classification_report(val_label_class , val_pred_class))


base_dir = '/kaggle/input/ai-vs-human-generated-dataset'
test_csv_path  = os.path.join(base_dir, 'test.csv')
df_test = pd.read_csv(os.path.join(base_dir, 'test.csv'))
df_test['id'] = df_test['id'].apply(lambda x: os.path.join(base_dir, x))
class TestImageDataset(Dataset):
    def __init__(self, file_list, transform=None):
        self.file_list = file_list
        self.transform = transform

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, idx):
        img_path = self.file_list[idx]
        img = Image.open(img_path).convert("RGB")
        if self.transform:
            img = self.transform(img)
        return img, os.path.basename(img_path)

test_dataset = TestImageDataset(df_test['id'].values, transform=test_transforms)
test_loader = DataLoader(test_dataset, batch_size=8, shuffle=False, num_workers=4)

model.eval()
```

```
predictions = []
image_names = []

with torch.no_grad():
    for data, names in tqdm(test_loader, desc="Predicting"):
        data = data.to(device)

        # On r cup re la classe pr dite
        output = model(data).squeeze(1)  # Ensure correct dimensions

        # Apply sigmoid activation to convert logits to probabilities
        probs = torch.sigmoid(output)

        # Convert probabilities to binary class (0 or 1) using threshold
    0.5
        preds = (probs > 0.5).int()

        predictions.extend(preds.cpu().numpy())
        image_names.extend([f"test_data_v2/{name}" for name in names])

# Cr er le DataFrame au format "id,label"
submission_df = pd.DataFrame({
    'id': image_names,
    'label': predictions
})

submission_df.head()
submission_df.to_csv("submission.csv", index=False)
print("Submission file generated: submission.csv")
```