Université Gustave Eiffel

Faculté des Sciences – Département de Mathématiques et Informatique

Mémoire de Fin d'Études

Master 2 – Probabilités et Statistiques

Titre :

Classification d'images déséquilibrées et incomplètes par réseaux de neurones convolutifs (CNN)

Présenté par : **Khalid Abdelli**

Encadré par : **M. Mohamed Hebiri**

Année universitaire 2024–2025

27 août 2025

Introduction

La classification d'images constitue l'un des champs les plus dynamiques de la vision par ordinateur, avec de nombreuses applications en santé, en sécurité, en transport ou encore en industrie. Grâce aux avancées en apprentissage profond (*deep learning*), et notamment aux réseaux de neurones convolutifs (CNN), il est désormais possible d'atteindre d'excellentes performances sur des tâches complexes de reconnaissance visuelle.

Cependant, dans les situations réelles, les jeux de données ne sont pas toujours parfaitement préparés. Deux problèmes apparaissent fréquemment :

— la présence de **zones corrompues ou masquées** sur certaines images ;
— un **déséquilibre dans la distribution des classes**, où certaines catégories sont sous-représentées.

Ce mémoire traite précisément de ces deux difficultés. Nous avons travaillé sur un dataset original contenant des images partiellement masquées par des rectangles noirs, et présentant une forte inégalité entre le nombre d'exemples par classe.

1. Prétraitement par Inpainting. La première étape de notre méthodologie consiste à corriger les images comportant des rectangles noirs (zones d'information manquante). Pour cela, nous avons appliqué une méthode d'*inpainting* basée sur l'algorithme de Telea, implémentée via la bibliothèque OpenCV. Cette technique permet de remplir les zones corrompues de manière visuellement cohérente à partir du voisinage. L'objectif est d'améliorer la qualité des entrées du modèle pour un apprentissage plus robuste.

2. Rééquilibrage du dataset par oversampling. Le second problème identifié est le déséquilibre entre les classes. En particulier, la classe 3 était significativement sous-représentée, rendant son apprentissage difficile. Nous avons donc utilisé une stratégie d'augmentation de données (*data augmentation*) combinée à un oversampling contrôlé, afin de créer des exemples artificiels diversifiés à partir d'images existantes. Ce rééquilibrage a pour but d'éviter le biais du modèle vers les classes dominantes.

3. Modélisation par CNN. Nous avons ensuite conçu et entraîné un réseau de neurones convolutif (CNN) profond, structuré en blocs convolutionnels, couches de pooling, activation ReLU, et régularisation par dropout. Ce modèle a été appliqué à la version corrigée du dataset afin d'en évaluer les performances sur une tâche de classification à 10 classes.

4. Comparaison des résultats. Une partie essentielle de ce mémoire consiste à comparer les performances du modèle dans plusieurs scénarios :

— Sans aucun traitement ;

— Avec inpainting uniquement ;
— Avec oversampling uniquement ;
— Avec les deux traitements combinés.

Cette comparaison permet de mesurer quantitativement les gains obtenus par chaque stratégie de prétraitement.

Organisation du mémoire. Ce mémoire est organisé de la manière suivante :

— Le **Chapitre 1** présente l'état de l'art sur les techniques d'inpainting, d'équilibrage des données, et sur les architectures CNN ;
— Le **Chapitre 2** décrit en détail la méthodologie adoptée pour corriger les images et équilibrer les classes ;
— Le **Chapitre 3** expose l'architecture du modèle CNN ainsi que les choix d'entraînement ;
— Le **Chapitre 4** présente les résultats expérimentaux et les comparaisons de performances ;
— Enfin, le **Chapitre 5** conclut et propose des perspectives d'amélioration.

ÈTAT DE L'ART

1 Traitement des images incomplétes : inpainting des zones masquées

Dans de nombreuses applications de vision par ordinateur, les images peuvent comporter des zones corrompues ou masquées. Ces artefacts, souvent dus à des erreurs de capteurs ou des masques appliqués, rendent l'apprentissage plus difficile. Pour y remédier, on utilise des techniques appelées **inpainting**, dont l'objectif est de restaurer les régions manquantes de manière visuellement plausible.

1.1. Inpainting par diffusion classique (OpenCV)

Deux algorithmes principaux sont disponibles dans OpenCV :

— **Algorithme de Telea (2004)** : c'est une méthode rapide de propagation de front. L'idée est de remplir les pixels manquants en propagent les informations voisines, en pondérant leur influence selon la distance à la frontière connue.

— **Algorithme de Navier-Stokes** : inspiré des équations de la dynamique des fluides, cette méthode réalise une diffusion des gradients dans les zones masquées, en maintenant une cohérence des structures.

Ces deux approches sont bien adaptées à des masques de petite taille. Nous avons retenu **Telea** pour sa vitesse, sa stabilité, et sa capacité à gérer nos rectangles noirs (bordures d'image).

1.2. Inpainting par apprentissage profond

Pour des restaurations plus complexes, on utilise des approches basées sur l'apprentissage :

— **Autoencoders convolutionnels** : ils apprennent à reconstruire une image à partir d'une entrée partiellement corrompue. Le goulot d'étranglement force le réseau à capturer les structures globales.

— **GANs (Generative Adversarial Networks)** : un générateur tente de remplir les zones manquantes, pendant qu'un discriminateur juge de la cohérence du résultat. Les GANs modernes sont capables de génerer des textures et structures fines.

— **Contextual Attention** : architecture plus avancée permettant d'utiliser dynamiquement des parties connues de l'image pour guider le remplissage.

Ces approches sont puissantes mais demandent beaucoup de données et de puissance de calcul. Pour notre cas, la solution classique (Telea) s'est avérée suffisante et plus rapide à mettre en œuvre.

2 Traitement du déséquilibre de classes : Oversampling et alternatives

Dans de nombreux datasets réels, certaines classes sont sous-représentées. Cela entraîne un biais d'apprentissage : le modèle tend à favoriser les classes majoritaires. Plusieurs stratégies existent pour compenser ce déséquilibre.

2.1. Approches simples

— **Oversampling naïf** : dupliquer les exemples de la classe rare jusqu'à égalisation.
— **Undersampling** : supprimer aléatoirement des exemples des classes majoritaires.
— **Pondération de la fonction de perte** : attribuer un poids plus important aux erreurs sur les classes rares.

2.2. Techniques avancées

— **SMOTE (Synthetic Minority Oversampling Technique)** : génère des points synthétiques en interpolant entre exemples proches de la classe minoritaire.
— **ADASYN** : extension de SMOTE qui cible plus fortement les zones où le modèle a du mal à généraliser.
— **Augmentation de données visuelle** : comme dans notre projet, on applique des transformations aléatoires (flip, rotation, crop, jitter) pour créer de la diversité.

Nous avons choisi l'approche **oversampling + augmentation** car elle est simple, visuelle, reproductible, et très efficace sur des images de petite taille (32x32).

3 Modèles de classification d'images : des CNN classiques aux modèles profonds

3.1. Réseaux de neurones classiques (MLP)

Les perceptrons multicouches (MLP) sont adaptés aux données tabulaires. Pour les images, ils ignorent la structure spatiale et deviennent vite inefficaces.

3.2. Réseaux Convolutifs (CNN)

Les CNN sont conçus pour capturer les motifs visuels locaux (bords, textures, formes). Ils utilisent des couches :

- **Conv2D** : extraction de motifs avec des noyaux 3x3, 5x5, etc.
- **ReLU** : activation non-linéaire $f(x) = \max(0, x)$.
- **MaxPooling** : réduction de dimension (ex : 2x2), garde le max.
- **Dropout** : réduction du sur-apprentissage, en désactivant des neurones.

3.3. Architectures notables

- **LeNet-5 (1998)** : premier CNN pour la reconnaissance de chiffres manuscrits.
- **AlexNet (2012)** : grand retour du deep learning avec ImageNet.
- **VGG, ResNet** : empilement de couches, résidual learning.

3.4. Alternatives à CNN

- **SVM + HOG** : anciennes méthodes efficaces sur petites images.
- **ViT (Vision Transformers)** : modèles sans convolution, inspirés du NLP. Encore très lourds pour des images simples.

3.5. Choix retenu dans notre projet

Nous avons conçu un **CNN personnalisé** avec plusieurs blocs convolutionnels + pooling, suivi de couches fully-connected. Ce modèle est bien adapté à la taille 32x32, tout en permettant une bonne expressivité. Sa structure est détaillée dans la section Architecture.

Conclusion. Cette section a présenté les principales solutions techniques connues pour traiter les images incomplètes, réquilibrer les classes, et construire un modèle de classification robuste. Dans la suite du mémoire, nous allons détailler concrètement comment nous avons implémenté ces choix dans notre propre pipeline.

4 Problème 1 : Zones noires dans les images

4.1 Impact des zones masquées

Certaines images de notre dataset présentent des zones rectangulaires noires, c'est-à-dire des régions où tous les pixels ont une intensité nulle. Ces anomalies sont souvent

dues à des défaillances de capteurs, à un mauvais traitement des données ou à des pertes d'information lors de la collecte.

Ces zones altèrent la structure visuelle des images et posent problème pour l'apprentissage automatique. En particulier, elles ont plusieurs effets négatifs :

— Le réseau CNN ne peut pas extraire de caractéristiques discriminantes dans ces régions masquées.

— Les gradients calculés pendant l'entraînement sont biaisés ou nuls, ce qui ralentit la convergence du modèle.

— Le réseau peut apprendre à associer artificiellement la présence de rectangles noirs à certaines classes.

4.2 Méthode utilisée : Inpainting par l'algorithme de Telea

Pour corriger ces défauts visuels, nous avons appliqué une méthode d'**inpainting** (reconstruction d'image) basée sur l'algorithme proposé par Alexandru Telea (2004), implémenté dans la bibliothèque `OpenCV`. Cette technique permet de remplir les pixels manquants à partir de leur voisinage de façon visuellement cohérente.

Principe général de l'inpainting

L'objectif est de propager l'information visuelle (couleurs, bords, textures) depuis les pixels valides (connus) vers les pixels invalides (à restaurer), tout en respectant les contours et la continuité de l'image.

L'algorithme repose sur les étapes suivantes :

1. Détection des régions corrompues via un masque binaire (pixels noirs = 1, autres = 0).
2. Remplissage itératif des pixels du contour vers le centre de la zone à reconstruire.
3. À chaque étape, la valeur d'un pixel inconnu est estimée à partir des voisins connus, pondérés selon leur distance et leur alignement avec le contour.

Formule mathématique

Pour un pixel inconnu p, la valeur reconstruite $I(p)$ est donnée par une moyenne pondérée :

$$I(p) = \frac{\sum_{q \in \mathcal{N}(p)} w(p,q) \cdot I(q)}{\sum_{q \in \mathcal{N}(p)} w(p,q)}$$

où :

— $\mathcal{N}(p)$ est l'ensemble des pixels connus voisins de p,

— $w(p, q)$ est un poids attribué à chaque pixel voisin q.

Le poids est défini par :

$$w(p,q) = \frac{1}{\|p-q\|^{\alpha}} \cdot |\nabla I(q) \cdot \vec{n}_p|$$

avec :

— $\|p - q\|$: distance euclidienne entre p et q (favorise les voisins proches),

— $\nabla I(q)$: gradient de l'image en q (préserve les bords),

— $\vec{n}_p$: vecteur normal au bord de la zone masquée en p (oriente la propagation),

— α : facteur de pondération (souvent fixé à 1).

Ce mécanisme assure une reconstruction fluide et naturelle des motifs manquants en exploitant la géométrie locale de l'image.

Exemple détaillé d'inpainting TELEA sur un pixel RGB

Soit une image avec les pixels suivants (positions et valeurs RGB) :

Position	Pixel	RGB
(0,0)	q_1	$[110, 10, 5]$
(0,1)	q_2	$[120, 20, 10]$
(0,2)	q_3	$[130, 30, 15]$
(1,0)	q_4	$[140, 40, 20]$
(1,1)	p (à remplir)	$[\mathbf{0}, \mathbf{0}, \mathbf{0}]$
(1,2)	q_5	$[160, 60, 30]$
(2,0)	q_6	$[170, 70, 35]$
(2,1)	q_7	$[180, 80, 40]$
(2,2)	q_8	$[190, 90, 45]$

La normale utilisée est : $\vec{n}_p = (1, 0)$ et le paramètre $\alpha = 1$.

Étape 1 : Calcul des distances $\|p - q\|$

$$\|p-q\| = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

Pixel	Position	Calcul	Distance	Valeur
q_1	(0,0)	$\sqrt{(1-0)^2+(1-0)^2}$	$\sqrt{2}$	1.414
q_2	(0,1)	$\sqrt{(1-0)^2+(1-1)^2}$	1	1
q_3	(0,2)	$\sqrt{(1-0)^2+(1-2)^2}$	$\sqrt{2}$	1.414
q_4	(1,0)	$\sqrt{(1-1)^2+(1-0)^2}$	1	1
q_5	(1,2)	$\sqrt{(1-1)^2+(1-2)^2}$	1	1
q_6	(2,0)	$\sqrt{(1-2)^2+(1-0)^2}$	$\sqrt{2}$	1.414
q_7	(2,1)	$\sqrt{(1-2)^2+(1-1)^2}$	1	1
q_8	(2,2)	$\sqrt{(1-2)^2+(1-2)^2}$	$\sqrt{2}$	1.414

Étape 2 : Calcul des gradients horizontaux $\partial I_c/\partial x(q)$

On approxime le gradient horizontal pour chaque canal $c \in \{R, G, B\}$ au pixel q par :

$$\frac{\partial I_c}{\partial x}(q) \approx I_c(x_q + 1, y_q) - I_c(x_q - 1, y_q)$$

En cas de bord, on utilise la valeur du pixel lui-même.

Pixel	$I_R(x_q-1,y_q)$	$I_R(x_q+1,y_q)$	$\partial I_R/\partial x$	$\partial I_G/\partial x$	$\partial I_B/\partial x$
q_1	110 (bord)	140	140 - 110 = 30	40 - 10 = 30	20 - 5 = 15
q_2	120 (bord)	p (inconnu, approx. 120)	0	0	0
q_3	130 (bord)	160	160 - 130 = 30	60 - 30 = 30	30 - 15 = 15
q_4	110	170	170 - 110 = 60	70 - 10 = 60	35 - 5 = 30
q_5	130	190	190 - 130 = 60	90 - 30 = 60	45 - 15 = 30
q_6	140	170 (bord)	170 - 140 = 30	70 - 40 = 30	35 - 20 = 15
q_7	p (inconnu, 0)	180 (bord)	180 - 0 = 180	80 - 0 = 80	40 - 0 = 40
q_8	160	190 (bord)	190 - 160 = 30	90 - 60 = 30	45 - 30 = 15

Étape 3 : Calcul des poids $w(p, q)$

Avec la formule :

$$w(p,q) = \frac{1}{\|p-q\|^\alpha} \times \frac{|\partial I_R/\partial x(q)| + |\partial I_G/\partial x(q)| + |\partial I_B/\partial x(q)|}{3}$$

où $\alpha = 1$.

Pixel	Distance	Moyenne absolue des gradients	Poids $w(p,q)$
q_1	1.414	$\frac{30+30+15}{3} = 25$	$25/1.414 = 17.68$
q_2	1	0	0
q_3	1.414	25	17.68
q_4	1	50	50
q_5	1	50	50
q_6	1.414	25	17.68
q_7	1	100	100
q_8	1.414	25	17.68

Étape 4 : Calcul de la couleur finale du pixel p

Pour chaque canal $c \in \{R, G, B\}$, on calcule :

$$I_c(p) = \frac{\sum_q w(p,q) \times I_c(q)}{\sum_q w(p,q)}$$

—

Calcul des sommes des poids :

$$\sum_q w(p,q) = 17.68 \times 4 + 50 + 50 + 100 + 0 = 270.72$$

—

Calcul des produits $w(p,q) \times I_c(q)$ pour chaque canal :

Pixel	$w(p,q)$	$I_R(q)$	$w \times I_R$	$I_G(q)$	$w \times I_G$	$I_B(q)$	$w \times I_B$
q_1	17.68	110	1945	10	177	5	88
q_2	0	120	0	20	0	10	0
q_3	17.68	130	2298	30	530	15	265
q_4	50	140	7000	40	2000	20	1000
q_5	50	160	8000	60	3000	30	1500
q_6	17.68	170	3005	70	1238	35	619
q_7	100	180	18000	80	8000	40	4000
q_8	17.68	190	3359	90	1591	45	796

—

Sommes finales :

$$\sum w \times I_R = 43607, \quad \sum w \times I_G = 16536, \quad \sum w \times I_B = 8268$$

—

Valeurs finales :

$$I_R(p) = \frac{43607}{270.72} \approx 161.1, \quad I_G(p) = \frac{16536}{270.72} \approx 61.1, \quad I_B(p) = \frac{8268}{270.72} \approx 30.5$$

—

Conclusion : Le pixel p est reconstruit avec la couleur RGB approximative :

$$\boxed{I(p) \approx [161, 61, 31]}$$

4.3 Implémentation OpenCV et choix des paramètres

La fonction utilisée dans notre projet pour réaliser l'inpainting est `cv2.inpaint()`, qui se définit comme suit :

Listing 1 – Inpainting avec OpenCV

```
corrected_img = cv2.inpaint(src=image, inpaintMask=mask,
                            inpaintRadius=radius,
                            flags=cv2.INPAINT_TELEA)
```

Paramètres principaux

- **image** : image originale à corriger (format `uint8`, couleur ou niveau de gris).
- **mask** : image binaire du même format que l'image, avec des pixels à corriger en blanc (255) et les pixels valides en noir (0).
- **inpaintRadius** : rayon d'inpainting. C'est un paramètre clé qui contrôle la taille du voisinage utilisé pour reconstruire un pixel masqué.
- **flags** : méthode utilisée pour l'inpainting. Deux options :
  - `cv2.INPAINT_NS` : méthode de Navier-Stokes (plus lente, pour les grandes zones continues),
  - `cv2.INPAINT_TELEA` : méthode rapide et efficace basée sur la distance géodésique (celle que nous avons utilisée).

Choix du rayon d'inpainting

Le paramètre `inpaintRadius` contrôle la taille du voisinage considéré pour estimer la valeur d'un pixel manquant. Un rayon trop grand produit un flou excessif ; un rayon trop petit ne permet pas une propagation cohérente. **Nous avons retenu la valeur de 1.0**, qui offre un bon compromis entre netteté et régularité.

4.4 Analyse visuelle avant/après inpainting

Pour évaluer visuellement l'effet de cette correction, nous illustrons ci-dessous un exemple d'image avant et après inpainting. On constate que la zone noire est reconstruite de façon réaliste et que les bords de l'objet sont bien prolongés.

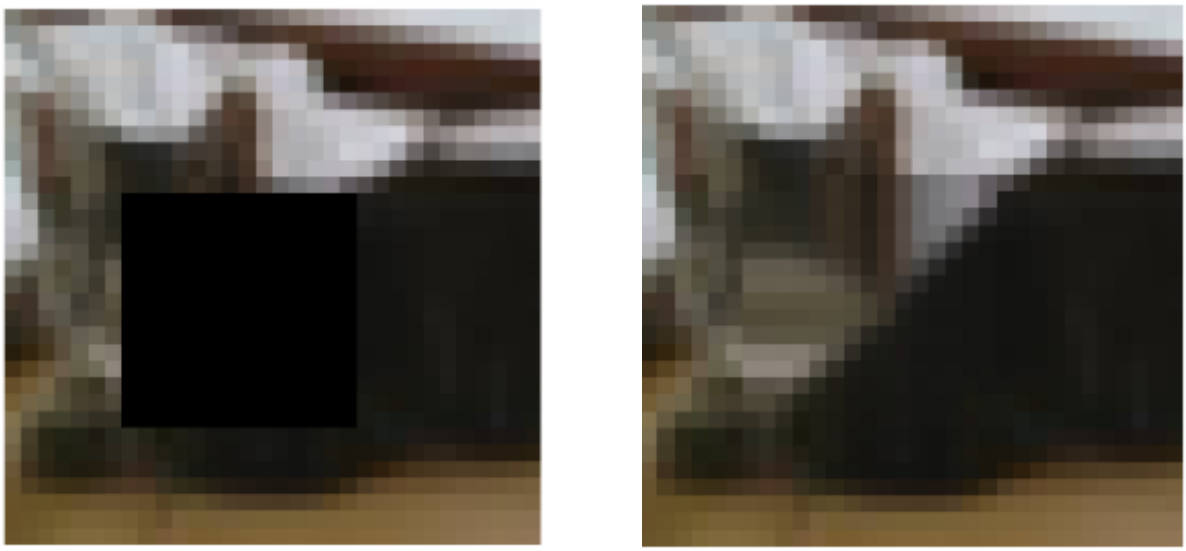

FIGURE 1 – Effet de l'inpainting sur une image contenant une zone noire

4.5 Évaluation de la qualité de l'inpainting (OpenCV)

Pour évaluer objectivement la qualité des images reconstruites par notre méthode d'inpainting, nous avons utilisé trois métriques standards issues du traitement d'images :

— **Erreur quadratique moyenne (MSE)** :

$$\text{MSE} = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} (I_{ij} - \hat{I}_{ij})^2$$

où I est l'image d'origine (sans défaut) et $\hat{I}$ l'image reconstruite. Plus le MSE est faible, plus la reconstruction est fidèle.

— **Peak Signal-to-Noise Ratio (PSNR)** : exprimé en décibels (dB), il mesure le rapport entre la puissance maximale du signal et le bruit introduit par la reconstruction.

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{\text{MSE}} \right)$$

où MAX_I est la valeur maximale possible d'un pixel (par exemple 255 pour des images 8 bits). Un PSNR plus élevé indique une meilleure qualité visuelle.

— **Structural Similarity Index (SSIM)** :

mesure de similarité perceptive entre deux images, prenant en compte la luminance, le contraste et la structure. Elle est comprise entre -1 et 1, et vaut 1 si les deux images sont identiques.

La formule du SSIM est la suivante :

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

où :

— μ_x, μ_y : moyennes locales des images x et y,
— σ_x^2, σ_y^2 : variances locales des images x et y,
— σ_{xy} : covariance entre x et y,
— C_1, C_2 : constantes de stabilisation (pour éviter la division par zéro).

5 Problème 2 : Déséquilibre des classes

5.1 Constat du déséquilibre

Dans de nombreux problèmes de classification, les données ne sont pas réparties équitablement entre les différentes classes. Ce phénomène, appelé **déséquilibre de classes**, peut nuire fortement aux performances d'un modèle d'apprentissage automatique, en le biaisant vers les classes majoritaires.

Formellement, soit un jeu de données $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, avec $y_i \in \{0, 1, \ldots, K-1\}$. On note $\mathcal{D}_k = \{(x_i, y_i) \in \mathcal{D} \mid y_i = k\}$ l'ensemble des exemples appartenant à la classe k. Si $\exists k$ tel que $|\mathcal{D}_k| \ll \max_j |\mathcal{D}_j|$, alors le dataset est dit déséquilibré.

À l'analyse initiale du dataset, nous avons observé une distribution très inégale des étiquettes de classes. En particulier, la classe **3** est significativement sous-représentée avec un nombre d'images bien inférieur aux autres classes. Ce déséquilibre peut induire plusieurs problèmes lors de l'apprentissage d'un classificateur :

— Le modèle devient biaisé en faveur des classes majoritaires.
— Les performances sont artificiellement élevées sur les classes fréquentes, mais faibles sur les classes rares.
— Le réseau peut apprendre à ignorer les classes peu représentées.

5.2 Méthode de correction : Oversampling ciblé

Pour pallier ce déséquilibre, nous avons utilisé une stratégie d'**oversampling ciblé** sur la classe 3. L'idée est de **générer artificiellement des images nouvelles** à partir d'images réelles existantes, en leur appliquant diverses transformations aléatoires.

Nous avons fixé un objectif de **5000 images** pour la classe 3. Pour cela, nous avons conservé les images originales, et généré des images supplémentaires par *data augmentation*, en utilisant une combinaison de transformations simples mais efficaces avec `Torchvision`.

Voici les transformations appliquées individuellement, chacune étant illustrée par un exemple visuel sur une même image de la classe 3.

5.3 Transformation 1 : Flip horizontal

Le retournement horizontal (`RandomHorizontalFlip`) consiste à inverser l'image de gauche à droite. C'est une transformation utile lorsque les objets sont symétriques ou que leur orientation n'est pas discriminante.

But : multiplier les cas de figure pour un même objet sans modifier son contenu sémantique.

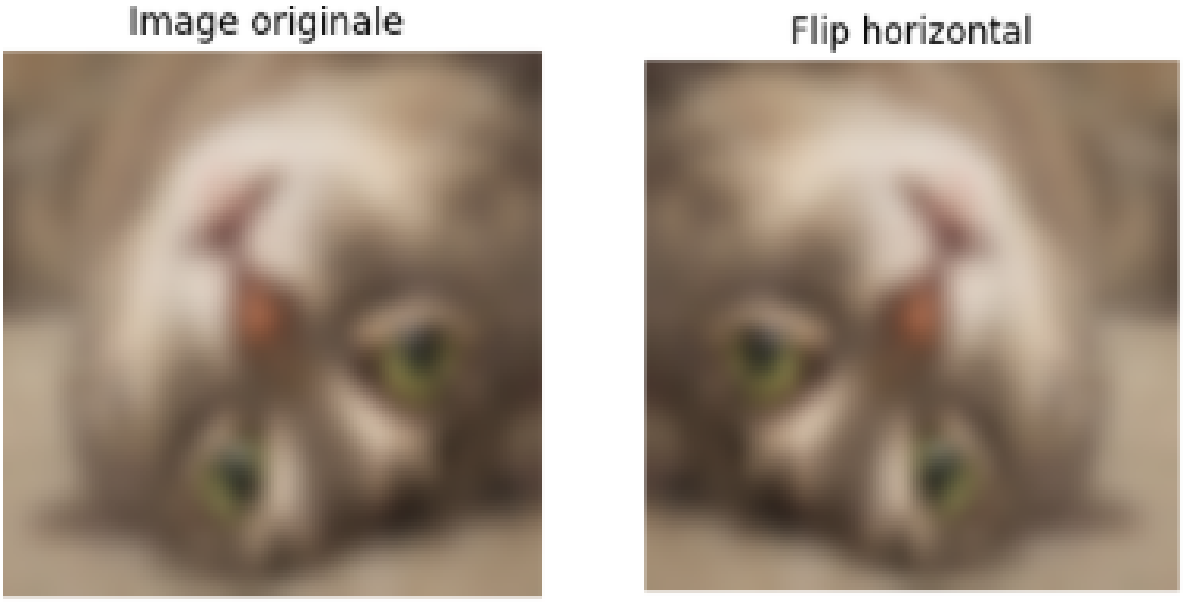

FIGURE 2 – Retournement horizontal de l'image de la classe 3

5.4 Transformation 2 : Rotation aléatoire

La rotation aléatoire (`RandomRotation`) applique une inclinaison aléatoire (dans notre cas entre -20 et +20 degrés). Elle permet de rendre le modèle robuste à la position de l'objet.

But : simuler une prise de vue légèrement inclinée ou un objet mal aligné.

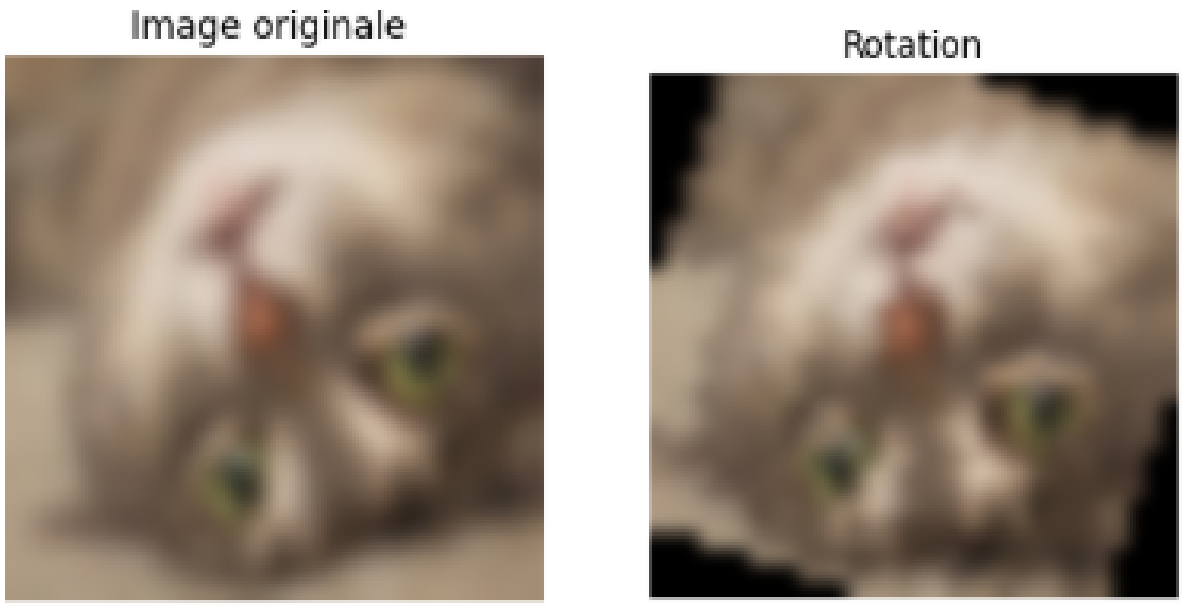

FIGURE 3 – Rotation aléatoire de l'image (angle ±20°)

5.5 Transformation 3 : Recadrage aléatoire

La transformation `RandomResizedCrop` applique un recadrage aléatoire de l'image, suivi d'un redimensionnement à la taille originale (ici 32×32). Elle simule un zoom partiel sur l'objet.

But : inciter le modèle à se concentrer sur des détails locaux ou des parties spécifiques de l'image.

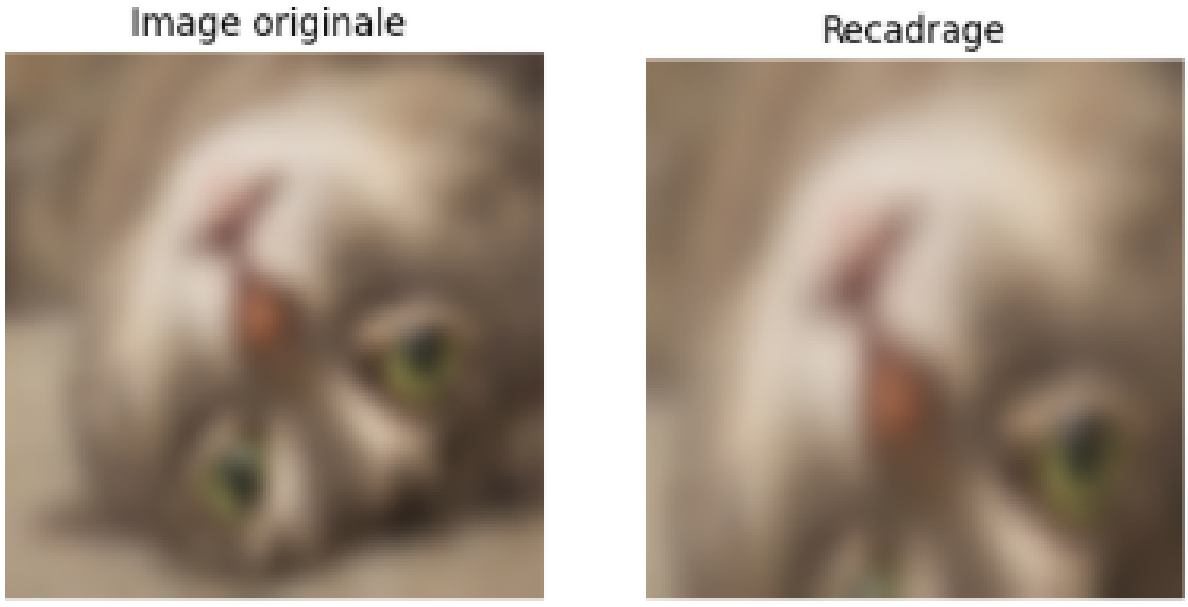

FIGURE 4 – Recadrage aléatoire suivi de redimensionnement

5.6 Transformation 4 : ColorJitter

Cette transformation aléatoire modifie la **luminosité** et le **contraste** de l'image via `ColorJitter`. Elle simule des variations d'éclairage ou de qualité de capture.

But : apprendre au modèle à être invariant aux variations de lumière.

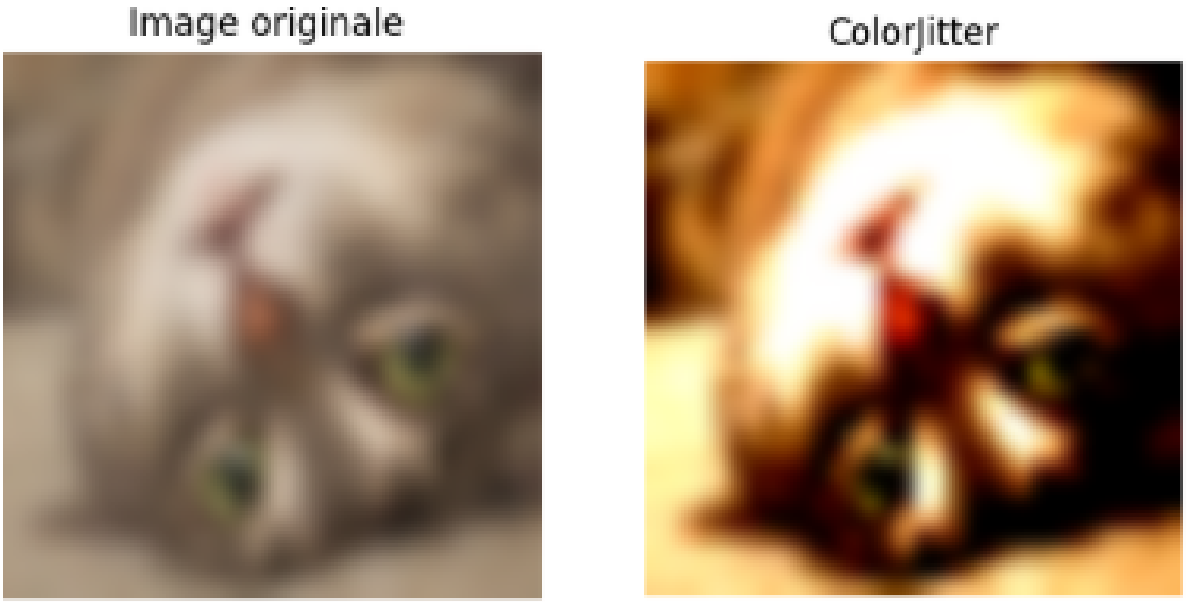

FIGURE 5 – Ajustement aléatoire de la luminosité et du contraste

5.7 Résumé de la méthode et intégration

L'oversampling a été appliqué uniquement à la classe 3, en répétant le processus de sélection et transformation jusqu'à atteindre 5000 images. Les autres classes n'ont pas été modifiées, ce qui permet de conserver leur distribution naturelle.

En fin de traitement :

— La classe 3 contient désormais 5000 images équilibrées.

— Les images synthétiques enrichissent la diversité visuelle de cette classe.

— Le modèle CNN est moins biaisé et apprend mieux à généraliser.

Ce rééquilibrage s'inscrit dans une stratégie globale de prétraitement des données, combinée à l'inpainting, que nous détaillons plus loin à travers l'évaluation expérimentale.

6 Architecture du modèle DeepCNN : une étude détaillée

6.1 Introduction au réseau convolutif

Le modèle développé repose sur un réseau de neurones convolutif profond (**Deep Convolutional Neural Network - DeepCNN**) destiné à la classification d'images issues du jeu de données CIFAR-10. Les CNN (Convolutional Neural Networks) sont aujourd'hui l'un des piliers de l'apprentissage profond pour le traitement d'images.

Contrairement aux modèles traditionnels basés sur des vecteurs tabulaires, les CNN opèrent directement sur les images sous forme de tenseurs tridimensionnels (hauteur $\times$ largeur $\times$ canaux). Par exemple, une image RGB de taille 32×32 est représentée par un tenseur $32 \times 32 \times 3$.

Notre architecture est composée de **trois blocs convolutifs**, chacun augmentant la complexité des motifs appris, suivis d'un classifieur dense qui prend les caractéristiques extraites pour effectuer la prédiction finale.

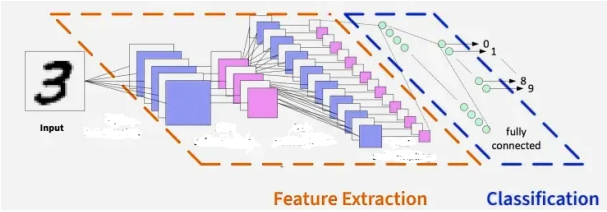

FIGURE 6 – Illustration d'un réseau de neurones convolutionnel (CNN). L'image d'entrée traverse une succession de couches convolutives (extraction de caractéristiques), de couches de pooling (réduction dimensionnelle), puis des couches entièrement connectées menant à la prédiction finale.

6.2 Principe mathématique de la convolution

La convolution 2D est une opération essentielle permettant d'extraire des motifs locaux dans une image. Elle consiste à appliquer un petit noyau de taille fixe (souvent 3×3) sur des sous-régions de l'image.

$$S(i,j) = (I * K)(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} I(i+m, j+n) \cdot K(m,n)$$

— I : l'image d'entrée (matrice ou tenseur)
— K : le noyau (filtre de convolution) appris par le réseau
— (i,j) : position du pixel sur la sortie
— k : demi-taille du noyau (ex : $k = 1$ pour un noyau 3×3)

Le filtre "glisse" sur l'image pour générer une nouvelle carte d'activation. Chaque filtre est entraîné pour détecter un motif spécifique (bord, texture, orientation...).

L'usage d'un **padding** (=1 ici) permet de conserver les dimensions de l'image après convolution, ce qui est utile pour un alignement correct des couches.

6.3 Bloc convolutif 1 : Extraction de caractéristiques simples

— `Conv2D(3, 64, kernel_size=3, padding=1)` : cette couche transforme une image avec 3 canaux (R, G, B) en 64 cartes de caractéristiques. Chaque filtre agit comme un détecteur de motif visuel de bas niveau.

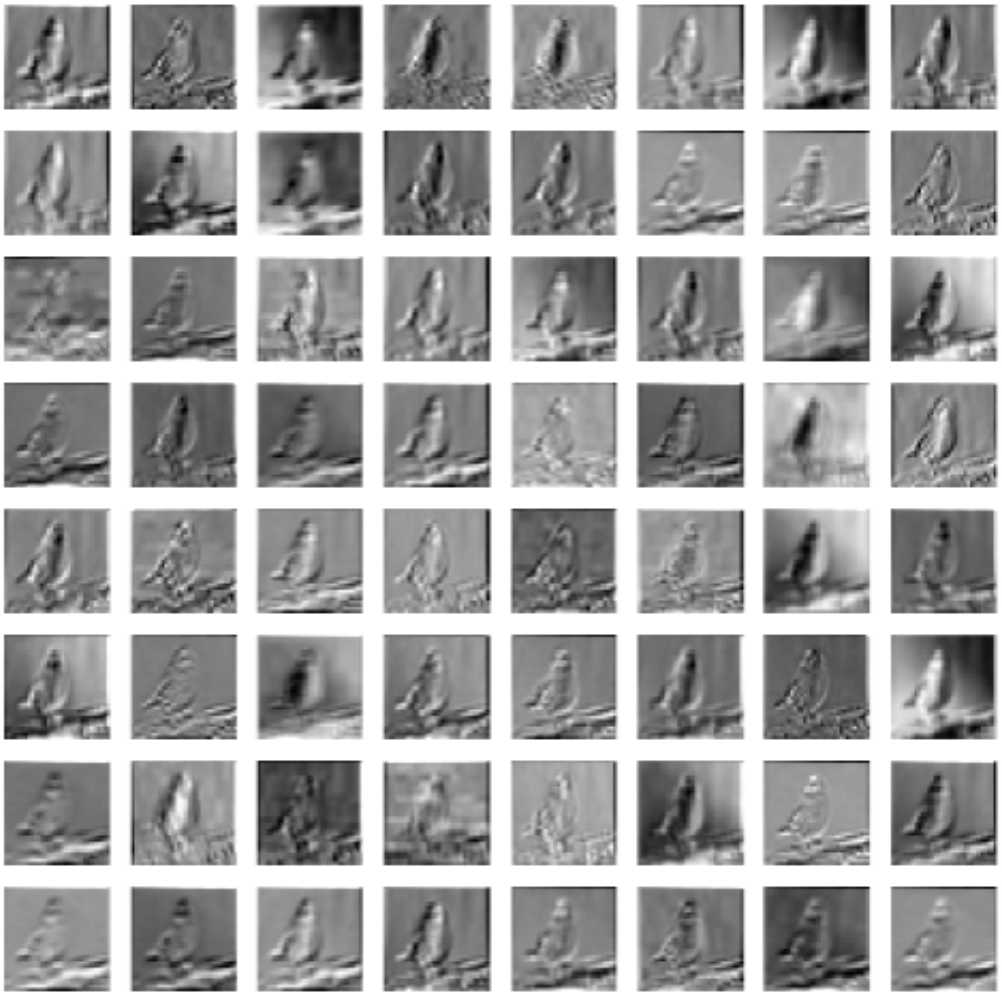

FIGURE 7 – Exemple des 64 cartes de caractéristiques extraites par le premier bloc convolutif. Les images apparaissent en niveaux de gris car chaque filtre produit une activation d'intensité (et non une image RGB). Chaque carte correspond donc à la réponse d'un filtre spécifique appliqué à l'image d'entrée.

— `BatchNorm2d(64)` : cette opération standardise les activations dans chaque canal pour accélérer l'apprentissage et stabiliser le réseau. Elle suit la formule suivante :

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{puis} \quad y = \gamma \hat{x} + \beta$$

Ici, μ_B est la moyenne du batch, σ_B l'écart-type, et les paramètres γ, β sont apprises par le réseau.

— `ReLU()` : Fonction d'activation non-linéaire qui applique :

$$\text{ReLU}(x) = \max(0, x)$$

Elle met à zéro toutes les valeurs négatives. Cela permet de supprimer les activations faibles ou inutiles, tout en conservant la structure hiérarchique du signal visuel.

— `Conv2D(64, 64, ...)` : une seconde couche convolutive affine les 64 cartes. À ce stade, le réseau commence à combiner plusieurs motifs pour détecter des structures légèrement plus complexes (coins, croisements...).

— `MaxPool2d(2)` : opération de sous-échantillonnage. Elle prend la valeur maximale dans chaque bloc 2×2, réduisant ainsi la taille spatiale de moitié ($32 \to 16$). Cela permet de diminuer le coût computationnel tout en conservant les activations les plus significatives.

6.4 Bloc convolutif 2 : Extraction intermédiaire

— Ce bloc est identique au précédent en structure, mais utilise **128 filtres**, permettant d'apprendre une plus grande variété de motifs complexes.

— Les dimensions spatiales de l'image sont encore réduites de moitié grâce au Max-Pooling ($16 \times 16 \to 8 \times 8$).

— À ce niveau, chaque pixel encode déjà des combinaisons de motifs bas niveau. Le réseau commence à apprendre des représentations plus abstraites.

6.5 Bloc convolutif 3 : Abstraction haute

— Ce bloc utilise **256 filtres**. On est ici dans des représentations de haut niveau : objets entiers, contours spécifiques d'animal ou de structure visuelle.

— Le format final des cartes est de taille 4×4, ce qui réduit l'image originale d'un facteur 8 tout en multipliant la richesse des canaux.

6.6 Passage au classifieur dense

Une fois les caractéristiques extraites, elles sont vectorisées avec :

$$\text{Flatten}(256 \times 4 \times 4) = 4096$$

— `Linear(4096, 512)` : première couche pleinement connectée. Chaque neurone reçoit toute l'information des 4096 caractéristiques pour apprendre des combinaisons complexes.

— `Dropout(0.5)` : durant l'entraînement, 50% des neurones sont éteints aléatoirement à chaque itération. Cela permet d'éviter que certains chemins neuronaux dominent l'apprentissage.

— `Linear(512, 10)` : couche de sortie. Chaque neurone correspond à une des 10 classes du dataset. La sortie brute est ensuite normalisée par une fonction softmax :

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

6.7 Fonction de perte : Cross Entropy

La fonction de perte choisie est l'entropie croisée, très utilisée pour les tâches de classification :

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Cette fonction punit fortement les prédictions incorrectes lorsque la confiance est élevée (i.e., $\hat{y}_i$ proche de 1 pour une mauvaise classe).

6.8 Optimiseur : AdamW

L'optimiseur `AdamW` combine les avantages d'Adam (adaptation dynamique du taux d'apprentissage pour chaque paramètre) avec une régularisation de type L2 pour éviter le sur-apprentissage :

$$\theta_{t+1} = \theta_t - \alpha \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

6.9 Scheduler : Réduction automatique du learning rate

Le scheduler utilisé est `ReduceLROnPlateau`. Il surveille une métrique (ex : validation loss) et, si celle-ci stagne pendant plusieurs époques, il réduit le learning rate. Cela affine l'optimisation sur les dernières étapes d'apprentissage.

6.10 Résumé complet de l'architecture

— **Entrée** : image $32 \times 32 \times 3$

— **Bloc 1** : Conv(3→64) → BN → ReLU → Conv(64→64) → BN → ReLU → MaxPool(2)

— **Bloc 2** : Conv(64→128) → BN → ReLU → Conv(128→128) → BN → ReLU → MaxPool(2)

— **Bloc 3** : Conv(128→256) → BN → ReLU → Conv(256→256) → BN → ReLU → MaxPool(2)

— **Densité** : Flatten(256×4×4) → Linear(4096→512) → Dropout → Linear(512→10)

7 Métriques de performance

Afin d'évaluer la qualité des prédictions de notre modèle de classification, nous nous appuyons sur plusieurs métriques classiques en apprentissage supervisé, calculées à partir de la *matrice de confusion*. Voici leurs définitions :

— **Précision (Precision)** : proportion de prédictions positives correctes parmi toutes les prédictions positives du modèle.

$$\text{Precision} = \frac{TP}{TP + FP}$$

— **Rappel (Recall)** : proportion de prédictions positives correctes parmi toutes les vraies instances positives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

— **F1-score** : moyenne harmonique entre la précision et le rappel. Elle permet de trouver un compromis entre les deux, en particulier dans le cas de classes déséquilibrées.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

— **Exactitude (Accuracy)** : proportion de prédictions correctes (toutes classes confondues) par rapport à l'ensemble des données.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Pour les problèmes de classification multi-classes comme le nôtre (10 classes), on utilise généralement deux types d'agrégations des scores individuels (calculés classe par classe) :

— **Moyenne macro (macro average)** : moyenne arithmétique non pondérée des scores de chaque classe. Chaque classe a le même poids, ce qui rend cette moyenne

sensible aux déséquilibres :

$$\text{Macro-F1} = \frac{1}{K} \sum_{i=1}^{K} \text{F1-score}_i$$

— **Moyenne pondérée (weighted average)** : moyenne pondérée par le nombre d'exemples dans chaque classe (le support). Cela permet de refléter l'importance relative des classes majoritaires :

$$\text{Weighted-F1} = \frac{1}{N} \sum_{i=1}^{K} n_i \cdot \text{F1-score}_i$$

où n_i est le nombre d'exemples de la classe i, et N le nombre total d'exemples.

Ces métriques ont été générées dans notre cas à l'aide de la fonction `classification_report` de `scikit-learn`, permettant une analyse détaillée des performances du modèle sur chaque classe.

7.1 Cas 1 : Évaluation sans inpainting et sans oversampling

Dans ce premier scénario, nous avons évalué notre modèle **DeepCNN** sur le jeu de test sans appliquer de techniques de prétraitement spécifiques, c'est-à-dire sans correction des zones noires (*inpainting*) et sans rééquilibrage des classes par *oversampling*.

Résultats globaux Le modèle obtient une **accuracy globale de 88.84%** sur l'ensemble de test. Cette performance globale semble satisfaisante, mais elle peut masquer des disparités significatives entre classes. Les métriques de classification (Tableau 1) montrent :

— **Macro F1** = 0.8388 : reflète une performance moyenne correcte sur toutes les classes, mais pénalisée par la classe 3.

— **Weighted F1** = 0.8870 : reflète la performance globale pondérée par le nombre d'exemples par classe, favorisant les classes majoritaires.

Analyse par classe Les classes majoritaires (0, 1, 4, 6, 7, 8 et 9) affichent des précisions et rappels supérieurs à 0.88, indiquant que le modèle reconnaît correctement la majorité des instances tout en limitant les faux positifs.

En revanche, la classe 3 présente une performance nettement inférieure :

— **Precision** = 0.5185 : les prédictions de la classe 3 sont souvent incorrectes.

— **Recall** = 0.2593 : la majorité des exemples de la classe 3 sont mal classés.

— **F1-score** = 0.3457 : synthèse faible entre précision et rappel, mettant en évidence la difficulté du modèle sur cette classe.

Cette faiblesse est due au **déséquilibre des classes** et à la **forte variabilité visuelle** des images de la classe 3.

Matrice de confusion La matrice de confusion (Figure 8) confirme ces observations : les exemples de la classe 3 sont souvent confondus avec les classes 2 et 5, ce qui traduit une **confusion inter-classes** probable. Certaines classes visuellement proches (ex. 0 et 8, 1 et 9) présentent également des confusions mineures, mais moins critiques.

Conclusion pour ce cas Sans inpainting ni oversampling, le modèle atteint une bonne performance globale mais échoue à généraliser sur les classes rares, en particulier la classe 3. Ce scénario constitue notre **référence** pour comparer l'impact des techniques de correction d'image et de rééquilibrage dans les cas suivants.

TABLE 1 – Rapport de classification — Cas 1 : sans inpainting et sans oversampling

Classe	Precision	Recall	F1-score	Support
0	0.8867	0.8987	0.8927	967
1	0.9421	0.9335	0.9378	993
2	0.8380	0.8388	0.8384	999
3	0.5185	0.2593	0.3457	108
4	0.8845	0.8584	0.8713	1017
5	0.8091	0.9016	0.8528	1006
6	0.9314	0.8700	0.8996	1015
7	0.8925	0.9068	0.8996	998
8	0.9277	0.9352	0.9315	988
9	0.9154	0.9217	0.9185	1009
Macro avg	0.8546	0.8324	0.8388	9100
Weighted avg	0.8875	0.8884	0.8870	9100

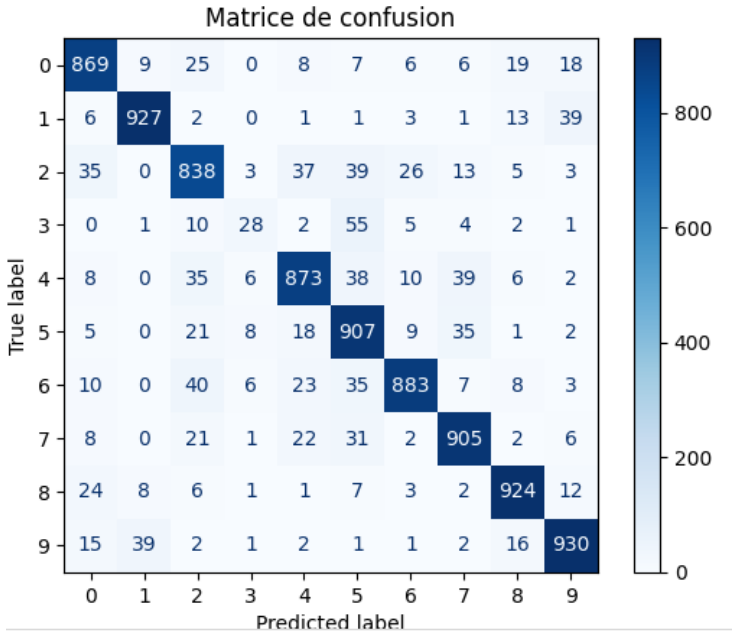

FIGURE 8 – Matrice de confusion — Cas 1 : sans inpainting et sans oversampling

Cas 2 : Avec inpainting et sans oversampling

Dans ce scénario, la technique d'inpainting de Telea a été appliquée afin de restaurer les zones noires présentes dans les images d'entrée. Aucun *oversampling* n'a été réalisé, ce qui signifie que les classes restent déséquilibrées dans l'ensemble d'apprentissage.

Résultats globaux Le modèle obtient une **accuracy globale de 89.25%** sur le jeu de test. Les moyennes macro et pondérée des métriques montrent :

- **Macro F1** = 0.8306 : reflète une performance moyenne correcte sur toutes les classes, mais encore pénalisée par les classes minoritaires comme la classe 3.
- **Weighted F1** = 0.8902 : reflète la performance globale pondérée par le nombre d'exemples par classe, favorisant les classes majoritaires.

Analyse par classe Les classes majoritaires (0, 1, 4, 6, 7, 8 et 9) présentent des précisions et rappels supérieurs à 0.88, ce qui montre que l'inpainting a permis d'améliorer légèrement la qualité des contours et des zones critiques dans les images.

Cependant, la classe 3 reste problématique :

- **Precision** = 0.3778 : les prédictions de la classe 3 sont encore largement incorrectes.
- **Recall** = 0.1589 : la majorité des images de la classe 3 sont mal classées.
- **F1-score** = 0.2237 : la faible valeur synthétise la difficulté persistante pour cette classe rare.

Le manque d'exemples représentatifs après inpainting et le déséquilibre de classes expliquent ces faibles résultats.

Matrice de confusion La Figure 9 illustre les confusions entre classes :

— Les confusions entre classes visuellement proches ont légèrement diminué grâce à l'inpainting.

— La classe 3 est majoritairement prédite comme classe 5, confirmant le problème des classes rares non rééquilibrées.

Conclusion pour ce cas L'inpainting améliore légèrement la reconnaissance des classes majoritaires en restaurant des informations visuelles importantes, mais ne résout pas le problème des classes minoritaires. La performance globale est légèrement supérieure au Cas 1, mais la classe 3 reste très faible, soulignant la nécessité de techniques de rééquilibrage pour améliorer la généralisation sur les classes rares.

TABLE 2 – Rapport de classification — Cas 2 : avec inpainting, sans oversampling

Classe	Precision	Recall	F1-score	Support
0	0.9040	0.8980	0.9010	1049
1	0.9415	0.9349	0.9382	998
2	0.8399	0.8390	0.8394	969
3	0.3778	0.1589	0.2237	107
4	0.8762	0.8636	0.8699	1041
5	0.8408	0.8772	0.8586	969
6	0.9157	0.9203	0.9180	979
7	0.9031	0.9013	0.9022	993
8	0.9257	0.9461	0.9358	1001
9	0.9078	0.9306	0.9190	994
Macro avg	0.8432	0.8270	0.8306	9100
Weighted avg	0.8891	0.8925	0.8902	9100

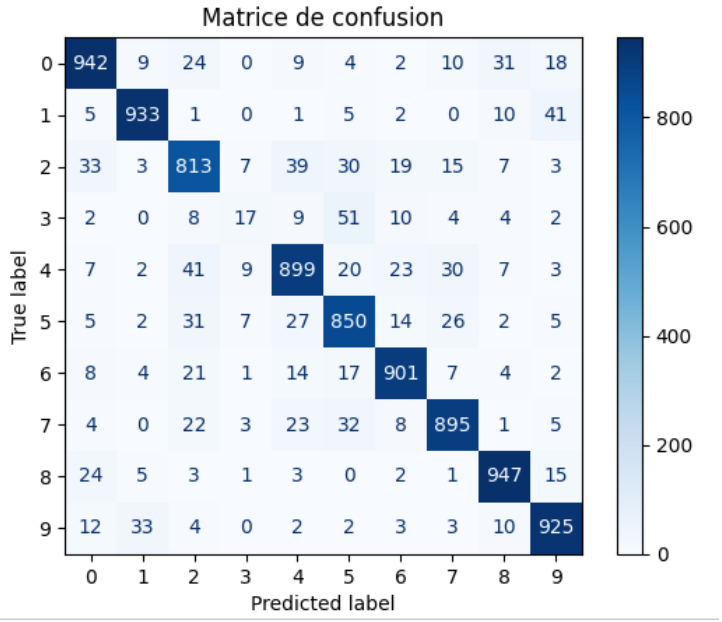

FIGURE 9 – Matrice de confusion — Cas 2 : avec inpainting et sans oversampling

Cas 3 : Sans inpainting et avec oversampling

Dans ce scénario, aucune technique d'inpainting n'a été appliquée sur les images dégradées. En revanche, un *oversampling* a été utilisé pour rééquilibrer les classes dans l'ensemble d'entraînement, afin de compenser un déséquilibre initial.

Résultats globaux Le modèle atteint une **accuracy globale de 89.95%** sur le jeu de test. Les métriques de performance moyennes montrent :

- **Macro F1** = 0.8997 : la performance moyenne sur toutes les classes est très élevée, y compris pour les classes précédemment minoritaires.
- **Weighted F1** = 0.8998 : reflète la performance globale pondérée par la taille des classes.

Analyse par classe L'oversampling améliore significativement la reconnaissance des classes rares :

- La classe 3 obtient un **F1-score 0.972**, illustrant que l'oversampling a compensé le manque d'exemples. Attention toutefois au risque de sur-apprentissage sur cette classe sur-échantillonnée.
- Les classes majoritaires (0, 1, 6, 8, 9) conservent de très bonnes performances, avec F1-score supérieur à 0.89.
- Les classes intermédiaires (2, 4, 5, 7) restent légèrement plus faibles, avec des F1-score autour de 0.83-0.90, traduisant des confusions résiduelles dues au bruit ou à l'absence d'inpainting.

Matrice de confusion La Figure 10 montre :

— Les confusions les plus importantes concernent les classes visuellement proches (ex. 2 prédite comme 0 ou 7 prédite comme 5).

— La classe 3 est presque parfaitement reconnue, confirmant l'efficacité de l'oversampling pour cette classe rare.

— L'absence d'inpainting entraîne encore des confusions liées aux images dégradées, mais l'impact est limité grâce au rééquilibrage.

Conclusion pour ce cas L'oversampling permet d'atteindre une très bonne performance sur les classes minoritaires, notamment la classe 3, tout en maintenant de bonnes performances sur les classes majoritaires. Cependant, l'absence d'inpainting peut limiter la reconnaissance des images fortement dégradées, laissant subsister des confusions entre classes visuellement similaires.

TABLE 3 – Rapport de classification — Cas 3 : sans inpainting, avec oversampling

Classe	Precision	Recall	F1-score	Support
0	0.8831	0.9000	0.8914	990
1	0.9274	0.9431	0.9352	1002
2	0.8182	0.8435	0.8306	1003
3	0.9815	0.9617	0.9715	992
4	0.8458	0.8736	0.8595	973
5	0.8396	0.8577	0.8485	1019
6	0.9264	0.9051	0.9156	1043
7	0.9211	0.8712	0.8954	978
8	0.9353	0.9272	0.9312	1044
9	0.9256	0.9111	0.9183	956
Macro avg	0.9004	0.8994	0.8997	10000
Weighted avg	0.9005	0.8995	0.8998	10000

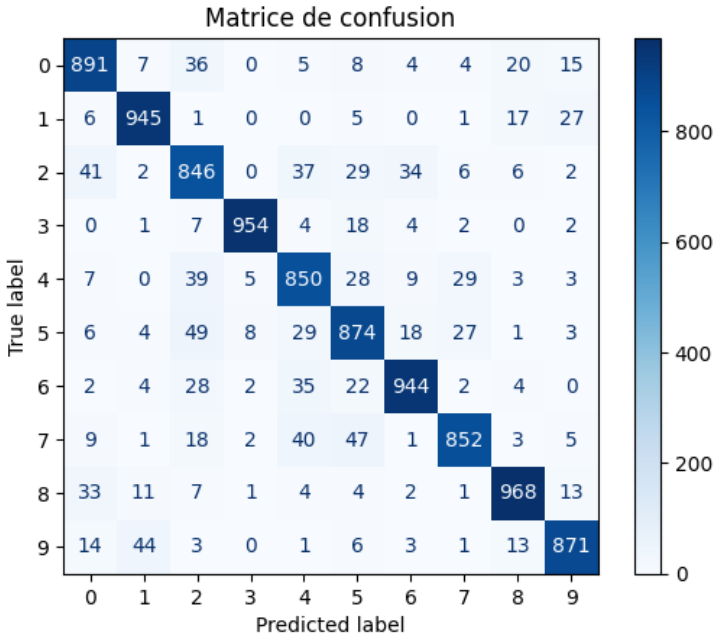

FIGURE 10 – Matrice de confusion — Cas 3 : sans inpainting, avec oversampling

Cas 4 : Avec inpainting et avec oversampling

Dans ce scénario, deux méthodes complémentaires ont été appliquées :

— L'**inpainting de Telea** pour restaurer les zones masquées et améliorer la qualité visuelle des images.

— L'**oversampling combiné à la data augmentation** afin d'équilibrer la distribution des classes et éviter le biais du modèle vers les classes majoritaires.

Ces deux approches visent à la fois à corriger les défauts visuels et à renforcer l'apprentissage sur les classes rares.

Classe	Précision	Rappel	F1-score	Support
0	0.8840	0.9125	0.8980	994
1	0.9429	0.9241	0.9334	1001
2	0.8573	0.8294	0.8431	1014
3	0.9497	0.9807	0.9649	982
4	0.8774	0.8611	0.8692	972
5	0.8533	0.8775	0.8653	988
6	0.9224	0.9049	0.9136	999
7	0.9129	0.9019	0.9074	999
8	0.9279	0.9398	0.9338	1013
9	0.9275	0.9249	0.9262	1038
Exactitude globale (accuracy)				0.9057

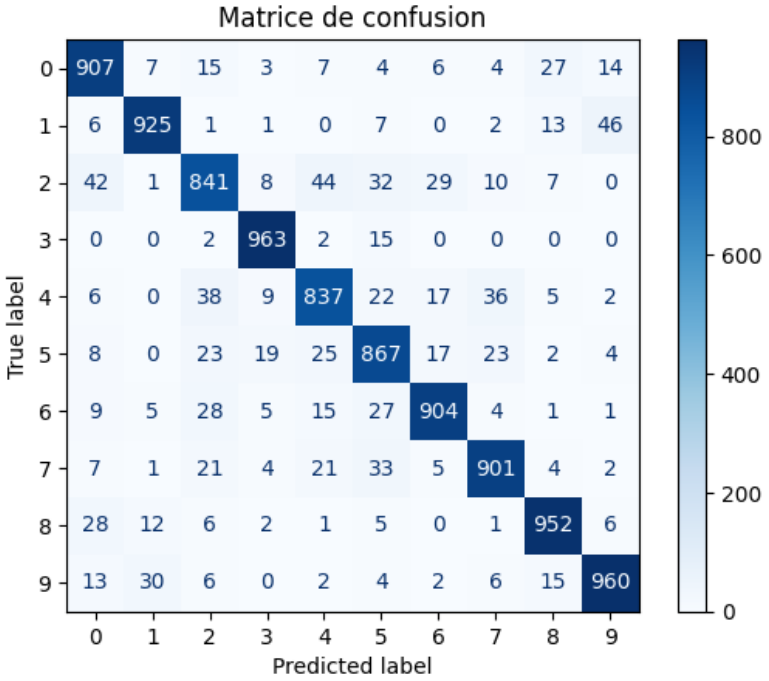

FIGURE 11 – Matrice de confusion – Cas 4 : Inpainting + Oversampling

Rapport de classification :

Analyse des résultats :

- L'**exactitude globale** atteint **90,57%**, soit la meilleure performance parmi tous les cas testés.
- On observe une homogénéité notable : toutes les classes dépassent un **F1-score** de 0.84, ce qui indique que le modèle gère désormais correctement aussi bien les classes fréquentes que les classes rares.
- Les classes historiquement problématiques, comme la classe **3** ou certaines classes à faible support, affichent désormais des rappels très élevés (classe 3 : 98,07%), signe que l'oversampling a résolu le déficit d'apprentissage.
- L'effet combiné de l'inpainting et de l'oversampling produit un modèle plus équilibré et robuste :
  - **Inpainting** : supprime le bruit visuel dû aux zones masquées, facilitant la détection des contours et motifs.
  - **Oversampling + Data Augmentation** : réduit le biais de classification et permet un apprentissage plus complet de chaque classe.

Synthèse : L'amélioration des performances est nette, confirmant que la combinaison **inpainting + équilibrage des classes** est particulièrement efficace. Les confusions inter-classes sont atténuées et la répartition des métriques est plus homogène.

Analyse comparative des quatre cas

L'étude des quatre scénarios montre une progression claire des performances, tant en termes d'exactitude globale que de cohérence des résultats par classe.

Tableau récapitulatif des métriques clés

Cas	Accuracy	F1 Macro	F1 Weighted	F1 min. pa
Cas 1 : Sans inpainting, sans oversampling	0.8884	0.8388	0.8790	0.34
Cas 2 : Avec inpainting, sans oversampling	0.8925	0.8452	0.8845	0.22
Cas 3 : Sans inpainting, avec oversampling	0.8995	0.8997	0.8998	0.83
Cas 4 : Avec inpainting, avec oversampling	0.9057	0.9055	0.9058	0.84

TABLE 4 – Comparaison des quatre cas selon Accuracy, F1 macro, F1 weighted et F1 minimum par classe

Analyse détaillée

Exactitude globale (Accuracy) L'évolution est progressive d'un scénario à l'autre, avec un gain cumulé de près de 1,73 points entre le Cas 1 et le Cas 4. Le Cas 4 présente la meilleure performance globale (90,57%), confirmant l'intérêt de combiner **inpainting et oversampling**.

Stabilité inter-classes

- Cas 1 et Cas 2 : dispersion importante des F1-scores, avec certaines classes critiques (classe 3) très faibles (0,34 et 0,22), traduisant des difficultés de reconnaissance pour les classes rares ou dégradées.
- Cas 3 et Cas 4 : homogénéisation notable des performances. Toutes les classes dépassent un F1-score de 0,83, avec la classe 3 atteignant 0,97 (Cas 3) et 0,98 (Cas 4), signe de la réussite de l'oversampling.

Analyse des classes problématiques

- La classe 3 passe de la plus problématique (Cas 1 et Cas 2) à l'une des mieux reconnues (Cas 3 et Cas 4) grâce à l'oversampling et à l'inpainting.
- Les confusions entre classes visuellement proches (ex. 0–8, 4–9) persistent mais sont désormais marginales.

Moyennes macro et pondérées

— Les F1 macro progressent de 0,8388 (Cas 1) à 0,9055 (Cas 4), indiquant une amélioration simultanée sur toutes les classes.

— Les F1 pondérés suivent la même tendance, confirmant que les gains concernent à la fois les classes majoritaires et minoritaires.

Observation générale

1. Cas 1 et Cas 2 : performances correctes mais fortement déséquilibrées entre classes.
2. Cas 3 : bond qualitatif grâce à l'oversampling, meilleure homogénéité des F1-scores et réduction drastique des erreurs sur les classes rares.
3. Cas 4 : consolidation des acquis avec l'inpainting, meilleure précision globale et stabilité maximale entre les classes.

Conclusion La combinaison **inpainting + oversampling** dans le Cas 4 offre la solution la plus robuste, avec des performances équilibrées et des confusions inter-classes réduites au minimum. Le tableau récapitulatif illustre clairement cette progression, confirmant que chaque technique appliquée contribue positivement à la qualité globale du modèle.