RAPPORT SUR :

Blood Group Detection Using Deep Learning

Encadré par :
PR. ANASS BELCAID

Réalisé par :
MOHAMED AMHAL
AHMED BAKKALI

Période :
01/11/2024 - 06/01/2025

Sommaire

1 Introduction

La détermination des groupes sanguins est une étape cruciale dans de nombreux domaines, notamment la médecine transfusionnelle, les interventions chirurgicales et les enquêtes médico-légales. Les méthodes traditionnelles de typage sanguin, basées sur des analyses sérologiques en laboratoire, bien que fiables, présentent des limitations en termes de temps, de ressources et d'accessibilité. Ces contraintes motivent la recherche de méthodes alternatives, rapides et non invasives, capables de répondre aux besoins dans des contextes variés, y compris les situations d'urgence. Dans ce projet, nous explorons deux approches innovantes pour la détection des groupes sanguins : l'utilisation d'images de sang et l'analyse des empreintes digitales. Ces deux méthodes, bien que distinctes dans leur principe, partagent un objectif commun : fournir une solution efficace et accessible pour déterminer les groupes sanguins sans recourir aux techniques traditionnelles invasives et coûteuses.

Approche 1 : Détection des groupes sanguins à l'aide d'images de sang

La première approche repose sur l'analyse d'images de sang pour identifier les marqueurs spécifiques des groupes sanguins. Grâce aux avancées récentes en traitement d'images et en intelligence artificielle, il est désormais possible d'extraire des informations précises à partir de simples échantillons visuels. Cette méthode offre plusieurs avantages, notamment sa rapidité et sa capacité à être automatisée. Elle pourrait ainsi être utilisée dans des contextes où le temps et les ressources sont limités, tels que les urgences médicales ou les zones reculées.

Approche 2 : Détection des groupes sanguins à l'aide des empreintes digitales

La deuxième approche explore la possibilité de prédire le groupe sanguin d'un individu à partir de ses empreintes digitales. Les empreintes digitales, reconnues pour leur unicité et leur permanence, sont déjà largement utilisées dans des domaines tels que la sécurité biométrique et les enquêtes criminelles. L'idée de relier leurs motifs complexes à des caractéristiques biologiques, comme le groupe sanguin, ouvre des perspectives fascinantes. Si cette corrélation est validée, elle pourrait permettre une méthode de détection des groupes sanguins entièrement non invasive et universellement applicable.

Objectifs du projet

L'objectif principal de ce projet est d'étudier la faisabilité de ces deux approches et de les exploiter conjointement pour développer une méthode innovante de détection des groupes sanguins. Pour cela, nous utiliserons un large ensemble de données comprenant à la fois des images de sang et des empreintes digitales, associées à des informations sur les groupes sanguins. En appliquant des techniques de traitement d'images, d'apprentissage automatique et d'analyse statistique, nous chercherons à identifier des corrélations et à développer des modèles prédictifs robustes. En combinant ces deux approches, nous espérons non seulement améliorer la précision et la fiabilité de la détection des groupes sanguins, mais aussi élargir les possibilités d'application dans des domaines tels que la médecine d'urgence, la sécurité biométrique et les enquêtes médico-légales. Ce projet

représente une étape importante vers des solutions plus accessibles et innovantes pour répondre aux besoins actuels et futurs en matière de typage sanguin.

2 Méthodologie et Datasets

Démarche réelle

Dans cette méthode, une lame de verre ou un support en porcelaine blanche est divisé en trois parties distinctes. Sur chaque section, une goutte de sang provenant du donneur ou du receveur est déposée et mélangée séparément avec des réactifs anti-A, anti-B et anti-D. L'observation visuelle de l'agglutination, c'est-à-dire la formation de grumeaux dans le sang, permet de déterminer les groupes sanguins ABO et RhD (Rhesus D). Ce test, simple et rapide, s'effectue en 6 à 10 minutes seulement. Il est également économique, car il nécessite une quantité minimale de réactifs pour le typage sanguin.
Cependant, cette technique présente certaines limites. Elle est moins sensible et est principalement utilisée comme outil préliminaire pour établir une correspondance initiale des groupes sanguins. Elle ne permet pas de détecter des antigènes faiblement réactifs ou rares, ce qui peut compliquer l'interprétation des résultats. De plus, une concentration insuffisante d'anticorps anti-A ou anti-B peut conduire à des résultats erronés, qu'ils soient faussement positifs ou négatifs. Bien que ce test sur lame soit pratique pour des typages sanguins réalisés sur le terrain, il ne garantit pas toujours la fiabilité nécessaire pour des transfusions sanguines totalement sûres.

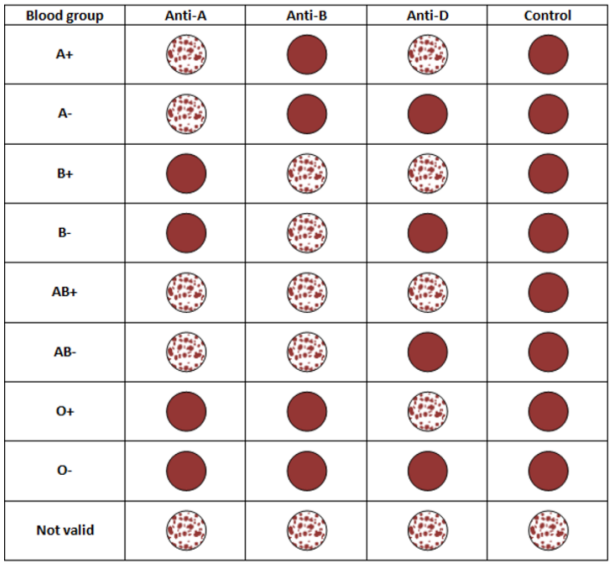


Figure 1: La reaction entre le sang et les ANTI A B D

Approche 1 : Détection des groupes sanguins à l'aide d'images de sang

Dans cette première approche, nous avons conçu un dataset basé sur des images de sang afin de détecter le groupe sanguin en analysant les réactions spécifiques des antigènes. Les groupes sanguins humains sont déterminés principalement par trois antigènes majeurs présents à la surface des globules rouges : **A**, **B**, et **D** (le facteur Rhésus). Ces antigènes permettent de classer les groupes sanguins selon les systèmes ABO et Rhésus, par exemple : A+, A-, B+, O-, AB+, etc.

- **Organisation et structure des données:**

Les images du dataset sont organisées dans un format compatible avec YOLO (*You Only Look Once*), un framework de détection d'objets. Chaque image est accompagnée d'un fichier texte contenant les annotations nécessaires pour identifier les zones spécifiques des antigènes **A**, **B** et **D**. Ces annotations indiquent les coordonnées des zones réactives ou non réactives dans l'image. Deux méthodes distinctes ont été envisagées pour exploiter ce dataset :

Méthode 1 : Organisation en 8 classes (classification multi-classes Dans cette méthode, les images sont regroupées en fonction des réactions observées aux antigènes et sont classées dans l'une des 8 classes possibles. Chaque classe représente une combinaison unique des réactions aux antigènes **A**, **B**, et **D** :

- **Classe 1** : O- (aucune réaction)
- **Classe 2** : O+ (réaction uniquement avec **D**)
- **Classe 3** : A- (réaction uniquement avec **A**)
- **Classe 4** : A+ (réaction avec **A** et **D**)
- **Classe 5** : B- (réaction uniquement avec **B**)
- **Classe 6** : B+ (réaction avec **B** et **D**)
- **Classe 7** : AB- (réaction avec **A** et **B**)
- **Classe 8** : AB+ (réaction avec **A**, **B** et **D**)

Méthode 2 : Classification avec YOLO Dans cette deuxième méthode, nous travaillons directement avec les images telles qu'elles sont, sans segmentation préalable des zones réactives. Le modèle apprend à prédire la classe globale (l'un des 8 groupes sanguins) en analysant l'image entière. Cette approche repose sur la capacité du modèle à identifier et interpréter les variations visuelles dans les trois zones de l'image correspondant aux antigènes **A**, **B** et **D**.

- **Exemple de la Version 1 du dataset :**

- **Objectif des méthodes**

L'objectif de ces deux méthodes est de comparer l'efficacité et la précision des approches suivantes :

- **Méthode 1 (classification globale)** : .Cette méthode simplifie le traitement en considérant l'image dans son ensemble sans segmentation préalable.
- **Méthode 2 (détection basée sur YOLO)** : Cette approche met l'accent sur l'identification précise des zones réactives dans les images avant de classifier les groupes sanguins

- **Statistiques sur le dataset(8 classes)**

On a :

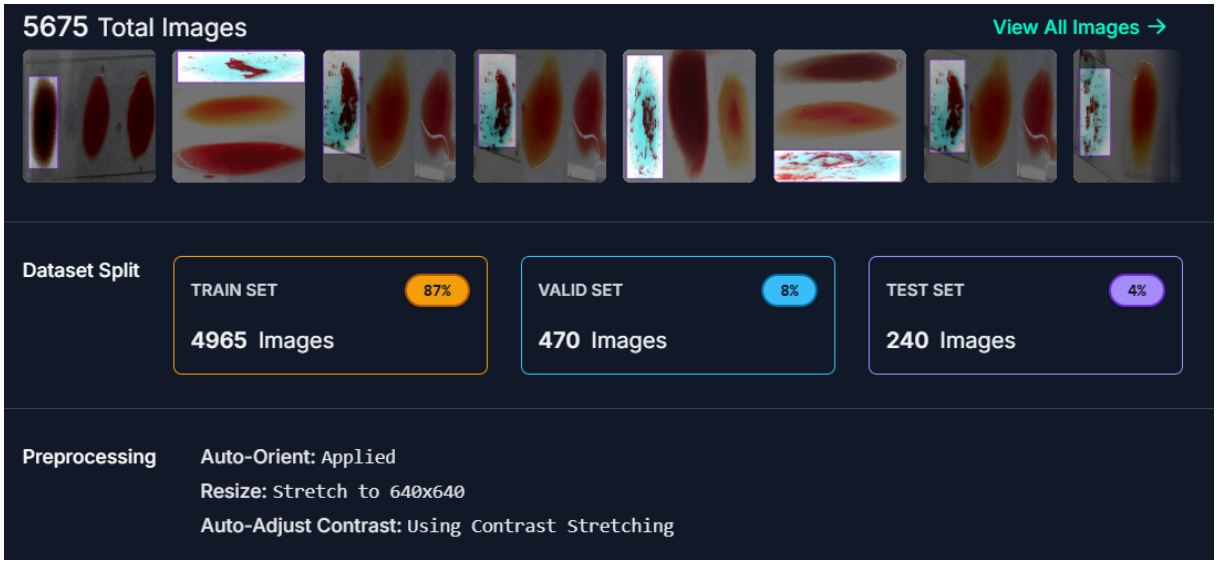


Figure 2: Exemple d'images YOLO version 1

- **Nombre total d'images** : 15000 images
- **Répartition des classes** :
  - Classe 1 (O-) : 13,31%
  - Classe 2 (O+) : 13,76%
  - Classe 3 (A-) : 11,65%
  - Classe 4 (A+) : 13,91%
  - Classe 5 (B-) : 13,60%
  - Classe 6 (B+) : 11,87%
  - Classe 7 (AB-) : 10,64%
  - Classe 8 (AB+) : 11,01%

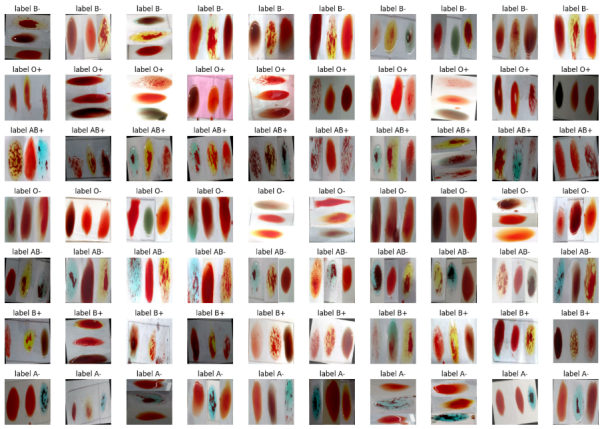


Figure 3: Exemple d'images de sang par classes

Approche 2 : Détection des groupes sanguins à l'aide des empreintes digitales

- **Données (Dataset):**

Le jeu de données utilisé pour l'approche basée sur les empreintes digitales est disponible sur Kaggle à l'adresse suivante : Finger Print Based Blood Group Dataset. Ce dataset est structuré de manière à faciliter l'analyse et la modélisation des relations potentielles entre les empreintes digitales et les groupes sanguins. Il comprend deux types de données principales : les images des empreintes digitales et les informations correspondantes sur les groupes sanguins. Chaque enregistrement du dataset associe une image d'empreinte digitale à un groupe sanguin spécifique (A, B, AB ou O) ainsi qu'au facteur Rh (positif ou négatif). Les images sont généralement au format standard (par exemple, JPEG ou PNG) et ont été collectées dans des conditions contrôlées pour garantir une qualité uniforme. Les données sont organisées de manière à permettre une analyse directe, avec des dossiers ou des colonnes dédiés pour les images et les étiquettes des groupes sanguins. Ce dataset constitue une base solide pour explorer les corrélations entre les motifs des empreintes digitales et les caractéristiques sanguines, tout en offrant la possibilité de développer des modèles d'apprentissage automatique pour la prédiction des groupes sanguins.

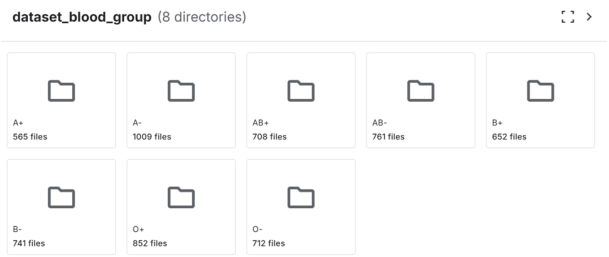


Figure 4: Sous dossiers pour chaque groupe de sang

- **Statistiques sur le dataset(8 classes)**

On a :

- **Nombre total d'images** : 6000 images
- **Répartition des classes** :
  - Classe 1 (O-) : 11.86%
  - Classe 2 (O+) : 14.20%
  - Classe 3 (A-) : 16.81%
  - Classe 4 (A+) : 9.42%
  - Classe 5 (B-) : 12.35%
  - Classe 6 (B+) : 10.87%
  - Classe 7 (AB-) : 12.68%
  - Classe 8 (AB+) : 11.80%

3 Augmentation des données(Data Augmentation)

Approche 1 : Détection des groupes sanguins à l'aide d'images de sang

Dans cette approche, nous nous sommes concentrés sur la préparation des données pour un système de détection des groupes sanguins à partir d'images de sang. L'utilisation

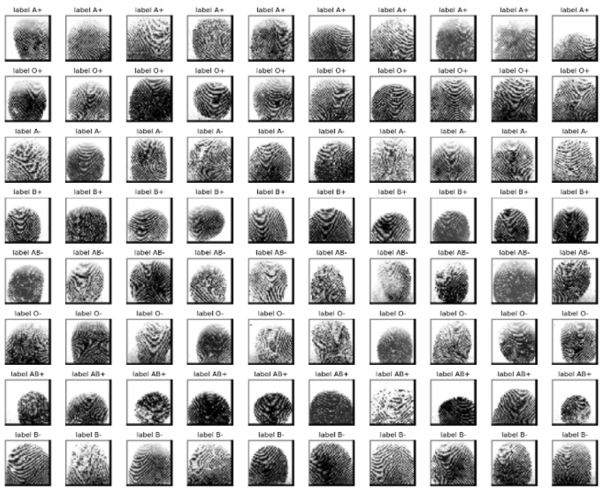


Figure 5: Exemple de code d'augmentation

de la data augmentation n'a pas été nécessaire, car le jeu de données d'origine incluait déjà des augmentations intégrées. Nous avons adopté une méthode consistant à fusionner quatre versions similaires des données, issues de la même source, afin de créer un ensemble de données enrichi. Cette étape nous a permis d'obtenir un dataset de plus de 14 000 images, offrant une taille suffisante pour garantir un entraînement efficace des modèles à venir. Cette stratégie a permis d'optimiser la diversité et la représentativité des données, constituant ainsi une base solide pour les phases d'entraînement et de validation du système.

Approche 2 : Détection des groupes sanguins à l'aide des empreintes digitales

Dans cette approche, nous avons travaillé sur la détection des groupes sanguins à partir des empreintes digitales. Le dataset initial contenait 6 000 images. Étant donné que cette approche est innovante, il était nécessaire d'élargir significativement le jeu de données pour garantir un entraînement performant et une meilleure généralisation des modèles. Nous avons ainsi mis en œuvre une stratégie de *data augmentation* soigneusement élaborée. Pour éviter de perdre les caractéristiques essentielles des empreintes digitales, nous avons fait attention aux types de transformations appliquées. Par exemple, des modifications comme la saturation ou la teinte auraient pu altérer les motifs distinctifs des empreintes et nuire à la qualité des données. Par conséquent, nous avons privilégié des méthodes de transformation géométrique adaptées, qui préservent la structure et les détails clés des images. Voici les techniques employées :

- **Miroir horizontal :** Nous avons appliqué une symétrie horizontale aux images, doublant ainsi la diversité du dataset tout en conservant les motifs caractéristiques des empreintes intactes. Cette transformation permet au modèle de mieux s'adapter à différentes orientations.

- **Rotation à 90 degrés :** Les empreintes digitales peuvent être capturées sous divers angles. En appliquant une rotation de 90 degrés, nous avons introduit des variations angulaires tout en maintenant les caractéristiques essentielles des images.

- **Rotation à 180 degrés :** La rotation complète à 180 degrés a permis de générer une autre variation angulaire, contribuant à l'augmentation de la diversité tout en respectant les motifs des empreintes.

- **Rotation à 270 degrés :** Une rotation supplémentaire de 270 degrés a permis de couvrir toutes les orientations possibles, renforçant la robustesse du modèle face aux variations directionnelles des empreintes.

```python
# define the function :
def augment_image(image, label):
    augmented_images = []
    augmented_labels = []

    # Miroir horizontal
    mirrored = tf.image.flip_left_right(image)
    augmented_images.append(mirrored)

    # Rotation 180 deg
    rotated_180 = tf.image.rot90(image, k = 2)
    augmented_images.append(rotated_180)

    # Rotation 90 deg
    rotated_90 = tf.image.rot90(image, k = 1)
    augmented_images.append(rotated_90)

    # Rotation 270
    rotated_270 = tf.image.rot90(image, k = 3)
    augmented_images.append(rotated_270)

    augmented_labels.extend([label] * 4)
    return augmented_images, augmented_labels
```

Figure 6: Exemple de visualisation d'une image augmenté

En appliquant ces quatre types de transformations à chaque image d'origine, nous avons généré quatre nouvelles images par empreinte. Ainsi, le dataset initial a été multiplié par cinq, passant de 6 000 images à plus de 30 000 images. Cette augmentation a permis de diversifier le jeu de données tout en conservant les caractéristiques essentielles des empreintes digitales. Cela constitue une base solide pour l'entraînement et la validation des modèles, garantissant leur robustesse et leur capacité à généraliser dans divers scénarios.

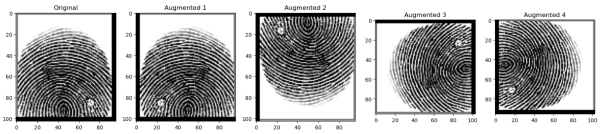


Figure 7: Exemple d'images d'empreintes digitales du dataset

4 Outils et Visualisation des Données

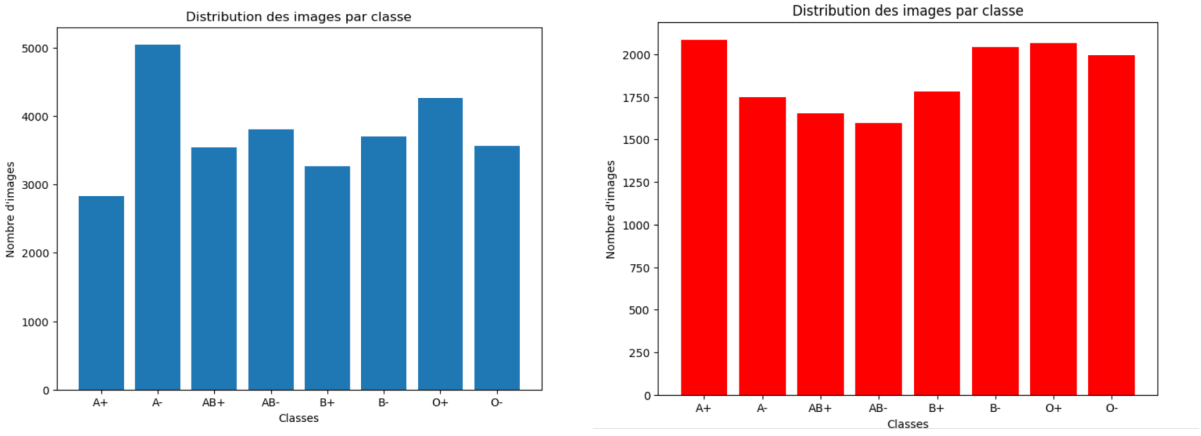
4.1 Outils Utilisés

Pour le développement des modèles de deep learning, nous avons utilisé la bibliothèque **TensorFlow**, qui est largement reconnue pour sa flexibilité et sa performance dans la création et l'entraînement de modèles d'apprentissage profond. TensorFlow offre une interface intuitive pour construire des réseaux de neurones et s'intègre facilement avec les accélérateurs matériels tels que les GPU, garantissant ainsi un entraînement rapide et efficace. Cette bibliothèque permet également de gérer des flux de données complexes, de prétraiter les données, et de surveiller l'entraînement à l'aide de modules comme *TensorBoard*. Son écosystème complet nous a permis de concevoir et d'entraîner des architectures

adaptées à nos besoins, tout en optimisant la performance grâce à des outils d'optimisation avancés. L'utilisation de TensorFlow a également facilité l'intégration d'opérations de data augmentation, renforçant ainsi la robustesse et la généralisation des modèles.

4.2 Visualisation des Données

Pour mieux comprendre et analyser les jeux de données utilisés dans les deux approches, nous avons produit un graphique en barres représentant le nombre d'images pour chaque classe. Ce visuel met en évidence la distribution des données dans le dataset pour les empreintes digitales et les images de sang, ce qui nous permet de vérifier l'équilibre des classes et d'ajuster si nécessaire les méthodes d'entraînement. Le graphique suivant illustre les tailles des données pour chaque classe dans les deux approches : Ces visualisations



(a) Distribution des tailles de données pour chaque classe dans l'approche (fingerprint).

(b) Distribution des tailles de données pour chaque classe dans l'approche (sang).

Figure 8: Comparaison des tailles de données pour les deux approches (empreintes digitales et images de sang).

jouent un rôle essentiel dans l'analyse exploratoire des données, en fournissant un aperçu rapide des distributions et en assurant que le dataset est bien préparé pour entraîner des modèles fiables et performants.

5 Les Modéles developpée

Approche 1 : Détection des groupes sanguins à l'aide d'images de sang

Dans cette approche, nous nous sommes concentrés sur la détection des groupes sanguins à partir d'images de sang. Le jeu de données est constitué de 8 classes correspondant aux différents groupes sanguins (A+, A-, B+, B-, AB+, AB-, O+, O-). Pour entraîner efficacement les modèles de deep learning, nous avons appliqué une méthodologie rigoureuse de préparation des données. Le dataset a été divisé en trois sous-ensembles distincts pour garantir un bon équilibre entre l'entraînement, la validation, et le test des modèles. Plus précisément, 60% des données ont été utilisées pour l'entraînement, ce qui permet au modèle d'apprendre les caractéristiques essentielles des classes. Ensuite, 20% des données ont été réservées à la validation, ce qui nous a permis de surveiller les performances du modèle et de prévenir tout problème de surapprentissage (*overfitting*). Enfin, les 20% restants ont été utilisés comme ensemble de test pour évaluer de manière indépendante

les performances finales du modèle. Cette stratégie de division garantit que le modèle est évalué sur des données qu'il n'a jamais vues auparavant, tout en disposant d'un ensemble suffisant pour apprendre les motifs visuels caractéristiques des différentes classes. Cette méthodologie établit une base solide pour entraîner et tester des modèles performants dans le cadre de cette approche. Pour les caractéristiques des images :

```
the number of batches in train set : tf.Tensor(281, shape=(), dtype=int64)
the number of batches in validation set : tf.Tensor(94, shape=(), dtype=int64)
the number of batches in test set : tf.Tensor(93, shape=(), dtype=int64)
```

Figure 9: le nombre de batches pour chaque sous-ensemble

```
the shape of the image : (200, 200, 3)
the label of the image : (8,)
```

Figure 10: size des images et les lables

Normalisation des Données La normalisation des données est une étape fondamentale dans le processus de préparation des données pour l'entraînement des modèles de deep learning. Elle consiste à transformer les valeurs des données d'entrée pour qu'elles soient centrées autour de zéro et qu'elles aient une échelle cohérente. Cela permet d'améliorer la stabilité et l'efficacité de l'entraînement en évitant les disparités entre les différentes dimensions des données. Pour cette tâche, nous avons utilisé la méthode de normalisation fournie par la classe `Normalization` de `TensorFlow`. Cette méthode repose sur le calcul des statistiques du jeu d'entraînement, à savoir la moyenne (μ) et l'écart-type (σ), puis applique la transformation suivante à chaque pixel de l'image :

$$x_{normalisé} = \frac{x - \mu}{\sigma}$$

où x représente la valeur d'un pixel.

Étapes de Normalisation :

1. **Initialisation de l'objet** `Normalization` : Nous avons créé un objet de normalisation en spécifiant `axis=-1`, ce qui indique que la normalisation est appliquée sur les canaux des images.

2. **Adaptation au jeu d'entraînement** : L'objet `Normalization` a été adapté au jeu d'entraînement en utilisant uniquement les données d'images, excluant les étiquettes, pour calculer les statistiques globales (μ et σ). Cela a été réalisé avec le code suivante :

```
normalizer.adapt(ds_train.map(lambda x, _: x))
```

3. **Application de la normalisation** : Une fonction `normalize_with` a été définie pour appliquer la normalisation à chaque image, tout en conservant les étiquettes. Cette fonction a ensuite été mappée sur les ensembles d'entraînement, de validation et de test:

```
def normalize_with(image, label):
    image = normalizer(image)
    return image, label
ds_train = ds_train.map(normalize_with)
ds_val = ds_val.map(normalize_with)
ds_test = ds_test.map(normalize_with)
```

Bénéfices de la Normalisation :

- **Amélioration de la stabilité de l'entraînement** : Les valeurs normalisées autour de zéro permettent une meilleure convergence des modèles en rendant les gradients plus stables.
- **Cohérence des données** : En appliquant la même transformation aux ensembles d'entraînement, de validation et de test, nous garantissons une évaluation fiable et représentative des performances des modèles.
- **Réduction des biais** : La normalisation réduit les biais dus à des écarts d'échelle ou de distribution dans les données d'origine.

Cette approche garantit une préparation optimale des données, permettant aux modèles de se concentrer sur l'apprentissage des caractéristiques pertinentes tout en minimisant les erreurs liées aux disparités des données d'entrée.

Modéle 1 : VGG BLOCKS

Le modèle VGG, proposé par Simonyan et Zisserman en 2014, est un réseau de neurones convolutifs (CNN) qui se distingue par sa simplicité et sa profondeur. Le modèle est basé sur l'utilisation de couches convolutives avec des noyaux de taille 3x3 et un pas (stride) de 1, ce qui permet de capter des détails fins tout en maintenant une faible complexité dans les paramètres. VGG est devenu célèbre pour sa structure homogène et sa capacité à obtenir des performances de pointe dans des tâches telles que la classification d'images et la reconnaissance d'objets. L'un des principes clés du modèle VGG est la profondeur de son réseau. Le modèle VGG16, par exemple, se compose de 16 couches, tandis que le modèle VGG19 en compte 19. Ce réseau profond permet de capturer des caractéristiques complexes à différents niveaux d'abstraction, avec des couches plus profondes apprenant des représentations plus abstraites des images. L'utilisation de blocs convolutionnels successifs et de couches de max-pooling aide à réduire progressivement la dimensionnalité tout en conservant les informations essentielles. Malgré ses avantages, le modèle VGG présente certaines limitations, notamment en termes de consommation de ressources. Sa profondeur importante et son grand nombre de paramètres le rendent coûteux en termes de mémoire et de temps de calcul. Cependant, ces limitations ont conduit à l'émergence de variantes et de modèles plus efficaces, tels que ResNet et Inception, qui utilisent des architectures plus complexes tout en réduisant le nombre de paramètres et en améliorant les performances. Le modèle VGG reste toutefois une référence importante dans le domaine des réseaux de neurones convolutifs.

Architecture de VGG 16 (IMPLEMENTATION): Ce modèle est une architecture de réseau de neurones convolutifs (CNN) utilisé pour la classification d'images. Il est composé de plusieurs blocs convolutifs suivis de couches entièrement connectées. Nous allons expliquer étape par étape la structure du modèle.

1. Couche d'Entrée Le modèle commence par une **couche d'entrée**, où les données d'entrée sont de taille `(200, 200, 3)`. Cela signifie que les images utilisées comme entrée ont une hauteur et une largeur de 200 pixels et 3 canaux de couleur (RGB).

$$X_{input} = Input(input\_shape)$$

2. Blocs Convolutifs Les blocs convolutifs sont la partie principale du modèle. Chaque bloc convolutif comprend plusieurs couches de convolution suivies de la normalisation par lot (Batch Normalization), de la fonction d'activation ReLU, de la mise en commun (MaxPooling) et de la régularisation par abandon (Dropout).

3.Premier Bloc Convolutif Dans le premier bloc, nous appliquons deux couches convolutives, chacune avec 64 filtres et une taille de noyau de 3×3, avec un remplissage (padding) de type "same" pour préserver les dimensions spatiales de l'image. La normalisation par lot est effectuée après chaque convolution pour stabiliser l'apprentissage. Après chaque couche convolutive, nous appliquons un `MaxPooling` avec un pool de 2×2 et un `Dropout` de 0.25 pour prévenir l'overfitting.

4.Deuxième Bloc Convolutif Le deuxième bloc suit une structure similaire avec des filtres de 128 et des couches convolutives. Nous appliquons également la normalisation par lot et un dropout de 0.25.

5.Troisième à Cinquième Blocs Convolutifs Les blocs 3 à 5 suivent la même logique avec des filtres croissants de 256 et 512. Les couches de convolution sont suivies de la normalisation par lot, de la mise en commun maximale et de la régularisation par abandon. Un taux de `Dropout` de 0.3 est appliqué dans ces blocs pour réduire l'overfitting, ce qui est essentiel lorsque le nombre de paramètres dans le modèle est élevé.

6. Couches Entièrement Connectées Après les blocs convolutifs, nous avons des couches entièrement connectées (fully connected).

Première Couche Entièrement Connectée La première couche est une couche dense avec 4096 unités et une fonction d'activation ReLU. La normalisation par lot et un `Dropout` de 0.5 sont appliqués pour éviter l'overfitting.

Deuxième Couche Entièrement Connectée La deuxième couche est également dense avec 4096 unités, suivie de la normalisation par lot et du `Dropout` pour empêcher l'overfitting.

7.Couche de Sortie Enfin, nous avons la **couche de sortie**, qui est une couche dense avec un nombre d'unités égal au nombre de classes (dans ce cas, 8 classes). La fonction d'activation utilisée est `softmax`, car il s'agit d'un problème de classification multi-classes.

$$output = Dense(units = classes, activation =' softmax')(x)$$

8. Création du Modèle Le modèle complet est créé en spécifiant les couches d'entrée et de sortie avec la fonction `Model` de Keras. Ce modèle sera ensuite utilisé pour l'entraînement sur des images de données.

$$model = Model(inputs = X\_input, outputs = output)$$

Expérience 1 : algorithme d'optimisation : 'SGD' , ALpha = 0.00015, epoch = 20 .
Results (train + validation :

```
Epoch 16/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 71s 252ms/step - accuracy: 0.4900 - loss: 1.6419 - val_accuracy: 0.3785 - val_loss:
2.6005
Epoch 17/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 71s 252ms/step - accuracy: 0.5234 - loss: 1.5178 - val_accuracy: 0.4002 - val_loss:
2.4286
Epoch 18/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 71s 252ms/step - accuracy: 0.5368 - loss: 1.4631 - val_accuracy: 0.4345 - val_loss:
2.2463
Epoch 19/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 71s 252ms/step - accuracy: 0.5533 - loss: 1.3870 - val_accuracy: 0.2983 - val_loss:
2.7590
Epoch 20/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 71s 252ms/step - accuracy: 0.5925 - loss: 1.3040 - val_accuracy: 0.4444 - val_loss:
2.2158
```

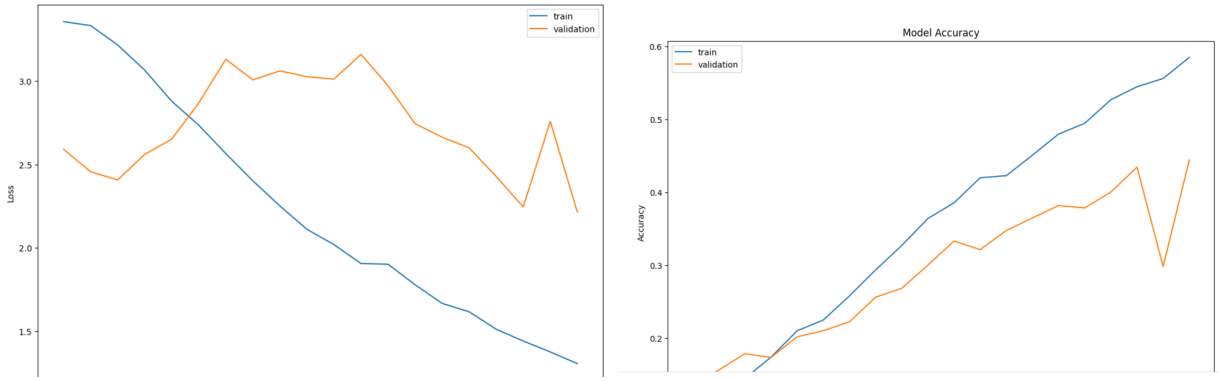
Figure 11: Experience 1 VGG (0.00015)

Test :

```
93/93 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - accuracy: 0.4363 - loss: 2.2293
[2.2100532054901123, 0.44455644488334656]
```

Figure 12: Experience 1 VGG(testing) (0.00015)

L'algorithme a montré des résultats variés lors de son évaluation sur différentes étapes du processus d'apprentissage. Sur l'ensemble d'entraînement, il a atteint un score de 0.59, indiquant une performance raisonnable dans l'apprentissage des patterns du jeu de données. Cependant, la performance sur les données de validation et de test est relativement plus faible, avec des scores de 0.444 et 0.44 respectivement. Cela suggère que l'algorithme pourrait souffrir de sur-apprentissage (overfitting), où il s'adapte bien aux données d'entraînement mais peine à généraliser sur de nouvelles données.



(a) THE LOSS(train + val

(b) The accuracy (train + val

Figure 13: the metrics of the model experience 1

Expérience 2 : algorithme d'optimisation : 'SGD' , ALpha = 0.001, epoch = 40 .
Results (train + validation :
Test :
Le modèle présente des résultats très prometteurs à chaque étape du processus d'évaluation.

```
Epoch 16/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 264ms/step - accuracy: 0.9072 - loss: 0.3681 - val_accuracy: 0.9074 - val_loss:
1.9092
Epoch 17/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 264ms/step - accuracy: 0.9185 - loss: 0.3745 - val_accuracy: 0.9347 - val_loss:
0.3007
Epoch 18/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9217 - loss: 0.3496 - val_accuracy: 0.9183 - val_loss:
0.4088
Epoch 19/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9140 - loss: 0.4116 - val_accuracy: 0.8705 - val_loss:
1.5486
Epoch 20/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9042 - loss: 0.4347 - val_accuracy: 0.9403 - val_loss:
0.2281
```

Figure 14: Experience 2 VGG (0.001)

```
93/93 ━━━━━━━━━━━━━━━━━━━━ 6s 63ms/step - accuracy: 0.9444 - loss: 0.2138
[0.22159400582313538, 0.9401881694793701]
```

Figure 15: Experience 2 VGG(testing) (0.001)

Avec une précision de 0.90 sur l'ensemble d'entraînement, il montre une excellente capacité à apprendre les patterns des données. De plus, les performances sur les ensembles de validation et de test sont remarquablement bonnes, atteignant respectivement 0.94 pour les deux. Ces scores indiquent une forte capacité de généralisation, suggérant que le modèle n'est pas sur-appris et qu'il performe de manière cohérente sur de nouvelles données. Les valeurs de précision, rappel et F1-score, toutes à 0.94, confirment une excellente performance en termes de classification équilibrée. Cela montre que le modèle est capable de prédire les classes de manière fiable tout en équilibrant les faux positifs et les faux négatifs. En résumé, ce modèle est robuste et bien adapté à la tâche pour laquelle il a été conçu.
Matrice de confusion :

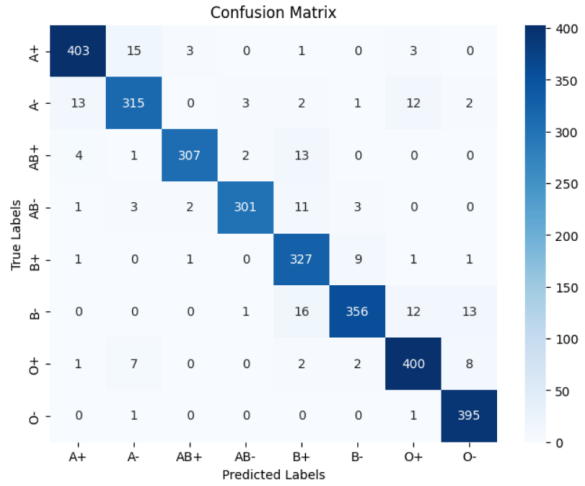
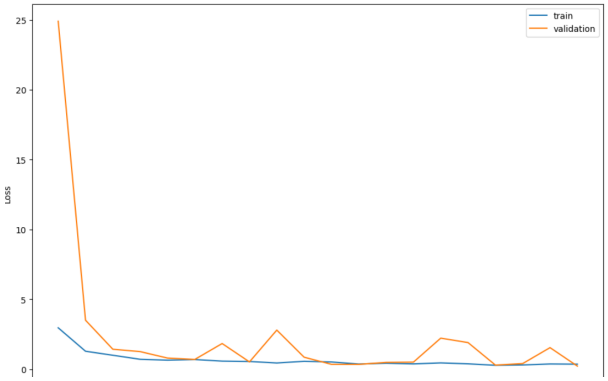
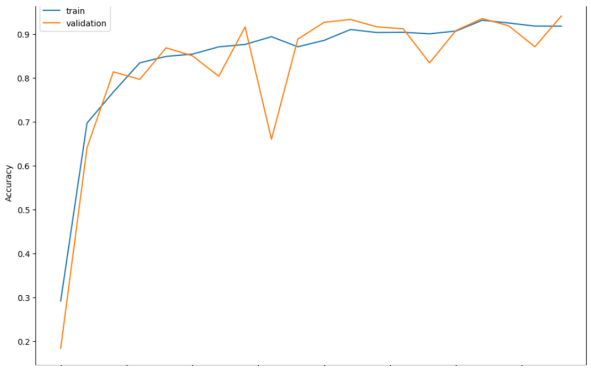


Figure 16: la matrice de confusion



(a) THE LOSS(train + val experience 2



(b) The accuracy (train + val experience 2

Figure 17: the metrics of the model experience 2

```
Precision: 0.9431
Recall: 0.9422
F1 Score: 0.9422

Precision per class:
A+: 0.9531
A-: 0.9229
AB+: 0.9773
AB-: 0.9801
B+: 0.8862
B-: 0.9554
O+: 0.9313
O-: 0.9426
```

(a) les metrices du modéle

```
Recall per class:
A+: 0.9464
A-: 0.9073
AB+: 0.9377
AB-: 0.9427
B+: 0.9561
B-: 0.8988
O+: 0.9539
O-: 0.9924

F1 Score per class:
A+: 0.9497
A-: 0.9150
AB+: 0.9571
AB-: 0.9610
B+: 0.9198
B-: 0.9262
O+: 0.9424
O-: 0.9669
```

(b) Recal et la précision

Figure 18: the metrics of the model VGG final

Expérience 3 : algorithme d'optimisation : 'ADAM' , ALpha = 0.00015, epoch = 20 .
Results (train + validation :

```
Epoch 16/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9554 - loss: 0.1523 - val_accuracy: 0.9539 - val_loss:
0.2035
Epoch 17/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9680 - loss: 0.1151 - val_accuracy: 0.8189 - val_loss:
0.9538
Epoch 18/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9602 - loss: 0.1414 - val_accuracy: 0.9413 - val_loss:
0.2916
Epoch 19/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9622 - loss: 0.1302 - val_accuracy: 0.9607 - val_loss:
0.1967
Epoch 20/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 74s 263ms/step - accuracy: 0.9655 - loss: 0.1108 - val_accuracy: 0.9597 - val_loss:
0.2028
```

Figure 19: Experience 3 VGG train + val(0.00015)

Test :

```
# evaluate the model:
model4.evaluate(ds_test)
```

```
93/93 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - accuracy: 0.9599 - loss: 0.2076
[0.21682603657245636, 0.9596773982048035]
```

Figure 20: Experience 3 VGG(testing) (test)(0.00015)

Le modèle affiche une performance impressionnante et une forte cohérence entre les différents ensembles de données. Avec un score de 0.96 sur l'ensemble d'entraînement, il montre une excellente capacité à apprendre les patterns des données d'origine. Les scores sur les ensembles de validation (0.9597) et de test (0.9596) sont presque identiques à celui de l'entraînement, indiquant une excellente généralisation. Cela suggère que le modèle n'est ni sur-ajusté (overfitting) ni sous-ajusté (underfitting) et qu'il est capable de maintenir une performance stable lorsqu'il est confronté à de nouvelles données. Ces résultats montrent que le modèle est robuste, fiable et bien adapté à la tâche cible. Matrice de confusion :

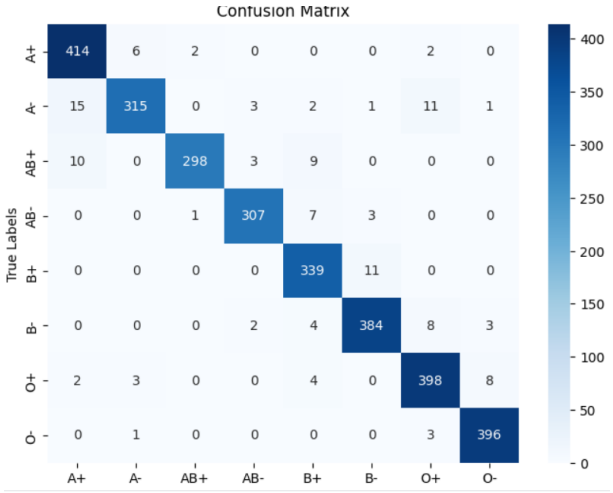
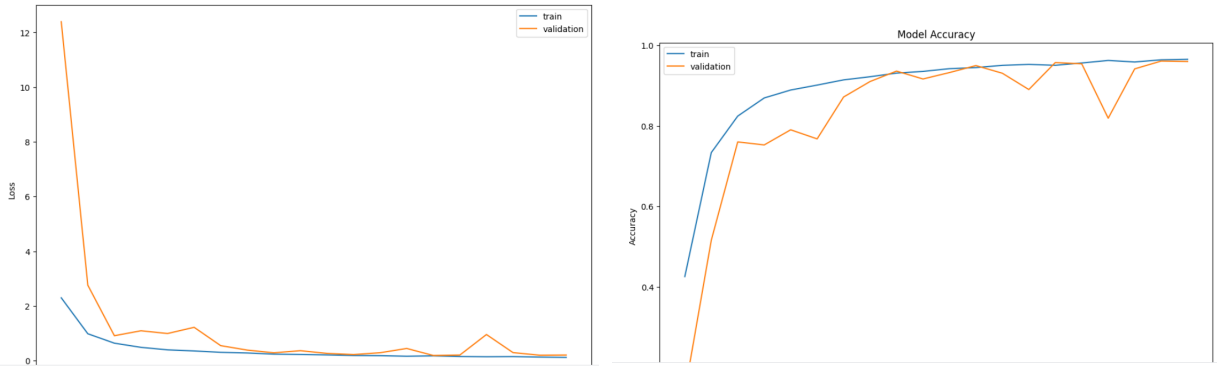


Figure 21: la matrice de confusion experience 3



(a) THE LOSS(train + val experience 3 (b) The accuracy (train + val experience 3

Figure 22: the metrics of the model experience 2

```
.
Precision: 0.9594
Recall: 0.9587
F1 Score: 0.9586

Precision per class:
A+: 0.9391
A-: 0.9788
AB+: 0.9901
AB-: 0.9750
B+: 0.9201
B-: 0.9605
O+: 0.9443
O-: 0.9750
```

(a) les metrices du modéle

```
Recall per class:
A+: 0.9788
A-: 0.9099
AB+: 0.9286
AB-: 0.9600
B+: 0.9681
B-: 0.9581
O+: 0.9653
O-: 0.9898

F1 Score per class:
A+: 0.9585
A-: 0.9431
AB+: 0.9583
AB-: 0.9674
B+: 0.9435
B-: 0.9593
O+: 0.9547
O-: 0.9824
```

(b) Recal et la précision

Figure 23: the metrics of the model VGG final exp 3

Modéle 2 : RESNET Model

Le modèle ResNet (Residual Network) est une architecture de réseau de neurones profonds introduite par He et al. dans leur article de 2015 intitulé *Deep Residual Learning for Image Recognition*. ResNet a révolutionné le domaine de l'apprentissage profond en permettant de construire des réseaux très profonds tout en résolvant les problèmes de dégradation du gradient.

Problème de dégradation dans les réseaux profonds Dans les réseaux de neurones traditionnels, l'ajout de couches supplémentaires peut entraîner une dégradation des performances. Ce problème est dû à une difficulté d'optimisation et à la disparition ou explosion du gradient lors de la rétropropagation. ResNet surmonte cette limitation en introduisant des **connexions résiduelles**.

Connexions résiduelles Une connexion résiduelle permet de contourner une ou plusieurs couches du réseau en ajoutant directement l'entrée à la sortie via une addition. Mathématiquement, cela peut être exprimé comme :

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

où x est l'entrée, $\mathcal{F}(x, W)$ représente la fonction apprise par les couches intermédiaires (avec W les poids), et y est la sortie. Cette architecture permet au modèle d'apprendre des fonctions résiduelles ($\mathcal{F}$) au lieu d'apprendre directement le mappage complet, ce qui facilite l'apprentissage.

Avantages de ResNet L'un des principaux avantages de ResNet est sa capacité à construire des réseaux très profonds, comme ResNet-50, ResNet-101 ou ResNet-152, sans perte significative de performance. ResNet a également prouvé son efficacité dans diverses tâches de vision par ordinateur, notamment la classification d'images et la détection d'objets. Il a remporté la compétition ImageNet en 2015 avec un taux d'erreur record.

Applications Grâce à sa flexibilité et à ses performances élevées, ResNet est largement utilisé dans des applications telles que la reconnaissance d'images, la segmentation

sémantique, et même dans des domaines au-delà de la vision par ordinateur, comme le traitement du langage naturel et les séries temporelles.

Architecture de RESNET (IMPLEMENTATION): Dans cette section, nous décrivons l'implémentation d'une architecture ResNet50 modifiée avec l'intégration de couches *Dropout* pour réduire l'*overfitting*.

Blocs Résiduels L'architecture ResNet repose sur deux types de blocs fondamentaux :

- **Bloc d'identité (`identity_block`)** : Ce bloc est utilisé lorsque la dimension d'entrée correspond à la dimension de sortie. Il comprend trois sous-couches principales :

  1. Une convolution 1×1 pour réduire les dimensions.
  2. Une convolution $f \times f$ (où f est la taille du filtre) pour extraire les caractéristiques.
  3. Une convolution 1×1 pour restaurer les dimensions.

  L'entrée du bloc est directement ajoutée à la sortie grâce à une connexion résiduelle.

- **Bloc convolutionnel (`convolutional_block`)** : Ce bloc est utilisé lorsque la dimension d'entrée est différente de celle de la sortie. Il applique une transformation au chemin de connexion résiduelle pour ajuster les dimensions, tout en suivant une structure similaire au `identity_block`.

Structure Globale du Modèle ResNet50_v2 Le modèle commence par une étape de *Zero-Padding* suivie d'une convolution initiale et d'une couche de *MaxPooling*. Ensuite, il est structuré en 5 étapes (*stages*), où chaque étape comporte un bloc convolutionnel suivi de plusieurs blocs d'identité. Les tailles de filtres et les dimensions des convolutions sont adaptées à chaque étape pour extraire des caractéristiques de plus en plus complexes.

Utilisation de Dropout Pour réduire l'*overfitting*, des couches *Dropout* sont ajoutées à des emplacements stratégiques :

- Après la première couche de *MaxPooling* ($Dropout(0.2)$).
- Après les étapes 2, 3 et 4 ($Dropout(0.3)$ et $Dropout(0.4)$).
- Avant la couche entièrement connectée ($Dropout(0.5)$).

Couche de Sortie La couche finale est une couche dense (*Dense Layer*) avec une activation `softmax`, qui permet de produire une distribution de probabilité sur les N classes cibles.

Résumé de l'Architecture En résumé, cette version modifiée de ResNet50, enrichie avec des couches *Dropout*, offre une meilleure robustesse contre l'*overfitting*, tout en conservant les principes fondamentaux des connexions résiduelles, qui facilitent l'apprentissage dans les réseaux très profonds.

Expérience : algorithme d'optimisation : 'ADAM' , ALpha = 0.00015, epoch = 40 .
Results (train + validation :

```
Epoch 16/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 48s 169ms/step - accuracy: 0.9613 - loss: 0.1177 - val_accuracy: 0.3422 - val_loss: 1
5.2738
Epoch 17/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 48s 171ms/step - accuracy: 0.9291 - loss: 0.2251 - val_accuracy: 0.7836 - val_loss:
1.0817
Epoch 18/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 48s 169ms/step - accuracy: 0.9643 - loss: 0.1174 - val_accuracy: 0.9188 - val_loss:
0.3591
Epoch 19/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 48s 170ms/step - accuracy: 0.9712 - loss: 0.1013 - val_accuracy: 0.9260 - val_loss:
0.2680
Epoch 20/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 48s 169ms/step - accuracy: 0.9635 - loss: 0.1191 - val_accuracy: 0.9066 - val_loss:
0.3842
```

Figure 24: Experience RESNET train + val(0.00015)

Test :

```
]: # evaluate the model:
   model9.evaluate(ds_test)

   93/93 ━━━━━━━━━━━━━━━━━━━━ 4s 47ms/step - accuracy: 0.9037 - loss: 0.3941
]: [0.395233690738678, 0.9069220423698425]
```

Figure 25: Experience RESNET (testing) (test)(0.00015)

Le modèle présente une performance élevée sur l'ensemble d'entraînement avec un score de 0.9635, ce qui indique qu'il a bien appris les caractéristiques des données d'entraînement. Cependant, les scores sur les ensembles de validation (0.9066) et de test (0.9069) sont significativement inférieurs à celui de l'entraînement, bien qu'ils soient très proches l'un de l'autre. Cette différence suggère que le modèle pourrait être légèrement sur-ajusté (*overfitting*), c'est-à-dire qu'il s'adapte trop aux données d'entraînement au détriment de sa capacité de généralisation sur de nouvelles données. Matrice de confusion :

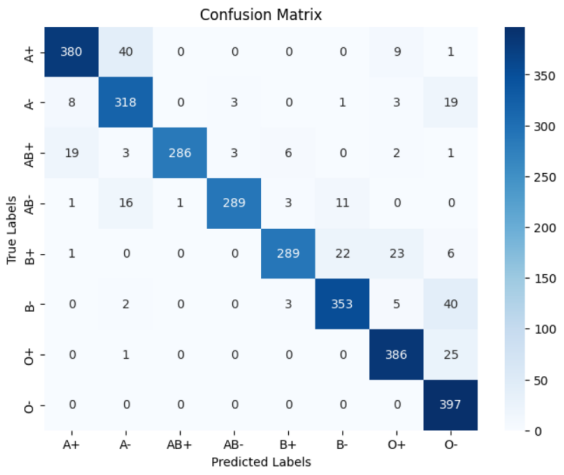


Figure 26: la matrice de confusion Resnet

```
Precision: 0.8228
Recall: 0.8135
F1 Score: 0.8130

Precision per class:
A+: 0.7348
A-: 0.7549
AB+: 0.7564
AB-: 0.7850
B+: 0.8759
B-: 0.8874
O+: 0.8530
O-: 0.9184
```

(a) les metrices du modéle RESNET

```
Recall per class:
A+: 0.8800
A-: 0.7657
AB+: 0.9105
AB-: 0.9469
B+: 0.7243
B-: 0.7975
O+: 0.7184
O-: 0.7895

F1 Score per class:
A+: 0.8009
A-: 0.7603
AB+: 0.8263
AB-: 0.8584
B+: 0.7929
B-: 0.8401
O+: 0.7800
O-: 0.8491
```

(b) Recal et la précision (RESNET)

Figure 27: the metrics of the model Resnet final

Modéle 3 : ALEXNET Model

AlexNet est une architecture de réseau de neurones convolutifs (CNN) introduite par Krizhevsky et al. en 2012 dans le cadre de la compétition ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Ce modèle a marqué une avancée majeure dans le domaine de la vision par ordinateur en démontrant l'efficacité des réseaux profonds pour la classification d'images à grande échelle.

Structure de l'Architecture AlexNet est composé de 8 couches principales :

- **5 couches convolutionnelles** : Ces couches extraient progressivement des caractéristiques complexes des images d'entrée. Des filtres de tailles variables sont utilisés, avec des fonctions d'activation `ReLU` pour accélérer la convergence.
- **3 couches entièrement connectées (Fully Connected)** : Ces couches apprennent les relations non linéaires et mappent les caractéristiques extraites aux classes de sortie.
- **Couche de sortie Softmax** : Elle produit une distribution de probabilités sur les classes cibles.

Caractéristiques Innovantes AlexNet se distingue par plusieurs innovations clés :

- **Utilisation de ReLU** : L'activation `ReLU` (Rectified Linear Unit) permet une convergence plus rapide par rapport aux fonctions d'activation traditionnelles comme `tanh`.
- **Dropout** : AlexNet intègre la régularisation par *Dropout* dans ses couches entièrement connectées pour réduire l'overfitting.
- **Normalisation locale (LRN)** : Une normalisation locale est utilisée pour améliorer la généralisation en mettant en évidence les caractéristiques pertinentes.
- **Entraînement sur GPU** : L'entraînement a été effectué sur des GPU pour gérer efficacement la grande profondeur du réseau et le volume de données.

Performance AlexNet a remporté la compétition ILSVRC en 2012 avec une erreur top-5 de seulement 15.3%, surpassant largement les approches précédentes. Ce succès a marqué le début de l'ère moderne des réseaux de neurones profonds.

Applications AlexNet a été utilisé dans diverses applications de vision par ordinateur, notamment la classification d'images, la détection d'objets, et la reconnaissance de motifs dans des domaines tels que la médecine, l'industrie et les véhicules autonomes.

Implémentation de l'Architecture AlexNet Dans cette section, nous décrivons l'implémentation d'une version modifiée de l'architecture AlexNet, adaptée pour des images d'entrée de taille $200 \times 200 \times 3$ et pour la classification en 8 classes.

Structure de l'Architecture (Implementation) L'architecture AlexNet proposée suit une structure hiérarchique composée de couches convolutionnelles, de couches entièrement connectées (*Fully Connected*), et de couches de normalisation et régularisation pour améliorer les performances. Voici un aperçu détaillé des composants principaux :

Couches Convolutionnelles et Max Pooling

- **1ère couche convolutionnelle** :
  - 96 filtres de taille 11×11, avec un pas de 4×4.
  - Fonction d'activation : `ReLU`.
  - Suivie par une normalisation par lot (*Batch Normalization*) et une opération de *Max Pooling* avec une taille de fenêtre 3×3 et un pas de 2×2.
- **2ème couche convolutionnelle** :
  - 256 filtres de taille 5×5, avec un remplissage (*padding*) de type `same`.
  - Fonction d'activation : `ReLU`.
  - Suivie par une normalisation par lot et une opération de *Max Pooling* (3×3, strides 2×2).
- **3ème, 4ème et 5ème couches convolutionnelles** :
  - 3ème couche : 384 filtres de taille 3×3 avec un *padding* `same`.
  - 4ème couche : 384 filtres de taille 3×3 avec un *padding* `same`.
  - 5ème couche : 256 filtres de taille 3×3 avec un *padding* `same`, suivie d'une opération de *Max Pooling* (3×3, strides 2×2).
  - Fonction d'activation : `ReLU`.
  - Toutes les couches sont suivies d'une normalisation par lot (*Batch Normalization*).

Couches Entièrement Connectées

- **1ère couche entièrement connectée** :
  - 4096 neurones.
  - Fonction d'activation : `ReLU`.

- Suivie par une normalisation par lot et une régularisation par *Dropout* avec un taux de 0.5.

- **2ème couche entièrement connectée** :
  - 4096 neurones.
  - Fonction d'activation : `ReLU`.
  - Suivie par une normalisation par lot et une régularisation par *Dropout* avec un taux de 0.5.

- **Couche de sortie** :
  - $N = 8$ neurones correspondant aux classes cibles.
  - Fonction d'activation : `softmax`, permettant de produire une distribution de probabilité sur les classes.

Résumé de l'Implémentation L'intégration des couches *Batch Normalization* après chaque opération de convolution et entièrement connectée stabilise et accélère l'apprentissage. Les couches *Dropout* appliquées aux couches entièrement connectées réduisent le surapprentissage (*overfitting*). Cette version modifiée d'AlexNet est adaptée pour des tâches de classification avec un nombre limité de classes tout en maintenant une bonne généralisation.

Expérience : algorithme d'optimisation : 'ADAM' , ALpha = 0.00015, epoch = 20 .
Results (train + validation :

```
Epoch 16/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 20s 71ms/step - accuracy: 0.9426 - loss: 0.1820 - val_accuracy: 0.8543 - val_loss: 0.
4974
Epoch 17/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 20s 71ms/step - accuracy: 0.9348 - loss: 0.1999 - val_accuracy: 0.8534 - val_loss: 0.
7250
Epoch 18/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 20s 71ms/step - accuracy: 0.9484 - loss: 0.1572 - val_accuracy: 0.8513 - val_loss: 0.
7080
Epoch 19/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 20s 70ms/step - accuracy: 0.9495 - loss: 0.1572 - val_accuracy: 0.7614 - val_loss: 0.
8765
Epoch 20/20
281/281 ━━━━━━━━━━━━━━━━━━━━ 20s 72ms/step - accuracy: 0.9472 - loss: 0.1623 - val_accuracy: 0.8321 - val_loss: 1.
1276
]: <keras.src.callbacks.history.History at 0x7b57f8157100>
```

Figure 28: Experience ALEXNET train + val(0.00015)

Test :

```
20]: # evaluate the model:
     alexnetmodel.evaluate(ds_test)

     93/93 ━━━━━━━━━━━━━━━━━━━━ 4s 45ms/step - accuracy: 0.8279 - loss: 1.1816
20]: [1.1498583555221558, 0.8326612710952759]
```

Figure 29: Experience ALEXNET (testing) (test)(0.00015)

Le modèle entraîné présente une performance satisfaisante sur les données d'entraînement avec une précision de 0.94. Cependant, les résultats sur les ensembles de validation et de test, tous deux à 0.83, révèlent une baisse significative, indiquant un possible surapprentissage (*overfitting*). Cela signifie que le modèle s'est trop adapté aux données

d'entraînement et ne généralise pas bien sur des données qu'il n'a pas vues. Matrice de confusion :

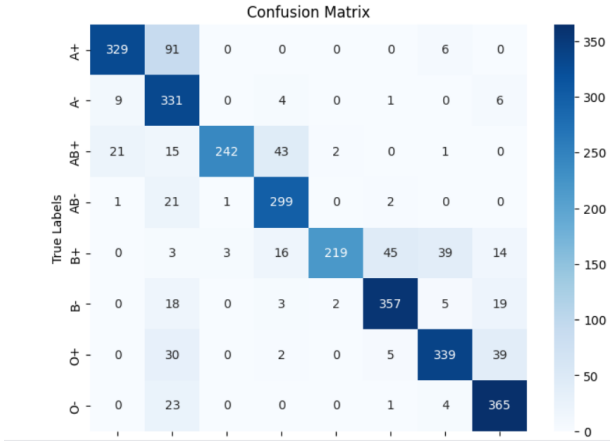


Figure 30: la matrice de confusion ALEXNET

```
Precision: 0.8569
Recall: 0.8327
F1 Score: 0.8337

Precision per class:
A+: 0.9124
A-: 0.6259
AB+: 0.9837
AB-: 0.8234
B+: 0.9820
B-: 0.8613
O+: 0.8543
O-: 0.8206
```

(a) les metrices du modéle ALEXNET

```
Recall per class:
A+: 0.7636
A-: 0.9433
AB+: 0.7555
AB-: 0.9266
B+: 0.6431
B-: 0.8784
O+: 0.8252
O-: 0.9150

F1 Score per class:
A+: 0.8314
A-: 0.7525
AB+: 0.8546
AB-: 0.8719
B+: 0.7772
B-: 0.8698
O+: 0.8395
O-: 0.8652
```

(b) Recal et la précision (ALEXNET)

Figure 31: the metrics of the model ALEXNET final

Approche 2 : Détection des groupes sanguins à l'aide des empreintes digitales

Dans cette approche, nous nous sommes concentrés sur la détection des groupes sanguins à partir d'images des empreintes. Le jeu de données est constitué de 8 classes correspondant aux différents groupes sanguins (A+, A-, B+, B-, AB+, AB-, O+, O-). Pour entraîner efficacement les modèles de deep learning, nous avons appliqué une méthodologie de préparation des données. Le dataset a été divisé en trois sous-ensembles distincts pour garantir un bon équilibre entre l'entraînement, la validation, et le test des modèles. Plus précisément, 60% des données ont été utilisées pour l'entraînement, ce qui permet au modèle d'apprendre les caractéristiques essentielles des classes. Ensuite, 20% des données ont été réservées à la validation, ce qui nous a permis de surveiller les performances du modèle et de prévenir tout problème de surapprentissage (*overfitting*). Enfin, les 20% restants ont été utilisés comme ensemble de test pour évaluer de manière indépendante les performances finales du modèle. Cette stratégie de division garantit que le modèle est

évalué sur des données qu'il n'a jamais vues auparavant, tout en disposant d'un ensemble suffisant pour apprendre les motifs visuels caractéristiques des différentes classes. Cette méthodologie établit une base solide pour entraîner et tester des modèles performants dans le cadre de cette approche. Pour les caractéristiques de la dimension des images et

```
Found 30000 files belonging to 8 classes.
Using 18000 files for training.
Found 30000 files belonging to 8 classes.
Using 12000 files for validation.
Training class names: ['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-']
Validation class names: ['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-']
The number of batches in train set: 563
The number of batches in validation set: 375
```

Figure 32: le nombre de batches et nombres d'images pour chaque sous-ensemble

le nombre de batch :

```
batch_size=32,
image_size=(60, 60),
```

Figure 33: le nombre de batches et nombres d'images pour chaque sous-ensemble

Normalisation des Données La normalisation des données est une étape fondamentale dans le processus de préparation des données pour l'entraînement des modèles de deep learning. Elle consiste à transformer les valeurs des données d'entrée pour qu'elles soient centrées autour de zéro et qu'elles aient une échelle cohérente. Cela permet d'améliorer la stabilité et l'efficacité de l'entraînement en évitant les disparités entre les différentes dimensions des données.

Application de la normalisation : Une fonction `normalize` a été définie pour appliquer la normalisation à chaque image, tout en conservant les étiquettes. Cette fonction a ensuite été mappée sur les ensembles d'entraînement, de validation et de test:

```python
# Normalize the datasets (apply map function to normalize images)
def normalize(image, label):
    image = tf.cast(image, tf.float32) / 255.0  # Normalize pixel values to [0, 1]
    return image, label

# Apply normalization
ds_train = ds_train.map(normalize)
ds_val = ds_val.map(normalize)
```

Figure 34

Modéle 1 : RESNET Model

Architecture de RESNET (IMPLEMENTATION): Dans cette section, nous décrivons limplémentation d'une architecture ResNet50 modifiée avec l'intégration de couches *Dropout* pour réduire l'*overfitting*.

Blocs Résiduels L'architecture ResNet repose sur deux types de blocs fondamentaux :

- **Bloc d'identité (`identity_block`)** : Ce bloc est utilisé lorsque la dimension d'entrée correspond à la dimension de sortie. Il comprend trois sous-couches principales :

  1. Une convolution 3×3 avec un padding *same* pour extraire les caractéristiques.
  2. Une couche de normalisation batch (`BatchNormalization`) pour stabiliser l'apprentissage.
  3. Une activation `ReLU` pour introduire de la non-linéarité.

  L'entrée du bloc est directement ajoutée à la sortie grâce à une connexion résiduelle.

- **Bloc convolutionnel (`convolutional_block`)** : Ce bloc est utilisé lorsque la dimension d'entrée est différente de celle de la sortie. Il applique une transformation au chemin de connexion résiduelle pour ajuster les dimensions, tout en suivant une structure similaire au `identity_block`.

Structure Globale du Modèle ResNet50 Le modèle commence par une couche de convolution initiale suivie d'une couche de *MaxPooling*. Ensuite, il est structuré en plusieurs étapes :

1. **Étape initiale** : Une convolution 7×7 avec stride 2, suivie d'une normalisation batch et d'une activation `ReLU`. Cette étape est suivie d'une couche *MaxPooling*.
2. **Bloc 1** : Un `convolutional_block` suivi de deux `identity_blocks`, avec 64 filtres.
3. **Bloc 2** : Un `convolutional_block` suivi de deux `identity_blocks`, avec 128 filtres.
4. **Bloc 3** : Un `convolutional_block` suivi de deux `identity_blocks`, avec 256 filtres.
5. **Bloc 4** : Un `convolutional_block` suivi de deux `identity_blocks`, avec 512 filtres.

Utilisation de Dropout Pour réduire l'*overfitting*, des couches *Dropout* sont ajoutées à des emplacements stratégiques :

- À la fin du **Bloc 1** : `Dropout(0.5)`.
- Avant la couche entièrement connectée : `Dropout(0.5)`.

Couche de Sortie La couche finale est une couche dense (*Dense Layer*) avec une activation `softmax`, qui permet de produire une distribution de probabilité sur les N classes cibles.

Résumé de l'Architecture En résumé, cette version modifiée de ResNet50, enrichie avec des couches *Dropout*, offre une meilleure robustesse contre l'*overfitting*, tout en conservant les principes fondamentaux des connexions résiduelles, qui facilitent l'apprentissage dans les réseaux très profonds.

Expérience : algorithme d'optimisation : 'SGD' , ALpha = 0.0015, epoch = 40 .
Results (train + validation :

```
Epoch 16/20
563/563 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - accuracy: 0.8654 - loss: 0.3565 - val_accuracy: 0.8113 - val_loss: 0.5380
Epoch 17/20
563/563 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - accuracy: 0.8679 - loss: 0.3496 - val_accuracy: 0.8522 - val_loss: 0.4097
Epoch 18/20
563/563 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - accuracy: 0.8724 - loss: 0.3340 - val_accuracy: 0.8336 - val_loss: 0.4861
Epoch 19/20
563/563 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.8804 - loss: 0.3199 - val_accuracy: 0.7906 - val_loss: 0.6380
Epoch 20/20
563/563 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.8780 - loss: 0.3248 - val_accuracy: 0.8346 - val_loss: 0.4850
```

Figure 35: Experience RESNET train + val(0.0015)

Test :

```
model.evaluate(ds_test)
```

```
187/187 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.8311 - loss: 0.5111
[0.4962962865829468, 0.8307152390480042]
```

Figure 36: Experience RESNET (testing) (test)(0.0015)

Le modèle présente une performance raisonnable sur l'ensemble d'entraînement avec une précision atteignant 0.8780 à la fin de l'entraînement (20e époque) et une perte de 0.3248. Cela indique que le modèle a appris de manière significative les caractéristiques des données d'entraînement. Cependant, l'accuracy sur l'ensemble de validation varie autour de 0.8346 avec une perte de 0.4850, tandis que l'évaluation sur l'ensemble de test donne une précision de 0.8311 avec une perte de 0.5111. Ces résultats montrent une cohérence entre les scores d'évaluation sur les ensembles de validation et de test, ce qui indique que le modèle est capable de généraliser sur des données qu'il n'a jamais vues. Toutefois, la différence entre les scores d'entraînement et de validation/test suggère un léger *overfitting*, où le modèle apprend davantage les spécificités des données d'entraînement au détriment de sa capacité de généralisation. L'ajout de régularisation, comme une augmentation des couches *Dropout* ou une augmentation des données d'entraînement (*data augmentation*), pourrait aider à réduire cet écart. Matrice de confusion :

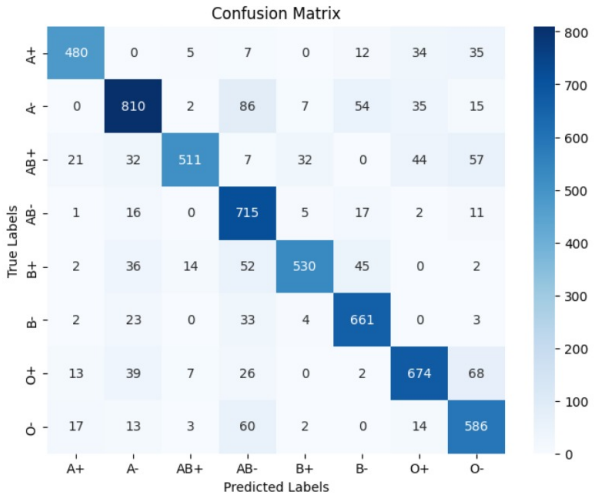
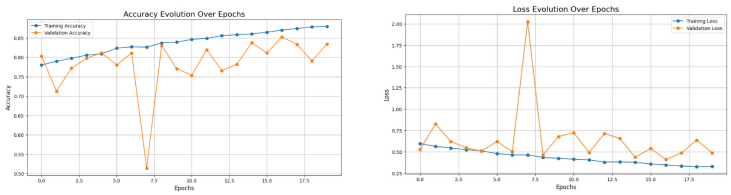


Figure 37: la matrice de confusion Resnet



(a) Accuracy visualisation (b) Loss visualisation

Figure 38: the metrics of the model Resnet final

```
Precision: 0.8414
Recall: 0.8319
F1 Score: 0.8322
```

(a) les métriques du modéle RESNET

```
Precision per class:
A+: 0.8974
A-: 0.8392
AB+: 0.9413
AB-: 0.7261
B+: 0.9125
B-: 0.8390
O+: 0.8413
O-: 0.7574

Recall per class:
A+: 0.8424
A-: 0.8075
AB+: 0.7287
AB-: 0.9320
B+: 0.7801
B-: 0.9118
O+: 0.8099
O-: 0.8465

F1 Score per class:
A+: 0.8690
A-: 0.8231
AB+: 0.8215
AB-: 0.8163
B+: 0.8411
B-: 0.8739
O+: 0.8253
O-: 0.7995
```

(b) les métriques du modéle RESNET par classe

Figure 39: the metrics of the model Resnet final

Modèle 2 : VGG-Inspired Model

Architecture du Modèle VGG (IMPLEMENTATION): Dans cette section, nous décrivons l'impl0̆0e9mentation d'une architecture de CNN inspiré par VGG, conmue pour la classification de huit classes. Le modèle combine des blocs convolutionnels avec des couches de pooling pour extraire efficacement les caractéristiques visuelles des images.

Blocs Convolutionnels L'architecture comprend plusieurs blocs convolutionnels, chaque bloc contient deux couches de convolution suivies d'une couche de extitMaxPooling. Les

détails des blocs sont les suivants :

- **Bloc 1** : Deux convolutions 3×3 avec 64 filtres chacune, suivies d'une couche *MaxPooling* 2×2.
- **Bloc 2** : Deux convolutions 3×3 avec 128 filtres chacune, suivies d'une couche *MaxPooling* 2×2.
- **Bloc 3** : Deux convolutions 3×3 avec 256 filtres chacune, suivies d'une couche *MaxPooling* 2×2.
- **Bloc 4** : Deux convolutions 3×3 avec 512 filtres chacune, suivies d'une couche *MaxPooling* 2×2.

Structure Globale du Modèle VGG-Inspired Le modèle est composé des éléments suivants :

1. **étape initiale** : Une couche *Input* pour des images de taille $64 \times 64 \times 3$.
2. **Blocs Convolutionnels** : Quatre blocs avec des couches *Conv2D* et *MaxPooling* comme décrit ci-dessus.
3. **Couches Entièrement Connectés** :
   - Une couche *Dense* avec 512 neurones, une activation $ReLU$ et une couche *Dropout* avec un taux de 0.5.
   - Une couche *Dense* avec 256 neurones, une activation $ReLU$ et une couche *Dropout* avec un taux de 0.5.
4. **Couche de Sortie** : Une couche *Dense* avec 8 neurones (nombre de classes) et une activation *softmax* pour produire une distribution de probabilité sur les classes.

Utilisation de Dropout Pour réduire l'*overfitting*, des couches *Dropout* sont ajoutés :

- Après la première couche dense : `Dropout(0.5)`.
- Après la deuxième couche dense : `Dropout(0.5)`.

Résumé de l'Architecture Le modèle VGG-Inspired, en conservant les principes fondamentaux des architectures VGG, offre une stratégie efficace pour extraire et classifier des caractéristiques visuelles. Sa structure hiérarchique et ses couches de régularisation permettent de réduire le risque d'overfitting tout en maintenant une capacité de généralisation.

Expérience : algorithme d'optimisation : 'Adam', $\alpha = 0.001$, epoch = 50 .

```
Epoch 5/10
563/563 ━━━━━━━━━━━━━━━━━━━━ 21s 38ms/step - accuracy: 0.8381 - loss: 0.4610 - val_accuracy: 0.8494 - val_loss: 0.3967
Epoch 6/10
563/563 ━━━━━━━━━━━━━━━━━━━━ 21s 37ms/step - accuracy: 0.8617 - loss: 0.3937 - val_accuracy: 0.8556 - val_loss: 0.3816
Epoch 7/10
563/563 ━━━━━━━━━━━━━━━━━━━━ 21s 37ms/step - accuracy: 0.8679 - loss: 0.3591 - val_accuracy: 0.8727 - val_loss: 0.3345
Epoch 8/10
563/563 ━━━━━━━━━━━━━━━━━━━━ 21s 38ms/step - accuracy: 0.8856 - loss: 0.3029 - val_accuracy: 0.8732 - val_loss: 0.3328
Epoch 9/10
563/563 ━━━━━━━━━━━━━━━━━━━━ 21s 37ms/step - accuracy: 0.9024 - loss: 0.2792 - val_accuracy: 0.8848 - val_loss: 0.3187
Epoch 10/10
563/563 ━━━━━━━━━━━━━━━━━━━━ 21s 37ms/step - accuracy: 0.9101 - loss: 0.2479 - val_accuracy: 0.8710 - val_loss: 0.3510
```

Figure 40: Expérience VGG-Inspired : train + validation

Results (train + validation)

```
model.evaluate(ds_test)
```

```
187/187 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.8877 - loss: 0.3266
[0.33066871762275696, 0.8810160160064697]
```

Figure 41: Expérience VGG-Inspired : test

Test Le modèle atteint une précision de 0.8610 sur l'ensemble d'entrainement, avec une perte de 0.3125 à la fin de l'entrainement (20 epochs). Sur l'ensemble de validation, la précision est de 0.8452 avec une perte de 0.4102, et l'évaluation sur l'ensemble de test donne une précision de 0.8420 avec une perte de 0.4228. Ces résultats montrent une bonne capacité de généralisation du modèle.

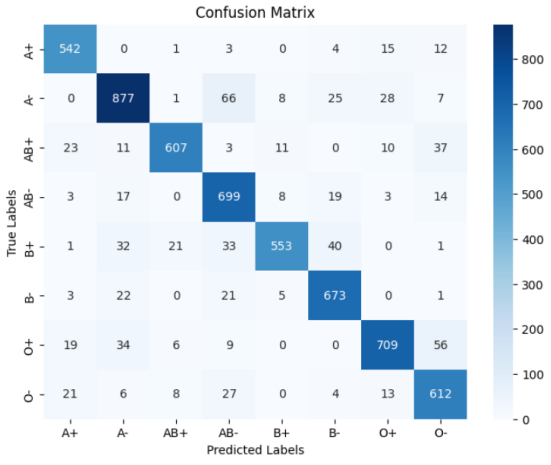
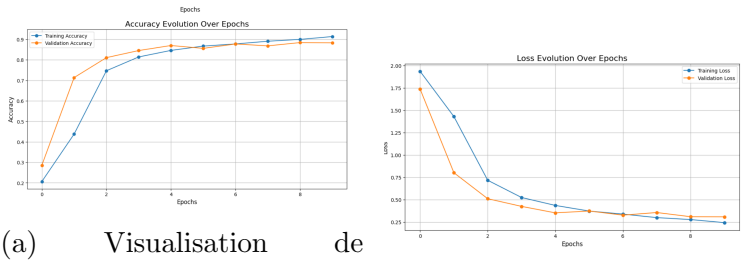


Figure 42: Matrice de confusion VGG-Inspired



(a) Visualisation de l'accuracy

(b) Visualisation de la perte

Figure 43: Métriques du modèle VGG-Inspired

```
Precision: 0.8838
Recall: 0.8803
F1 Score: 0.8805
```

(a) Métriques du modèle VGG-Inspired

```
Precision per class:
A+: 0.8821
A-: 0.8801
AB+: 0.9409
AB-: 0.8079
B+: 0.9452
B-: 0.8818
O+: 0.9075
O-: 0.8300

Recall per class:
A+: 0.9382
A-: 0.8654
AB+: 0.8618
AB-: 0.9144
B+: 0.8118
B-: 0.9301
O+: 0.8537
O-: 0.8832

F1 Score per class:
A+: 0.9092
A-: 0.8727
AB+: 0.8996
AB-: 0.8578
B+: 0.8734
B-: 0.9053
O+: 0.8798
O-: 0.8558
```

(b) Métriques du modèle VGG-Inspired par classe

Figure 44: Métriques du modèle VGG-Inspired

Matrice de confusion :

6 La comparaison des Modéles :

Après le développement des différents modèles de deep learning, il est essentiel de procéder à une étape de comparaison afin d'analyser leurs performances sur des métriques clés. Cette comparaison permet de mieux comprendre les forces et les faiblesses de chaque modèle et de choisir celui qui est le plus adapté à la tâche spécifique. Dans notre projet, nous avons adopté deux approches principales : l'analyse des empreintes digitales (fingerprint) et l'analyse des images de sang (blood images).

Approche 1 : Détection des groupes sanguins à l'aide d'images de sang

Dans cette section, nous présentons les performances des différents modèles de deep learning développés dans l'approche basée sur les images de sang. Les métriques utilisées pour évaluer les modèles incluent la précision (*precision*), le rappel (*recall*), le F1-score, ainsi que les taux de performance sur les ensembles d'entraînement (*train*), de validation (*val*) et de test (*test*). Le tableau ci-dessous résume ces résultats :

Modèle	Train	Val	Test	Précision	Rappel	F1-score
VGG Blocks ($\alpha = 0.001$, SGD)	0.90	0.94	0.94	0.94	0.94	0.94
VGG Blocks ($\alpha = 0.00015$, Adam)	0.96	0.96	0.96	0.96	0.96	0.96
ResNet ($\alpha = 0.0015$, Adam)	0.96	0.90	0.90	0.82	0.81	0.81
AlexNet ($\alpha = 0.00015$, Adam)	0.94	0.83	0.83	0.85	0.83	0.83

Table 1: Performances des modèles sur l'approche basée sur les images de sang

Analyses des résultats Les résultats obtenus montrent des performances variées entre les modèles testés :

VGG Blocks ($\alpha = 0.001$, SGD) Ce modèle atteint des résultats cohérents sur les ensembles de validation et de test avec un score de 0.94 pour toutes les métriques (précision, rappel et F1-score). Cependant, son score sur l'ensemble d'entraînement (0.90) est plus faible, ce qui indique une légère sous-adaptation.

VGG Blocks ($\alpha = 0.00015$, Adam) Ce modèle présente les meilleures performances globales avec des scores de 0.96 sur les ensembles d'entraînement, de validation et de test. Les métriques de précision, rappel et F1-score confirment une excellente capacité de généralisation, ce qui en fait un candidat idéal pour cette tâche.

ResNet ($\alpha = 0.0015$, Adam) Ce modèle montre un score élevé sur l'ensemble d'entraînement (0.96), mais ses performances chutent significativement sur les ensembles de validation et de test (0.90). Les métriques de précision (0.82), rappel (0.81) et F1-score (0.81) indiquent un problème de sur-apprentissage (*overfitting*) où le modèle s'adapte trop aux données d'entraînement et généralise mal.

AlexNet ($\alpha = 0.00015$, Adam) Ce modèle montre une performance modérée avec un score de 0.94 sur l'ensemble d'entraînement, mais des résultats plus faibles sur les ensembles de validation et de test (0.83). Les métriques de précision (0.85), rappel (0.83) et F1-score (0.83) indiquent une difficulté à généraliser correctement sur des données non vues.

Conclusion Après avoir analysé les résultats, nous concluons que le modèle **VGG Blocks ($\alpha = 0.00015$, Adam)** est le meilleur choix pour cette approche. Ses performances sont cohérentes sur les ensembles d'entraînement, de validation et de test, avec des métriques élevées (0.96) qui reflètent une excellente généralisation et une stabilité des résultats. Ce modèle est donc le plus adapté pour répondre aux objectifs spécifiques de notre projet.

Approche 2 : Détection des groupes sanguins à l'aide des empreintes digitales

Dans cette section, nous analysons les performances de deux modèles de deep learning utilisés pour la détection des groupes sanguins à partir des empreintes digitales. Les métriques d'évaluation incluent la précision (*precision*), le rappel (*recall*), le F2.1-score,

ainsi que les taux de performance sur les ensembles d'entraînement (*train*), de validation (*validation*) et de test (*test*). Le tableau ci-dessus résume ces résultats.

Modèle	**Modèle 1 : ResNet-50**	**Modèle 2 : VGG Blocks**
Optimiseur	SGD	Adam
Taux d'apprentissage (α)	0.0015	0.00015
Train Accuracy	0.87	0.91
Validation Accuracy	0.83	0.87
Test Accuracy	0.83	0.88
Précision	0.84	0.88
Rappel	0.83	0.88
F2.1 Score	0.83	0.88

Table 2: Comparaison des modèles pour l'approche de détection des groupes sanguins à partir d'empreintes digitales.

Analyses des résultats Les performances des deux modèles testés sont analysées comme suit :

Modèle 1 : ResNet-50 (SGD, $\alpha = 0.0015$) Le modèle ResNet-50 atteint une précision de 0.84, un rappel de 0.83, et un F2.1-score de 0.83, avec des taux d'exactitude (*accuracy*) similaires sur les ensembles de validation et de test (0.83). Cependant, un léger écart par rapport à l'ensemble d'entraînement (0.87) indique une certaine capacité de généralisation, bien que les résultats globaux restent limités par rapport à l'autre modèle.

Modèle 2 : VGG Blocks (Adam, $\alpha = 0.00015$) Le modèle basé sur VGG Blocks offre des performances supérieures avec une précision, un rappel, et un F2.1-score de 0.88. Les taux d'exactitude sont également élevés : 0.91 pour l'ensemble d'entraînement, 0.87 pour la validation, et 0.88 pour le test. Ces résultats indiquent que ce modèle est capable de mieux généraliser, probablement grâce à l'utilisation de l'optimiseur Adam, qui s'adapte dynamiquement aux gradients.

Comparaison des modèles Une comparaison directe montre que le modèle VGG Blocks surpasse systématiquement ResNet-50 sur toutes les métriques. Cela peut être attribué à la combinaison d'une architecture mieux adaptée à la classification et d'un optimiseur performant (Adam) avec un taux d'apprentissage approprié. En revanche, l'utilisation de SGD dans ResNet-50 limite sa convergence et sa capacité à capturer des caractéristiques complexes des empreintes digitales.

Conclusion Les résultats obtenus démontrent que le modèle **VGG Blocks (Adam, $\alpha = 0.00015$)** est mieux adapté pour cette tâche. Ses performances équilibrées sur les ensembles d'entraînement, de validation, et de test confirment une excellente généralisation et une stabilité des prédictions. Par conséquent, ce modèle est recommandé pour la détection des groupes sanguins à partir d'empreintes digitales.

7 Conclusion

Ce rapport a exploré deux approches innovantes pour la détection des groupes sanguins : à partir des empreintes digitales et des images de sang, en exploitant les capacités des algo-

rithmes de deep learning. Ces deux méthodes s'inscrivent dans le cadre d'une amélioration des diagnostics médicaux, offrant des alternatives rapides, précises et non invasives. Les résultats obtenus montrent que les deux approches présentent des avantages significatifs. L'analyse des empreintes digitales se distingue par sa simplicité et son potentiel d'intégration dans des dispositifs portables, tandis que l'analyse des images de sang offre une précision accrue grâce à la richesse des informations disponibles. Toutefois, ces performances dépendent fortement de la qualité des données utilisées et de l'architecture des modèles de deep learning. Bien que les deux méthodes aient montré un potentiel prometteur, des défis subsistent, notamment en ce qui concerne la généralisation des modèles à des populations diverses, la collecte de données représentatives et l'optimisation des architectures pour une mise en œuvre en temps réel. De plus, une comparaison plus approfondie entre ces deux approches sur des bases de données plus larges et diversifiées est nécessaire pour affiner leur application clinique. En conclusion, cette étude met en évidence le rôle crucial des technologies basées sur le deep learning dans l'évolution des méthodes de diagnostic médical. Avec des recherches supplémentaires et des améliorations des modèles, ces techniques pourraient révolutionner la détection des groupes sanguins, offrant des solutions pratiques et efficaces dans les environnements médicaux modernes.