Rapport de projet S6 : HEMAI

Une intelligence artificielle pour simplifier le diagnostic de la leucémie aiguë lymphoblastique

Tom CONNERY, Antonin DECOUVELAERE, DJEZVEDJIAN Emmanuel,
LETEURE-CORTELLA Maté

Table des matières

1 Introduction

1.1 Contexte

La leucémie lymphoblastique aiguë (LLA) est un cancer du sang et de la moelle osseuse qui se caractérise par la prolifération rapide de lymphocytes immatures. Elle est la forme la plus courante de leucémie chez les enfants, mais elle affecte également les adultes, avec une incidence et un pronostic qui varient selon l'âge. Avec l'avènement du "Big Data" et l'augmentation exponentielle de la puissance de calcul au cours des deux dernières décennies que l'IA a véritablement révolutionné de nombreux secteurs, y compris la médecine.

1.2 Explication biologique

La leucémie lymphoblastique aiguë est un cancer prenant naissance dans les lymphoïde des cellules souches présentent dans la moelle osseuse. Les cellules souches lymphoïdes se transforment normalement en lymphocytes, un type de globule blanc. Les lymphocytes aident à combattre les infections et à détruire les cellules anormales. Dans le cas de la leucémie la différentiation des cellules souches n'a pas lieu, et les cellules restent au stade de lymphoblastes, qui sont des cellules sanguines immatures.

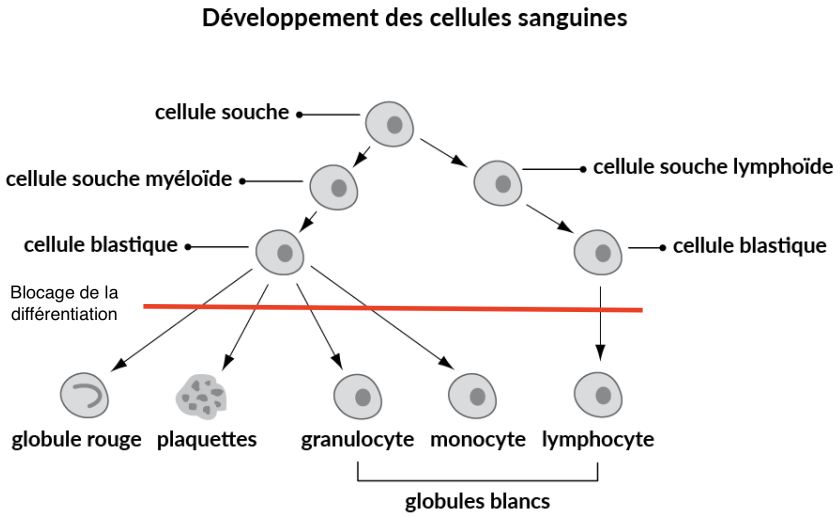


FIGURE 1 – Schéma du développement des cellules bloqué par le cancer

Avec le temps, les cellules blastiques prennent la place des cellules sanguines normales, les empêchant ainsi d'accomplir leurs tâches. Ces cellules présentent donc des différentes morphologiques qui peuvent être détectées par un modèle de reconnaissance d'image bien entraînée.

1.3 Problème

Le diagnostic et le traitement de la LLA sont compliqués par l'hétérogénéité de la maladie, qui se manifeste par diverses anomalies génétiques et moléculaires. Les méthodes traditionnelles de diagnostic, bien qu'efficaces, peuvent manquer de précision pour identifier les sous-types spécifiques de LLA qui nécessitent des approches thérapeutiques distinctes. Il est donc essentiel de développer des outils avancés capables d'analyser des ensembles de données complexes pour identifier des biomarqueurs prédictifs et optimiser les stratégies de traitement.

1.4 Relations entre l'IA et la médecine

L'application de l'IA en médecine a ouvert de nouvelles opportunités pour la recherche et la pratique clinique, en particulier dans le domaine de l'oncologie. En effet, l'IA facilite le développement de modèles prédictifs pour la réponse aux traitements, ouvrant la voie à une médecine de précision. Les collaborations interdisciplinaires entre informaticiens, biologistes et cliniciens sont essentielles pour exploiter pleinement le potentiel de l'IA en médecine et pour traduire les avancées technologiques en bénéfices concrets pour les patients atteints de LLA.

1.5 Répartition des tâches

Pour mener à bien ce projet, les tâches ont été réparties entre les membres de l'équipe comme suit :

- **Antonin** : Responsable des mathématiques derrière l'intelligence artificielle, assurant le développement et l'optimisation des algorithmes.
- **Tom** : En charge de la construction de notre modèle d'IA, intégrant les algorithmes développés et assurant leur implémentation technique.
- **Emmanuel** : Effectue la recherche bibliographique, collectant et analysant les données et les études pertinentes pour guider le développement du modèle.
- **Maté** : Gère le traitement d'image et l'entraînement du modèle, préparant les données visuelles et optimisant les performances du modèle à travers des techniques d'apprentissage automatique.

2 Etude bibliographique

2.1 Contexte et Objectifs

La première étape de notre travail a consisté à effectuer une revue des études existantes afin de mettre en évidence les différents éléments essentiels à la mise en place d'un CNN efficace de prédiction de leucémie.

Nous avons choisi de nous appuyer sur les travaux d'une équipe iranienne ayant récupéré 3242 images microscopiques de lymphoblastes, collectées dans plusieurs hôpitaux de Téhéran. Ces images représentent des cellules saines, ou atteintes de leucémie lymphoblastique aiguë de type B (B-ALL). Grâce à ce jeu de données, nous pouvons mettre en place un CNN simple à partir des indications de l'équipe iranienne.

Ce choix est motivé par le fait que les images de la base de données n'étaient pas traitées, ce qui nous a permis dans un premier temps de nous concentrer sur la préparation des données et ainsi de maîtriser au final notre système depuis la collecte même de données. De plus, grâce à une utilisation de modèles légers et préentraînés, cette première approche est un bon équilibre entre efficacité, légèreté et qualité.

2.2 Description du Dataset

Le jeu de données est divisé en deux classes principales : Bénin et Malin (LAL), cette dernière étant subdivisée en trois sous-types : Early Pre-B, Pre-B et Pro-B ALL. Le tableau ci-dessous résume la répartition :

Type	Classes	Échantillons	Dimensions des Images
Bénin	Hématogones	512	1024×768
Malin	Early Pre-B ALL	979	1024×768
Malin	Pre-B ALL	955	1024×768
Malin	Pro-B ALL	796	1024×768

TABLE 1 – Description du dataset local de B-ALL et de ses sous-classes.

On remarque que les classes sont à peu près équi-représentées, ce qui limite une possibilité de sous-représentation de classe lors de l'entraînement du CNN.

2.3 Traitement des images

2.3.1 Préparation des données

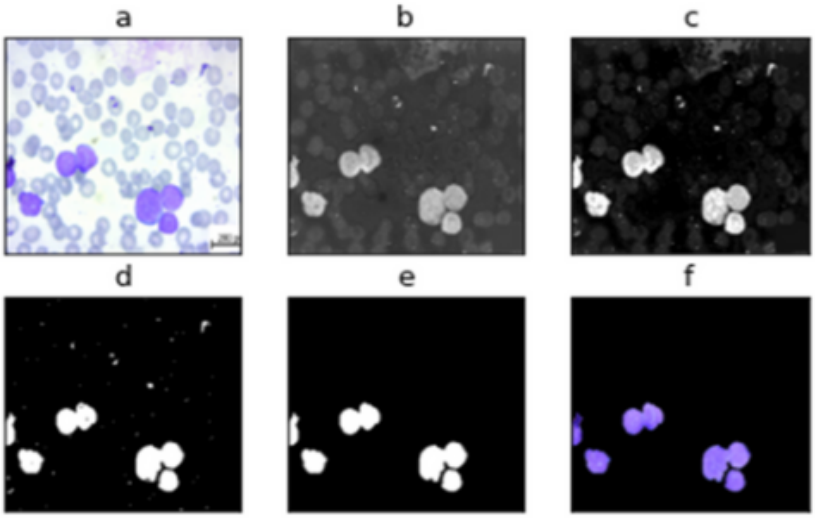
Les images ont été décodées et redimensionnées à 224x224 pixels, afin d'être compatibles avec les architectures CNN pré-entraînées. Nous avons augmenté artificiellement les données par des flips horizontaux et verticaux pour diversifier les images présentées au modèle. Enfin, les images de chaque classe ont été séparées dans différents dossiers de train, test et validation.

2.3.2 Segmentation

Les images étant non-traitées dans le data-set, il est nécessaire de réduire au maximum le bruit des données présentées au CNN, tel que les éléments non-pertinents, ou encore l'éclairage.

Tout d'abord, une conversion de l'espace couleur RGB vers LAB a été réalisée, une conversion classique en traitement d'images. En effet, LAB est un espace de couleur se rapprochant plus fidèlement à la perception de la lumière par l'oeil humain, et qui permet par extraction de la composante [a] de ne plus prendre en compte l'intensité de la lumière de l'image. Par la suite, la combinaison d'un algorithme de clustering K-means et d'un seuillage binaire permet d'éliminer les composants de fond et d'extraire les régions d'intérêt (ROI).

En appliquant le résultat de la segmentation à l'image initiale, on obtient une image qui ne contient que le nécessaire en ayant enlevé le plus possible de bruit.



Segmentation stages: a) RGB color space, b) LAB color space, c) K-means clustering d) Binary thresholding, e) Cleaning methods and mask generation, f) Applied Mask on the RGB original image. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

FIGURE 2 – Etapes du traitement des images

2.4 Choix du modèle CNN

Trois architectures CNN légères ont été testées : EfficientNetB0, MobileNetV2 et NASNet Mobile. MobileNetV2 a été sélectionné pour sa haute efficacité, son nombre raisonnable de paramètres pris en compte par le modèle (3.5 millions), sa faible taille mémoire (14 Mo), et sa compatibilité avec les appareils mobiles.

2.5 Résultats

Afin de quantifier la réussite du modèle, l'équipe utilise plusieurs paramètres classiques : accuracy (proportion de prédictions correctes), recall (détection correcte des cas malins), specificity (détection correcte des cas bénins). Le tableau ci-dessous résume les caractéristiques et résultats des différents modèles entraînés.

Réseaux	**Époque**	**Train Accuracy**	**Batch size**	**Size (MB)**	**Depth**	**Paramètres**	**Test Accuracy**
EfficientNetB0	30	99.1	32	29	132	5.3 million	100
MobileNetV2	30	99.62	32	14	88	3,538,984	100
NASNet Mobile	30	86.3	32	23	771	5,326,716	94.1

TABLE 2 – Comparaison des performances des extracteurs de caractéristiques

Nos résultats sur le modèle MobileNetV2 concordent avec ceux de l'équipe comme la figure ci-dessous l'illustre. Nous obtenons notamment une précision de test de 0.9938.

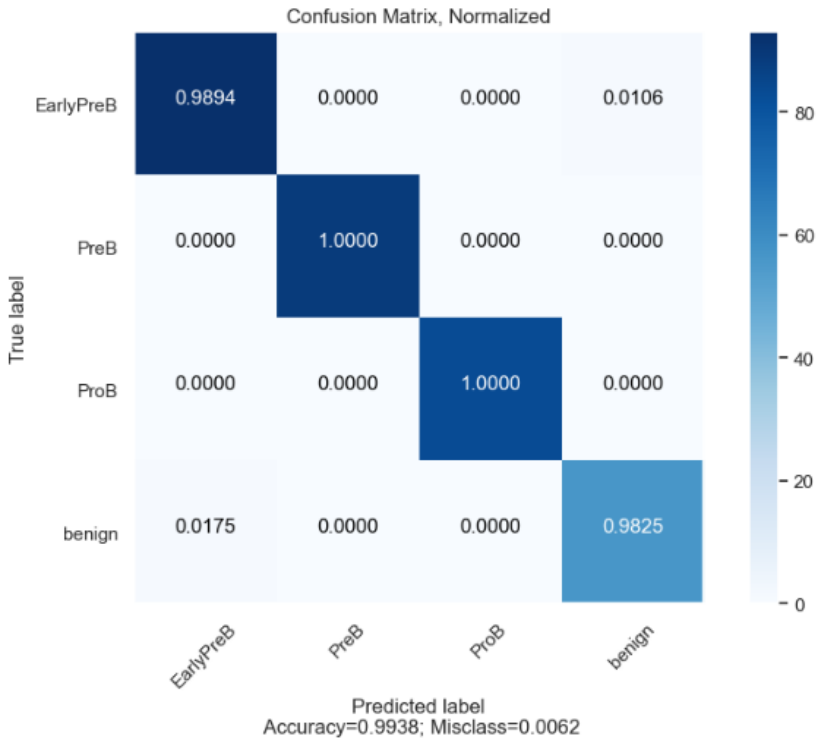


FIGURE 3 – Matrice de confusion sous le modèle MobilNetV2

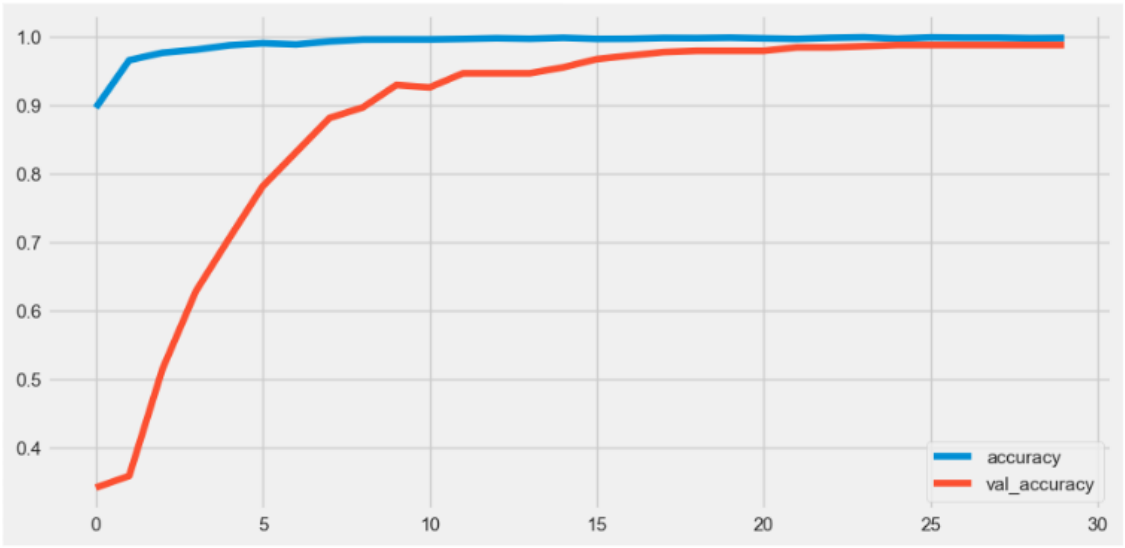


FIGURE 4 – Performance du modèle MobileNetV2

Nous pouvons donc valider le travail de l'équipe iranienne, et utiliser ce premier modèle afin de vérifier si d'autres set de données sont compatibles avec une mise en place d'un modèle CNN de reconnaissance d'image.

3 Notre modèle

3.1 Notre base de donnée

Les images utilisées dans cette étude proviennent de microscopes de laboratoires médicaux spécialisés dans l'hématologie. Ces images ont été collectées dans le cadre de recherches cliniques visant à améliorer le diagnostic de la leucémie lymphoblastique aiguë (ALL). Les échantillons de cellules sanguines ont été prélevés chez des patients atteints de leucémie ainsi que chez des sujets sains, permettant ainsi une comparaison rigoureuse entre cellules cancéreuses et normales.

Notre base de données est composée d'images classées en deux catégories principales : les cellules cancéreuses, spécifiquement des B-lymphoblastes leucémiques, et les cellules normales, identifiées comme des précurseurs B-lymphoïdes. La composition détaillée du dataset est présentée ci-dessous :

- **Nombre total de sujets** : 118
  - **Patients atteints de leucémie (ALL)** : 69
  - **Patients sains** : 49

Le dataset a été divisé en trois ensembles distincts pour faciliter l'entraînement, la validation et le test du modèle :

- **Ensemble d'entraînement** : 73 sujets (47 ALL, 26 normaux) - 10,661 cellules
- **Ensemble de pré-test** : 28 sujets (13 ALL, 15 normaux) - 1,867 cellules
- **Ensemble de test final** : 17 sujets (9 ALL, 8 normaux) - 2,586 cellules

chaque image du train dataset est étiquetée. Un exemple de cellule est représenté ci-dessous :

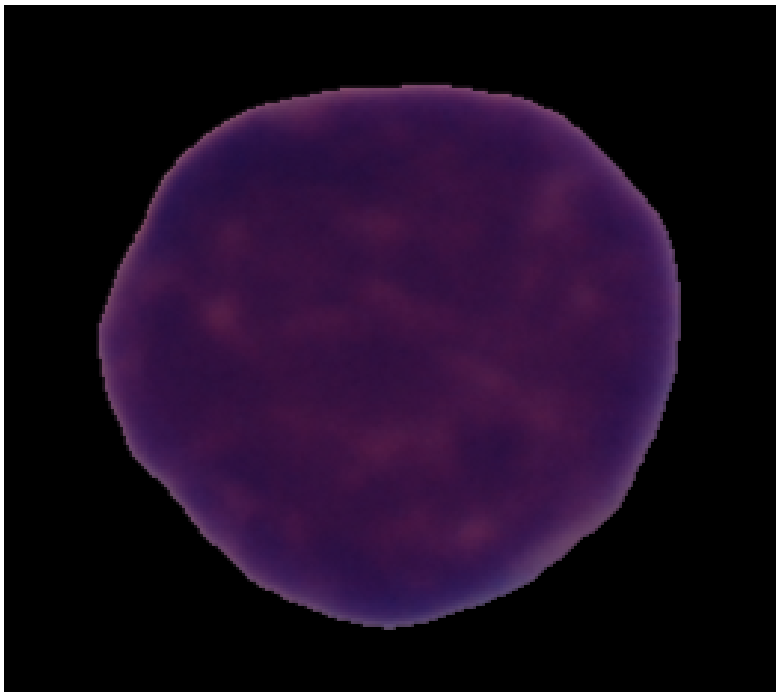


FIGURE 5 – Image étiquetée : UID_1_9_1_all.bmb

3.2 Traitement des données et création des datasets

Dans le cadre de cette étude, plusieurs librairies Python ont été employées pour faciliter la manipulation des données, la construction du modèle de réseau de neurones, ainsi que l'entraînement et l'évaluation de ce dernier. Voici une description des principales librairies utilisées et de leurs rôles respectifs :

- **torch** : La librairie PyTorch est au cœur de notre implémentation. Elle fournit les outils nécessaires pour la création et l'entraînement de réseaux de neurones, notamment grâce à ses fonctionnalités de calcul tensoriel et d'auto-différentiation.

— **torch.nn** : Ce module de PyTorch contient des couches et des fonctions de perte pré-définies, essentielles pour la construction de notre modèle de réseau de neurones convolutifs (CNN).
— **torch.nn.functional** : Ce module fournit des fonctions utilitaires pour les opérations de réseau de neurones, telles que les fonctions d'activation et les fonctions de perte.
— **torch.optim** : Ce module de PyTorch inclut divers algorithmes d'optimisation, tels que l'optimiseur Adam, utilisé pour ajuster les poids de notre modèle pendant l'entraînement.
— **torchvision** : Cette librairie, associée à PyTorch, offre des outils pour le traitement d'images et la manipulation de datasets d'images, incluant des transformations et des datasets pré-chargés.
— **matplotlib.pyplot** : Cette librairie est utilisée pour la visualisation des données et des résultats, notamment pour tracer les courbes de perte et de précision pendant l'entraînement et l'évaluation du modèle.
— **sklearn.metrics** : Ce module de scikit-learn fournit des fonctions pour calculer diverses métriques d'évaluation, telles que la matrice de confusion, essentielle pour évaluer la performance de notre modèle de classification.
— **seaborn** : Cette librairie est utilisée pour créer des visualisations statistiques attrayantes et informatives, complétant ainsi les capacités de matplotlib.
— **numpy** : Cette librairie est fondamentale pour la manipulation de tableaux numériques et de matrices, fournissant des structures de données et des fonctions mathématiques essentielles pour le traitement des données.

La préparation des données est une étape cruciale dans le processus de développement de notre modèle de classification. Cette section détaille les transformations appliquées aux images et la division du dataset en ensembles d'entraînement, de validation et de test.

Dans cette étude, nous avons utilisé exclusivement des images étiquetées pour assurer la qualité et la fiabilité de notre modèle de classification. Par conséquent, nous avons sélectionné uniquement les données "train" de la base de données initiale, car ce sont les seules à être étiquetées.

Les images de notre dataset subissent une série de transformations pour les préparer à l'analyse par notre modèle de réseau de neurones convolutifs. Ces transformations sont définies à l'aide du module `transforms` de la librairie `torchvision` :

```
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
    ])
```

Les transformations appliquées incluent :

— **Redimensionnement** : Les images sont redimensionnées à une résolution uniforme de 128x128 pixels. Cette étape est essentielle pour assurer une cohérence dans les dimensions des entrées du modèle, facilitant ainsi le traitement par les couches convolutives.
— **Conversion en tenseurs** : Les images redimensionnées sont converties en tenseurs PyTorch. Cette conversion est nécessaire pour que les images puissent être traitées par les couches de notre réseau de neurones.
— **Normalisation** : Les valeurs des pixels des images sont normalisées pour avoir une moyenne de 0.5 et un écart-type de 0.5. La normalisation est une étape cruciale pour améliorer la performance du modèle, car elle facilite la convergence de l'algorithme d'optimisation en assurant que les données d'entrée ont une distribution similaire.

Le dataset est divisé en trois ensembles distincts : un ensemble d'entraînement, un ensemble de validation et un ensemble de test. Cette division est effectuée à l'aide de la fonction `random_split` de PyTorch, qui permet de diviser le dataset de manière aléatoire tout en respectant les proportions spécifiées :

```
train_size = int(0.7 * len(full_dataset))
val_size = int(0.15 * len(full_dataset))
test_size = len(full_dataset) - train_size - val_size
train_dataset, val_dataset, test_dataset = random_split(full_dataset, [
   train_size, val_size, test_size])
```

Les proportions utilisées pour la division du dataset sont les suivantes :

- **Ensemble d'entraînement** : 70% du dataset complet.
- **Ensemble de validation** : 15% du dataset complet.
- **Ensemble de test** : 15% du dataset complet.

Enfin, nous créons des DataLoaders pour chaque ensemble de données. Les DataLoaders sont des objets PyTorch qui facilitent l'itération sur les ensembles de données pendant l'entraînement et l'évaluation du modèle. Ils permettent également de spécifier la taille des batches et de mélanger les données :

```
train_dl = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_dl = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_dl = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Les paramètres utilisés pour la création des DataLoaders sont les suivants :

- **Taille des batches** : 32 images par batch.
- **Mélange des données** : Les données d'entraînement sont mélangées (`shuffle=True`) pour assurer une distribution aléatoire des images pendant l'entraînement, ce qui est essentiel pour éviter le sur-ajustement (overfitting). Les ensembles de validation et de test ne sont pas mélangés (`shuffle=False`) pour assurer une évaluation cohérente et reproductible du modèle.

3.3 Présentation du modèle

Le modèle est basé sur un réseau de neurones convolutifs qui va permettre de détecter les caractéristiques spatiales des images.

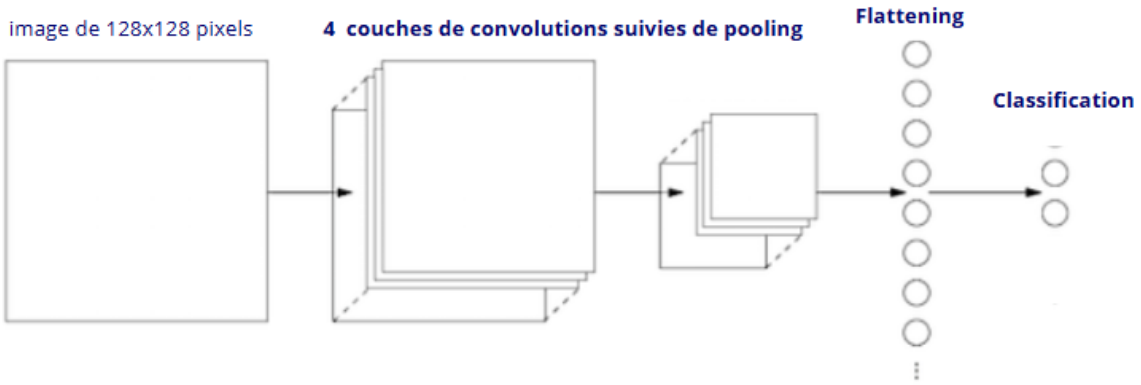


FIGURE 6 – détail de la structure du réseau de neurones

Cette structure est implémenté en **Pytorch** :

```
class Network(nn.Module):
    def _init_(self):
        super(Network, self)._init_()
        self.conv1 = nn.Conv2d(3, 8, kernel_size=3)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3)
        self.conv3 = nn.Conv2d(16, 32, kernel_size=3)
        self.conv4 = nn.Conv2d(32, 64, kernel_size=3)

        self.dropout_rate = 0.25
        self.pool = nn.MaxPool2d(2, 2)
        self._to_linear = self._get_conv_output()

```

```
        self.fc1 = nn.Linear(self._to_linear, 100)
        self.fc2 = nn.Linear(100, 2)

    def _get_conv_output(self):
        x = torch.zeros(1, 3, 128, 128)
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        return x.numel()

    def forward(self, X):
        x = self.pool(F.relu(self.conv1(X)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, self.dropout_rate, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x,dim=1)
```

3.3.1 Les couches de convolutions

La première étape consiste à appliquer un **filtre de convolution** sur nos images d'entrée décomposées en 3 canaux de 128x128 (chaque canal correspond à une couleur au format RGB). On applique un filtre (kernel) de taille $(3, 3, 3)$ qui va se déplacer sur l'image en entrée. On obtient ainsi une première couche de convolution de taille 126×126. Cette couche de convolution est définie par une matrice de poids W de taille $(3, 3, 3)$ et un biais b. Ainsi l'activation des neurones de cette couche est calculée par la formule :

$$a^1 = \sigma(b + w * a^0) \tag{1}$$

Avec a^1 l'activation de cette couche, σ la fonction d'activation et a^0 l'activation correspondant à l'image d'entrée. On répéte cette opération 8 fois pour obtenir 8 couches de convolution qui permettent de capturer 8 caractéristiques à partir de l'image d'entrée.

3.3.2 Etape de Max-pooling

L'étape de max-pooling succède la première étape de convolution, elle permet de se séparer de l'information exacte de position pour se concentrer sur les relations spatiales entre les features (caractéristiques). Le max-pooling consiste simplement à extraire la valeur maximale d'un kernel de taille (2,2) pour diminuer les dimensions de sortie. On obtient ainsi 8 couches de taille (63,63).
On répète cette étape de convolution suivie d'un max-pooling 3 fois pour finalement obtenir 64 couches de convolutions de taille $(6, 6)$.

3.3.3 Flattening et couche dense

Le Flattening est nécessaire pour relier le tenseur-3D de taille (64,6,6) issu des étapes de convolutions à une couche de neurones complétement connectée. Ainsi le tenseur-3D est transformé en un tenseur-1D de taille $64 \times 6 \times 6 = 2304$
Ce tenseur-1D est l'entrée d'un réseau entiérement connecté (réseau dense). La première couche est constituée de 2304 neurones, tous reliées à 100 neurones de la couche intermédiaire. Enfin la dernière couche est reliée à deux neurones pour la classification.

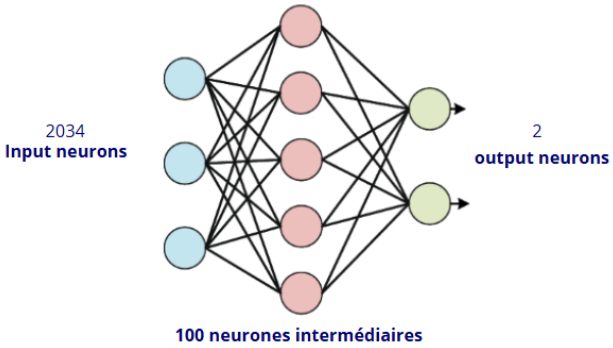


FIGURE 7 – Structure des couches de neurones entiérement connectées

La fonction d'activation entre la première couche et la couche de neurones intermédiaire est reLu, définit par : ReLU(x)= $max(0, x)$. Cette fonction permet d'ajouter de la non-linéarité entre les couches. Enfin, la dernière fonction d'activation est la fonction *Softmax*, définie par : $a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$. Elle permet d'assimiler la couche de classification à une densité de probabilité. Ainsi, la prédiction correspond au neurone (Sain ou malade) avec la probabilité la plus élevée.

3.3.4 Dropout

L'objectif de la méthode de dropout est de limiter le sur-apprentissage (overfitting) sur les données d'entraînements en forçant le réseau à apprendre des caractéristiques générales. Le **Dropout** correspond à la suppression temporaire de certains neurones dans la couche intermédaire de notre réseau dense. Le choix des neurones supprimés est aléatoire avec une probabilité $p = 0.3$. Lors de la phase de test tout les neurones sont réactivés avec leur activation normalisée.

3.3.5 Fonction de coût

La fonction de coût permet de calculer l'erreur entre les prédictions de notre modèle. La fonction de coût utilisée est la **Negative Log-Likelyhood Loss** (NLLL). Elle est définie par :

$$C(w, b) = -\ln(a_y^L)$$

Avec a_y^L l'activation de la dernière couche, assimilée à une densité de probabilité. Cette fonction à l'avantage de pénaliser fortement les prédictions erronées avec une haute probabilité, et à l'inverse récompenser les prédictions correctes avec une grande confiance.

3.3.6 Descente de gradient et optimisation

L'objectif de l'algorithme de **descente de gradient** est de minimiser la fonction de coût, donc de maximiser la vraisemblance de notre modéle. Au lieu de calculer les dérivés partielles de la fonction de coût par rapport aux poids et aux biais pour toutes les données d'entraînements, on applique l'algorithme à une petite taille d'échantillon qui va permettre d'ajouter les poids du modèle dans la direction souhaitée. C'est **l'algorithme de descente de gradient stochastique**. Ainsi les poids w et biais b des neurones sont mis-à-jour de la façon suivante :

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \to b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

Avec m la taille de l'échantillon et η, le learning-rate.
En pratique, on utilise l'optimiseur **Adam** de PyTorch qui fonctionne de façon analogue à l'algorithme de descente stochastique avec un taux d'apprentissage η qui varie :

```
opt=optim.Adam(cnn_model.parameters(), lr=3e-4)
```

3.4 Les résultats de notre modèle

Dans cette section, nous présentons les résultats obtenus à partir de l'évaluation de notre modèle de réseau de neurones convolutifs (CNN) entraîné pour la classification des cellules sanguines en cellules leucémiques et cellules normales. L'évaluation a été effectuée sur un ensemble de test distinct, et les résultats sont présentés sous forme de précision de test, de courbes de perte et de précision, ainsi que d'une matrice de confusion.

Pour évaluer la performance de notre modèle sur l'ensemble de test, nous avons calculé la précision du test (accuracy), qui mesure la proportion d'images correctement classées par le modèle. Le code suivant illustre le processus d'évaluation :

```
model.eval()
all_preds = []
all_labels = []
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_dl:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

print(f"\  n     Test Accuracy: {100 * correct / total:.2f}%")
```

Le modèle est d'abord placé en mode évaluation avec `model.eval()`. Ensuite, nous itérons sur l'ensemble de test, prédisons les classes des images, et comparons les prédictions avec les étiquettes réelles pour calculer la précision du test. La précision du test obtenue est affichée à la fin du processus.

Pour visualiser la performance du modèle pendant l'entraînement et la validation, nous avons tracé les courbes de perte et de précision. Ces courbes permettent de suivre l'évolution de la perte et de la précision au fil des époques et d'identifier d'éventuels problèmes d'overfitting ou d'underfitting.

```
sns.set(style='whitegrid')
epochs = params_train["epochs"]
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

sns.lineplot(x=list(range(1, epochs+1)), y=loss_hist["train"], ax=ax[0],
    label='Train Loss')
sns.lineplot(x=list(range(1, epochs+1)), y=loss_hist["val"], ax=ax[0], label
    ='Val Loss')
ax[0].set_title("Loss")

sns.lineplot(x=list(range(1, epochs+1)), y=metric_hist["train"], ax=ax[1],
    label='Train Accuracy')
sns.lineplot(x=list(range(1, epochs+1)), y=metric_hist["val"], ax=ax[1],
    label='Val Accuracy')
ax[1].set_title("Accuracy")

plt.show()
```

Les courbes de perte et de précision sont tracées à l'aide de la librairie `seaborn`. Les courbes de perte montrent l'évolution de la perte d'entraînement et de validation, tandis que les courbes de précision montrent l'évolution de la précision d'entraînement et de validation. Ces courbes permettent de visualiser la performance du modèle et d'identifier les tendances au fil des epochs.

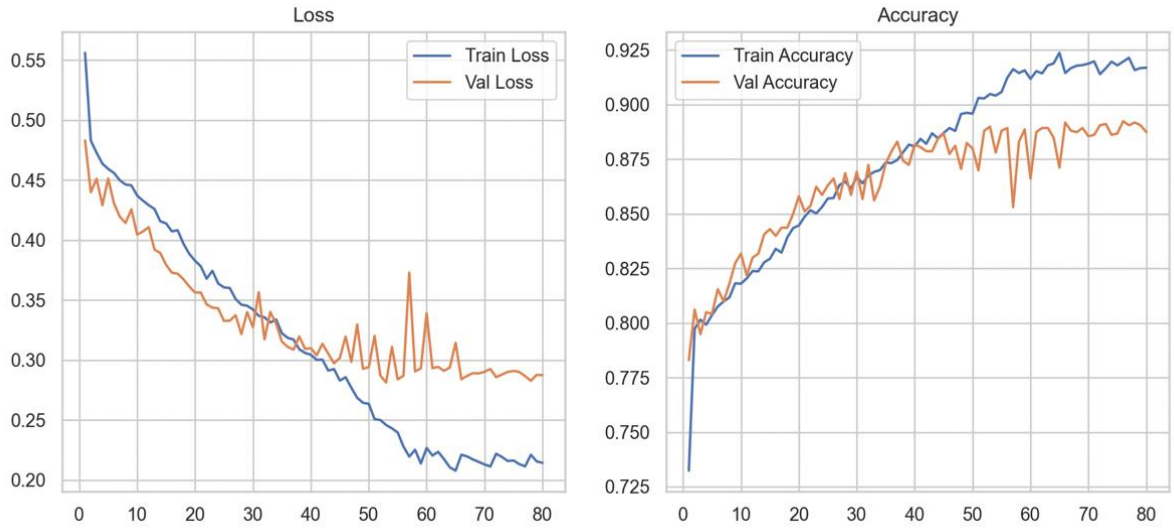


FIGURE 8 – Pertes et précisions en fonction du nombre d'époch

Enfin, nous avons tracé une matrice de confusion pour évaluer la performance du modèle en termes de vrais positifs, vrais négatifs, faux positifs et faux négatifs. La matrice de confusion fournit une vue d'ensemble des performances de classification du modèle et permet d'identifier les classes qui sont bien ou mal classées.

```
cm = confusion_matrix(all_labels, all_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
    full_dataset.classes)
disp.plot(cmap=plt.cm.Blues)
plt.title('Matrice de Confusion')
plt.show()
```

La matrice de confusion est tracée à l'aide de la librairie `sklearn.metrics`. Elle montre les performances de classification du modèle pour chaque classe et permet de visualiser les erreurs de classification. La matrice de confusion est un outil précieux pour évaluer la performance du modèle et identifier les domaines à améliorer.

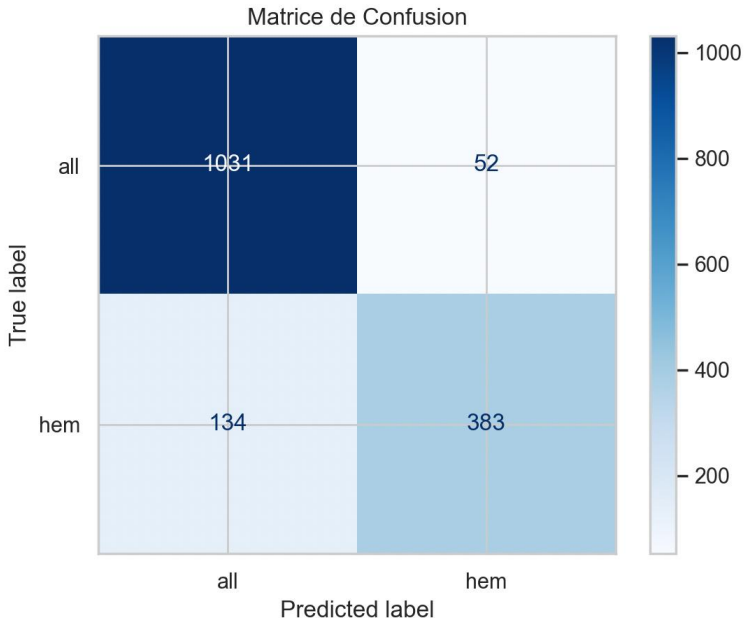


FIGURE 9 – Matrice de confusion

Les courbes de perte et de précision (Figure 8) montrent l'évolution de la performance du modèle pendant l'entraînement et la validation. Les courbes de perte indiquent une diminution progressive de la perte d'entraînement et de validation, ce qui suggère que le modèle apprend efficacement à classifier les images. Les courbes de précision montrent une augmentation progressive de la précision d'entraînement et de validation, ce qui indique que le modèle améliore sa capacité à classifier correctement les images au fil des epochs.

La matrice de confusion (Figure 9) montre les performances de classification du modèle pour chaque classe. Les valeurs diagonales de la matrice représentent les vrais positifs et les vrais négatifs, tandis que les valeurs hors diagonale représentent les faux positifs et les faux négatifs. La matrice de confusion permet de visualiser les erreurs de classification et d'identifier les classes qui sont bien ou mal classées.

Afin de faciliter l'utilisation de ce modèle, nous le sauvegardons sous le nom `HEMAI.pth`, ce qui permet de le réutiliser dans différents scripts annexes. Pour cela, nous utilisons la commande suivante :

```
torch.save(model.state_dict(), 'HEMAI.pth')
```

En conclusion, les résultats obtenus à partir de l'évaluation du modèle montrent une performance prometteuse pour la classification des cellules sanguines en cellules leucémiques et cellules normales. Nous avons obtenu une précision finale de 88,38%, ce qui indique que le modèle est capable de distinguer efficacement les cellules leucémiques des cellules normales. La précision du test, les courbes de perte et de précision, ainsi que la matrice de confusion fournissent une vue d'ensemble de la performance du modèle et permettent d'identifier les domaines à améliorer pour des recherches futures.

4 Améliorations et Perspectives

Dans cette section, nous discutons des améliorations potentielles et des perspectives futures pour notre modèle de réseau de neurones convolutifs (CNN) développé pour la classification des cellules sanguines. Nous abordons également la valeur créée par notre travail et les défis rencontrés, tout en mettant en lumière les opportunités de recherche et développement futures.

4.1 La Valeur Créée

Notre travail a permis de créer un modèle de réseau de neurones convolutifs avec une précision de 88,38%, développé entièrement de A à Z avec les ressources et les connaissances disponiblesen un temps limité. Cette réalisation est à prendre en considération car nous nous avons été limité par la technologie et le temps disponible.

L'un des principaux avantages de notre modèle est sa rapidité. Grâce à une architecture optimisée, notre modèle est capable de traiter et de classifier un grand nombre d'images en un temps relativement court. Cette rapidité est cruciale pour des applications cliniques où le temps de diagnostic peut être un facteur déterminant.

De plus, notre modèle a été conçu pour être accessible. En utilisant des technologies et des bibliothèques open-source, nous avons veillé à ce que notre solution puisse être facilement déployée et utilisée dans divers environnements cliniques, même ceux disposant de ressources limitées.

Un autre aspect important de notre travail est la réduction des coûts associés au diagnostic des cellules leucémiques. En automatisant une partie du processus de diagnostic, notre modèle permet de diminuer la charge de travail des professionnels de santé et de réduire les coûts opérationnels. Cela peut potentiellement conduire à une meilleure allocation des ressources et à une amélioration de l'efficacité globale des processus de diagnostic.

4.2 Perspectives et Améliorations

Bien que notre modèle ait atteint une précision remarquable de 88,38%, il existe plusieurs directions pour des améliorations et des recherches futures.

4.2.1 Outils Technologiques

L'une des principales limitations de notre travail a été la puissance de calcul disponible. Nos ordinateurs ne nous permettent pas d'entraîner des modèles capables d'obtenir une précision proche des 100%. L'utilisation de GPU (Graphics Processing Units) est fortement recommandée pour ce type de calcul intensif. Les GPU, avec leur capacité à effectuer des calculs parallèles massifs, peuvent significativement réduire le temps d'entraînement et permettre l'exploration de modèles plus complexes et potentiellement plus précis.

4.2.2 Structure de Notre Modèle

Améliorer la structure de notre modèle est une autre piste pour augmenter la précision tout en diminuant le nombre de calculs nécessaires. Plusieurs approches peuvent être envisagées :

- **Optimisation des Couches** : Réévaluer le nombre et le type de couches dans notre réseau pour améliorer l'efficacité sans sacrifier la précision.
- **Techniques de Régularisation** : Implémenter des techniques de régularisation supplémentaires pour éviter le sur-ajustement et améliorer la généralisation du modèle.
- **Architectures Avancées** : Explorer des architectures de réseaux de neurones plus avancées, telles que les réseaux résiduels (ResNet) ou les réseaux à attention, qui peuvent offrir des performances supérieures.

4.2.3 Recherche et Développement

La participation à des projets de recherche et le développement continu sont essentiels pour continuer à innover et améliorer les technologies de diagnostic. Voici quelques pistes de recherche futures :

- **Collaborations Académiques et Industrielles** : Travailler en collaboration avec d'autres institutions académiques et partenaires industriels pour partager les connaissances et les ressources.
- **Intégration de Nouvelles Données** : Incorporer de nouvelles données et types de données pour améliorer la robustesse et la précision du modèle.
- **Validation Clinique** : Effectuer des validations cliniques supplémentaires pour évaluer la performance du modèle dans des environnements réels et obtenir des certifications nécessaires pour un déploiement à grande échelle.

4.2.4 Remerciments

Nous tenons à exprimer notre gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet. Un grand merci à nos encadrants pour leurs conseils avisés et leur soutien tout au long de ce projet. Leurs réunions de suivi régulières nous ont été d'une aide précieuse. Nous remercions également Joel Le Grand pour son cours qui nous a fourni des conseils intéressants et pertinents pour notre travail. Enfin, merci à tous ceux qui, de près ou de loin, ont apporté leur aide et leur soutien pour la réussite de ce projet.

En conclusion, bien que notre modèle actuel représente une avancée significative avec une précision de 88,38%, il existe de nombreuses opportunités pour des améliorations et des recherches futures. En exploitant des outils technologiques avancés, en optimisant la structure de notre modèle, et en participant activement à des projets de recherche et développement, nous pouvons continuer à innover et à améliorer les technologies de diagnostic médical assisté par l'intelligence artificielle.

Références

[1] Hosseini, A., Eshraghi, M. A., Taami, T., Sadeghsalehi, H., Hoseinzadeh, Z., Ghaderzadeh, M., & Rafiee, M. (2023). A mobile application based on efficient lightweight CNN model for classification of B-ALL cancer from non-cancerous cells : A design and implementation study. *Informatics in Medicine Unlocked*, 39, 101244. Disponible à : `https://www.sciencedirect.com/science/article/pii/S2352914823000862`.

[2] Mourya, S., Kant, S., Kumar, P., Gupta, A., & Gupta, R. (2019). ALL Challenge dataset of ISBI 2019 (C-NMC 2019) (Version 1) [dataset]. The Cancer Imaging Archive. Disponible à : `https://www.cancerimagingarchive.net/collection/c-nmc-2019/`.

[3] Société Française d'Hématologie (SFH). (2024). Intelligence Artificielle : Quelles applications pour le diagnostic ? *Cancérologie Pratique*. Disponible à : `https://www.cancerologie-pratique.com/journal/article/007934-sfh-2024-intelligence-artificielle-quelles-applications-diagnostic`.

[4] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Disponible à : `https://www.deeplearningbook.org/`.

[5] Pourquoi Docteur. (n.d.). Leucémie, lymphome : l'intelligence artificielle pour mieux diagnostiquer. Disponible à : `https://www.pourquoidocteur.fr/Articles/Question-d-actu/37302-Leucemie-lymphome-l-intelligence-artificielle-mieux-diagnostiquer`.

[6] Business Decision. (n.d.). Tutoriel Machine Learning : Comment mettre en place l'apprentissage d'un réseau de neurones ? Disponible à : `https://fr.blog.businessdecision.com/tutoriel-machine-learning-comment-mettre-en-place-lapprentissage-dun-reseau-de-neurones/`.

[7] PyTorch. (n.d.). Documentation et ressources pour l'apprentissage profond. Disponible à : `https://pytorch.org/`.

[8] Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. *arXiv preprint arXiv :1311.2901*. Disponible à : `https://arxiv.org/abs/1311.2901`.