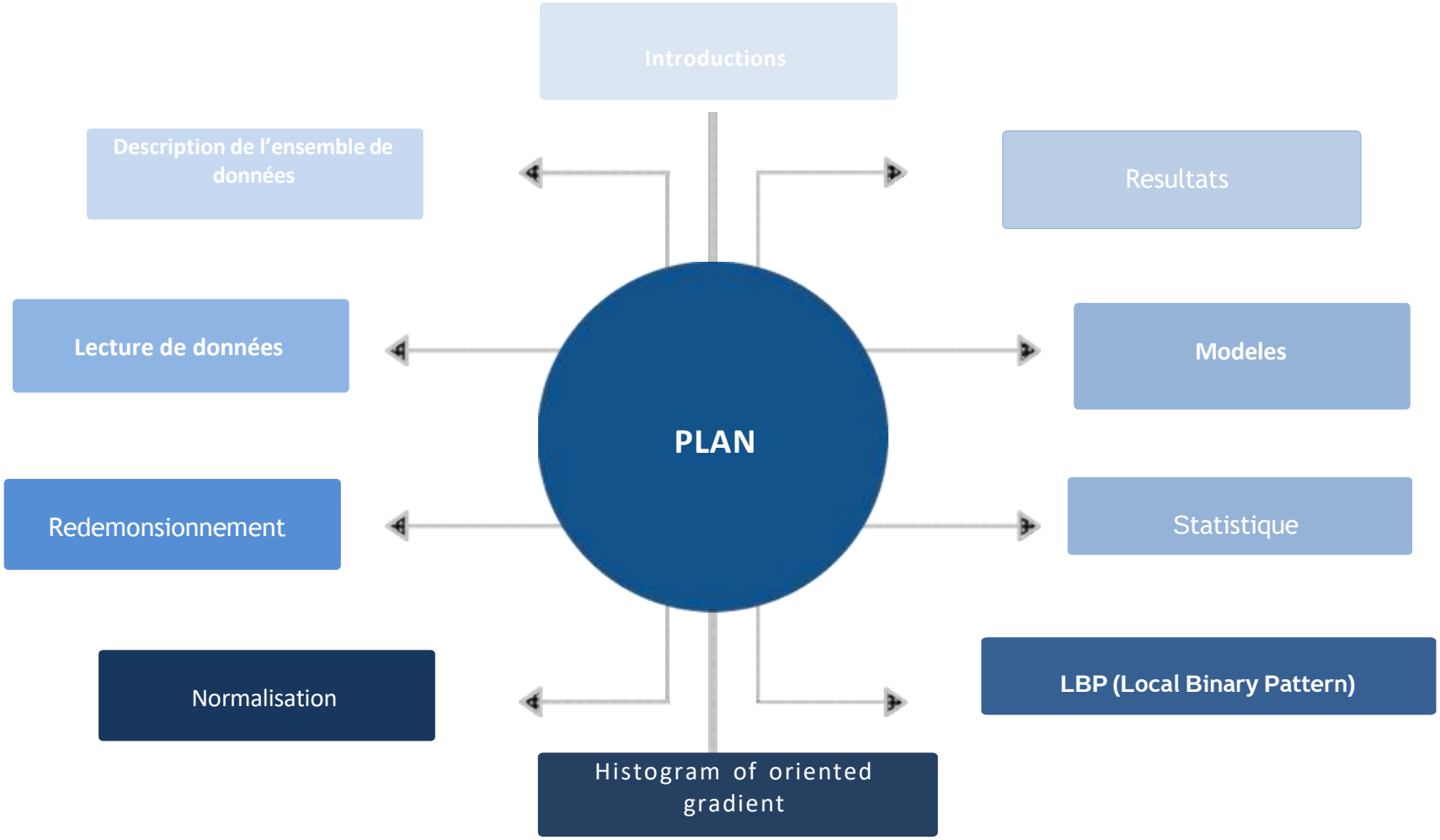
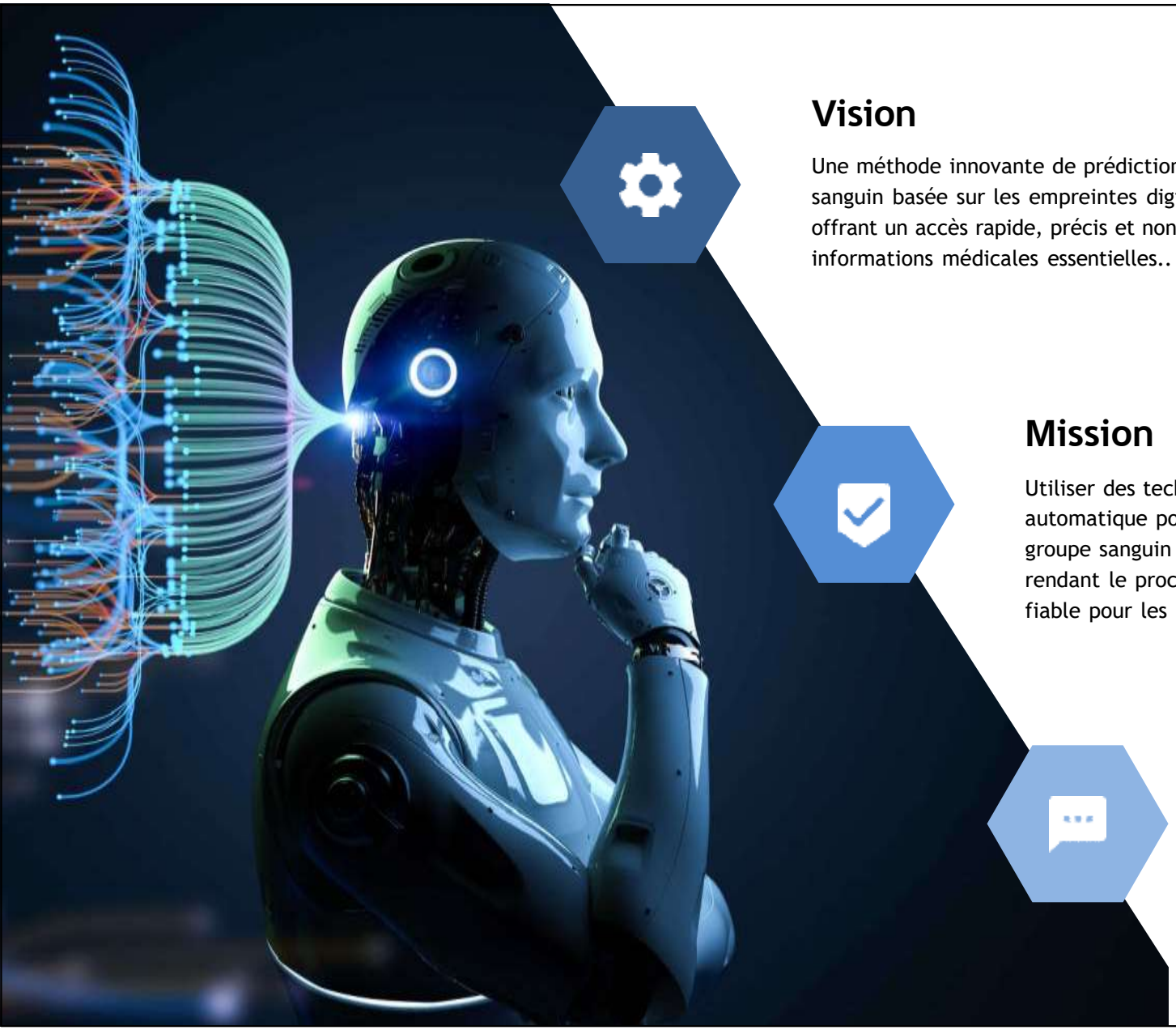
Machine Learning

Prédiction de groupe sanguin à partir d'empreintes digitales


Introductions
Description de l'ensemble de données
Resultats
Lecture de données
PLAN
Modeles
Redemonsionnement
Statistique
Normalisation
LBP (Local Binary Pattern)
Histogram of oriented gradient



Vision

Une méthode innovante de prédiction du groupe sanguin basée sur les empreintes digitales, offrant un accès rapide, précis et non invasif aux informations médicales essentielles..

Mission

Utiliser des techniques avancées d'apprentissage automatique pour prédire avec précision le groupe sanguin à partir d'empreintes digitales, en rendant le processus plus rapide, accessible et fiable pour les besoins médicaux.

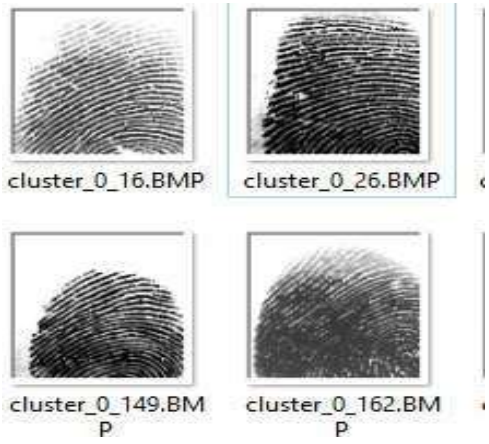
Objectifs

- Développer un modèle capable de prédire le groupe sanguin à partir d'empreintes digitales avec une haute précision.

Description de l’ensemble de données

Source

Données des empreintes issues de kagge



Richesse du Dataset

L'ensemble du dataset comprend environ 6000 images au total, ce qui offre une base de données riche et équilibrée pour entraîner et évaluer les modèles de classification supervisée.

Types de données .

- Il est organisé en plusieurs dossiers, chacun nommé selon un groupe sanguin spécifique (par exemple : A+, A-, B+, B-, AB+, AB-, O+, O-).
- Chaque dossier contient des images d’empreintes digitales d’individus appartenant au groupe sanguin correspondant

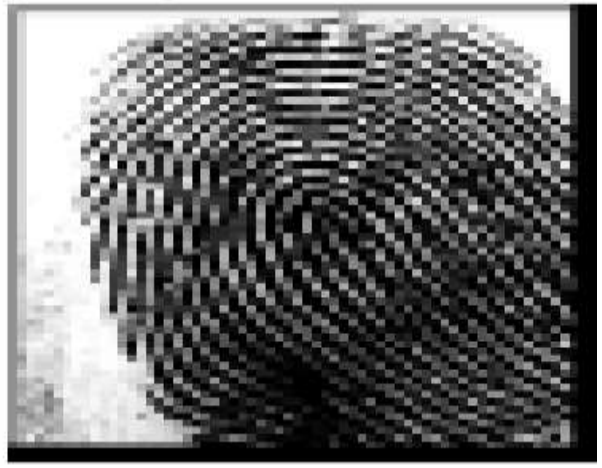
Préparation de données

Dans un premier temps, les images d’empreintes digitales ont été converties en niveaux de gris pour simplifier leur traitement tout en conservant les informations importantes.

Redimonsionnement

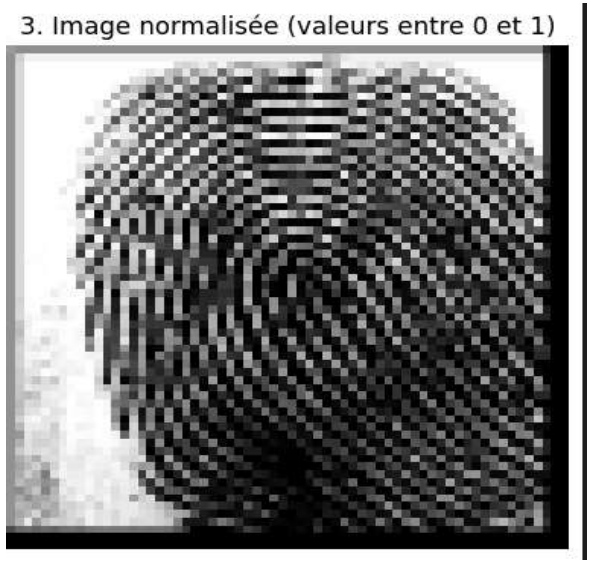
Afin d'assurer une homogénéité dans les dimensions des données d'entrée, toutes les images d'empreintes digitales ont été redimensionnées à 64x64 pixels.
Ce redimensionnement permet de réduire la charge computationnelle, d'accélérer l'entraînement des modèles et de garantir que chaque image possède le même format, indispensable pour le traitement automat·ique.

2. Image redimensionnée (64x64)



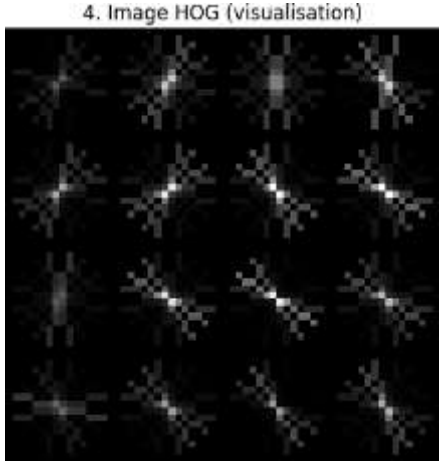
Normalisation

Après le redimensionnement des images, nous avons effectué une normalisation en divisant les valeurs des pixels par 255. Cette étape est essentielle car elle permet de ramener les valeurs des pixels dans une plage de 0 à 1. Ce processus facilite l'entraînement des modèles en améliorant leur stabilité, tout en accélérant leur vitesse de convergence. En normalisant les données de cette manière, nous optimisons l'apprentissage du modèle, ce qui se traduit par une meilleure performance dans la tâche de classification.



HOG (Histogram of Oriented Gradients)

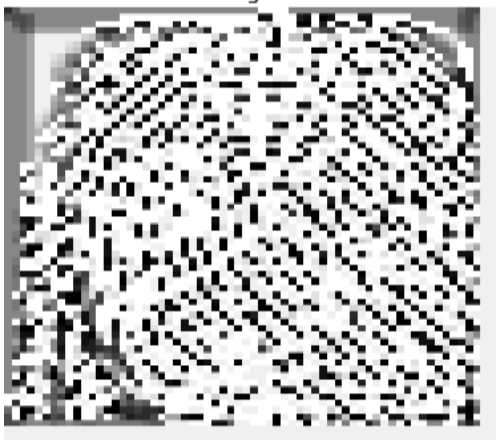
Pour extraire les caractéristiques des empreintes digitales, la technique HOG (Histogram of Oriented Gradients) a été appliquée. Elle permet de capturer la structure locale de l'image en analysant les directions des variations d'intensité. En divisant l'image en petites cellules et en calculant des histogrammes de gradients pour chacune d'elles, HOG extrait des informations précises sur la forme et les motifs, essentielles pour améliorer la performance des modèles de classification.



LBP (Local Binary Pattern)

Pour enrichir l'extraction des caractéristiques, la méthode LBP (Local Binary Pattern) a été utilisée. Cette technique analyse la texture locale de l'image en comparant chaque pixel à ses voisins proches. En attribuant des codes binaires selon l'intensité relative des pixels, LBP permet de capturer des motifs fins et répétitifs présents dans les empreintes digitales, offrant ainsi une représentation complémentaire aux caractéristiques extraites par HOG.

5. Image LBP



Les statistiques

En complément des méthodes avancées d’extraction, des caractéristiques statistiques simples ont été calculées à partir des images.
Ces caractéristiques comprennent la moyenne, l'écart type, la médiane, le maximum et le minimum des intensités des pixels. Elles permettent de résumer globalement l'information lumineuse de l'image et offrent des indicateurs utiles sur la distribution des niveaux de gris, renforçant ainsi la capacité des modèles à différencier les empreintes digitales selon leur groupe sanguin

```
6. Statistiques de l'image :
 - Moyenne     : 0.4042
 - Écart-type  : 0.3508
 - Médiane     : 0.3216
 - Maximum     : 1.0000
 - Minimum     : 0.0000
```

Modèles

1. Méthodologie

Trois modèles de classification ont été implémentés et évalués :

Support Vector Machine (SVM): est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Il cherche à trouver l'hyperplan optimal dans un espace de grande dimension qui sépare les différentes classes avec la marge maximale.

Random Forest : C'est une méthode de machine learning qui combine plusieurs arbres de décision (decision trees) pour améliorer la précision et réduire le surapprentissage.

K-Nearest Neighbors (KNN) : Algorithme simple de classification (ou régression) qui attribue à un point la classe majoritaire parmi ses *k* plus proches voisins dans l'espace des caractéristiques.

Modèles

2. Optimisation des parametres :

```python
def optimize_hyperparameters(model, param_grid, X_train, y_train):

    total_combinations = 1
    for values in param_grid.values():
        total_combinations *= len(values)

    random_search = RandomizedSearchCV(
        model, param_grid,
        n_iter=min(3, total_combinations),
        cv=3, n_jobs=-1,
        scoring='accuracy', random_state=42
    )

    random_search.fit(X_train, y_train)


    return {
        'best_params': random_search.best_params_,
        'best_score': random_search.best_score_
    }
```

- Calcule le nombre total de combinaisons d'hyperparamètres possibles.
- Limite les essais à 3 maximum (ou moins si l'espace est petit).
- Teste aléatoirement des combinaisons avec RandomizedSearchCV (en 3 folds par défaut).
- Retourne les meilleurs hyperparamètres et leur score.

Objectif : Trouver rapidement une bonne configuration sans explorer toutes les possibilités.

Modèles

3.Entrainement et évaluation des modèles :

```python
def train_and_evaluate_models(X_train, y_train, X_val, y_val, X_test, y_test, classes):

    models_config = {
        'SVM': {
            'model': SVC(probability=True, random_state=42),
            'params': {
                'C': [0.1, 1],
                'kernel': ['rbf', 'linear']
            }
        },
        'Random Forest': {
            'model': RandomForestClassifier(random_state=42, n_jobs=-1),
            'params': {
                'n_estimators': [50, 100],
                'max_depth': [None, 10]
            }
        },
        'KNN': {
            'model': KNeighborsClassifier(n_jobs=-1),
            'params': {
                'n_neighbors': [3, 5],
                'weights': ['uniform']
            }
        }
    }
```

Objectif principal :

Comparer et sélectionner le meilleur modèle ML (SVM, Random Forest, KNN)

Configuration des modèles

Définit 3 modèles (SVM, Random Forest, KNN) avec leurs hyperparamètres à tester.

Modèles

3. Entrainement et évaluation des modèles :

```
results = {}

# Entrainement et évaluation de chaque modèle
for name, config in models_config.items():
    logging.info(f"\nEntrainement du modèle {name}...")

    # Optimisation des hyperparamètres
    logging.info("Optimisation des hyperparamètres...")
    optimization_results = optimize_hyperparameters(
        config['model'], config['params'], X_train, y_train
    )

    # Configuration du modèle avec les meilleurs paramètres
    model = config['model'].set_params(**optimization_results['best_params'])

    # Validation croisée
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, n_jobs=-1)

    # Entrainement final
    model.fit(X_train, y_train)

    # Prédictions
    y_pred_val = model.predict(X_val)
    y_pred_test = model.predict(X_test)
```

Entrainement des modèles

- Entraînement : Validation croisée (5 folds)
- Entraînement final sur tout le jeu d'entraînement
- Prédiction : Génère les prédictions sur validation et test

Modèles

3.Entrainement et évaluation des modèles :

```
# Évaluation
val_report = classification_report(y_val, y_pred_val, target_names=classes, output_dict=True)
test_report = classification_report(y_test, y_pred_test, target_names=classes, output_dict=True)

# Matrice de confusion
cm = confusion_matrix(y_test, y_pred_test)

# Sauvegarde des résultats
results[name] = {
    'model': model,
    'val_accuracy': val_report['accuracy'],
    'test_accuracy': test_report['accuracy'],
    'val_report': val_report,
    'test_report': test_report,
    'confusion_matrix': cm,
    'cv_scores': cv_scores.tolist(),
    'best_params': optimization_results['best_params'],
    'optimization_score': optimization_results['best_score']
}
```

Cette partie évalue chaque modèle optimisé en générant des rapports de classification et une matrice de confusion, puis stocke les métriques clés (accuracy, paramètres, scores) dans un dictionnaire structuré pour comparaison finale.

Modèles

3.Entrainement et évaluation des modèles :

```
# Évaluation
val_report = classification_report(y_val, y_pred_val, target_names=classes, output_dict=True)
test_report = classification_report(y_test, y_pred_test, target_names=classes, output_dict=True)

# Matrice de confusion
cm = confusion_matrix(y_test, y_pred_test)

# Sauvegarde des résultats
results[name] = {
    'model': model,
    'val_accuracy': val_report['accuracy'],
    'test_accuracy': test_report['accuracy'],
    'val_report': val_report,
    'test_report': test_report,
    'confusion_matrix': cm,
    'cv_scores': cv_scores.tolist(),
    'best_params': optimization_results['best_params'],
    'optimization_score': optimization_results['best_score']
}
```

Cette partie évalue chaque modèle optimisé en générant des rapports de classification et une matrice de confusion, puis stocke les métriques clés (accuracy, paramètres, scores) dans un dictionnaire structuré pour comparaison finale.

Modèles

3.Entrainement et évaluation des modèles :

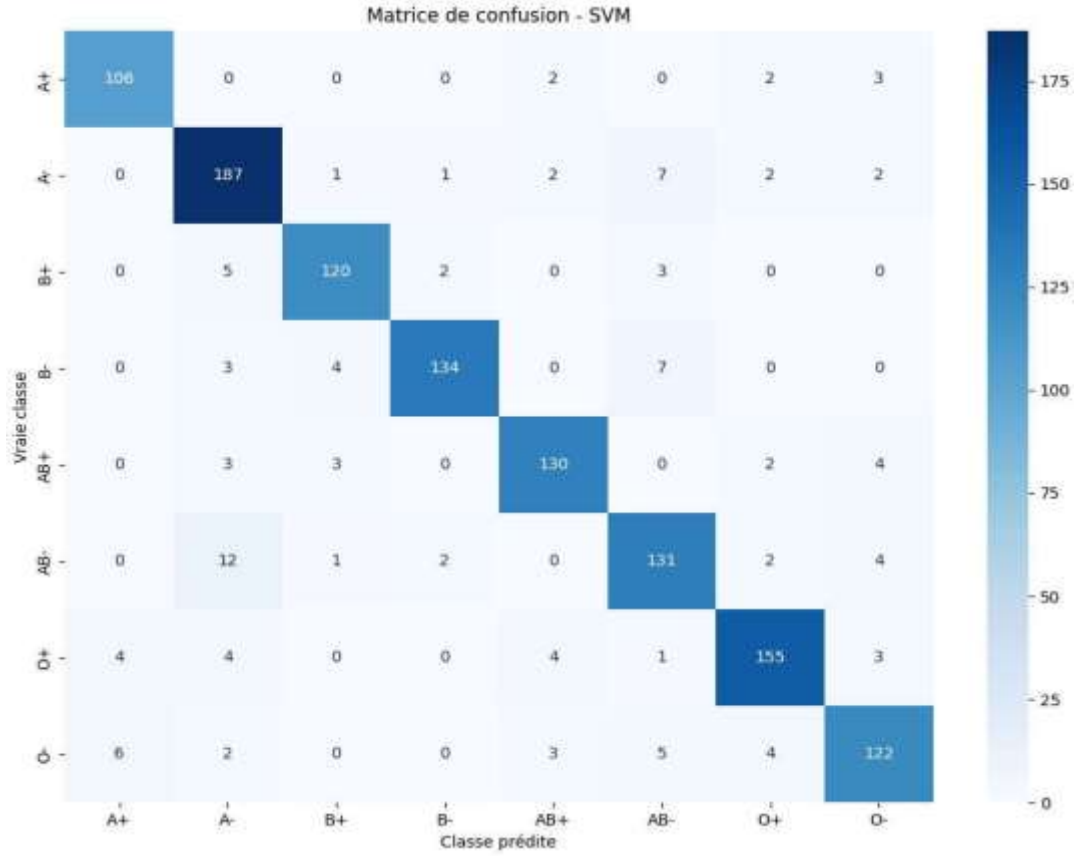
```
    # Affichage des résultats
    logging.info(f"\nPerformance du modèle {name}:")
    logging.info(f"Meilleurs paramètres: {optimization_results['best_params']}")
    logging.info(f"Score CV moyen: {cv_scores.mean():.4f} (±{cv_scores.std():.4f})")
    logging.info(f"Précision sur validation: {val_report['accuracy']:.4f}")
    logging.info(f"Précision sur test: {test_report['accuracy']:.4f}")

return results
```

Résultats

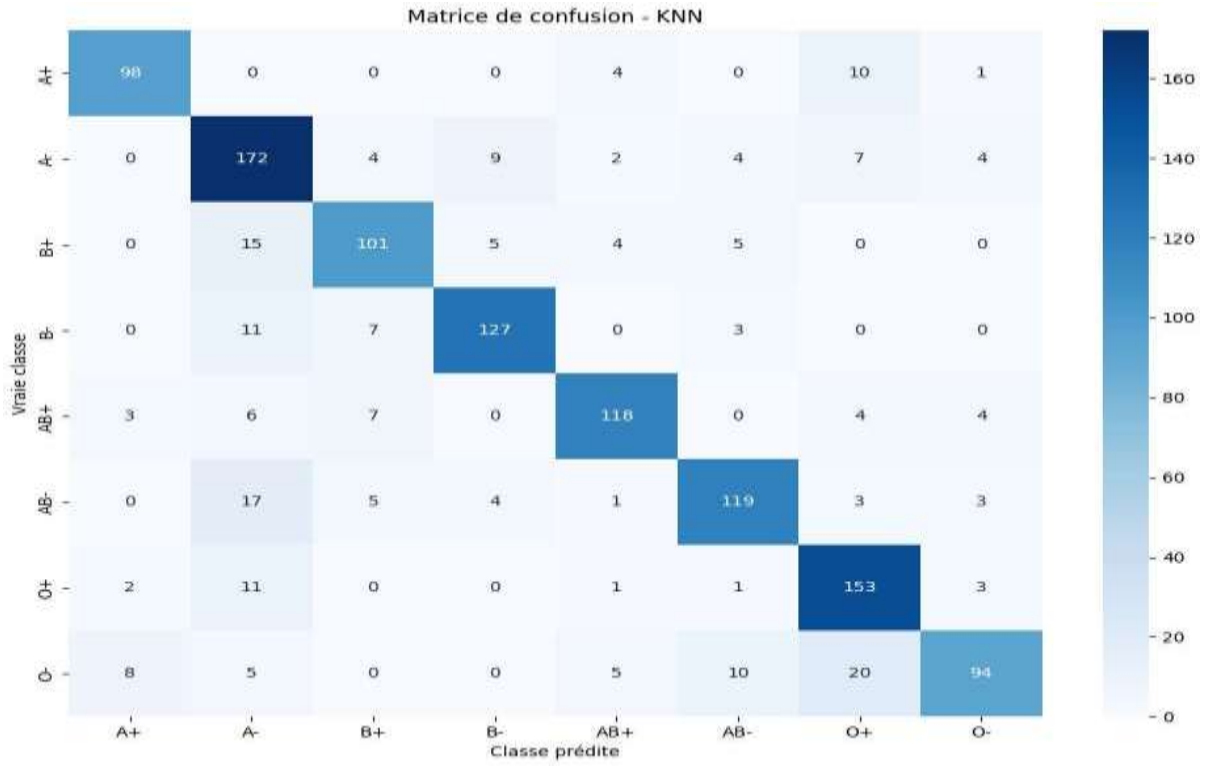
Résultats d'optimisation des parametres:

Modèle	Meilleurs Paramètres	Précision(Validation)	Précision(Test)	Score CV moyen
SVM	{'kernel': 'linear', 'C': 0.1}	90.08%	90.42%	0.8947 (±0.0048)
Random Forest	{'n_estimators': 100, 'max_depth': None}	88.67%	88.92%	0.8733 (±0.0051)
KNN	{'weights': 'uniform', 'n_neighbors': 5}	83.50%	81.83%	0.8172 (±0.0128)

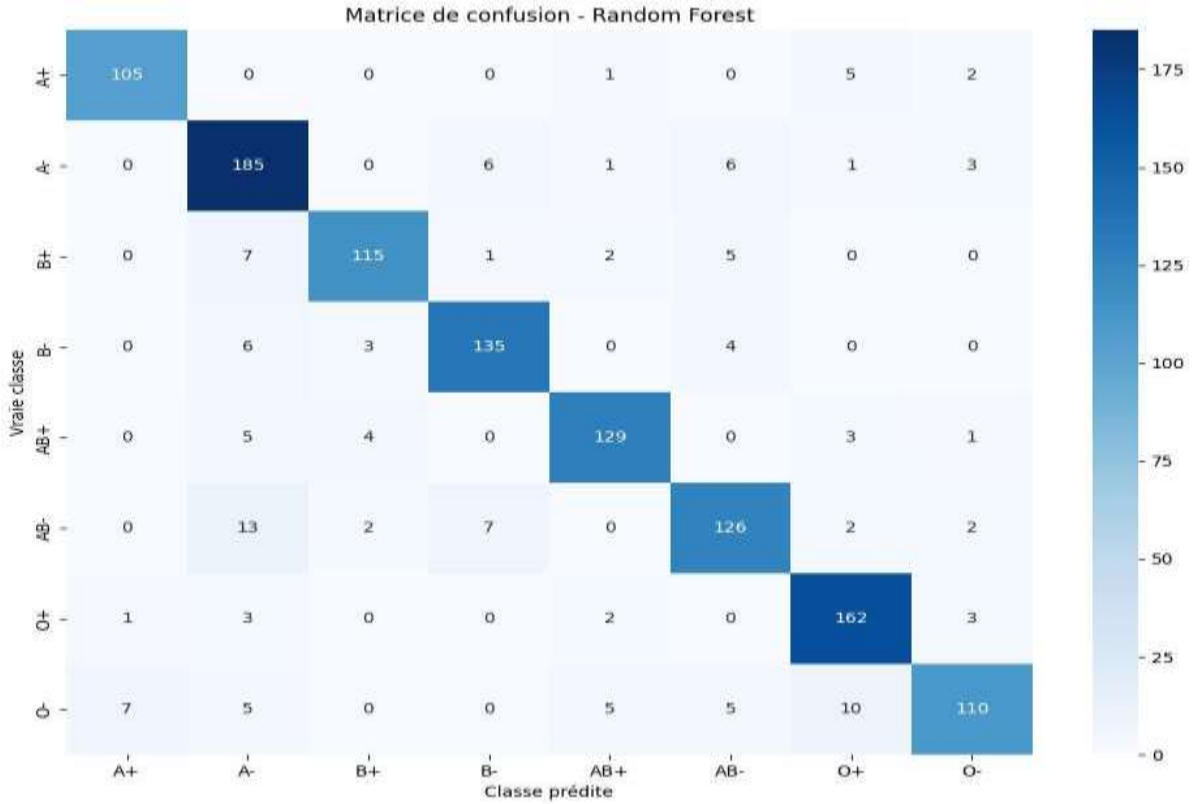
Résultats


Matrice de confusion - SVM
Vraie classe
A+
A-
B+
B-
AB+
AB-
O+
O-
106 0 0 0 2 0 2 3
0 187 1 1 2 7 2 2
0 5 120 2 0 3 0 0
0 3 4 134 0 7 0 0
0 3 3 0 130 0 2 4
0 12 1 2 0 131 2 4
4 4 0 0 4 1 155 3
6 2 0 0 3 5 4 122
A+ A- B+ B- AB+ AB- O+ O-
Classe prédite
175
150
125
100
75
50
25
0

Résultats


Matrice de confusion - KNN
Vraie classe
Classe prédite
A+
A-
B+
B-
AB+
AB-
O+
O-
98 0 0 0 4 0 10 1
0 172 4 9 2 4 7 4
0 15 101 5 4 5 0 0
0 11 7 127 0 3 0 0
3 6 7 0 118 0 4 4
0 17 5 4 1 119 3 3
2 11 0 0 1 1 153 3
8 5 0 0 5 10 20 94
160
140
120
100
80
60
40
20
0

Résultats


Matrice de confusion - Random Forest
Vraie classe
Classe prédite
A+
A-
B+
B-
AB+
AB-
O+
O-
105 0 0 0 1 0 5 2
0 185 0 6 1 6 1 3
0 7 115 1 2 5 0 0
0 6 3 135 0 4 0 0
0 5 4 0 129 0 3 1
0 13 2 7 0 126 2 2
1 3 0 0 2 0 162 3
7 5 0 0 5 5 10 110
175
150
125
100
75
50
25
0

Résultats

Conclusions clés :

-SVM est clairement supérieur pour cette tâche
-Les paramètres optimaux sont simples (pas de surajustement détecté)
-Le modèle SVM a été sauvegardé pour déploiement

Les raisons principales qui font de ce modèle le meilleur sont :

1. Performance supérieure : Score de 90.42% en test, dépassant le Random Forest de 1.5% et le KNN de 8.6%

2. Stabilité : Faible écart entre les performances d'entraînement (90.08%) et de test (90.42%), indiquant une bonne généralisation

3. Simplicité : Configuration simple avec un noyau linéaire et un paramètre de régularisation C=0.1, évitant le surapprentissage"

Interface de prédiction

Conclusion

- Ce projet a démontré qu'il est possible de prédire le groupe sanguin simplement en analysant une empreinte digitale, avec une précision encourageante (environ 85-90%). Cette méthode présente 3 avantages majeurs :

- Simple : Pas besoin de prise de sang, juste un scan du doigt

- Rapide : Résultats en quelques secondes au lieu de 30 minutes

- Économique : Réduit considérablement le coût des tests

Applications pratiques :

- Dans les hôpitaux : Pour accélérer les transfusions en urgence

- Dans les zones reculées : Où les tests sanguins classiques sont impossibles

- Pour les dons de sang : Permet un pré-tri rapide des donneurs

Merci !

Pour votre attention