Identification de maladies de la rétine avec des Algorithmes de Machine Learning

Amine OUAKIB[1] *Groupe: amineoua[2]

[1] UDEM,Université de Montréal, Montréal, Canada
[2]DIRO,Département d'informatique et de recherche opérationnelle,UDEM,Université de Montréal, Montréal, Canada.

Introduction

Les maladies de la rétine, telles que la rétinopathie diabétique et la dégénérescence maculaire, sont des causes majeures de perte de vision. Leur détection précoce est cruciale, mais le diagnostic repose souvent sur l'analyse d'images complexes, ce qui peut être long et sujet à des erreurs. Les modèles de machine learning, notamment les réseaux de neurones profonds, peuvent automatiser cette analyse, offrant une détection précise et rapide des anomalies, comparable à celle des experts humains. Cette technologie permettrait de standardiser les diagnostics, de réduire les délais et d'élargir l'accès à des outils de dépistage, notamment dans les régions où les spécialistes sont peu nombreux. Elle pourrait ainsi transformer le diagnostic et le traitement des maladies de la rétine.

1. Méthodologie

1.1. Prétraitement et préparation des données

Le prétraitement des données est une étape essentielle dans tout projet d'apprentissage machine, particulièrement lorsque les données sont issues de sources complexes telles que des images. Dans ce projet, plusieurs techniques ont été appliquées afin de préparer les données d'images pour l'entraînement du modèle, tout en maximisant leur qualité et leur utilité.

1.1.1. Redimensionnement des images

Les images ont été redimensionnées pour correspondre à la forme attendue par les réseaux de neurones convolutifs (CNN), soit $(\text{batch\_size}, \text{channels}, \text{height}, \text{width})$. Dans ce cas, les images d'entrée ont une dimension de 28×28 pixels avec une seule couche (image en niveaux de gris). Le redimensionnement a été effectué avec la commande suivante :

```
X = X.reshape(-1, 1, image_height, image_width)
```

Cette étape permet de préparer les données pour être traitées par un CNN, en assurant que chaque image est correctement formée et prête à l'emploi.

1.1.2. Mélange aléatoire des données

Afin de réduire les biais potentiels dans l'entraînement du modèle, les données ont été mélangées aléatoirement. Cela garantit que le modèle ne se base pas sur l'ordre des données pendant l'entraînement, ce qui pourrait affecter la performance. Le mélange a été réalisé en utilisant la fonction `np.random.shuffle()` :

```
indices = np.arange(len(X))
np.random.shuffle(indices)
X, y = X[indices], y[indices]
```

1.1.3. Sous-échantillonnage

Afin d'équilibrer la distribution des classes, une méthode de sous-échantillonnage a été utilisée, consistant à réduire la taille de la classe majoritaire en supprimant certains échantillons. Cette approche permet d'obtenir un nombre équivalent d'exemples pour chaque classe. Étant donné que certaines classes étaient sur-représentées, le nombre minimal d'échantillons par classe a été calculé, et chaque classe a été sous-échantillonnée pour garantir une répartition équilibrée des classes.

```
min_samples_per_class = np.min(class_counts)
```

Pour chaque classe, les indices des échantillons ont été choisis aléatoirement afin de maintenir une distribution équilibrée dans le jeu de données final.

1.1.4. Normalisation des données

Les images ont été normalisées en divisant les valeurs des pixels par 255, afin de les ramener dans une plage de valeurs comprise entre 0 et 1. Cette normalisation est cruciale pour améliorer la stabilité et la rapidité de l'apprentissage du modèle, car elle permet aux algorithmes d'apprentissage automatique de mieux converger, en évitant que certaines caractéristiques dominent sur d'autres en raison de différences d'échelle.

La formule générale de la normalisation des données est la suivante :

$$x' = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

où :

*Corresponding author: amine.ouakib@umontreal.ca

- x_i est la valeur originale de la caractéristique,
- $\min(x)$ est la valeur minimale de la caractéristique dans l'ensemble de données,
- $\max(x)$ est la valeur maximale de la caractéristique dans l'ensemble de données,
- x' est la valeur normalisée.

Dans cette étude, les images ont été normalisées en appliquant la transformation suivante :

$$X = \frac{X}{255.0}$$

Cette opération permet de garantir que toutes les caractéristiques (c'est-à-dire les pixels des images) sont sur la même échelle. En effet, les pixels d'image, qui ont initialement des valeurs entre 0 et 255, sont réduits à une plage de valeurs comprises entre 0 et 1. Cela est essentiel pour le bon fonctionnement des modèles d'apprentissage machine, en assurant une convergence plus rapide et une performance optimale.

1.1.5. Augmentation des données

Pour améliorer la robustesse du modèle et éviter le sur-apprentissage, des techniques d'augmentation des données ont été appliquées. Ces techniques modifient légèrement les images d'entraînement afin de créer de nouvelles variantes tout en préservant les informations pertinentes. Les transformations suivantes ont été utilisées :

Rotation aléatoire : Rotation des images de manière aléatoire jusqu'à 15 degrés. Flip horizontal aléatoire : Inversion horizontale des images. Transformation affine aléatoire : Déplacement des images de manière aléatoire pour simuler des variations légères dans l'orientation ou la position des objets. Cela permet de générer une plus grande diversité d'exemples d'entraînement, ce qui peut améliorer la capacité du modèle à généraliser sur de nouvelles données.

Division de l'ensemble de données

La division des données est une étape cruciale dans les méthodologies de machine learning et de deep learning. Elle consiste à diviser le jeu de données en deux ensembles. Dans les tâches de classification, le jeu de données est divisé en un ensemble d'entraînement (80 pourcents des données) et un ensemble de test (les 20 pourcents restants). La performance du modèle et sa capacité à se généraliser à des données inconnues peuvent être évaluées grâce à cette division.

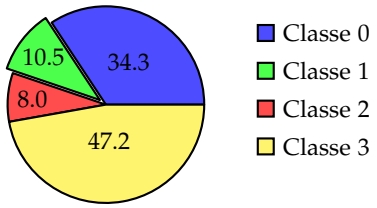

Figure 1: *Répartition des classes avant undersampling*

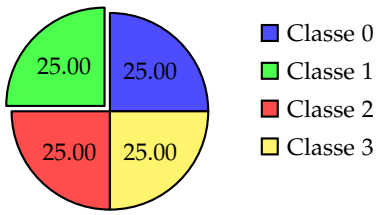

Figure 2: *Répartition des classes après undersampling*

1.1.5. validation croisée

La validation croisée est une méthode pour tester la précision d'un modèle de machine learning. Elle consiste à diviser les données en plusieurs parties : on entraîne le modèle sur certaines parties, puis on le teste sur les autres. Cela permet de mieux évaluer la performance du modèle en utilisant toutes les données disponibles pour l'entraînement et le test, mais de manière alternée.

1.1.6. Stop Early (ou arrêt précoce)

Stop Early (ou arrêt précoce) est une technique utilisée en apprentissage automatique et en apprentissage profond pour prévenir le surapprentissage (overfitting) lors de l'entraînement d'un modèle. L'idée est d'arrêter l'entraînement du modèle avant qu'il n'atteigne le nombre d'itérations ou d'époques prévu, dès que la performance du modèle commence à se dégrader sur des données de validation.

1.1.7. Sélection des hyperparamètres

Grid Search (ou Recherche par grille) est une méthode d'optimisation des hyperparamètres en apprentissage automatique, utilisée pour trouver la meilleure combinaison d'hyperparamètres pour un modèle donné. Elle consiste à définir un ensemble de valeurs possibles pour chaque hyperparamètre, puis à tester toutes les combinaisons de ces valeurs. À chaque combinaison, le modèle est évalué, généralement en utilisant la validation croisée, pour déterminer celle qui donne les meilleures performances.

2. Classification Multiclass des Images de Maladies de la Rétine

La classification multiclass des images de maladies de la rétine consiste à attribuer une étiquette parmi plusieurs classes prédéfinies à chaque image, représentant différentes pathologies. Ce problème peut être formalisé comme une tâche d'apprentissage supervisé où l'ensemble de données est constitué de n exemples (x_i, y_i), où $x_i \in \mathbb{R}^d$ est une

image représentée sous forme vectorielle dans un espace de dimension d, et $y_i \in \{1, 2, \ldots, C\}$ est l'étiquette associée correspondant à l'une des C classes.

La fonction de décision d'un classifieur f est définie comme suit :

$$f(x) = \arg \max_{c \in \{1,\ldots,C\}} P(y = c \mid x, \theta),$$

où $P(y = c \mid x, \theta)$ est la probabilité conditionnelle estimée par le modèle pour la classe c, paramétrée par θ.

2.1. Méthodes

Dans le cadre de ce projet, plusieurs algorithmes d'apprentissage ont été utilisés, notamment :

2.1.1. Random Forest

Un ensemble de N arbres de décision indépendants, où la prédiction finale est obtenue par vote majoritaire. La fonction de perte peut être exprimée comme une entropie croisée pour chaque arbre :

$$\mathcal{L} = -\sum_{i=1}^{n} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}),$$

où $y_{i,c}$ est l'étiquette réelle (one-hot encoded) et $\hat{y}_{i,c}$ est la probabilité prédite pour la classe c.

2.1.2. SVM Multiclass (Support Vector Machine)

Le SVM cherche à maximiser la marge entre les classes. La fonction objectif est donnée par :

$$\min_{\mathbf{w},\mathbf{b}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w}^T x_i + b)),$$

où $\mathbf{w}$ représente le vecteur de poids, b est le biais, et C est un hyperparamètre de régularisation.

2.1.3. Arbres de Décision

Basés sur la minimisation de l'impureté de Gini ou du gain d'information, ces modèles sont particulièrement interprétables. La fonction de Gini pour une division S est définie comme :

$$G(S) = 1 - \sum_{c=1}^{C} p_c^2,$$

où p_c est la proportion des données dans S appartenant à la classe c.

2.1.4. Réseaux de Neurones Profonds

Un réseau de neurones entièrement connecté ou convolutif peut être entraîné pour minimiser la perte d'entropie croisée. Dans le cadre de fine-tuning de ResNet50, les couches supérieures ont été remplacées pour s'adapter à C classes, et la rétropropagation a été utilisée pour ajuster les poids pré-entraînés :

$$\mathcal{L}_{\text{cross-entropy}} = -\sum_{i=1}^{n} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}),$$

où $\hat{y}_{i,c}$ est la sortie du réseau après application de la fonction softmax.

Ces méthodes ont été appliquées sur un ensemble d'images de rétine normalisées, transformées et segmentées. Le fine-tuning de ResNet50 a permis d'améliorer les performances grâce à son architecture pré-entraînée sur des millions d'images, exploitant ainsi des représentations hiérarchiques efficaces. La comparaison des modèles a été effectuée en utilisant des métriques comme la précision moyenne pondérée, le rappel et la matrice de confusion, pour analyser les erreurs entre classes.

2.2. Performance Measurement

L'analyse statistique, notamment l'exactitude, a été utilisée pour évaluer la performance des modèles développés pour classifier les pathologies rétiniennes. Cependant, la spécificité, la sensibilité, la précision et la matrice de confusion ont également été prises en compte pour affiner l'évaluation.

Accuracy : Mesure globale de la performance du modèle, calculée en divisant le nombre de prédictions correctes par le nombre total de prédictions. Elle est utile lorsque toutes les classes sont considérées avec la même importance.
Precision : Proportion de prédictions correctes parmi celles faites pour chaque classe.
Recall : Proportion d'exemples corrects d'une classe parmi tous les exemples réels de cette classe.
F1 score : Moyenne harmonique de la précision et du rappel, offrant une mesure équilibrée de la performance.

3. Phase I: Algorithme Random Forest avec Ajustement des Hyperparamètres

Dans cette phase de la compétition, j'ai utilisé un classifieur **Random Forest** pour effectuer la classification multiclasses des pathologies rétiniennes. Ce modèle repose sur une collection d'arbres de décision indépendants entraînés sur des sous-échantillons des données. La classification finale est déterminée par un vote majoritaire parmi ces arbres.

Table 1: *Performance du Random forest avec différentes valeurs pour les hyperparametres.*

n_estimators	max_depth	val_accuracy	test_accuracy
10	5	42,44	53,479
30	5	43.57	59.443
50	10	55.42	62.624

3.1. Justification du Choix de l'Algorithme

Le Random Forest est particulièrement efficace pour les problèmes où les relations entre les caractéristiques sont complexes et non linéaires. Contrairement à un arbre de décision unique, le Random Forest réduit le risque de surapprentissage en agrégeant les prédictions de plusieurs arbres, chacun entraîné sur un sous-échantillon différent des données. Cette approche améliore généralement la robustesse et la précision du modèle, en particulier sur des jeux de données déséquilibrés.

3.2. Modèle et Hyperparamètres

Le modèle de Random Forest est défini par deux principaux hyperparamètres :

- `n_estimators` : le nombre d'arbres dans la forêt.
- `max_depth` : la profondeur maximale des arbres de décision.

Dans ce projet, j'ai exploré différentes combinaisons d'hyperparamètres pour optimiser la performance du modèle :

- Pour `n_estimators`, j'ai testé des valeurs comme 30 et 50.
- Pour `max_depth`, j'ai expérimenté avec des profondeurs telles que 5 et 10.

Chaque fois que le modèle était entraîné, j'ai évalué ses performances à l'aide de la précision (*accuracy*). Voici un exemple de structure d'entraînement utilisée dans mon implémentation :

```
    rf = RandomForest(n_estimators=50, max_depth=10)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    accuracy = np.mean(y_pred == y_test)
    print(f"Accuracy: {accuracy * 100:.2f}%")
```

Listing 1: *Exemple d'entraînement avec Random Forest*

3.3. Comparaison Basée sur la Précision

Les performances du modèle ont été comparées en fonction des différentes valeurs d'hyperparamètres. J'ai observé que :

- Une augmentation de `n_estimators` améliore généralement la stabilité des prédictions, mais au prix d'un temps d'entraînement plus long.
- Une profondeur maximale modérée (comme 10) équilibre bien la complexité et la généralisation.

3.4. Conclusion

Grâce à l'ajustement des hyperparamètres, le modèle Random Forest a permis d'obtenir une précision optimale pour le problème de classification multi-classe. En combinant la robustesse d'un ensemble d'arbres de décision et un ajustement attentif des hyperparamètres, le modèle s'est avéré efficace pour capturer les relations complexes entre les caractéristiques tout en maintenant de bonnes performances sur les données.

4. Phase II : Sélection des hyperparametres et Évaluation des Algorithmes pour la Classification multi-classe

Dans cette phase, j'ai adopté une approche de recherche sur grille (*Grid Search*) pour identifier les meilleures configurations d'hyperparamètres pour chaque algorithme. Cette méthode systématique teste toutes les combinaisons possibles dans l'espace des hyperparamètres défini, garantissant ainsi une optimisation exhaustive.

4.1. Recherche d'hyperparamètres pour Random Forest

```
param_grid = {'n_estimators': [50, 100, 150],'max_depth': [20, 40, 80],'
    min_samples_split': [5, 10, 15],'min_samples_leaf': [4, 8, 12],'bootstrap': [True
    , False]}
```

Les hyperparamètres ont été soigneusement définis pour explorer différentes configurations du modèle Random Forest, notamment :

- **n_estimators** : le nombre d'arbres dans la forêt, testé avec 50, 100, et 150 arbres.
- **max_depth** : profondeur maximale des arbres, pour contrôler le surapprentissage.
- **min_samples_split** : nombre minimum d'échantillons requis pour diviser un nœud, avec des valeurs de 5, 10, et 15.
- **min_samples_leaf** : nombre minimal d'échantillons nécessaires dans une feuille terminale, exploré avec 4, 8, et 12.
- **bootstrap** : activation ou désactivation de l'échantillonnage avec remise.

Meilleurs hyperparamètres : 'bootstrap': False, 'max_depth': 80, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 150

4.2. Recherche d'hyperparamètres pour SVM

```
param_grid = {'C': [0.1, 1, 10],'kernel': ['rbf'],'gamma': ['scale', 'auto']}
```

Pour le modèle SVM, la grille de recherche incluait les paramètres suivants :

- **C** : le paramètre de régularisation, exploré avec les valeurs 0.1, 1 et 10.
- **kernel** : le noyau radial (*rbf*), connu pour ses performances avec des données non linéaires.
- **gamma** : le coefficient du noyau *rbf*, testé avec les valeurs *scale* et *auto*.

Meilleurs hyperparamètres : 'C': 10, 'gamma': 'scale', 'kernel': 'rbf'

4.3. Recherche d'hyperparamètres pour l'arbre de décision

```
param_grid = {'criterion': ['gini', 'entropy'],'max_depth': [None, 10, 20, 30],'
    min_samples_split': [2, 5, 10]}
```

Pour l'arbre de décision, l'objectif était d'évaluer l'impact des paramètres suivants :

- **criterion** : la méthode pour mesurer la qualité des splits, soit *gini* ou *entropy*.
- **max_depth** : la profondeur maximale des arbres, permettant de limiter leur complexité.
- **min_samples_split** : le seuil minimal pour diviser un nœud, testé avec les valeurs 2, 5 et 10.

Meilleurs hyperparamètres : 'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 2

4.4. Recherche d'hyperparamètres pour ResNet

```
num_epochs = 5
param_grid = {"learning_rate": [0.001, 0.0005, 0.0001],"weight_decay": [1e-4, 1e-5],"
    dropout": [0.3, 0.5],"batch_size": [32, 64]}
```

Les meilleurs hyperparamètres obtenus sont : batch_size à 64, dropout à 0.2, learning_rate à 0.001, et weight_decay à 1e-04. Par la suite, j'ai augmenté le nombre d'itérations (epochs) à 25 puis à 50. De plus, j'ai utilisé l'arrêt précoce (early stopping) afin d'éviter le surapprentissage (overfitting). Pour ResNet, j'ai exploré plusieurs combinaisons basées sur :

- **batch_size** : définit combien d'exemples de données seront utilisés dans chaque itération, testé avec 32 et 64.
- **lr** : le taux d'apprentissage, ajusté entre 0.001 et 0.0001.
- **weight_decay** : la régularisation pour réduire le surapprentissage.
- **dropout** : le taux d'abandon, évalué avec des valeurs de 0.3, 0.4, et 0.5.

la structure de mon modele est :

```
class CustomResNet50(nn.Module):
    def __init__(self, num_classes):
        super(CustomResNet50, self).__init__()
        self.model = resnet50(pretrained=True)
        self.model.conv1 = nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1, bias=
            False)
        self.model.fc = nn.Linear(self.model.fc.in_features, num_classes)
        self.dropout = nn.Dropout(0.2)
    def forward(self, x):
        x = self.model(x)  # Passage dans ResNet50
        x = self.dropout(x)  # Application du Dropout
        return x
```

Table 2: *Performance des modèles de machine learning utilisés pour prédire la classe des textes.*

Algorithms	Accuracy	Precision	Recall	F1-score
SVM	0.60	0.61	0.60	0.60
Decision Tree	0.46	0.47	0.46	0.46
Random Forest	0.63	0.64	0.63	0.64
Resnet50	0.80	0.81	0.80	0.80

4.5. Résultats et Conclusions

Les résultats montrent que ResNet50 surpasse les autres algorithmes avec une précision, un rappel et un F1-score de 0,80 ou plus, indiquant sa meilleure performance globale. Random Forest suit avec des scores autour de 0,63, surpassant légèrement le SVM (0,60) en précision et rappel. Decision Tree est le moins performant, avec des scores autour de 0,46, soulignant sa faible capacité à généraliser par rapport aux autres modèles.

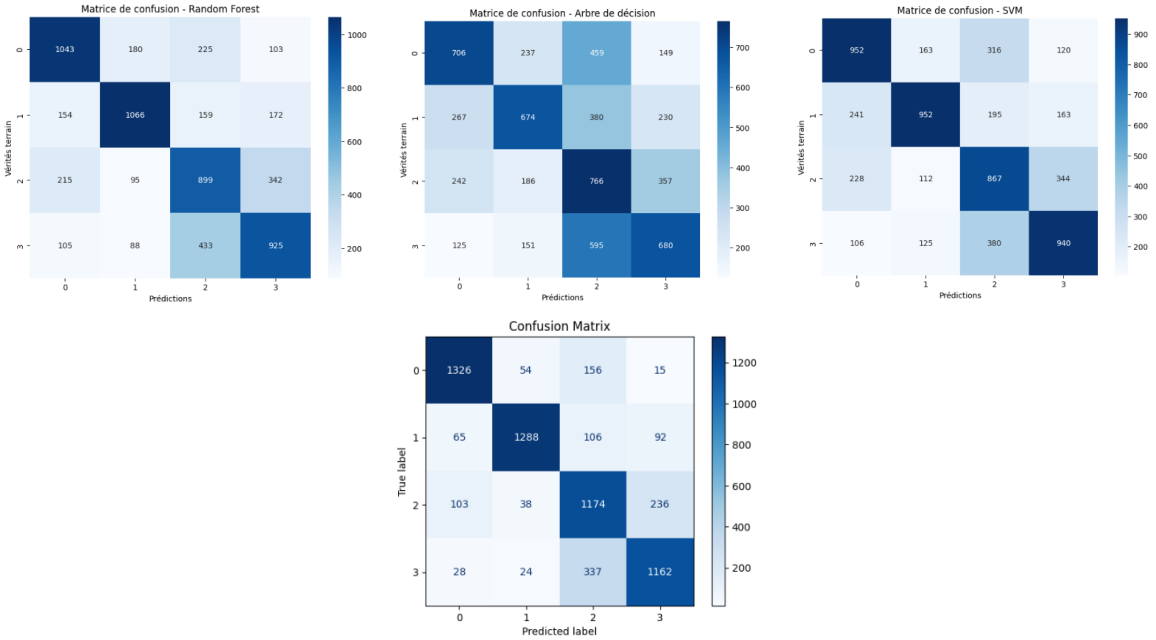

Figure 3: *Confusion Matrix for all models.*

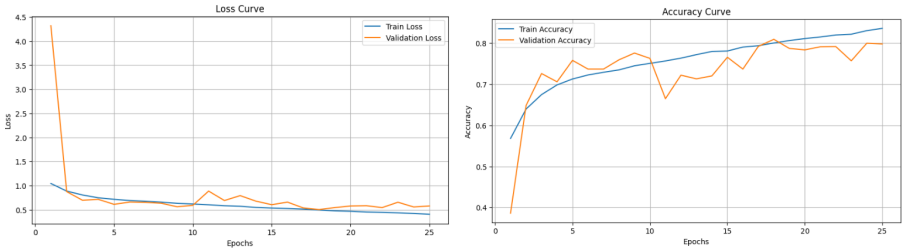

Figure 4: *Courbes d'entraînement et de validation ensemble.*

J'ai effectué plusieurs tests séparés pour évaluer le modèle ResNet50. Dans un premier test, j'ai utilisé l'early stopping pour prévenir le sur-apprentissage, tout en appliquant la validation croisée pour obtenir une évaluation plus fiable de la performance du modèle. Dans un autre test, j'ai ajusté différents hyperparamètres pour optimiser les résultats. Les résultats de chaque test, accompagnés de leurs figures et métriques, sont présentés dans le notebook.

Conclusion et Perspectives

Les modèles profonds, comme ResNet50, ont surpassé les méthodes traditionnelles pour l'identification de la rétine, offrant de meilleures performances globales malgré des limites dans les cas complexes. L'utilisation de réseaux plus avancés, l'enrichissement des données et des techniques comme l'augmentation ou le préentraînement pourraient améliorer la précision et la robustesse du système.

Références

- https://www.kaggle.com/code/muhammadfaizan65/retinal-disease-classification
- https://www.kaggle.com/code/ubaidshakil/eye-diseases