INF8245E COMPETITION - FALL 2023
REPORT TEAM LEKIP2

Antoine LEBLANC (2310186)
antoine.leblanc@polymtl.ca

Maxime MONTOIS (2306823)
maxime.montois@polymtl.ca

Hugo PETRILLI (2306643)
hugo-cedric.petrilli@polymtl.ca

1 INTRODUCTION

Ce projet, réalisé dans le cadre du cours INF8245E - Machine Learning à Polytechnique Montréal, consistait en une compétition Kaggle de classification d'images tenue du 26 octobre au 25 novembre 2023. Les fichiers `train_features.csv` et `train_labels.csv` représentaient respectivement les attributs et les labels des données d'entraînement, tandis que `test_features.csv` contenait les attributs des données de test. L'objectif était de former un modèle de Machine Learning sur les données d'entraînement pour prédire les labels des données de test. Ce rapport expose notre démarche et notre travail au cours de cette compétition.

2 EXPLORATION DES DONNÉES

2.1 DONNÉES D'ENTRAÎNEMENTS

La première phase de notre approche a consisté à examiner attentivement les données d'entraînement, qui étaient constituées de 10 000 exemples, chacun étant caractérisé par 1024 attributs. Dans un premier temps, nous avons visualisé les images afin de mieux appréhender la nature des éléments que nous classifiions. Le résultat obtenu est illustré dans la **Figure 1**.

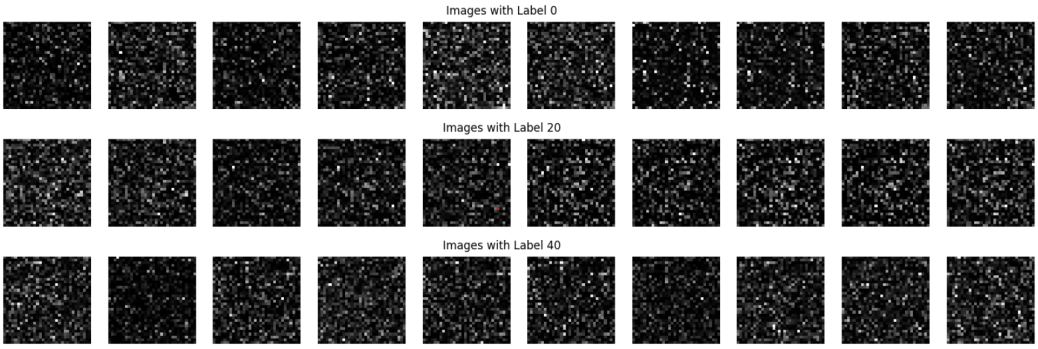

Figure 1: Visualisations des images

Nous n'avons pas réussi à extraire des informations directement par observation de ces images. A l'œil nu, aucune différence entre les différents labels n'est perceptible. Suite à quelques recherches Si-Yao et al. (2017) Russo (2003), nous avons emis l'hypothèse qu'elles avaient subi un traitement au bruit gaussien. Ce processus implique l'application de fluctuations aléatoires à l'intensité des pixels, compliquant ainsi la distinction visuelle entre les différents labels sur les images. Nous avons effectué divers traitements à l'aide de la bibliothèque `OpenCV` afin de créer de nouveaux ensembles de données.

Ensuite, nous avons essayé d'identifier des corrélations entre les attributs de nos données d'entraînement. En fixant un seuil arbitraire à 0.7, nous n'avons trouvé que trois attributs présentant

une corrélation significative parmi les 1024 disponibles. Étant donné que ce nombre est relativement faible et que la suppression de ces trois attributs n'a pas entraîné d'améliorations significatives des performances, nous avons choisi de les conserver.

Nous nous sommes ensuite, intéressés aux labels et à leur répartition. La **Figure 2** contient affiche la distribution des différentes classes.

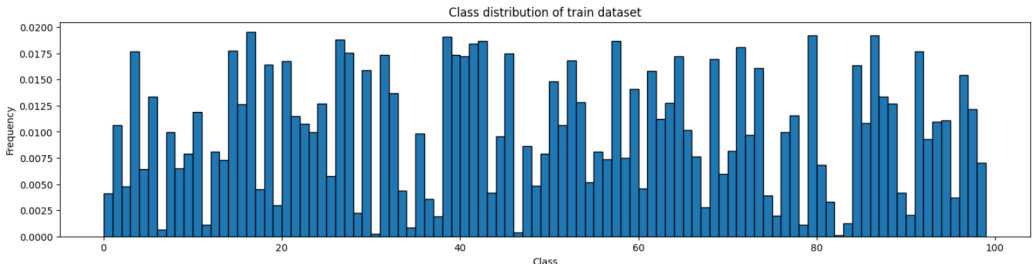

Figure 2: Répartition des labels d'entraînement

Nous avons été particulièrement frappés par la disparité de répartition entre les classes, certaines étant nettement sous-représentées, d'autres sur-représentées. Par exemple, la classe 82 est la moins représentée avec seulement 3 exemples sur les 25 000 des données d'entraînement. La classe 16 est la mieux représentée avec 98 exemples. Nous avons donc travaillé afin de rétablir l'équilibre entre ces différentes classes.

2.2 Comparaison des données d'entraînement et de test

Nous n'avons relevé aucune découverte significative concernant les données de test, qui se révèlent très similaires aux données d'entraînement. Néanmoins, tout de même chercher à les comparer. Nous avons initialement examiné les valeurs minimales, maximales, les variances, les moyennes, ainsi que les premier et troisième quartiles de nos attributs dans les deux ensembles de données, et nous avons constaté des courbes semblables. À titre d'exemple, la **Figure 3** compare la moyenne des attributs entre les données d'entraînements de tests.

Figure 3: Comparaison des moyennes des 100 premiers attributs entre données de test et d'entraînement

Face à l'absence de différences significatives, nous avons opté pour une seconde approche en sommant les valeurs de chaque attribut et en comparant les densités de répartition entre données de test et d'entraînement.

En traçant les résultats sur la **Figure 4**, nous avons constaté que les densités de répartition étaient pratiquement identiques tant pour les données de test que pour celles d'entraînement, ce qui signifie qu'aucune différence majeure n'a pu être relevée entre ces deux ensembles de données. Néanmoins, le graphique souligne que certaines caractéristiques présentaient des valeurs nettement divergentes par rapport aux autres.

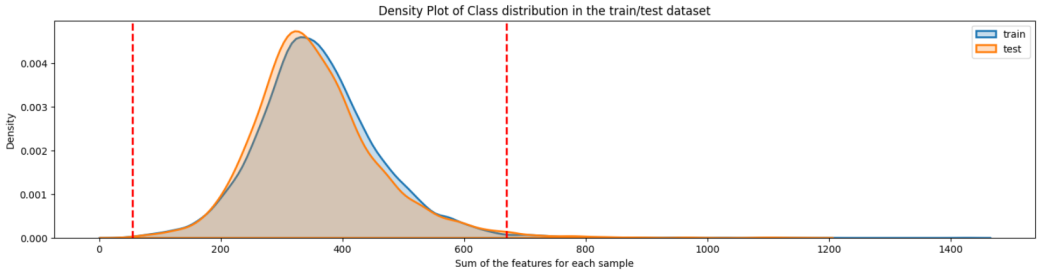

Figure 4: Répartition des sommes des valeurs des attributs entre données d'entraînement et de test

2.3 Pre-processing des données

Suite à l'exploration approfondie des données, plusieurs idées de prétraitement ont émergé :

1. **Atténuation du bruit gaussien** : Nous avons envisagé de supprimer le bruit gaussien des ensembles de données d'entraînement et de test avec la librairie `OpenCV`.
2. **Équilibrage des classes** : Afin d'obtenir une répartition plus équitable des classes, nous avons exploré l'utilisation de deux fonctions du module `imblearn`. La fonction `RandomOverSampler` vise à équilibrer les classes en augmentant le nombre d'exemples des classes sous-représentées, tandis que la fonction `RandomUnderSampler` vise à équilibrer les classes en réduisant le nombre d'exemples de la classe sur-représentée.
3. **Identification et suppression des outliers** : Nous avons examiné la possibilité de retirer les valeurs aberrantes, c'est-à-dire les attributs présentant des valeurs significativement supérieures aux autres lorsqu'on en fait la somme.

Par ailleurs, nous avons exploré des méthodes de prétraitement plus conventionnelles, notamment :

1. **Normalisation des données** : Nous avons appliqué la normalisation avec `StandardScaler` et `QuantileTransformers` du module `scikit-learn`.
2. **Analyse en composantes principales (PCA)** : Nous avons expérimenté la réduction de la dimensionnalité de nos données en utilisant l'algorithme `PCA` de `scikit-learn`. Des tentatives ont été faites pour réduire nos données à des dimensions arbitraires, notamment 300 et 400.

3 Algorithmes et Méthodologie

3.1 Méthodologie de travail

Afin de trouver le meilleur algorithme possible, nous commençons par définir les indicateurs utilisés pour mesurer nos algorithmes. Nous prenons l'accuracy et le "f1-score weigthed". Par la suite, nous constituons un ensemble de tests, avec 80% des données attribuées à l'échantillon de test et 20% à l'échantillon de validation. Cette démarche nous permet d'évaluer l'accuracy et le f-score de divers algorithmes de machine learning, en facilitant la sélection du meilleur candidat.

Nous procédons ensuite à l'optimisation de cet algorithme en ajustant ses paramètres et en évaluant l'impact de différentes méthodes de prétraitement des données. Après cela, nous utilisons Pytorch et essayons de construire un réseau de neurones que nous testons de la même manière que l'algorithme précédent. Enfin, nous entraînons les modèles finaux sur l'ensemble de la base de données train afin de prédire la base de données test et nous soumettons les résultats sur Kaggle pour évaluation.

3.2 Survol des différentes méthodes

Afin d'évaluer les performances des différents modèles, nous avons testé différents algorithmes sur un échantillon des données correspondant à 80% des données de test. Parmi les modèles examinés

figuraient les arbres de décision, le bagging, le random forest, le gradient boosting, l'adaboost, les k-nearest neighbours et le SVM. Étant donné que ces algorithmes comportent divers paramètres, nous avons essayer de les faire varier pour combattre l'overfitting et affiner les approximations.

En regardant leur accuracy et leur f-score sur les données de validation, nous avons remarqué que le SVM était l'algorithme le plus performant.

3.3 Optimisation de SVM

Étant donné que le SVM semblait être plus performant que les autres modèles, nous avons choisi de poursuivre avec ce dernier, en cherchant à l'optimiser davantage. Nous avons ainsi testé le modèle SVM en normalisant les données à l'aide de `Quantiletransformer` et en éliminant les valeurs aberrantes. Par la suite, notre attention s'est portée sur l'optimisation des hyper-paramètres. Nous avons donc fait varier quatre paramètres principaux : le paramètre de régularisation `C`, le type de noyau à utiliser dans l'algorithme `kernel`, le coefficient de noyau `gamma` et `decision_function_shape`. L'objectif est de maximiser le f1-score.

Nous avons commencé par une première recherche de paramètres avec `GridSearchCV` (**Table 1**)puis nous avons raffiner les plages de valeurs testées.

Paramètre	Valeurs
`C`	1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 40, 60, 80, 100
`kernel`	'linear', 'poly', 'rbf', 'sigmoid'
`gamma`	0.001, 0.01, 0.1, 1, 'scale', 'auto'
`decision_function_shape`	'ovo', 'ovr'

Table 1: Valeurs pour la première recherche de paramètres optimaux pour SVM.

Finalement, les paramètres sélectionnés sont réunis dans la Table 2. Le paramètre `kernel` avec la valeur `rbf` (fonction de base radiale) permet de transformer les données dans un espace de dimension supérieure. La fonction rbf permet au modèle SVM de gérer des relations complexes entre les caractéristiques.

C	kernel	gamma	decision_function_shape
5.6	'rbf'	'scale'	'ovo'

Table 2: Paramètres optimaux pour SVM

3.4 Réseaux de neurones

Le principal inconvénient de l'algorithme SMV est que le temps de prédiction des données est longue, rendant le travail et l'évaluation de cet algorithme particulièrement chronophages. Pour remédier à cette situation, nous avons initié la création d'un réseau de neurones simple à l'aide de PyTorch, formé selon la procédure suivante :

1. D'un couche linéaire prenant les 1024 caractéristiques et sortant 128 grandeurs.
2. Une couche d'activation `ReLu`.
3. D'un couche linéaire prenant 128 grandeurs et sortant 100 grandeurs correspondant au score de chaque classe.

Ensuite, nous avons procédé à l'entraînement du réseau sur 200 epochs en utilisant l'optimiseur Adam. La durée totale d'entraînement s'est établie à environ 30 secondes. Nous avons ensuite entrepris des expériences visant à faire varier le nombre de neurones et de couches. Cette exploration s'est déroulée de manière non délibérée, étant donné la vaste gamme de configurations possibles pour les réseaux neuronaux, et le manque de recul nous empêchant de cibler un type spécifique de réseau. Par conséquent, le réseau de neurones décrit précédemment a été maintenu.

4 RÉSULTATS

Les résultats principaux obtenues pour les algorithmes testés sont présenté dans la **Table 3**.

Model	**Accuracy Validation Set**	**F1-Score Validation Set**	**F1-Score Test Set (Kaggle)**
Decision Tree Classifier	75.0 %	74.8 %	53.2 %
Random Forest	92.1 %	91.8 %	-
K Nearest Neighbors ($K = 20$)	87.2 %	86.9 %	72.8 %
Gradient Boosting Classifier	45.3 %	36.0 %	-
AdaBoost Classifier	5.5 %	1.1 %	-
Bagging Classifier	80.2 %	79.4 %	-
SVM	**92.9 %**	**92.7 %**	**79.4 %**
SVM avec Quantile Transformer	94.2 %	94.0 %	77.8 %
SVM en supprimant les outliers	93.3 %	93.2 %	78.8 %
SVM avec paramètrres optimisés	94.1 %	94.0 %	79.0 %
Réseau de Neurones	93.2 %	93.0 %	78.0 %
Réseau de Neurones sans outliers	93.4 %	93.3 %	79.1 %
Réseau de Neurones avec PCA	91.5 %	91.3 %	-
Réseau de Neurones avec débruitage des données	92.4 %	92.1 %	-

Table 3: Résultats des différents algorithmes

5 DISCUSSION

Avec notre approche, nous avons réussi à développer des modèles performants sur l'ensemble d'entraînement et de validation. En sélectionnant les meilleurs modèles, en éliminant les valeurs aberrantes, en explorant la normalisation des données et en optimisant les paramètres, nous avons atteint des f-scores dépassant les 93% sur le set de validation. Cependant, ces performances ne se reflètent pas sur la base de données de test, où notre meilleur résultat atteint seulement 79.4%.

Il semble qu'il y ait une différence entre les données train et test que nous n'avons pas réussi à interpréter et à traiter. Il en résulte qu'en cherchant à trop optimiser nos modèles sur les données de validation, nous pouvions obtenir de moins bons résultats. Cela se voit puisque le modèle SVM sans avoir travaillé les données est meilleur sur l'espace de test que les réseaux de neurones ou que SVM optimisé alors qu'il est moins bon sur les données de validation.

Pour améliorer notre modèle, nous pouvons nous intéresser aux différences entre les données test et train. En réduisant la différence entre ces deux bases de données, nous pourrions améliorer grandement nos résultats puisque nos modèles sont bien meilleurs sur l'ensemble de validation que sur les données tests.

Pendant la compétition, nous n'avons pas eu le temps d'essayer de mettre sur Kaggle l'idée de modifier le jeu de données d'entraînement et de test, malgré la sur-représentation de certains labels et la sous-représentation d'autres, comme illustré à la **Figure 2**. Travaillant sur l'ensemble du jeu de données, il est probable que le modèle n'ait pas été suffisamment entraîné sur les classes peu représentées. Bien que nous aurions pu améliorer cela en ajustant les données d'entraînement, la contrainte de soumissions limitées sur Kaggle rendait cette approche difficile à mettre en œuvre.

6 ÉNONCÉ DES CONTRIBUTIONS

Durant ce projet, nous avons effectué une grande partie du travail et des discussions sur les idées à avoir ensemble. Nous nous étions fixé un point hebdomadaire après le cours de Machine Learning. Nous avons tous les trois rédigé ce rapport. Les différentes tâches réalisées par chacun sont détaillés dans la **Table 4**.

Hugo	**Antoine**	**Maxime**
Mise en place d'un premier modèle faisant un traitement des données rapide.	Transformation des données avec des réductions de dimensions.	Interprétation et recherche à donner un sens aux données.
Séparation des données en train et validation.	Équilibrage de la répartition des classes dans le training set.	Optimisation de SVC sur différents paramètres.
Tests de différents modèles en variant différents paramètres pour éviter l'overfitting.	Mise en place de KNN et optimisation de quelques-uns de ses paramètres.	Construction d'un réseau de neurones simple et étude de différentes architectures.
Code de création de `csv` avec la solution demandée.	Mise en place d'un réseau de neurones convolutifs.	Variation des données d'entrées dans le réseau.

Table 4: Énoncé des contributions.

We hereby state that all the work presented in this report is that of the authors.

REFERENCES

F. Russo. A method for estimation and filtering of gaussian noise in images. *IEEE Transactions on Instrumentation and Measurement*, 52(4):1148–1154, 2003. doi: 10.1109/TIM.2003.815989.

Li Si-Yao, Qian Yin, and Ping Guo. Understanding and eliminating the large-kernel effect in blind deconvolution. *CoRR*, abs/1706.01797, 2017. URL `http://arxiv.org/abs/1706.01797`.