Rapport TP 5 - Apprentissage Automatique

Noura FAIZ **Nouh CHELGHAM**

Université Toulouse III Paul Sabatier IAFA

Noura.Faiz@univ-tlse3.fr
Nouh.Chelgham@univ-tlse3.fr

Résumé

Cette étude compare les performances de deux architectures de réseaux de neurones — le perceptron multicouche (MLP) et le réseau de neurones convolutif (CNN) — pour une tâche de classification sonore. L'objectif est de guider le choix de l'architecture pour des applications en classification audio. Après avoir optimisé les architectures traditionnelles et expérimenté avec divers paramètres d'entraînement et optimiseurs, nous avons analysé les avantages et les inconvénients de chaque modèle. La comparaison de leur efficacité est basée sur la précision obtenue lors des tests. Nos résultats indiquent que le CNN est supérieur au MLP dans ce contexte.

Mots Clef

Apprentissage Automatique , MLP , CNN , réseau-neuronal

1 Introduction

Au fil des années, la classification automatique de données a connu des avancées significatives. Bien que l'enseigner à un ordinateur demeure un défi de taille, les réseaux de neurones se sont révélés être des outils efficaces pour cette tâche. Dans notre étude, nous nous sommes concentrés sur deux architectures spécifiques : le MLP (perceptron multicouche) et le CNN (réseau de neurones à convolution). Le MLP est largement utilisé pour traiter des données structurées, grâce à ses multiples couches de neurones interconnectées, tandis que les CNN, initialement développés pour la classification d'images, se distinguent par leur capacité à extraire des caractéristiques des données d'entrée via des opérations de convolution.[2]
Notre objectif principal était de déterminer laquelle de ces deux architectures est la plus performante pour la classification sonore. Pour ce faire, nous avons effectué une comparaison directe en les évaluant sur le même jeu de données. Ce jeu de données nous a permis de mesurer leur efficacité respective dans la classification d'événements sonores en dix catégories différentes. En analysant les performances des deux modèles sur cet ensemble de données commun, nous avons pu tirer des conclusions quant à leur aptitude à traiter efficacement les tâches de classification sonore.

2 Le Jeu de Données

Nous avons exploré différents jeux de données pour notre recherche sur la classification sonore, notamment ESC-10, ESC-50 et UrbanSound8K (US8K). Cependant, étant donné que notre tâche implique de classer les sons en seulement 10 catégories, On peut conclure que notre jeu de données est structuré selon ESC-10 (Environmental Sound Classification). Pour extraire des caractéristiques de ces extraits sonores, nous les avons convertis en spectrogrammes, une représentation graphique de la fréquence du signal sonore en fonction du temps. Ces spectrogrammes, de dimensions 216 par 128, ont ensuite été utilisés comme entrées pour les deux architectures de réseaux neuronaux (MLP et CNN) que nous avons comparées. Les modèles ont été entraînés pour prédire la classe d'un événement sonore donné en fonction de ces caractéristiques.[1]

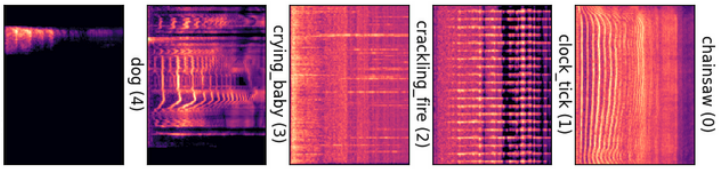

FIGURE 1 – Exemple de spectogrammes

3 Les Modèles

Dans cette section, nous avons examiné en détail deux architectures clés pour notre tâche de classification sonore : le perceptron multicouche (MLP) et le réseau de neurones à convolution (CNN). Nous avons commencé par décrire la structure de base de chaque architecture, en mettant en évidence leurs principaux composants et leur fonctionnement général. Ensuite, nous avons discuté des modifications spécifiques que nous avons apportées à chaque architecture pour les adapter à notre problème de classification sonore

3.1 Perceptron Multi-Couche MLP

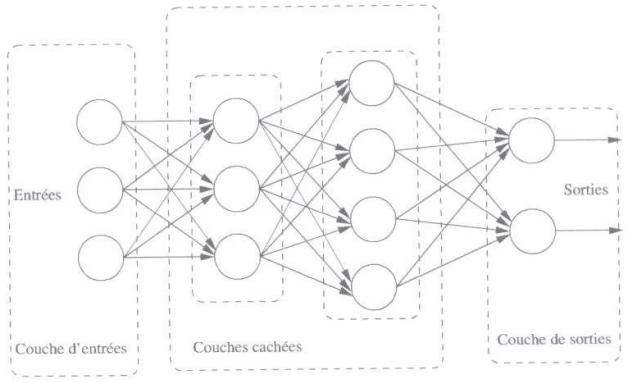

FIGURE 2 – Strcuture d'un MLP

Un Perceptron Multi-Couches (MLP), également connu sous le nom de réseau neuronal à propagation avant, est structuré en plusieurs couches. Chaque couche est reliée à la suivante, et chaque neurone dans une couche est connecté à tous les neurones de la couche suivante. Il comprend au moins une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Les MLP sont utilisés pour des tâches telles que la classification, la régression et d'autres types de modélisation de données complexes.

1ére Version du model MLP :
Nous avons développé un modèle comportant deux couches linéaires utilisant la fonction d'activation ReLU. Les paramètres clés que nous avons configurés sont num-hidden, input-size et num-classes. Par défaut, num-hidden est initialisé à 50, input-size correspond à la dimension des données d'entrée (128 x 216), et num-classes représente le nombre de classes en sortie, fixé à 10 dans ce cas.
Ce modèle implémente une fonction forward qui prend en entrée un tenseur de dimensions (batch-size, 128, 216) et renvoie un tenseur de dimensions (batch-size, num-classes), assignant à chaque entrée les scores associés à chaque classe.
Dans la première couche linéaire, nous devons prédire input-sizenum-hidden paramètres, tandis que la deuxième couche en nécessite num-hiddennum-classes. Malgré cela, les performances du MLP de base ne se sont pas révélées satisfaisantes. Par conséquent, nous avons entrepris d'améliorer son architecture dans l'espoir d'optimiser ses résultats.

2éme Version du model MLP :
Pour rechercher la version optimale de notre MLP, nous avons procédé à des expérimentations en modifiant plusieurs aspects, notamment le nombre de couches cachées, le nombre de neurones dans ces couches, le taux d'apprentissage, le nombre d'époques, ainsi que l'optimiseur utilisé.
Notre meilleure version améliorée du MLP se compose désormais de trois couches linéaires avec la fonction d'activation ReLU, complétées par deux couches de Normalisation par lots.
Le modèle est composé de trois couches linéaires (fully connected) intercalées de fonctions d'activation et de couches de normalisation par lots (Batch Normalization) pour stabiliser l'apprentissage et accélérer la convergence du modèle. La première couche linéaire, self.fc1, prend une entrée de taille 128*216 et produit une sortie de taille 256, suivie d'une fonction d'activation LeakyReLU et d'une normalisation par lots. Ensuite, la deuxième couche linéaire, self.fc2, réduit la dimensionnalité de la représentation à 128, suivie des mêmes opérations de LeakyReLU et de normalisation par lots. Enfin, la dernière couche linéaire, self.fc3, produit une sortie de taille 10, correspondant au nombre de classes dans le problème de classification. Le modèle est entraîné à l'aide de l'algorithme de rétropropagation du gradient pour ajuster les paramètres du réseau afin de minimiser la perte lors de la classification des données d'entrée. Le modèle a un total de 7,113,098 paramètres entraînables et effectue environ 7.11 millions de multiplications-additions lors de son fonctionnement.

3.2 Réseaux de Neuronnes Convolutif CNN

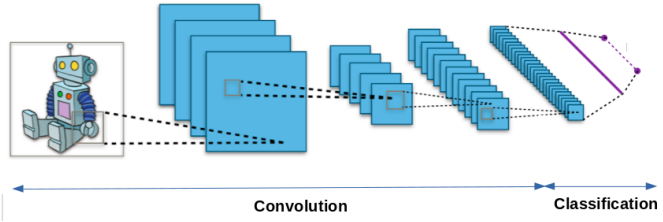

FIGURE 3 – Strcuture d'un CNN

Un réseau de neurones convolutif (CNN) est une architecture de réseau neuronal qui utilise des couches de convolution pour extraire des caractéristiques significatives des données en appliquant des filtres locaux, suivies de couches de pooling pour réduire la dimensionnalité. Typiquement, un CNN est composé de couches de convolution, de couches de pooling, de couches entièrement connectées et de couches de sortie. Cette structure en cascade permet au CNN d'apprendre des hiérarchies de caractéristiques à différentes échelles, en faisant des CNN l'architecture dominante pour des tâches telles que la classification d'images, la détection d'objets et la segmentation sémantique en vision par ordinateur.

1ére Version du model CNN : Le fragment de code en annexe illustre la construction d'un réseau de neurones à convolution spécialement conçu pour la classification. Cette architecture comprend trois niveaux de convolution.

Chaque niveau emploie des filtres de dimensions 3×3, avec une augmentation progressive du nombre de filtres de 8, à 16, puis à 32 pour chaque couche successive. Chaque étage convolutif est directement suivi par une opération de réduction dimensionnelle via une couche de pooling (`MaxPool2d`), qui utilise des fenêtres de 2×2 pour simplifier les sorties des couches de convolution.

Ces données traitées sont ensuite converties en un vecteur plat, puis dirigées vers un ensemble de couches densément connectées. La première couche de ce segment comporte 50 neurones, tandis que la seconde, conçue pour la sortie finale, contient 10 neurones, représentant les différentes classes possibles.

Paramétrage du Modèle

- **Taille des noyaux de convolution :** Tous les noyaux ont une dimension de 3×3.
- **Pooling :** Utilisation de `MaxPool2d` avec des fenêtres de 2×2 pour le sous-échantillonnage spatial après chaque convolution.
- **Caractéristiques de sortie :** $32 \times 30 \times 52$, indiquant le nombre de caractéristiques générées par la dernière couche de convolution. Cette quantité détermine les dimensions de la première couche entièrement connectée.
- **Couches entièrement connectées :** La première couche dispose de 50 neurones, suivie par une couche de 10 neurones correspondant aux classes cibles.

2éme Version du model CNN : Le code présenté en annexe détaille la mise en place d'une version avancée d'un réseau de neurones à convolution, destiné à la classification avancée. Cette architecture se distingue par l'intégration de batch normalization à chaque couche convolutive, améliorant ainsi la stabilité et la performance du réseau. Elle est composée de trois couches de convolution utilisant des filtres progressivement plus denses, avec des tailles de 16, 32 et 64. Chaque couche de convolution est suivie d'une normalisation et d'une couche de pooling (`MaxPool2d`) pour réduire la dimensionnalité spatiale des données.

Les données ainsi traitées sont aplaties et ensuite traitées par une couche dense équipée de 256 neurones, intégrant une normalisation et un dropout de 50% pour éviter le surapprentissage. La sortie passe par une dernière couche dense de 10 neurones, correspondant aux classes cibles du problème.

Paramétrage du Modèle

- **Taille des noyaux de convolution :** Utilisation de noyaux 3×3 pour toutes les couches convolutives.
- **Batch Normalization :** Appliquée après chaque couche convolutive pour normaliser les activations.
- **Pooling :** Utilisation de `MaxPool2d` avec des fenêtres de 2×2 pour réduire la taille des caractéristiques spatiales après chaque groupe de convolution et normalisation.
- **Caractéristiques de sortie :** $64 \times 30 \times 52$, déterminant le nombre de caractéristiques entrant dans la première couche entièrement connectée.
- **Couches entièrement connectées :** Une première couche dense de 256 neurones avec dropout de 50%, suivie par une couche de 10 neurones pour la classification finale.

4 Résultats

Suite à l'entraînement de nos modèles avec diverses valeurs de learning rate et d'époques, ainsi qu'à des tests utilisant différents optimiseurs tels qu'Adam et SGD, nous avons constaté une diversité de résultats. Nos meilleurs MLP atteignent approximativement une précision de 0,6 sur l'ensemble de test, tandis que nos meilleurs CNN affichent une précision d'environ 0,7-0.8. Dans le cas des deux modèles améliorés, l'optimiseur Adam semble offrir les performances les plus efficaces.

Pour notre modèle MLP amélioré, nous avons un total de 7,113,098 paramètres avec une perte (loss) de 0,08. Ce modèle a atteint une précision maximale de 0,88 lors de la phase d'entraînement, mais affiche une précision de 0,62 lors de la phase de test.(Learning Rate : 0.001, Num Epochs : 30, optimizeur : 'adam', Test Accuracy : 0.62).

Tandis que notre modèle CNN amélioré a un total de 25,585,898 paramètres avec une loss de 0.07. le modèle arrive à toucher une précision maximale de 1 lors de la phase d'entrainementet et obtient une précision de 0.84 lors de la phase de test .(Learning Rate : 0.001, Num Epochs : 20, optimizeur : 'adam', Test Accuracy : 0.84).

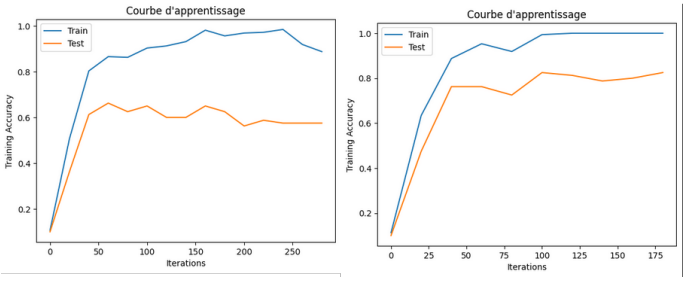

FIGURE 4 – Courbe d'apprentissage MLP / CNN

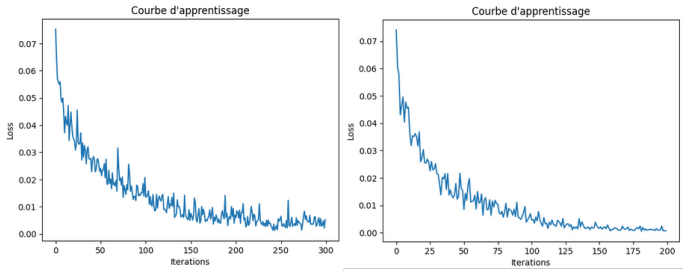

FIGURE 5 – Courbe de loss MLP / CNN

Les différentes configurations évaluées nous ont conduit à la conclusion que le CNN est le choix optimal pour la tâche de classification audio étudiée. Cependant, pour

mieux comprendre les performances relatives des deux architectures, nous avons examiné les matrices de confusion générées par le MLP et le CNN. Ces matrices nous per-

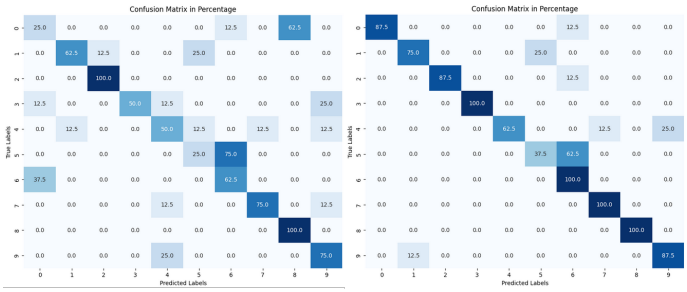

FIGURE 6 – Matrices de confusion MLP / CNN

mettent d'observer la proportion de prédictions correctes par rapport aux prédictions incorrectes pour chaque classe. L'analyse des matrices de confusion révèle des différences significatives dans la capacité des deux modèles à classifier les sons. En effet, certaines prédictions incorrectes effectuées par le MLP sont correctement identifiées par le CNN. Cette observation suggère que le CNN est capable de mieux discriminer entre les différentes classes de sons, même lorsque les spectogrammes d'entrée présentent des similitudes apparentes. En examinant de manière plus détaillée les prédictions incorrectes, nous constatons que le CNN parvient à capturer des caractéristiques subtiles dans les données audio qui échappent au MLP. Ces caractéristiques peuvent être cruciales pour la classification précise des sons, notamment lorsque les classes sont proches ou partagent des similitudes acoustiques. En fin de compte, la matrice de confusion du CNN présente une capacité de classification supérieure, reflétant sa capacité à mieux discriminer entre les différentes classes de sons par rapport à celle du MLP. Ces résultats renforcent notre conclusion initiale selon laquelle le CNN est le choix optimal pour cette tâche de classification audio

5 Conclusion

Pour conclure, dans le contexte de la classification audio que nous avons examinée, il semble que les réseaux de neurones convolutifs (CNN) soient plus efficaces que les perceptrons multi-couches (MLP). Lorsque nous comparons les performances des CNN avec celles des MLP, nous observons généralement que les CNN obtiennent des résultats supérieurs en termes de précision du test, de perte (loss) et de matrice de confusion. De plus, cette supériorité est souvent réalisée avec un nombre de paramètres plus restreint, ce qui indique une meilleure efficacité dans l'apprentissage et la généralisation des caractéristiques audio pertinentes. En effet, les CNN sont particulièrement bien adaptés à la reconnaissance de motifs dans les données spatiales telles que les images et peuvent être adaptés de manière similaire à l'analyse de signaux temporels comme ceux trouvés dans les données audio. Ainsi, leur capacité à capturer les relations locales et à hiérarchiser les caractéristiques semble offrir un avantage significatif dans la classification précise des sons, par rapport aux structures plus simples des MLP.[2] Par ailleurs, nos investigations nous ont menés à conclure que bien que les CNN surclassent les MLP dans notre cas, ces derniers sont conçus pour traiter des images 2D classiques, capturant à la fois les informations spatiales et temporelles. En revanche, les images spectrogrammes représentent des données séquentielles, avec une dimension temporelle, ce qui les rend distinctes. Ainsi, pour mieux appréhender ces données séquentielles, l'utilisation de réseaux de neurones récurrents (RNN) est privilégiée, voire leur combinaison avec des CNN. Les RNN sont spécifiquement adaptés à la modélisation de séquences, permettant de saisir les dépendances temporelles dans les données audio. L'association d'un RNN et d'un CNN offre une représentation plus riche des caractéristiques temporelles et spatiales, potentiellement améliorant significativement les performances de classification.[3]

Annexe

```
[ ] # Perceptron multi-couche
    class MLP(nn.Module):
        def __init__(self, num_hidden=50, input_dim=height*width, num_targets=10):
            super(MLP, self).__init__()
            self.layer1 = nn.Linear(input_dim, num_hidden)
            self.layer2 = nn.Linear(num_hidden, num_targets)
            # vous pouvez définir d'autres couches dans un deuxième temps

        def forward(self, spectro):
            flattened = spectro.view(-1, height*width) # flatten le spectro
            out = self.layer1(flattened)
            out = torch.relu(out)
            out = self.layer2(out)
            return out
```

FIGURE 7 – Code MLP de base

```
class ImprovedMLP(nn.Module):
    def __init__(self):
        super(ImprovedMLP, self).__init__()
        self.fc1 = nn.Linear(128*216, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.relu = nn.LeakyReLU()
        self.bn1 = nn.BatchNorm1d(256)
        self.bn2 = nn.BatchNorm1d(128)

    def forward(self, x):
        x = x.view(-1, 128*216)
        x = self.relu(self.bn1(self.fc1(x)))
        x = self.relu(self.bn2(self.fc2(x)))
        x = self.fc3(x)
        return x

# Création de l'instance du modèle
Improve_model = ImprovedMLP().to(device)

# Affichage du modèle pour vérifier la structure
print(Improve_model)
```

FIGURE 8 – Code MLP aprés Améliorations

```
[ ]  class CNN(nn.Module):
        def __init__(self):
            super(CNN, self).__init__()
            self.conv1 = nn.Conv2d(1, 8, kernel_size=3)
            self.conv2 = nn.Conv2d(8, 16, kernel_size=3)
            self.conv3 = nn.Conv2d(16, 32, kernel_size=3)
            self.pool = nn.MaxPool2d(2, 2)

            self.fc1 = nn.Linear(32*30*52, 50)
            self.fc2 = nn.Linear(50, 10)

        def forward(self, x):
            x = torch.relu(self.conv1(x))
            x = self.pool(torch.relu(self.conv2(x)))
            x = self.pool(torch.relu(self.conv3(x)))

            x = x.view(-1, 32*30*52) # flatten
            x = torch.relu(self.fc1(x))
            x = self.fc2(x)

            return x
```

FIGURE 9 – Code CNN de base

```
[ ]  class ModifiedCNN(nn.Module):
        def __init__(self):
            super( ModifiedCNN, self).__init__()
            self.conv1 = nn.Conv2d(1, 16, kernel_size=3)
            self.bn1 = nn.BatchNorm2d(16)
            self.conv2 = nn.Conv2d(16, 32, kernel_size=3)
            self.bn2 = nn.BatchNorm2d(32)
            self.conv3 = nn.Conv2d(32, 64, kernel_size=3)
            self.bn3 = nn.BatchNorm2d(64)
            self.pool = nn.MaxPool2d(2, 2)

            self.fc1 = nn.Linear(64*30*52, 256)
            self.bn4 = nn.BatchNorm1d(256)
            self.dropout = nn.Dropout(0.5)
            self.fc2 = nn.Linear(256, 10)

        def forward(self, x):
            x = torch.relu(self.bn1(self.conv1(x)))
            x = self.pool(torch.relu(self.bn2(self.conv2(x))))
            x = self.pool(torch.relu(self.bn3(self.conv3(x))))

            x = x.view(-1, 64*30*52) # flatten
            x = self.dropout(torch.relu(self.bn4(self.fc1(x))))
            x = self.fc2(x)

            return x
```

FIGURE 10 – Code CNN aprés Améliorations

Références

[1] Zohaib Mushtaq , Shun-Feng Su, Quoc-Viet Tran , Spectral images based environmental sound classification using CNN with meaningful data augmentation

[2] Minkyu Lim1, Donghyun Lee1, Hosung Park1, Yoseb Kang1, Junseok Oh1, Jeong-Sik Park2, Gil-Jin Jang3, and Ji-Hwan Kim ,Convolutional Neural Network based Audio Event Classification

[3] Kamalesh Palanisamy†, Dipika Singhania† and Angela Yao , Rethinking CNN Models for Audio Classification