Compte rendu TP 5 - Apprentissage Automatique

Chloé COLOMBEL[1] Khalil MAACHOU[2]

[1] Université Paul Sabatier - IMA
[2] Université Paul Sabatier - IAFA

chloe.colombel@univ-tlse3.fr
khalil.maachou@univ-tlse3.fr

Résumé

Dans cette étude comparative, nous avons évalué les performances de deux architectures de réseau de neurones, le perceptron multicouche (MLP) et le réseau de neurones à convolution (CNN), pour une tâche de classification sonore. L'objectif de cette étude est d'orienter le choix d'architecture de réseau de neurones pour les tâches de classification audio. Ainsi, après avoir amélioré les architectures classiques et testé différents paramètres d'entraînement et optimiseurs, nous avons regardé les avantages et les inconvénients de chaque architecture et évalué leur pertinence en comparant leurs résultats en terme de précision de test. Nous avons conclu que dans notre cas le CNN surpasse le MLP.

Mots Clef

MLP, CNN, Apprentissage Automatique, Classification sonore, Réseaux de neurones, ESC-10

1 Introduction

La classification automatique de données est un domaine de recherche qui a connu des progrès significatifs au cours des dernières années. Bien que ce soit une tâche difficile à enseigner à un ordinateur, il existe des réseaux de neurones capables de réaliser cette tâche. Nous allons nous intéresser à deux architectures de réseaux de neurones en particulier, à savoir le MLP et le CNN [1] [4].

Le MLP est une architecture de réseau de neurones, utilisée pour traiter des données structurées, en utilisant plusieurs couches de neurones connectées les unes aux autres [1]. D'un autre côté, les CNN ont été initialement développés pour la classification d'images. Ils sont basés sur des opérations de convolution qui permettent d'extraire des caractéristiques des données d'entrée [3].

Nous pouvons donc nous demander laquelle de ces deux architectures est la plus performante pour une tâche de classification sonore. Dans le but de répondre à cette question, nous avons comparé ces architectures sur un même jeu de données nommé ESC-10 (Environmental Sound Classification).

2 Description et extraction des données

Le jeu de données ESC-10 (Environmental Sound Classification) est un ensemble de données conçu pour la classification d'événements sonores en 10 classes différentes. Il est composé de 400 extraits sonores, chacun ayant une durée de 5 secondes et une fréquence de 44,1 kHz [5].

Pour extraire des caractéristiques de ces extraits sonores, nous avons converti chaque extrait sous la forme d'un spectrogramme (figure 1), qui est une représentation graphique de la fréquence du signal sonore en fonction du temps. Ces spectrogrammes, de dimensions 216 par 128, ont été utilisés comme entrées pour les deux architectures de réseaux de neurones (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.

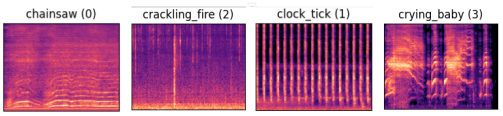

FIGURE 1 – Affichage de quelques spectrogrammes

3 Architectures

Dans cette partie nous avons décris les deux architectures que nous avons choisies pour effectuer notre tâche de classification, à savoir le MLP et le CNN. Nous avons présenté leur architecture de base ainsi que les modifications que nous y avons apportées.

3.1 MLP

Le multi-layer perceptron, MLP, est un type de réseau de neurones composé de plusieurs couches. Chaque couche est connectée à la suivante, et chaque neurone de la couche est connecté à tous les neurones de la couche suivante [1][4].

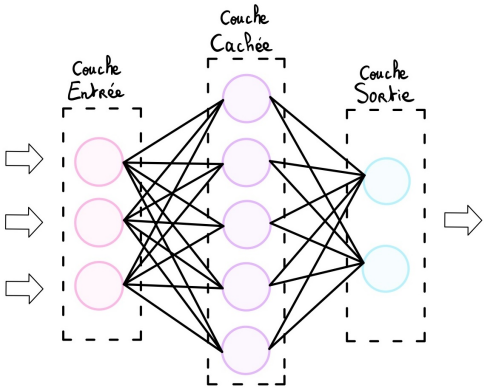

FIGURE 2 – Schéma d'un MLP

Modèle de base. Nous avons implémenté un modèle contenant deux couches linéaires avec une fonction d'activation ReLu. Les hyperparamètres que nous avons utilisé sont num-hidden, input-size et num-classes. Le num-hidden est par défaut initialisé à 50, input-size représente la taille des données en entrée (128 x 216) et le num-classes représente le nombre de classes en sortie qui est ici de 10. Ce modèle utilise une fonction passe-avant forward qui prend en paramètre un tensor de (batch-size, 128, 216), et retourne un tensor de (batch-size, num-classes) qui associe à chaque entrée les scores obtenues pour chaque classe. Pour la première couche linéaire nous avons input-size*num-hidden paramètres à prédire, et num-hidden*num-classes paramètres pour la deuxième couche (cf annexe - figure 8). Les résultats ont montré que le MLP de base ne semble pas performant, nous avons donc amélioré son architecture pour tenter d'améliorer ses performances.

Modèle amélioré. Pour trouver la meilleure version de notre MLP nous avons donc testé différentes configurations, notamment en changeant le nombre de couches cachées, le nombre de neurones dans les couches cachées, le taux d'apprentissage, le nombre d'époques ou encore l'optimiseur. Notre meilleure version améliorée du MLP est composée de trois couches linéaires (Fully-Connected) avec comme fonction d'activation la ReLu ainsi que de deux couches de Batch Normalisation. Nous avons obtenu les meilleurs résultats de test en utilisant l'optimiseur Adam avec un learning rate de 0.001 et 20 époques (cf annexe - figure 9).

La composition des couches est la suivante :

- 128*216 : nombre de neurones de la couche en entrée
- : 256 : nombre de neurones de la première couche cachée
- 128 : nombre de neurones de la deuxième couche linéaire
- 10 : nombre de neurones de la couche de sortie correspondant au nombre de classes

3.2 CNN

Un réseau de neurones à convolution, CNN, est un type de réseau de neurones composé de plusieurs couches et utilisant des couches de convolution. Il s'agit d'un des réseaux de neurones les plus utilisé pour faire de la classification. Les couches de convolutions sont utilisées pour extraire les caractéristiques d'une image en appliquant des filtres de convolutions. Ces données sont ensuite réduites grâce à des couches de pooling puis transmises à des couches dites fully-connected pour faire la classification [4].

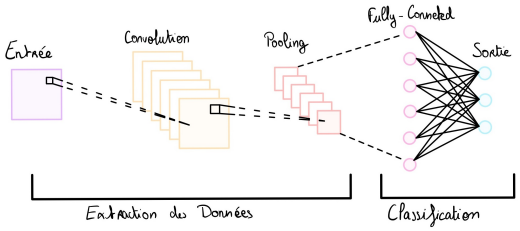

FIGURE 3 – Schéma d'un CNN

Modèle de base. Le code en annexe (cf annexe - figure 10) correspond à l'implémentation d'un réseau de neurones convolutif (CNN) pour la classification de données. L'architecture contient 3 couches de convolution avec des kernels de 3x3 et des tailles de filtres de sortie de 8, 16 et 32 respectivement, chaque couche de convolution est suivie d'une couche de pooling (MaxPool2d) avec une taille de fenêtre de 2x2. Après les couches convolutives et de pooling, les sorties sont aplaties et passées à travers deux couches linéaires (Fully-Connected) où la première couche à 50 neurones et la deuxième à 10 neurones correspondant aux 10 classes de sortie.

Les hyperparamètres utilisés dans ce modèle sont les suivants :

- kernel-size=3 : taille du noyau de convolution
- nn.MaxPool2d(2, 2) : taille de la fenêtre et stride du pooling
- 32 x 30 x 52 : nombre de features de sortie de la dernière couche de convolution, utilisé pour déterminer la taille de la couche linéaire suivante
- 50 : nombre de neurones de la première couche linéaire
- 10 : nombre de neurones de la deuxième couche linéaire, correspondant au nombre de classes de sortie.

Modèle amélioré 1. Le code en annexe (cf annexe - figure 11) correspond à notre version améliorée du CNN. L'architecture contient également 3 couches de convolution avec des kernels de 3x3 et des tailles de filtres de sortie de 8, 16 et 32 respectivement. Chaque couche de convolution est suivie d'une couche de batch normalisation (BatchNorm2d) pour rendre la phase d'entraînement plus rapide est donner de la stabilité. Ensuite une couche de max-pooling est utilisé avec une taille de fenêtre de 2x2. Après

les couches convolutives et de pooling, les sorties sont aplaties et passées à travers deux couches fully-connected où la première couche à 50 neurones et la deuxième à 10 neurones correspondant aux 10 classes de sortie. Ces deux couches denses fully-connected permettent de faire la classification. Nous avons utilisé une fonction d'activation ReLu.
Les hyperparamètres utilisés dans ce modèle sont les suivants :

- kernel-size=3 : taille du noyau de convolution
- nn.MaxPool2d(2, 2) : taille de la fenêtre et stride du pooling
- 32 x 30 x 52 : nombre de features de sortie de la dernière couche de convolution, utilisé pour déterminer la taille de la couche linéaire suivante
- 50 : nombre de neurones de la première couche linéaire
- 10 : nombre de neurones de la deuxième couche linéaire, correspondant au nombre de classes de sortie.

Modèle amélioré 2. Pour cette deuxième version améliorée du CNN (cf annexe - figure 12) nous avons utilisé une architecture similaire à celle précédente, qui ne diffère de celle-ci que par le nombre de canaux en sortie, ici de 16, 32 et 64. Chaque couche est suivie par une couche de normalisation puis comme la version précédente, nous avons utilisé un MaxPooling de taille 2 x 2 sur la sortie de chaque couche ainsi que deux couches fully-connected pour faire passer le nombre de neurones à 10 en sortie. Cette version amélioré est donc similaire à la première version amélioré mais possède un nombre de paramètres plus important.
Les hyperparamètres utilisés dans ce modèle sont les suivants :

- kernel-size=3 : taille du noyau de convolution
- nn.MaxPool2d(2, 2) : taille de la fenêtre et stride du pooling
- Le nombre de filtres de chaque couche de convolution : 16, 32 et 64
- 64 x 30 x 52 : nombre de features de sortie de la dernière couche de convolution, utilisé pour déterminer la taille de la couche linéaire suivante
- 256 : nombre de neurones de la première couche linéaire
- 10 : nombre de neurones de la deuxième couche linéaire, correspondant au nombre de classes de sortie.

4 Résultats

Après avoir entraîné nos modèles avec différentes valeurs de learning rate, d'époques et en testant sur différents optimiseurs à savoir Adam et Sgd, nous avons obtenu des résultats variés. Alors que nos meilleurs MLP tournent aux alentours de 0.6 pour la valeur du test, nos meilleurs CNN sont environ à 0.8 (figure 4). Pour nos deux modèles améliorés, l'optimiseur Adam semble être le plus efficace.

	MLP	CNN
Valeur du meilleur Train	0.9781	1.0
Valeur du meilleur Test	0.6375	0.8125
Valeur de la meilleure Loss	0.17	0.06
Nombre de paramètres	7113098	2502660
Valeur moyenne du Test	0.5583	0.7222

TABLE 1 – Comparaisons des performances entre MLP et CNN

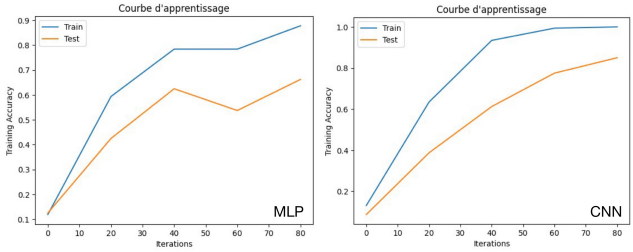

FIGURE 4 – Comparaison des courbes d'accuracy

Sur les 20 configurations différentes que nous avons testées, nous avons obtenu les meilleurs résultats pour notre modèle MLP avec cette configuration :

- taille du bach : 32
- optimiseur : Adam
- taux d'apprentissage : 0.001
- nombre d'époques : 20

De la même manière, nous avons obtenu les meilleurs résultats pour notre modèle CNN avec cette configuration :

- taille du bach : 32
- optimiseur : Adam
- taux d'apprentissage : 0.0001
- nombre d'époques : 20

La comparaison de nos meilleurs résultats, nous montre bien que le modèle utilisant un CNN semble bien meilleur que celui utilisant le MLP (table 1), que ce soit en terme de précision du test, du nombre de paramètres ou de loss. Cela peut venir du fait que les architectures CNN ont été créé pour gérer les problèmes de classification d'images (Spectrogrammes en entrée). Ainsi, le nombre de paramètres augmente facilement pour le modèle MLP par rapport au CNN, ce qui peut créer des redondances et donc des difficultés lors de la phase d'entraînement (overfitting).
En effet, nous pouvons voir que les CNN nécessitent en général moins de paramètres que les MLP pour atteindre des résultats supérieurs. Le CNN peut atteindre de très bons résultats avec moins de paramètres, comme le montre le meilleur modèle retenu avec 2502660 paramètres et une précision de 0,8125 au test contre 7113098 paramètres pour le MLP. En augmentant le nombre de neurones du MLP, en utilisant par exemple 1000 neurones dans

la couche cachée, le nombre de paramètres augmente jusqu'à 28000000, et pourtant n'atteint toujours pas les performances du CNN. En revanche, lorsque nous avons réduit le nombre de paramètres du MLP, la précision n'a pas dépassé 0,5 au test.

Afin de montrer les erreurs de nos modèles, nous avons également fait une matrice de confusion, à partir des résultats obtenus par le meilleur modèle de chaque architecture, qui montre la proportion de réponses correctes et incorrectes pour chaque catégorie de son pour extraire les différentes confusions entre les classes.
En comparaison, la matrice de confusion montre que les classes ont été moins bien classé avec le MLP. Plus généralement, nous remarquons que le MLP a tendance à confondre tous les sons dont les spectrogrammes présentent des similitudes (figure 5). Par exemple en regardant les images des spectrogrammes la confusion entre le son de la tronçonneuse et bruit des vagues est justifié par la grande similitude entre les deux spectrogrammes en entrées.
L'architecture du CNN quant-à elle, semble donné de meilleurs résultats au vu de sa matrice de confusion (figure 5). Nous voyons que les classes ont été en grande partie bien classées pour le CNN.

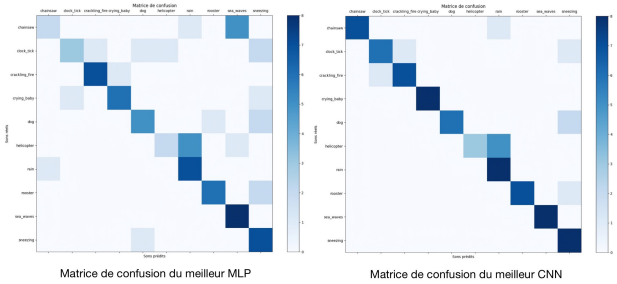

FIGURE 5 – Comparaisons des matrices de confusion.

Tous ces résultats tendent à montrer que pour des tâches de classification audio, le choix d'un CNN semble être une meilleure option en termes de précision et de nombre de paramètres.

5 Conclusion

En conclusion, nous pouvons dire que les CNN semblent être plus efficaces que les MLP pour la classification audio que nous avons étudiée. Lorsque nous comparons les performances des CNN avec ceux des MLP, nous pouvons voir que les CNN ont généralement obtenu de meilleurs résultats en termes de précision du test, de loss et de matrice de confusion. et ce pour un plus petit nombre de paramètres.

Néanmoins d'autres architectures sont également utilisées aujourd'hui pour faire de la classification sonore, telles que les réseaux de neurones récurrents (RNN) ou les réseaux de neurones transformers. Des recherches existantes tendent à montrer que selon les paramètres choisis, les RNN et les CNN peuvent obtenir des performances similaires pour la classification sonore [6]. Enfin, d'autres études suggèrent que les transformers peuvent être plus performants et seraient donc une alternative efficace aux CNN et aux RNN pour la classification sonore [2].

Annexe

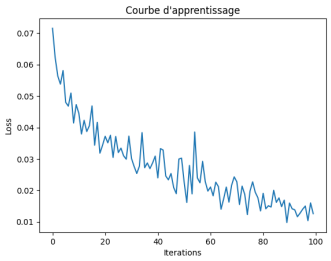

FIGURE 6 – Graphe de Loss pour MLP amélioré

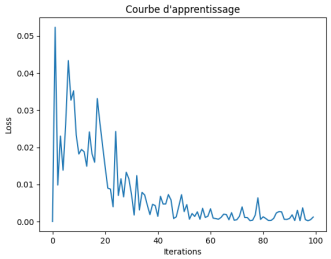

FIGURE 7 – Meilleur Courbe de loss obtenu avec modèle CNN

```python
# Perceptron multi-couche
class MLP(nn.Module):
    def __init__(self, num_hidden=50, input_size = 128*216, num_classes = 10):
        super(MLP, self).__init__()
        self.layer1 = nn.Linear(input_size, num_hidden)
        self.layer2 = nn.Linear(num_hidden, num_classes)
        # vous pouvez définir d'autres couches dans un deuxième temps

    def forward(self, spectro):
        flattened = spectro.view(-1, 128*216) # flatten le spectro
        out = self.layer1(flattened)
        out = torch.relu(out)
        out = self.layer2(out)
        return out
```

FIGURE 8 – Code de la class MLP de base

```
[ ]  class MLP_ameliore(nn.Module):
        def __init__(self):
            super(MLP_ameliore, 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
```

FIGURE 9 – Code de la class MLP amélioré

```
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 10 – Code de la class CNN basique

```
class CNN_ameliore(nn.Module):
    def __init__(self):
        super(CNN_ameliore, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, kernel_size=3)
        self.bn1 = nn.BatchNorm2d(8)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3)
        self.bn2 = nn.BatchNorm2d(16)
        self.conv3 = nn.Conv2d(16, 32, kernel_size=3)
        self.bn3 = nn.BatchNorm2d(32)
        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(32*30*52, 50)
        self.bn4 = nn.BatchNorm1d(50)
        self.fc2 = nn.Linear(50, 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, 32*30*52) # flatten
        x = torch.relu(self.bn4(self.fc1(x)))
        x = self.fc2(x)

        return x
```

FIGURE 11 – Code de la class CNN amélioré 1

```
class CNN_ameliore2(nn.Module):
    def __init__(self):
        super(CNN_ameliore2, 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 12 – Code de notre meilleure CNN amélioré

Références

[1] F. Chollet, *Deep Learning with Python,* Manning Publications, 2018.

[2] Y Gong, S Khurana, A Rouditchenko, *Cmkd : Cnn/transformer-based cross-model knowledge distillation for audio classification,* 2022.

[3] A. Krizhevsky, I. Sutskever, and G. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks,* Advances in Neural Information Processing Systems, 2012.

[4] Y. LeCun, Y. Bengio, and G. Hinton, *Deep Learning,* Nature, vol. 521, pp. 436-444, May 2015.

[5] Karol J. Piczak, *ESC : Dataset for Environmental Sound Classification,* Proceedings of the ACM International Conference on Multimedia, 2015.

[6] Z Zhang, S Xu, S Zhang, *Comparison of CNN and RNN for environmental sound classification,* Proceedings of the IEEE Conference on Multimedia Information Processing and Retrieval (MIP), 2019.