Comparaison des performances entre un MLP et un CNN sur une tâche de classification d'évènements sonores

Maxime Chourré[1] Anne-Sophie Dusart[1]

[1] Université Paul Sabatier

18 avril 2022

Résumé

Ce rapport compare les performances entre un MLP et un CNN sur une tâche de classification d'évènements sonores. Pour cela, nous implémentons sur PyTorch un MLP et un CNN, ainsi qu'une fonction d'entraînement. Nous comparons ensuite nos deux réseaux en faisant varier les paramètres d'entraînement afin d'estimer au mieux les performances de chacun.

Mots Clef

MLP, CNN, Performance, Classification, Sons, Spectrogramme, Comparaison

1 Description des données et des paramètres

1.1 Le corpus

Pour tester les performances, nous utiliserons un corpus contenant des enregistrements audio répartis en 10 catégories.[1] Le but est donc d'entraîner un modèle capable de ranger un enregistrement dans une des 10 catégories.

1.2 L'apprentissage

Pour entraîner les modèles, nous faisons une fonction d'entraînement utilisant la "Cross Entropie Loss" et pouvant utiliser deux optimiseurs : Adam ou SGD. Nous lui donnons 320 enregistrements d'entraînement, et 80 enregistrements de test. Nous allons ensuite créer des groupes aléatoires d'enregistrements, appelés "batch". Pour chaque groupe, nous allons passer les enregistrements dans le modèle, afin de calculer les nouveaux poids de notre réseau. Nous détaillerons les paramètres que nous feront varier dans chaque sous-partie.

2 Description des modèles

2.1 MLP

Définition. Le perceptron multicouche (multilayer perceptron, noté MLP), est un réseau de neurones organisé en couches. Une couche est un groupe de neurones qui n'ont pas de liens les uns avec les autres. Un MLP contient au minimum deux couches : une couche cachée et une couche de sortie, mais on peut ajouter autant de couches cachées que l'on veut. Chaque couche cachée interagit avec la couche précédente seulement, ce qui nous donne pour la $k^{\text{ième}}$ couche cachée :

$$h^{(k)}(x) = g(b^{(k)} + W^{(k)} h^{(k-1)}(x))$$

où $h^{(k)}$ est la $k^{\text{ième}}$ couche cachée, $g(x)$ la fonction d'activation, $b^{(k)}$ le biais et $W^{(k)}$ la matrice de poids. On initialise $h^{(0)}(x)) = x$.
Une fonction d'activation différente peut être appliquée sur la couche de sortie.

Notre implémentation. Nous avons implémenté une classe avec différents paramètres pour personnaliser le nombre de couches et leur valeur. Il est possible de personnaliser le nombre de couches cachées, ainsi que leur nombre de neurones. On peut aussi changer la fonction d'activation, et décider où avoir un BatchNorm.

```python
class MLPperso(nn.Module):
 '''
 @param num_hidden Liste avec le nombre de neurones par couche cachee
 @param activation Fonction d'activation
 @param batchNorm Liste des layers qui subiront un BatchNorm1d
 '''
 def __init__(self, num_hidden=[50], activation=torch.relu, batchNorm=[]):
  super(MLPperso, self).__init__()

  num_hidden_layer = len(num_hidden)
  self.activation = activation
  self.num_layers = num_hidden_layer
  self.batchNorm = batchNorm # Liste des layers suivis d'un batchnorm

  if num_hidden_layer == 0:
   self.layer1 = nn.Linear(128*216, 10)
  else:
   # On ajoute la premiere couche
   self.layer1 = nn.Linear(128*216, num_hidden[0])
   if 1 in batchNorm:
    self.bn1 = nn.BatchNorm1d(num_hidden[0])

   # On ajoute les couches cachees
   for i in range(1, num_hidden_layer):
    self.__setattr__(f"layer{i+1}",nn.Linear(num_hidden[i-1],num_hidden[i]))
    if i+1 in batchNorm:
     self.__setattr__(f"bn{i+1}", nn.BatchNorm1d(num_hidden[i]))

   # On ajoute la derniere couche
   self.__setattr__(f"layer{num_hidden_layer+1}",nn.Linear(num_hidden[-1],10))

 def forward(self, spectro):
  activ = spectro.view(-1, 128*216)
  for i in range(self.num_layers):
   # On parcourt les layers avec notre spectre d'entree
   activ = self.__getattr__(f"layer{i+1}")(activ)
   activ = self.activation(activ)
   if i+1 in self.batchNorm:
    activ = self.__getattr__(f"bn{i+1}")(activ)
  return self.__getattr__(f"layer{self.num_layers+1}")(activ)
```

2.2 CNN

Définition. Un réseau de convolution (convolutional neural network, noté CNN) est un type de réseau de neu-

rones créé au départ pour traiter des images. Le principe du CNN est assez semblable a celui du MLP : on retrouve une couche de sortie et une ou plusieurs couche(s) cachée(s). Dans les couches cachées nous avons : plusieurs couches de convolution, suivies de couches d'activations, du "pooling" qui n'est autre que du sous-échantillonnage, des couches de normalisation qui servent à accélérer et éventuellement, améliorer l'apprentissage, et des couches "fullyconnected" pour faire la classification. Afin de mieux visualiser ce qu'est un CNN, une représentation graphique est présentée en Figure 1.

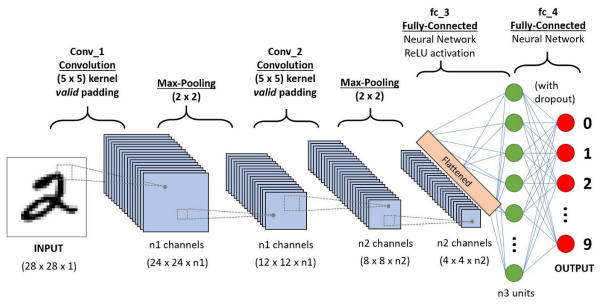

FIGURE 1 – Représentation d'un CNN[2]

Notre implémentation. Nous avons implémenté une classe avec différents paramètres pour personnaliser le nombre de couches et leur valeur. Il est possible de personnaliser la taille du noyau, le nombre de neurones dans une couche cachée, le nombre de convolutions ainsi que la taille de leurs canaux en sortie. On peut aussi changer la fonction d'activation.

```
class CNNperso(nn.Module):
 def __init__(self, channels=[8,16,32], kernelSize=3, linearHidden=50,
      activation=torch.relu):
  super(CNNperso, self).__init__()

  # On utilise une image de test pour connaitre la taille finale
  img = iter(train_loader).next()[0]

  self.pool = nn.MaxPool2d(2, 2)
  self.num_channels = len(channels)
  self.activation = activation

  if self.num_channels == 0:
   raise Exception("Il faut au moins un channel")

  # Premiere convolution
  self.conv1 = nn.Conv2d(1, channels[0], kernelSize)
  img = self.pool(self.conv1(img))

  # On enregsitre les convolutions suivantes
  for i in range(1, self.num_channels):
   self.__setattr__(f"conv{i+1}", nn.Conv2d(channels[i-1], channels[i],
       kernelSize))
   img = self.pool(self.__getattr__(f"conv{i+1}")(img))

  # La taille de l'image finale permet de connaitre l'entree de fc
  img = img.shape
  self.imgSize = img[1]*img[2]*img[3]

  self.fc1 = nn.Linear(self.imgSize, linearHidden)
  self.fc2 = nn.Linear(linearHidden, 10)

 def forward(self, x):
  for i in range(self.num_channels):
   x = self.activation(self.__getattr__(f"conv{i+1}")(x))
   x = self.pool(x)

  # On transforme notre sortie en une liste
  x = x.view(-1, self.imgSize)
  x = self.activation(self.fc1(x))
  x = self.fc2(x)

  return x
```

3 Résultats

3.1 MLP

Notre MLP de base dispose d'une couche cachée de 50 neurones, et a également une fonction d'activation ReLu. Cet MLP contient 1382960 paramètres.
Nous l'avons entraîné avec des groupes de 32 "batchs". Voici certains tests que nous avons réalisés, avec 20 epochs, l'optimiseur "Adam" (sauf contre-indication) et un degré d'apprentissage de 0.0001 :

Modifications	Préc. Train	Préc. Test
Modèle de base	0.31875	0.2625
Optimiseur SGD	0.18125	0.1
Activation Sigmoid	0.2875	0.25
2 hid. layers (50,50)	0.85625	0.5375
2 hid. layers (500,50)	0.49375	0.4125
3 hid. layers (1000, 200,50)	0.7125	0.4
BatchNorm après 1er layer	0.88125	0.5875
3 hid. layers + BatchNorm 2	**0.896875**	**0.6125**

TABLE 1 – Résultats obtenus avec différentes versions d'un modèle MLP

Le meilleur résultat est obtenu avec le modèle de base auquel on a modifié le nombre de neurones dans les couches cachées. Nous avons donc trois couches avec respectivement 1000, 200 et 50 neurones. Nous avons également utilisé une "BatchNorm" après la deuxième couche.
On peut voir sur la Figure 2 les courbes représentant la précision du modèle en fonction des itérations pour les données d'entraînements et les données de tests, dans la version du MLP donnant les meilleurs résultats.

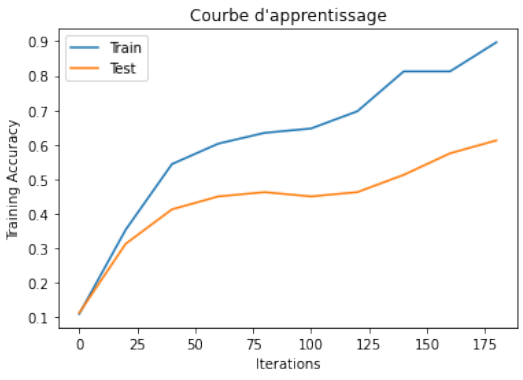

FIGURE 2 – Meilleur résultat obtenu pour notre modèle MLP

3.2 CNN

Notre CNN de base dispose de trois couches de convolutions avec des noyaux de taille 3x3 et à 8, 16 et 32 canaux en sortie respectivement, un pooling qui s'applique après chaque couche de convolution ainsi que deux couches "ful-

lyconnected" fc1 et fc2. fc1 a 50 neurones et fc2 10 neurones. Notre modèle a également une fonction d'activation ReLu appliqué après chaque couche de convolution et après fc1. Notre CNN de base contient 566448 paramètres, soit plus de 2 fois moins que le MLP.

Nous l'avons entraîné avec des groupes de 32 "batchs". Voici certains tests que nous avons réalisés, avec 20 epochs, l'optimiseur "Adam" (sauf contre-indication) et un degré d'apprentissage de 0.0001 :

Modifications	Préc. Train	Préc. Test
Modèle de base	0.96875	0.6125
Optimiseur SGD	0.703125	0.55
Activation Sigmoid	0.203125	0.2
Noyaux 5x5	0.9125	0.55
Noyaux 7x7	0.9125	0.4875
Noyaux 9x9	0.865625	0.575
Neurones conv (8,16,32,64)	0.815625	0.5
Neurones conv (2,4,8)	0.71875	0.6125
Neurones conv (32,16,8)	0.9125	0.5875
Une couche cachée (500)	0.984375	0.6875
Une couche cachée (1000)	0.9875	0.675
Couche cachée (500) + (4,8,16)	**0.975**	**0.65**

TABLE 2 – Résultats obtenus avec différentes versions d'un modèle CNN

Le meilleur résultat est obtenu avec le modèle de base auquel on a modifié le nombre de neurones dans la couche cachée du "fullyconnected" (nous en avons utilisé 500) ; et nous avons utilisé trois convolutions avec respectivement 4, 8 et 16 canaux de sortie.

La Figure 3 nous montre les courbes représentant la précision du modèle en fonction des itérations pour les données d'entraînements et les données de tests dans la version du CNN donnant les meilleurs résultats. On observe un léger sur-apprentissage.

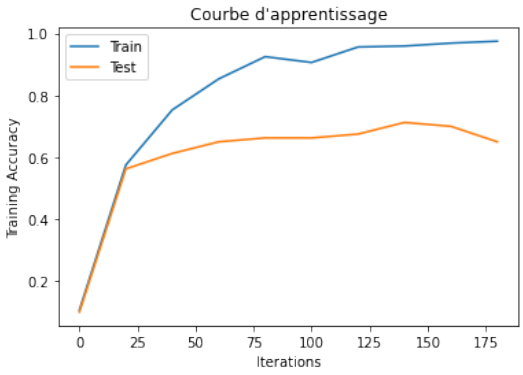

FIGURE 3 – Meilleur résultat obtenu pour notre modèle CNN

3.3 Analyse

On voit que malgré les améliorations apportées, le modèle CNN reste meilleur que le modèle MLP pour la classification d'évènements sonores. Cela peut venir du fait que CNN a été crée pour la classification d'image et que l'on classe les évènements sonores grâce à leur spectrogramme.

De plus, on voit que notre MLP contient plus du double de nombre de paramètres que notre CNN, cela vient du fait que MLP est entièrement connecté. En effet, chaque noeud est connecté à tous les autres noeuds de la couche suivante et de la couche précédente. Ce surplus de paramètre entraîne des redondances qui causent des difficultés lors de l'entraînement. Notre CNN, quant à lui, ne contient que les deux dernières couches entièrement connectées, ce qui réduit le nombre de paramètres et améliore l'apprentissage.

Afin de montrer les difficultés de nos modèles, nous avons fait un histogramme, à partir de la version de chaque modèle donnant les meilleurs résultats, qui montre la proportion de réponses correctes et incorrectes pour chaque catégorie de son. Le nombre de résultats corrects pour chaque son est indiqué par la couleur bleue "Correct". Lorsque le résultat est incorrect, l'estimation faite par le modèle est indiqué par une couleur différente dans la colonne du résultat attendu.

Concernant le MLP, l'histogramme (Figure 4) indique que le modèle a des difficultés à reconnaître les sons de "chainsaw" et les sons de "rain". Plus généralement, on remarque que le MLP a tendance à confondre tous les sons dont les histogrammes présentent des similitudes, a savoir "dog", "craying baby", "clock trick", et "crackling fire" d'un côté puis "chainsaw", "helicopter", "rain" et "sea waves" d'un autre. (cf. Figure 6 en Annexe).

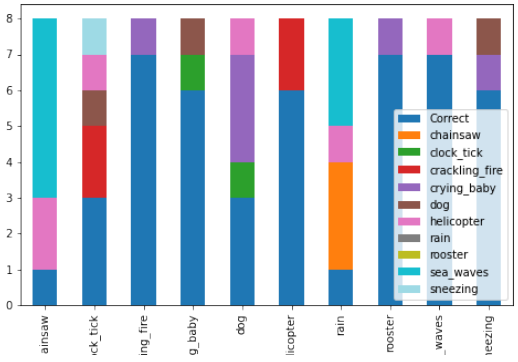

FIGURE 4 – Meilleur résultat obtenu pour notre modèle MLP

Sur le même principe, la Figure 5 illustre les résultats pour le modèle CNN. On remarque que les sons de "sea waves" ont souvent été confondus par des sons "helicopter", on peut supposer que c'est à cause de leurs spectrogrammes qui se ressemblent (cf. Figure 6 en Annexe). Cependant, le CNN réussit à mieux distinguer les nuances dans les histogrammes.

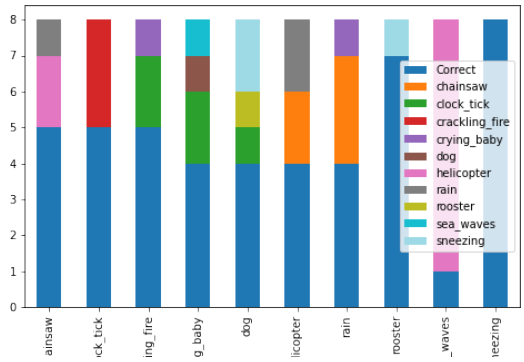

FIGURE 5 – Meilleur résultat obtenu pour notre modèle CNN

4 Conclusion

Après avoir implémenté nos deux réseaux et avoir essayé plusieurs architectures différentes afin d'améliorer les résultats, nous pouvons en déduire que le CNN donne de meilleurs résultats pour la classification d'évènements sonores que le MLP.

Si on souhaite obtenir de meilleurs résultats, il faudrait envisager d'utiliser un jeu de données plus grand, et éventuellement utiliser un autre type de réseau de neurones (RNN par exemple) qui ne se sert pas du spectrogramme.

Annexe

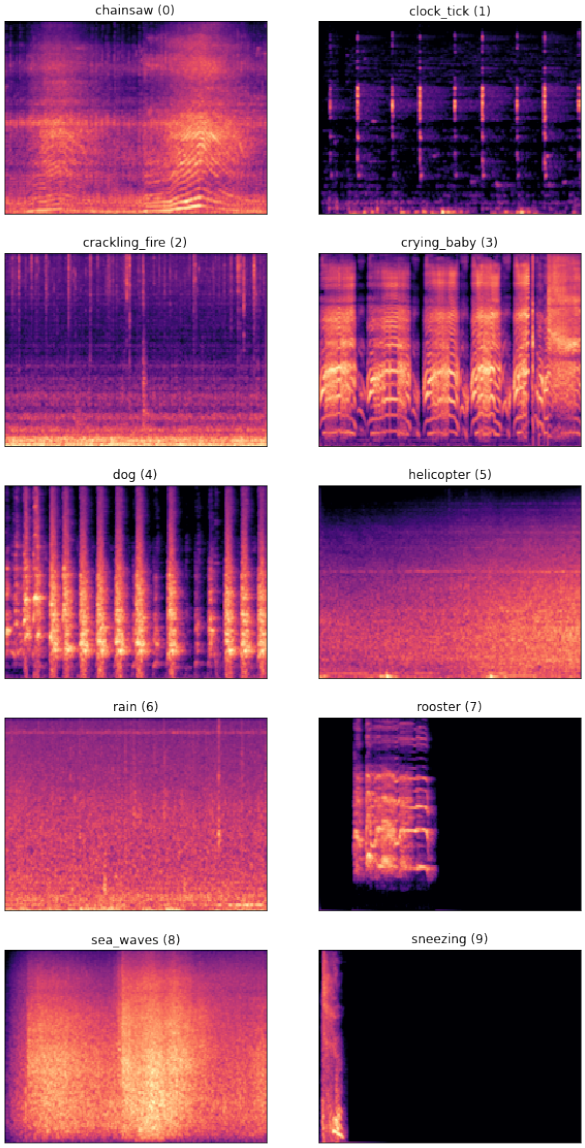

FIGURE 6 – Spectrogrammes des différentes catégories d'enregistrement

Références

[1] Pellegrini, T. : Audio dataset (Oct 2018), https://www.irit.fr/ Thomas.Pellegrini/ens/M2RFA/dataset.zip

[2] Saha, S. : A comprehensive guide to convolutional neural networks (Dec 2018), https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53