Sorbonne Université

Rapport de projet

PLDAC

Zhirui QI
Zhihao ZHU
Master 1 Semestre 2

Tuteur : M. Benjamin Piwowarski

Février 2021 — Mai 2021

Index

Table des matières

1 Introduction

Notre sujet de recherche est Audio, mais en fait il s'agit d'un problème un peu spécifique pour classifier les audios de son dans différents scènes comme *Indoor, Outdoor ou Transport* (3 classes), ou même plus difficile, les scènes plus précises comme (*Airport, Subway, Station...*). C'est bien la tâche qu'on fait dans site de DCASE pour traiter les problèmes s'agit de son.

Par rapport aux caractéristiques des sons qu'on va étudier, il s'agit plutôt de traiter les extraits des identifications dans les sons. Par exemple, on peut entendre le son de décollement d'avion seulement dans les sons enregistrés dans l'aéroport. Comme ça, notre machine learning est capable de classifier les sons d'après ces caractéristiques distincts. Selon les consignes, on doit modifier la forme des datas de son par **LMFB** (Log-mel filter bank Logmel Librosa) et les chargés tout d'abord. Quant à Logmel, c'est un technique qui change la forme de data par **FFT** (Fast Fourier Transform) pour faciliter notre chargé de donnée. Dans le traitement du signal, Mel-Frequency Cepstrum (MFC) est un spectre qui peut être utilisé pour représenter l'audio à court terme. Son principe est basé sur le spectre logarithmique représenté par l'échelle de mel non linéaire et son cosinus linéaire. Conversion (cosinus linéaire transformer).Les coefficients de Mel cepstrum sont généralement obtenus par la méthode suivante :

1. Transformée de Fourier un signal
2. Utilisez la fenêtre de chevauchement triangulaire pour mapper le spectre à l'échelle de Mel
3. Prendre le Logarithme
4. Prendre une transformation cosinus discrète
5. MFCC est le spectre converti.

Selon la documentation au dessous de partie 3.1, on connait que logmel est la première étape indispensable. Logmel Documentation

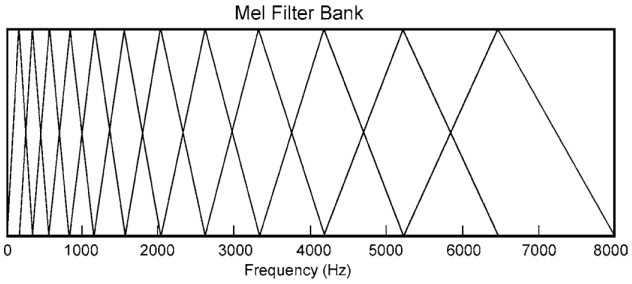

FIGURE 1 – Logmel filter bank

Après, il faut trouver les caractéristiques des différentes scènes par utiliser **CNN** (convolutional neural network) et enfin bien finir la classification d'après les caractéristiques qu'on obtient. Ce sont les démarches de traitements globales. Du coup, c'est la première fois qu'on utilise **CNN** pour traiter le problème. Convolutional Neural Network (CNN) est un réseau de neurones à réaction directe, ses neurones artificiels peuvent répondre à une partie des unités environnantes dans la zone de couverture,a d'excellentes performances pour le traitement d'image à grande

échelle. Le réseau neuronal convolutif se compose d'une ou plusieurs couches convolutives et d'une couche entièrement connectée au sommet (correspondant à un réseau neuronal classique), et comprend également des poids associés et une couche de regroupement. Cette structure permet aux réseaux de neurones convolutifs d'utiliser la structure bidimensionnelle des données d'entrée. Comparés à d'autres structures d'apprentissage en profondeur, les réseaux de neurones convolutifs peuvent donner de meilleurs résultats en matière de reconnaissance d'image et de parole.ça nous donne une impression globale de **CNN** et on trouve que en général, on utilise **RNN** (Recurrent neural network) au lieu de **CNN** pour traiter le problème de son. Dans le site de DCASE, on cherche les documentations qui utilise plutôt **CNN** pour le traitement de data processing et on a trouvé quelque documentations utiles et enrichissantes.CNN1 et et on a ainsi traiter notre donnée selon un **CNN** donné dans une autre documentaion qu'on va indiquer après.

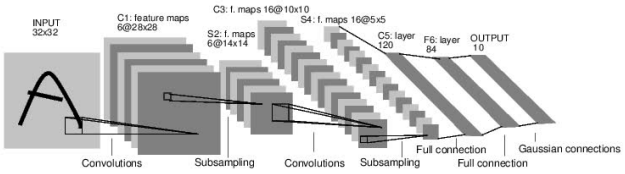

FIGURE 2 – convolutional neural network(CNN)

D'aillleurs, la classification des sons est appliquée souvent dans le cas de recherche des sons et afin de faciliter les recherches plus subtiles par séparer les sons circonstanciels ou même plus évident pour trouver identifier la circonstance où l'utilisateur se trouve.

Enfin,comme résultat, on va évaluer l'accuray de notre data processing et traitement, comme on a vu dans le site DCASE, les docteurs et professeurs qui sont capable d'arriver au niveau d'accuracy de autour de 80

2 Développement

On donne ici le lien de notre code et on voit la première partie de code. Code de projet Ce sont les environements qu'on utilise pour développer le code et on a ainsi *Librosa* principalement pour le traitement **LMFB** (Log-mel filter bank) des datas et *pytorch* principalement pour exécuter le **CNN** (convolutional neural network).

2.1 Prétraitement

2.1.1 lire les fichier

Ici c'est le code pour lire les datas et les charger. On parcourt tous les données de document et forme une dictionnaire qui contient tout. Il retourne une dictionnaire de forme `{classe : liste de chemins des fichiers audios de cette classe}` Code de projet

```
['airport', 'bus', 'metro_station', 'park']
```

A cause de taille de data, on tire seulement 4 classes de données pour les tester et entrainer.

2.1.2 triater la donnée de un fichier

La fonction sert à bien changer la forme de données par **LMFB** et ça forme une matrice de dimension (1,128,431). D'ailleurs, on utilise directement la fonction logmel depuis le source de source **Librosa** en parcourant toutes les fichiers. Code de projet

Algorithm 1: get_logmel_data

Entrée:
filepath : string
Paramètre :
- duration=10
- num_freq_bin=128
- num_fft=2048
- num_channel=1

Sortie :
logmel_data : matrices de donnée de dimension (1,128,431)

1 Lire le fichier de $filepath$
2 hop_length est defini par $\left\lfloor \frac{num\_fft}{2} \right\rfloor$
3 num_time_bin est defini par $\left\lceil \frac{duration \times sr}{hop\_length} \right\rceil$ avec sr est taux d'échantillonnage
4 $logmel\_data$ est initialisé par 0 de dimension ($num\_channel$, num_freq_bin, num_time_bin).
5 On calcule la valeur mel avec paramètre qu'on a défini avant.
6 On calcule le logarithme de la valeur mel

Rappel :

- **hop_length** : Le nombre d'échantillons entre les trames successives
- **num_time_bin** : Le nombre de temps on considère = dimension y de données
- les paramètres `librosa.feature.melspectrogram` besoin :
  y, sr, num_fft,hop_length, num_freq_bin, fmax=sr/2

2.1.3 Fonction pour lire les fichiers et traiter les données des fichiers

Cette fonction est presque la même chose mais on intègre ou concatène les deux processus. Il retourne une matrice de dimension (2000,1,128,431) (`2000=nombre d'échantillion`)

Le processus de prétraitement :
1. lire tous les fichier
2. traiter les données par la méthode `get_logmel_data`
3. enregistrer les données dans une matrice de dimension (nombre d'échantillion, 1, 128, 431)

Code de projet

2.2 Analyse des données (2000 échantillions)

2.2.1 la variance

On calcule la somme de variance et affiche la variance pour chaque "pixel" dans les données (taille (128,431)) entre les échantillions de même classe. Code de projet

airport : la somme de variance = 235197.98

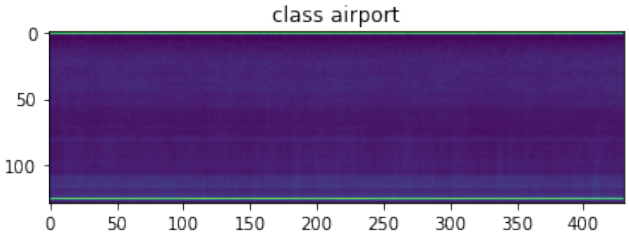

FIGURE 3 – la variance de airport

bus : la somme de variance = 270929.2

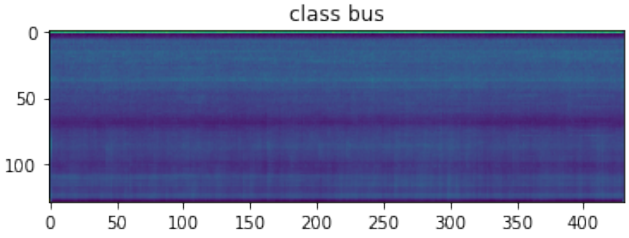

FIGURE 4 – la variance de bus

metro station : la somme de variance = 347128.25

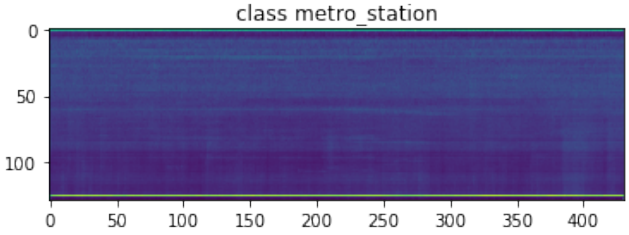

FIGURE 5 – la variance de metro station

la variance de metro station est le plus haut, on peut voire la précision de metro station est le plus pas dans les tests suivants

park : la somme de variance = 269944.06

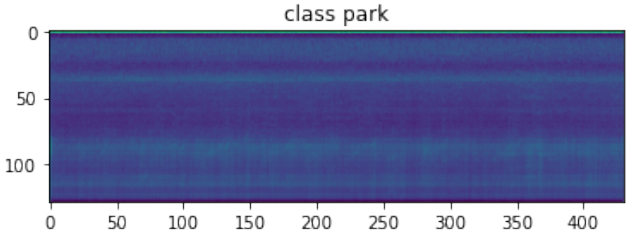

FIGURE 6 – la variance de park

2.2.2 la similarité

on calcule la similarité d'après modèle cosinus :

(plus haut le score est, les données sont plus similaries)

classe airport vs classe bus : 1.3630869e-06
classe airport vs classe metro station : 1.3439948e-06

classe airport vs classe park : 1.0636244e-06
classe bus vs classe metro station : 1.2277217e-06
classe bus vs classe park : 9.777448e-07
classe metro station vs classe park : 9.379438e-07

Il explique que le réseau a tendance à confondre
bus et metro station
airport et bus
airport et metro station

2.3 Construction de CNN

2.3.1 net 1 (essai et exemple)

Comme c'est la première fois qu'on développe un CNN donc selon les consignes et exemples données, on crée un petit CNN avec 2 couches cachés et 3 couches poolings comme un essai.On peut établir un tableau pour ce net.

input module
Conv(1,6,3) - ReLU Max pooling(2, 2)
Conv(6,16,3) - ReLU Max pooling(2, 2)
Linear(16*3180, 120) - ReLU Linear(200, 84) - ReLU Linear(84, nb_classe)

Code de projet On voit ici quelque paramètre de ce net.

```
Net(
(conv1): Conv2d(1, 6, kernel_size=(3, 3), stride=(1, 1))
(conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1))
(fc1): Linear(in_features=50880, out_features=120, bias=True)
(fc2): Linear(in_features=120, out_features=84, bias=True)
(fc3): Linear(in_features=84, out_features=4, bias=True)
)
```

2.3.2 net 2

D'après l'expérience de CNN simple avant, on crée ici un CNN un peu plus compliqué .Ainsi on établit ici le tableau correspondant.

input module
Conv(1,6,3) - ReLU Max pooling(2, 2)
Conv(6,18,3) - ReLU Max pooling(2, 2)
Linear(18*3180, 120) - ReLU

Linear(200, 84) - ReLU
Linear(84, nb_classe)

Code de projet On voit ici quelque paramètre de ce net.

```
 Net2(
(conv1): Conv2d(1, 6, kernel_size=(3, 3), stride=(1, 1))
(conv2): Conv2d(6, 18, kernel_size=(3, 3), stride=(1, 1))
(fc1): Linear(in_features=57240, out_features=200, bias=True)
(fc2): Linear(in_features=200, out_features=84, bias=True)
(fc3): Linear(in_features=84, out_features=4, bias=True)
)
```

tester le Réseau est bien valide

On a ainsi testé la validation de notre CNN et ça bien nous donne une matrice de taille (2000,4). (on prend 2000 échantillion)

```
input = torch.from_numpy(np.array(list_data))
output = net(input)
print (output)
```

```
tensor([[-0.0237,  0.0619,  0.0182, -0.0847],
[-0.0535,  0.0673,  0.0054, -0.0734],
[-0.0246,  0.0282, -0.0030, -0.0892],
...,
[-0.0429,  0.0650,  0.0069, -0.0895],
[-0.0188,  0.0601, -0.0307, -0.0670],
[-0.0228,  0.0814,  0.0167, -0.1040]], grad_fn=<AddmmBackward>)
```

2.3.3 net compliqué

Enfin, on pose un CNN vraiment compliqué sert à traiter les données.
CNN2 De même on établit ici le tableau correspondant.

input module

Conv(1,42,5)(pad-2,stride-2)
Batchnormalisation(42) - ReLU

Conv(42,42,3)(pad-1,stride-1)
Batchnormalisation(42) - ReLU

Max pooling(2, 2) + GaussianNoise(1.00)

Conv(42,84,3)(pad-1,stride-1)
Batchnormalisation(84) - ReLU

Conv(84,84,3)(pad-1,stride-1)
Batchnormalisation(84) - ReLU

Max pooling(2, 2) + GaussianNoise(0.75)

Conv(84,168,3)(pad-1,stride-1)
Batchnormalisation(168) - ReLU
Drop out(0.3)

Conv(168,168,3)(pad-1,stride-1)
Batchnormalisation(168) - ReLU
Drop out(0.3)

Conv(168,168,3)(pad-1,stride-1)
Batchnormalisation(168) - ReLU
Drop out(0.3)

Max pooling(2, 2) + GaussianNoise(0.75)

Conv(168,336,3)(pad-1,stride-1)
Batchnormalisation(336) - ReLU
Drop out(0.5)

Conv(336,336,1)(pad-1,stride-1)
Batchnormalisation(336) - ReLU
Drop out(0.5)

Conv(336,10,1)(pad-1,stride-1)
Batchnormalisation(10)
GaussianNoise(0.3)
Global-Average-Pooling

10-way Soft-Max

Code de projet

2.4 Train (peut faire validation)

2.4.1 processus de train (par tous les échantillions)

On procède ici la fonction de train par minimiser la loss fonction et après chaque itération (backward) on met les gradients à zéro et on peut ainsi choisir la façon de descente de gradient. D'ailleurs, si on a plusieurs classes on peut établir une matrice de confusion pour traiter les données. On affiche ici le graphe le lien entre le nombre d'itération et le coût quand on prend le nombre de batch est 50. D'après le graphe au dessous, on sait pour le batch 50, on a le coût de fonction descent quand le nombre d'itération augmente. On donne ici l'algorithme pour notre entraînement des données.

```
net1 = Net(nb_calsse)
optimizer = optim.Adam(net1.parameters(), lr=0.001,eps = 1e-16)
matrices_confusion,net1 = train_visual (net1,optimizer,train_x=list_data,
    train_y=list_label,pourcentage_validation=1,nbite=4,batch_size=120,visual
     = True)
```

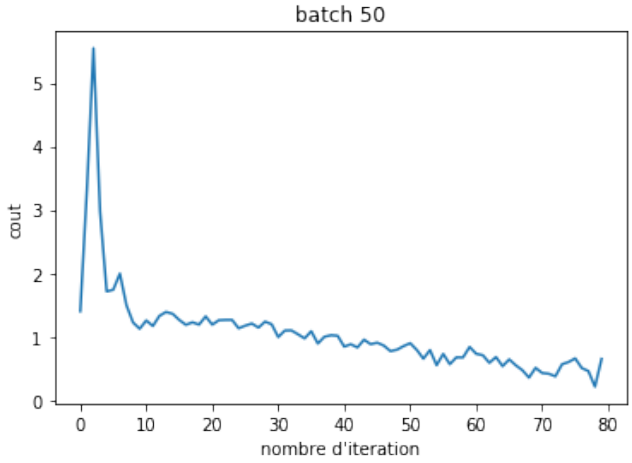

FIGURE 7 – net 1 entrainé par tous les échantillions

Algorithm 2: Train_visualisation

Entrée:

net : Réseau

optimizer : type optim pour défini la méthode de la descente de gradient (Adam, SGD etc.)

train_x : matrice taille (nombre d'echantillion train, nombre de channel , dimension x de donnee , dimension y de donnee) : matrice des données

train_y : liste taille (nombre d'echantillion train, 1) : liste des labels

val_x : matrice taille (nombre d'echantillion validation, nombre de channel , dimension x de donnee , dimension y de donnee) : matrice des données

val_y : vliste taille (nombre d'echantillion validation, 1) : liste des labels

paramètre :

- lossfonction=nn.CrossEntropyLoss() : fonction du coût
- pourcentage_validation=0.9 : pourcentage de validation (utilise si val_x ou val_y est vide)
- nbite=1 : nombre d'époque de backward
- batch_size=1 : taille de batch pour chaque itération
- visual=True : decide si on affiche le graphe du coût

Sortie :

matrices_confusion (dimension (nb_classe, nb_classe)), net

if *val_x ou val_y est vide* **then**
    Separation des données d'après la paramètre *pourcentage_validation* ;
end
Tester si *batch_size* est plus grand que la taille de donnée et corriger le *batch_size*;
Confirmer que les echantillions de train sont de ordre arbitraire;
for *poque* $\in$ *nbite* **do**
    for $i, data \in enumerate(dataset)$ **do**
        Extraiter dataTrain et label;
        Mettre le paramètre gradients en zéro;
        forward + backward + optimize;
        Calculer le coût de validataion pour chaque renouvellement de pararmète
    end
    Calculer la precision de validataion pour chaque époque;
    Calculer le coût de validataion pour chaque époque;
end
visualiser et affichage;

Code de projet

2.4.2 enregistrer le Reseau

Etape pour enregistrer le Réseau.

```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

2.5 Train et validation

Après le train, on teste ici la validation de notre ensemble train et comme la fonction `séparer validation` juste avant, on sépare les données en 2 parties à propos de validation 90% et ensemble de test 10%. Code de projet On teste la validation de 2 CNN qu'on propose avant.

Premier net :

```
net1 = Net(nb_calsse)
optimizer = optim.Adam(net1.parameters(), lr=0.001,eps = 1e-16)
matrices_confusion,net1 = train_visual (net1,optimizer,train_x=list_data,
    train_y=list_label,nbite=4,batch_size=50,visual = True)
```

On fait 90% validation, et 4 époque avec 50 mini-batch :

On affiche ici les données pour la validation et précision et ainsi les graphes pour ces deux caractéristiques.

```
validation  90.0  % :
loss =  0.13427214324474335
precision validation :  95.55555555555556  %
```

On constate que la précision de validation augmente quand le nombre d'itération augmente. C'est un phénomène comme prévu.

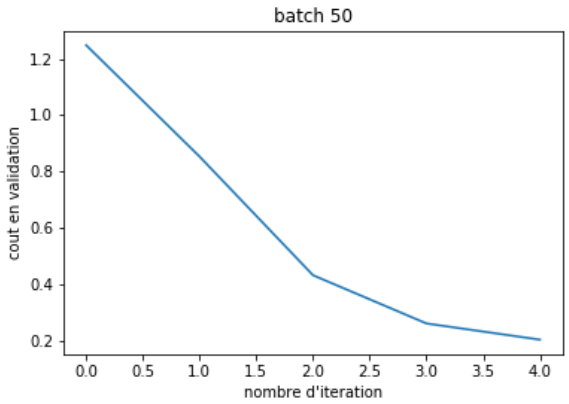

FIGURE 8 – net 1 coût d'exécution chaque époque

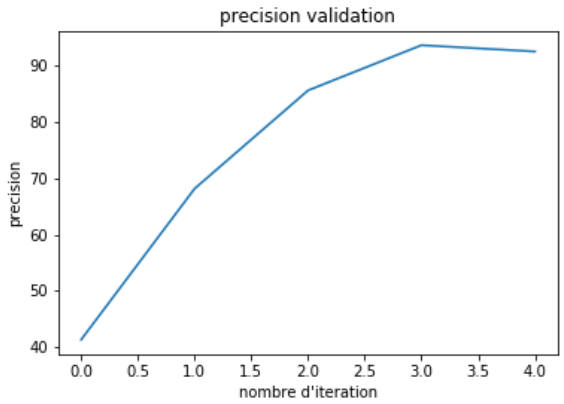

FIGURE 9 – net 1 precision validation chaque époque

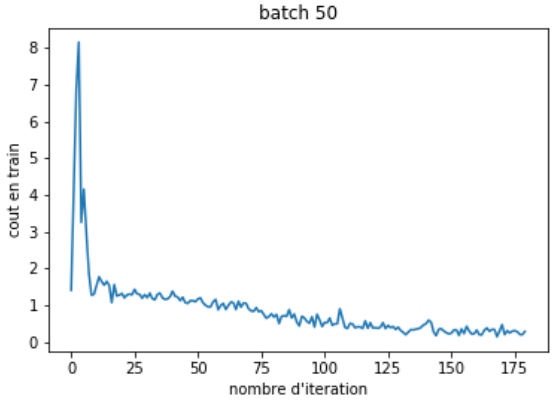

FIGURE 10 – net 1 coût d'exécution chaque itération

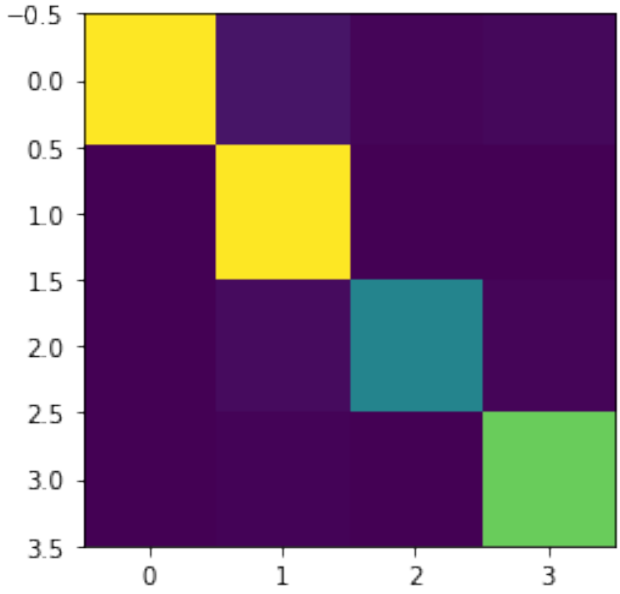

FIGURE 11 – net 1 matrice confusion

La matrice confusion `[0,1,2,3]` répresente `['airport', 'bus', 'metro_station', 'park']`. La valeur précise :

```
[[533  30   8  12]
 [  0 534   1   1]
 [  1  15 240   7]
 [  0   5   0 413]]
```

D'apres la partie **Analyse des données**, la similarité explique que le réseau a tendance à confondre

bus et metro station
airport et bus
airport et metro station

Deuxième net :

On fait 90% validation, et 5 époque avec 50 mini-batch :

Pareil pour le deuxième net.

```
validation  90.0  % :
loss =  0.1327710896730423
precision validation :  96.33333333333334  %
```

On peut savoir ici pour le deuxième net c'est presque la même chose mais la précision n'est pas assez bien que le premier.

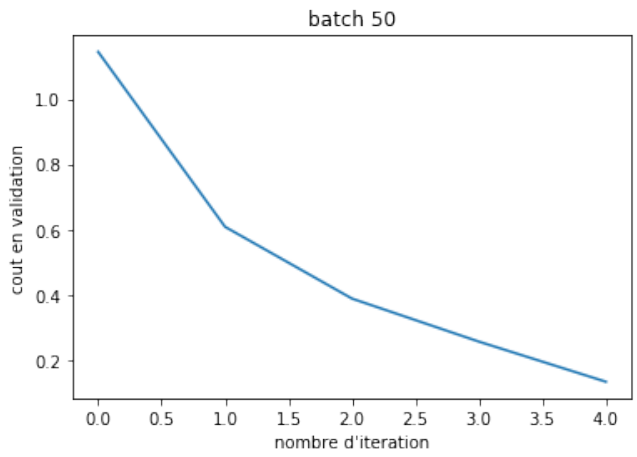

FIGURE 12 – net 2 coût d'exécution chaque époque

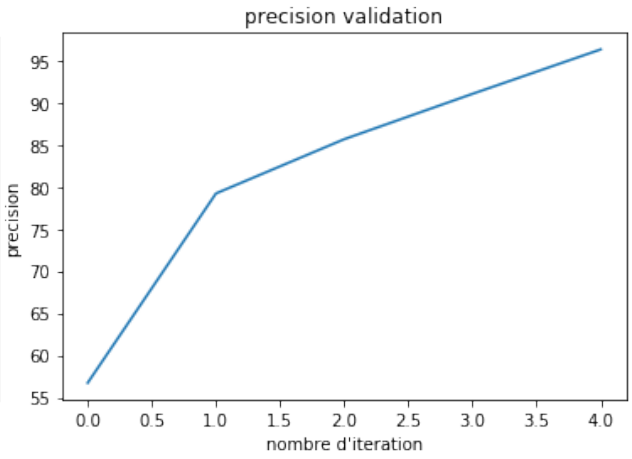

FIGURE 13 – net 2 precision validation chaque époque

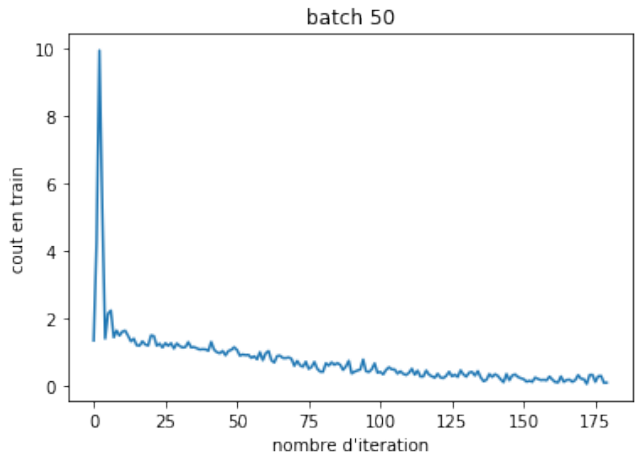

FIGURE 14 – net 2 coût d'exécution chaque itération

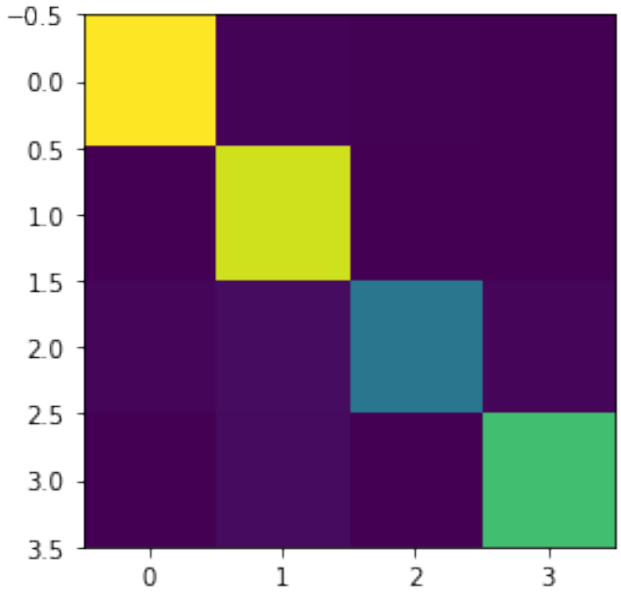

FIGURE 15 – net 2 matrice confusion

La matrice confusion `[0,1,2,3]` répresente `['airport', 'bus', 'metro_station', 'park']`. La valeur précise :

```
[[575   5   3   0]
 [  0 533   1   2]
 [ 11  20 225   7]
 [  1  16   0 401]]
```

D'apres la partie **Analyse des données**, la similarité explique que le réseau a tendance à confondre

bus et metro station
airport et bus
airport et metro station

2.6 Test de précision

Enfin, on teste la précision exacte distincte de Réseau (tous les classes), de classe aéroport, de classe bus, de classe métro et de classe parc de notre CNN différents et les affiche. Code de projet

Pour net 1 :

```
Precision de Reseau : 96 %

Precision de airport : 100 %
Precision de   bus : 100 %
Precision de metro_station : 84 %
Precision de  park : 100 %
```

Pour net 2 :

```
Precision de Reseau : 89 %

Precision de airport : 100 %
Precision de   bus : 100 %
Precision de metro_station : 64 %
Precision de  park : 93 %
```

2.7 Test avancé

On fait 90% validation, et 50 époque avec batch(1800) :

On voit bien les coûts de validation et test baise avec le nombre d'itération augmente.

Premier net :

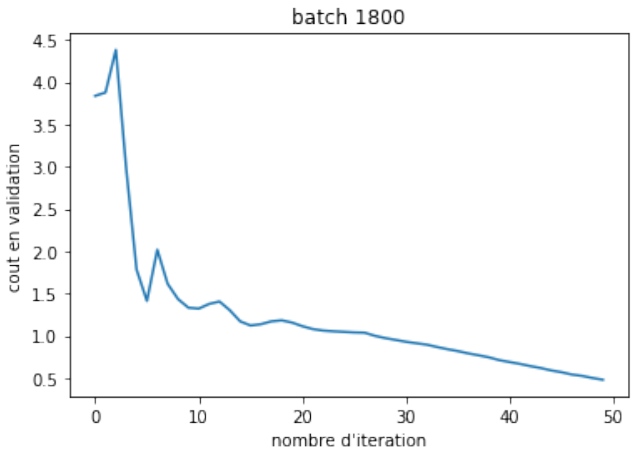

FIGURE 16 – net 1 coût d'exécution chaque époque

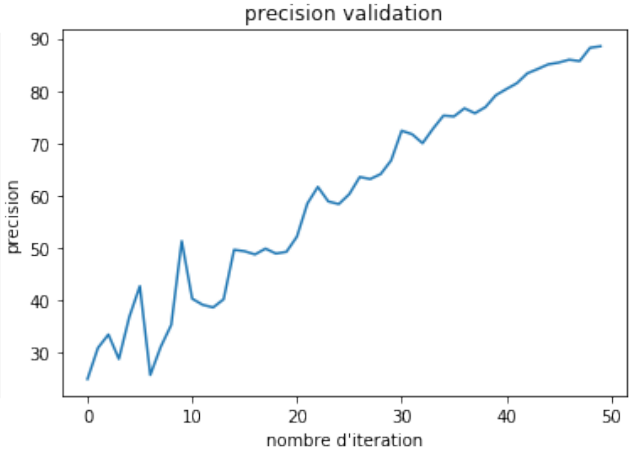

FIGURE 17 – net 1 precision validation chaque époque

```
validation  90.0  % :
loss =  0.47955480217933655
precision validation :  88.5  %
```

```
Precision de Reseau : 53 %

Precision de airport : 70 %
Precision de   bus :  8 %
Precision de metro : 62 %
Precision de  park : 72 %
```

Deuxième net :

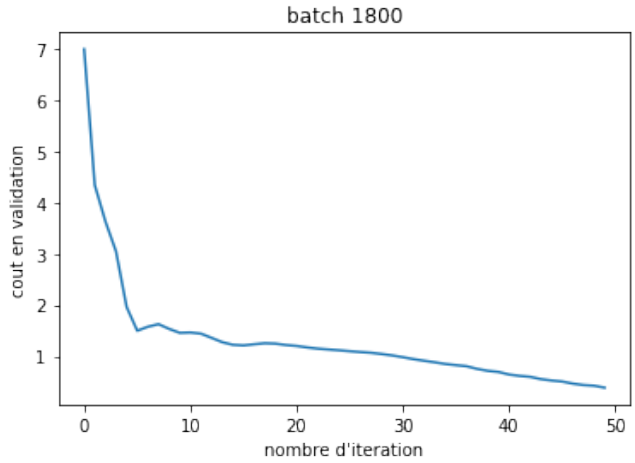

FIGURE 18 – net 2 coût d'exécution chaque époque

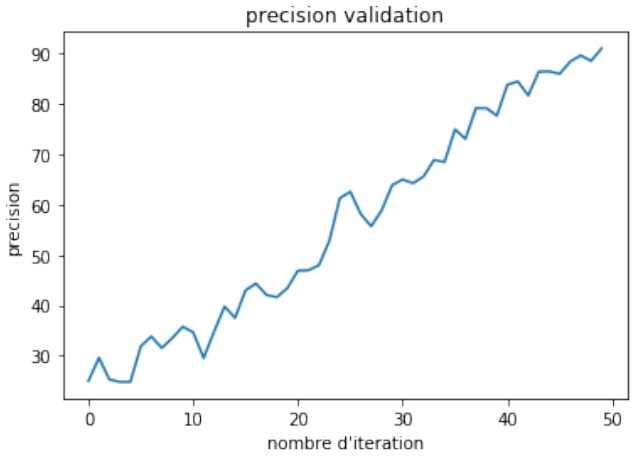

FIGURE 19 – net 2 precision validation chaque époque

```
validation  90.0  % :
loss =  0.3835340738296509
precision validation :  90.94444444444446  %
```

```
Precision de Reseau : 51 %

Precision de airport : 72 %
Precision de   bus : 32 %
Precision de metro : 32 %
Precision de  park : 68 %
```

On voit bien les coûts de validation et test baise avec le nombre d'itération augmente c'est de même avec les premiers graphes.Mais on trouve quelque chose un peu intéressante, le coût de deuxième partie est plus petit que celui de premier même si on a moins de batch.

2.8 Tableau de la précision

On étalit ici un tableau qui donne la précision en train/validation/test en fonction des paramètres pour le network1 et network2.

Net / domaines	$Net\ 1$	$Net\ 2$
Cas : 90% validation, et 5 époque avec 50 mini-batch		
pourcentage de la validation	90%	90%
loss	0.13427214324474335	0.1327710896730423
precision validation	95.55555555555556 %	96.33333333333334 %
Precision de Reseau	96%	89 %
Precision de airport	100%	100 %
Precision de Bus	100%	100 %
Precision de metro station	84%	64 %
Precision de park	100%	93 %
Cas : 90% validation, et 50 époque avec batch(1800)		
pourcentage de la validation	90%	90%
loss	0.47955480217933655	0.3835340738296509
precision validation	88.5%	90.94444444444446 %
Precision de Reseau	53%	51 %
Precision de airport	70%	72 %
Precision de Bus	8%	32 %
Precision de metro station	62%	32 %
Precision de park	72%	68 %

D'après ce tableau de précision, on voit bien que pour le même network et pourcentage de validation, méthode 1 toujours donne une précision plus haute. Ca veut dire que ce nombre de époque et batch convient plus à cette recherche. Et puis, en même temps, Net 2 donne une précision autant que Net 1 donc on constate que les deux réseau sont presque mêmes choses.

On préfère d'analyse cas 1 : 90% validation, et 5 époque avec 50 mini-batch, Parce que les résultats sont plus fiables que cas 2.

On peut observer que la precision de metro station n'est pas très bien. On peut expliquer d'après partie **Analyse des données** : la variance de metro station est le plus haut, donc c'est plus difficle de trouver les features de cette classe. Par conséquent, le Réseau va confondre plus des échantiilions de metro station que les autres. **(on a déja analysé l'influence de la similarité pour la matrice confusion)**

3 Conclusion

En résumé, on utilise 3 CNN pour traiter notre donnée et on a enfin les performances sont environ 90%. D'ailleurs, on recherche sur la relation entre les coûts et nombre d'itération et batch. On sait ainsi l'importance de prétraitement des données comme **LMFB**, par conséquent, ça facilite notre étude après. Après tout, comme on a indiqué, c'est la première fois qu'on utilise CNN pour traiter le problème et ça nous pose intéressant et enrichissant à la fin.