Yassine El-Khanoussi
Samer Saleh
SICOM 3A

Rapport: Projet Deep Learning

1. Les données

1.1 Création des données d'entraînement, validation et test.

Pour la création des données, nous avons téléchargé les fichiers audio depuis la **librairie libri-speech** [1], plus précisément le dataset “dev-clean” qui comporte environ 5h de parole en anglais de bonne qualité. Ce dataset comporte plusieurs locuteurs. Nous avons pris environ 50% des fichiers pour l'entraînement (environ 2h de parole), puis 5% pour la validation(environ 15 minutes) et 5% pour le test (locuteur et audios différents que ceux utilisés pour l'entraînement).

Nous avons pris pour les différents sets des locuteurs différents. Pour créer de la parole bruitée, nous avons mixé avec une **RSB aléatoire suivant une loi uniforme dans [0; 10]**, la parole propre et un bruit (**babble**) de cafeteria.

On extrait les spectrogrammes des différents fichiers et on effectue une **normalisation des spectrogrammes (spectrogramme par spectrogramme) par méthode min-max**, qui met à l'échelle les données de manière à ce qu'elles soient bornées entre [0,1]. Les spectrogrammes des audios sont ensuite enregistrés dans un fichier pickle qui permet d’avoir en un seul fichier toutes les données.

1.2 Paramètres de STFT

Les données d’entrée que nous allons utiliser sont des spectrogrammes (amplitude de la STFT), pour faire le calcul des STFT nous avons choisi ces paramètres:

- Taille de fenêtre: 512 échantillons. Nous avons choisi cette taille car c’est une taille qui correspond à une durée de 32ms, et que la tranche de durée 20 à 40 ms contient toute l’information sur l’intelligibilité de la parole.
- Taille du saut: ¼ de la taille de la fenêtre. Cela permet de garder une bonne résolution temporelle tout en conservant une taille de spectrogramme (en trames) raisonnable pour réduire le temps d'entraînement.
- Taille de la FFT: 512 échantillons. Nous avons décidé de ne pas faire de zéro-padding, la résolution fréquentielle de 512 échantillons semble être pertinente car c’est un nombre suffisant pour conserver de l’information pour le débruitage et car augmenter considérablement le zéro-padding aurait pour conséquence de alourdir les calculs lors de l'entraînement.

2. Un réseau de neurones profond totalement connecté (Fully Connected neural network)

Un réseau de neurones profond totalement connecté est un modèle de deep learning qui est constitué de plusieurs couches de neurones, où chaque neurone de chaque couche est connecté à chaque neurone de la couche suivante.

Dans cette partie, on va expliquer comment on a mis en place les données à l'entrée de notre FC network puis on va étudier l'influence des différents hyperparamètres sur la performance globale du réseau.

2.1 Mise en place

Entrée et sortie du réseau: Comme mentionnée précédemment, l'entrée du réseau sera un certain nombre de trames de STFT bruité. Plus précisément ca sera un tensor (mixed audio) de dimension (257, 865730), ce qui représente presque 2 heures d'audio. Les données de sortie prédite du réseau (clean audio) elles aussi seront représentées par un tensor de même dimension que le tensor d'entrée. Et donc la couche d'entrée et la couche de sortie du réseau ont une même taille que le nombre de points dans une trame de STFT (257).
Optimizer: Dans ce projet, on utilise **ADAM** (Adaptive Moment Estimation) optimizer pour toutes les architectures. Il ajuste automatiquement le **learning rate qui sera initialisé à 0.001** pour tous nos modèles.
Loss function: MSE (Erreur quadratique moyenne**),** pour les données d'entraînement et d'évaluation(validation).

2.2 Expérimentations et résultats

Nous allons tester et analyser différents modèles et plusieurs hyperparamètres. On commencera par tester l'influence du nombre de couches cachées, puis l'influence du nombre de perceptron par couche cachée et finalement l'influence du batch size.

Influence du nombre de couches cachées

- **Expérience 1** (*Fully Connected*):
    - Forme (shape) des tenseurs en entrée: [x, 257]
    - 1 couche dense en entrée (257 perceptrons)
    - **Variable**: 1 couches cachées **puis 2 puis** 3 **puis** 4 (257 perceptrons chaque)
    - 1 couche dense en sortie (257 perceptrons)
    - Taille des batch: 1000
    - Nombre de paramètres: **`198,918 puis 265,224 puis 331,530 puis 397,836`**
    -
- **Résultats et analyse:**

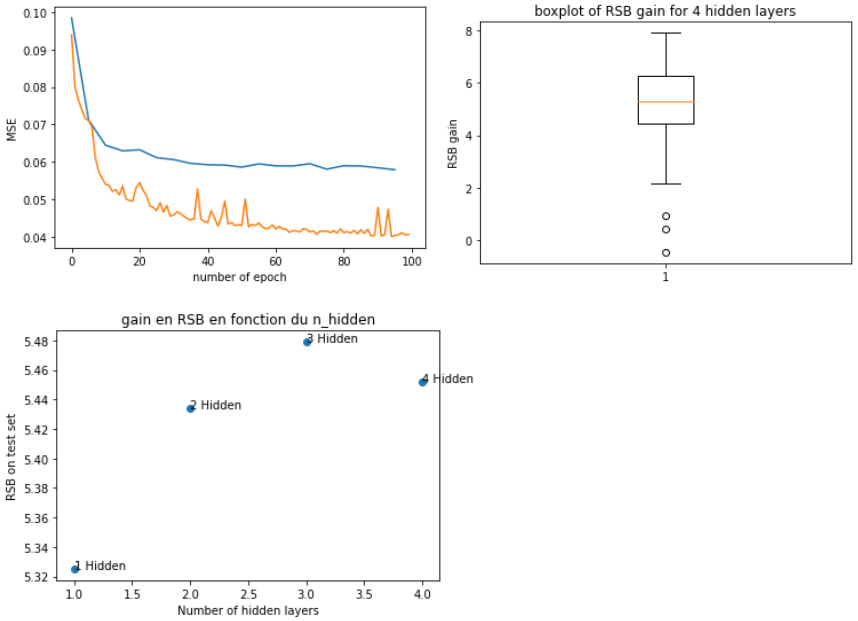


Figure 1: MSE loss for training set(orange) and validation set(blue), boxplot of RSB gain for 4 hidden layers on test set, mean RSB gain on test set for n= 1,2,3 and 4 hidden layers of 257 perceptrons each

- On voit bien une que le nombre de couches cachées a bien une influence sur les performances du modèle (différence de MSE pour le test et validation set et le gain en RSB sur le test set).
- L'augmentation du nombre de couches implique plus de paramètres sans nécessairement améliorer les performances, pour 3 couches on a un gain en RSB moyen (RSB_gain = 5.48) qui est meilleurs que pour 4 couches(RSB_gain = 5.45) . En fait, le nombre optimal de couches cachées pour une tâche spécifique varie en fonction de la complexité du problème.
- On voit bien que sur le boxplot du test set , que les gains en RSB sur les différents audios varie entre des gains de 2 jusqu'à des gains de 8 selon les audios. Et que la médiane est entre 5 et 6 dB. On rappelle bien que le test set est composé d'audio de différents locuteurs que celle du training set avec des RSB de mixture entre 0 et 10 dB.
- Finalement on voit bien un écart de MSE entre celui de l'entraînement et celui de la validation. Cela est normal car les données de validation sont nouvelles au modèle. Cette métrique mesure à quelle point le modèle peut se généraliser à de nouvelles données. Cet écart est appelé **"avoidable bias"** et peut être réduit en augmentant la taille des données d'entraînement par exemple.

Influence de la taille des batch

La taille des batch est le nombre de données d'entraînement que le réseau prend en compte avant de mettre à jour les poids du modèle. Une taille de batch plus importante entraîne des gradients plus stables et des temps d'entraînement plus rapides mais augmente la computation. La taille de batch optimale est un hyperparamètre qu'on essaiera de trouver pour notre base de données. Pour minimiser le temps d'exécution, on teste avec 2 couches cachées (257 perceptrons chacune).

- **Expérience 2** (*Taille des batch Fully connected*):
  - Forme (shape) des tenseurs en entrée: [x, 257]
  - 1 couche dense en entrée (257 perceptrons)

- 2 couches cachées (257 perceptrons chaque)
- 1 couche dense en sortie (257 perceptrons)
- **Variable:** Taille des batch: 1000 puis 2000 puis 4000 puis 8000
- Nombre de paramètres: `265,224`

- **Résultats et analyse:**

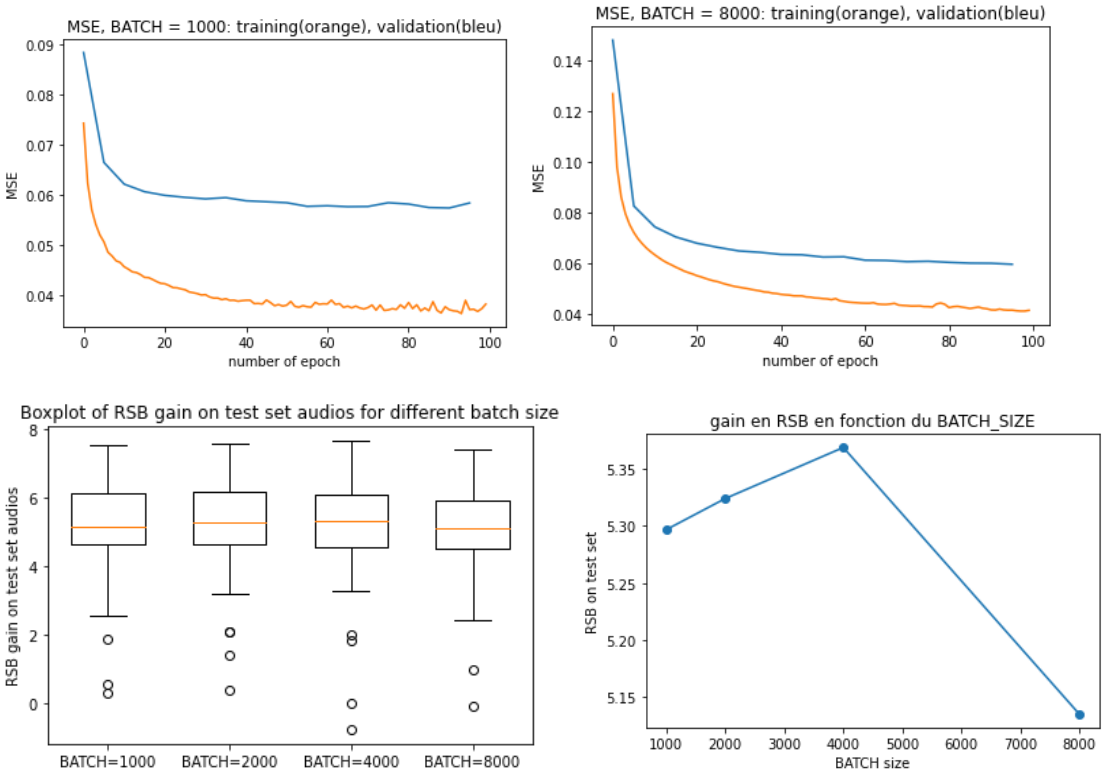


Figure 2: MSE loss for training set(orange) and validation set(blue) for batch = 1000 and 8000, boxplot of RSB gain on test set, average RSB gain on test set for batch =1000, 2000,4000,8000 and 4 hidden layers of 257 perceptrons each.

- Cet hyperparamètre influence énormément le temps d'entraînement vu que qu'on on augmente la taille du batch, la mise à jour du gradient sera moins fréquente et donc moins de calcule.
- En augmentant la taille des batch au début, le gain en RSB du modèle augmente légèrement et le temps d'entraînement diminue jusqu'à ce que celui-ci devienne trop grand et commence à dégrader les performances (BATCH size = 8000).
- On voit bien que pour batch size= 8000 on a moins de variation par epoch du training et validation loss. En effet, en augmentant le batch size, on aura des modèle plus stable et plus généraliste. Cependant comme on le voit bien un size de batch élevé peut aussi dégrader les résultat à cause de l'**overfitting** (pour batch_size = 800 le gain en RSB moyen est le plus bas).

Influence du nombre de neurones par couche cachée

Cet hyperparamètre détermine la capacité du réseau de neurones. Ce nombre là est choisi selon la complexité de la tâche. Cet hyperparamètre est choisi par expérimentation.

- **Expérience 3** (*Nombre de perceptron par couche Fully connected*):
  - Forme (shape) des tenseurs en entrée: [x, 257]
  - 1 couche dense en entrée (257 perceptrons)

    - **Variable:** 3 couches cachées (257 **puis** 512 **puis** 1024 perceptrons chaque)
    - 1 couche dense en sortie (257 perceptrons)
    - Taille des batch: 4000
    - Nombre de paramètres: `331,530 puis 1,051,905 puis 3,676,417`
- **Résultats et analyse:**

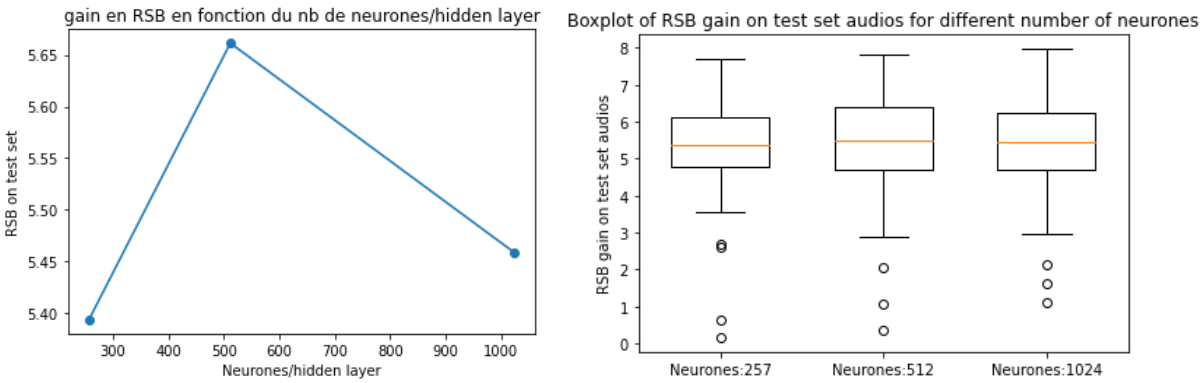


Figure 3: boxplot of RSB gain on test set, average RSB gain on test set for number of neurons/hidden layer = 257, 512 and 1024 and 3 hidden layers, batch_size= 4000.

- On voit bien figure 3, que les meilleurs résultats sont obtenus pour un nombre de neurones/couche cachée = 512 pour les trois couches cachées du réseau. Le gain moyen en RSB est de 5,662 ce qui est le meilleur résultat obtenu jusqu'à présent. On obtient aussi la plus haute médiane dans le boxplot parmi les trois modèles. En prenant moins que 512, le modèle apprend moins de représentation complexe, en prenant plus que 512, le modèle commence à faire du overfitting sur les données d'entraînement.
- Finalement le modèle fully connecté avec un batch=4000, nombre de couches cachées = 3 et nombre de neurones/ couche cachées= 512 sera le modèle final qu'on utilisera dans la suite.

3. Long Short Term Memory (LSTM)

Nous avons fait l'entraînement avec un modèle LSTM.Ces modèles sont capables de maintenir un état interne qui peut capturer des informations sur les entrées précédentes dans la séquence, ce qui peut être utile dans une application telle que la nôtre. Nous avons choisi de tester 3 expériences différentes (on aperçoit en gras les paramètres modifiés):

- Première expérience (*gain_LSTM_win_125_layers_2*):
    - Forme (shape) des tenseurs en entrée: [x, **125**, 257] *(séquence de 125 trames de STFT d'un même audio)*
    - 2 couches LSTM
    - 1 couche dense en sortie (257 perceptrons)
    - Taille des batch: 100 tenseurs de shape [125,257]
    - Nombre de paramètres: `1193508`
- Deuxième expérience (*gain_LSTM_win_25_layers*):
    - Forme (shape) des tenseurs en entrée: [x, **25**, 257]
    - 2 couches LSTM
    - 1 couche dense en sortie (257 perceptrons)
    - Taille des batch: 100 tenseurs de shape [25,257]
    - Nombre de paramètres: `1193508`
- Troisième expérience (*gain_LSTM_win_25_layers_4*):

- Forme (shape) des tenseurs en entrée: [x, **25**, 257]
- **4** couches LSTM
- 1 couche dense en sortie (257 perceptrons)
- Taille des batch: 100 tenseurs de shape [25,257]
- Nombre de paramètres: **2254404**

Voici les résultats obtenus lors de l'entraînement.

■ loss/validation LSTM 1ere experience ◆ loss/train LSTM 1ere experience
◆ loss/validation LSTM 2eme experience ■ loss/train LSTM 2eme experience
■ loss validation LSTM 3eme experience ◆ loss/train LSTM 3eme experience

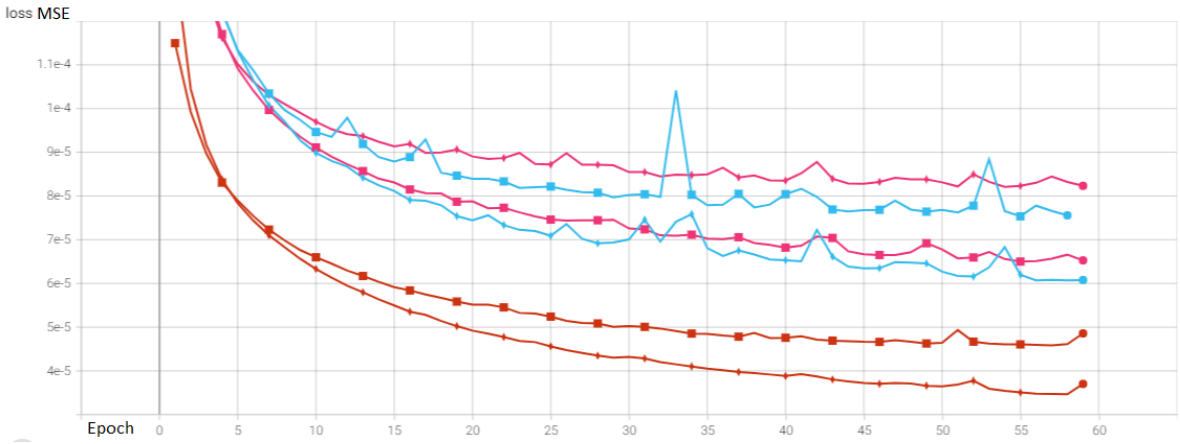


Figure 4 : MSE Loss en fonction des epochs pour les 3 expériences

Comme on peut l'observer en figure 4, le modèle ayant le plus de paramètres obtient une MSE plus basse et il converge plus rapidement. On constate également que le modèle avec une taille de fenêtre plus petite (influant sur le nombre de vecteurs à considérer en mémoire) performe une moins bonne MSE. Contrairement aux résultats de MSE, on voit figure 5 que le modèle de l'expérience 2 (2 couches LSTM et taille de fenêtre 25) obtient les meilleurs résultats en termes de RSB sur le test set. En effet, on voit que la médiane, la moyenne ainsi que les 4 quartiles se situent plus haut que pour les 2 autres expériences.

Ce qu'on peut donc en déduire c'est que prendre une taille de fenêtre **plus petite fenetre (de l'ordre des centaines de millisecondes) conduit à de meilleurs résultats comparé à des grandes fenêtres (1s) dans notre cas.** Cela est lié au fait que 125 correspond à 1 seconde de parole, et que considérer toute l'information dans cette tranche n'est pas pertinent pour un débruitage et peut causer un overfitting du modèle. Cependant, en prenant une taille de fenêtre trop petite (taille de séquence de 5), le modèle ne capture pas assez de contexte dans cette courte période.

On voit également que l'utilisation de 2 couches conduit également à des meilleurs résultats que l'utilisation de 4 couches. Cela est peut être dû à du sur-apprentissage (overfitting). Ainsi le modèle plus simple se généralise mieux à des données inconnues.

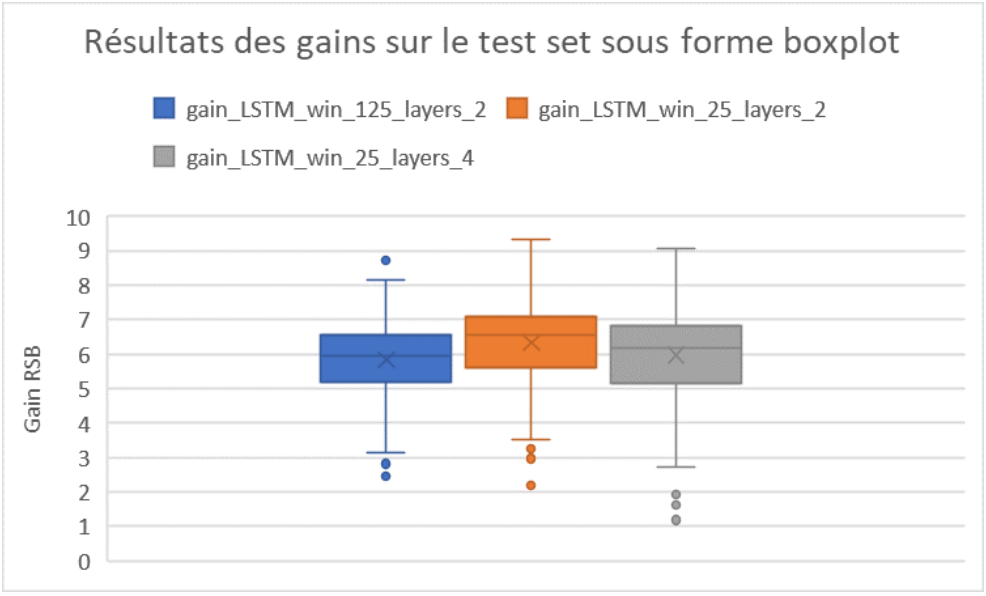


Figure 5 : Résultats des gains en RSB en boxplot sur le test set pour les 3 expériences

4. Convolutional Neural Network (CNN)

Dans cette partie nous avons expérimenté l'utilisation d'un réseau de neurones de type convolutionnel. Nous avons opté pour l'architecture suivante:

Layer (type)	Output Shape	Param #
Input	[-1, 1, 25, 257]	0
Conv2d-1	[-1, 64, 19, 251]	3,200
BatchNorm2d-2	[-1, 64, 19, 251]	128
ReLU-3	[-1, 64, 19, 251]	0
Conv2d-4	[-1, 128, 15, 247]	204,928
BatchNorm2d-5	[-1, 128, 15, 247]	256
ReLU-6	[-1, 128, 15, 247]	0
Conv2d-7	[-1, 256, 13, 245]	295,168
BatchNorm2d-8	[-1, 256, 13, 245]	512
ReLU-9	[-1, 256, 13, 245]	0
ConvTranspose2d-10	[-1, 128, 15, 247]	295,040
BatchNorm2d-11	[-1, 128, 15, 247]	256
ReLU-12	[-1, 128, 15, 247]	0
ConvTranspose2d-13	[-1, 64, 19, 251]	204,864
BatchNorm2d-14	[-1, 64, 19, 251]	128
ReLU-15	[-1, 64, 19, 251]	0
ConvTranspose2d-16	[-1, 1, 25, 257]	3,137
BatchNorm2d-17	[-1, 1, 25, 257]	2
ReLU-18	[-1, 1, 25, 257]	0

Total params: 1,007,619

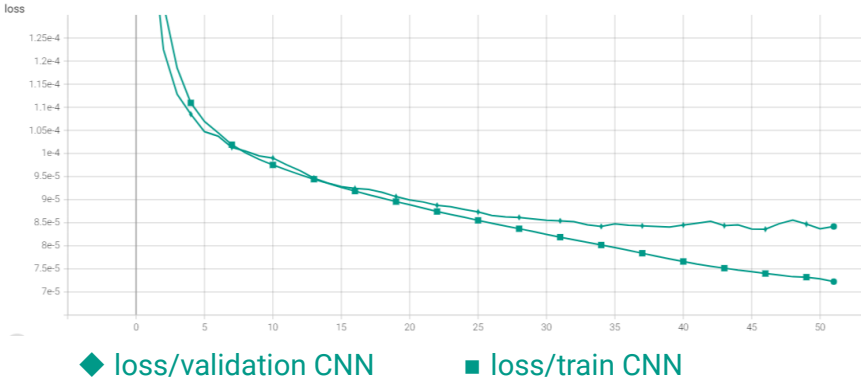


Figure 6 : Architecture du modèle CNN ainsi que la Loss (MSE) en fonction des epoch

On peut noter que nous avons utilisé une taille de filtres 7x7, puis 5x5 et 3x3, cela permet de capturer des features de plus en plus fines sur les couches successives. La taille des filtres est relativement petite (comparée à la taille des spectrogrammes en entrée) afin de capturer des détails fins, cela permet aussi de réduire le nombre de paramètres à entraîner. Le nombre de canaux de sorties passe de 1 à 64 afin de couvrir des zones larges et être plus précis sur la capture de features plus abstraites. Les batch de normalizations sont utilisées pour stabiliser l'entraînement en réduisant les variations de la distribution des activations entre les différentes couches du réseau, ce qui permet aussi une convergence plus rapide. Les résultats sur le test set seront discutés dans la section suivante en les comparant aux autres architectures.

5. Comparaison des modèles et conclusion

Voici les résultats obtenus sur le test set pour les 3 architectures différentes:

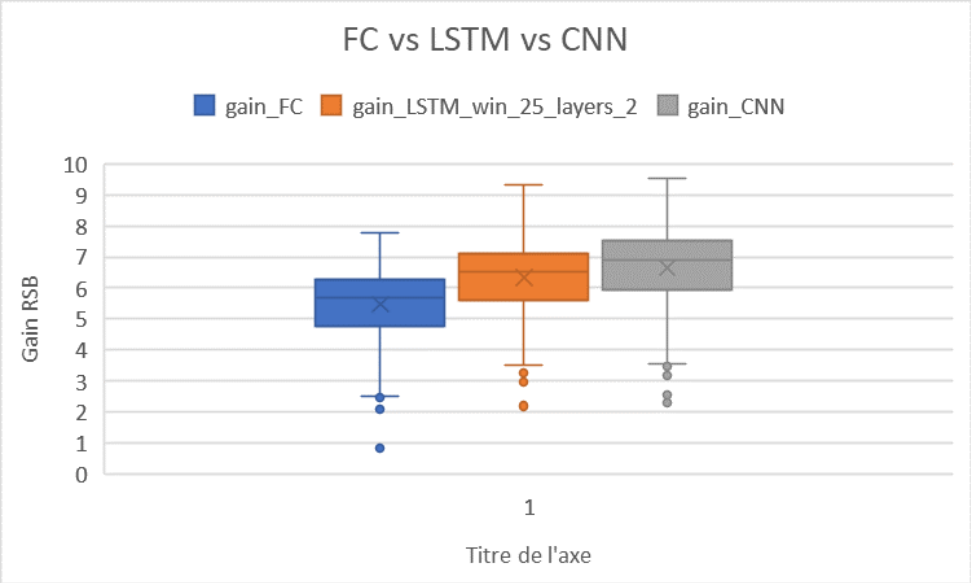
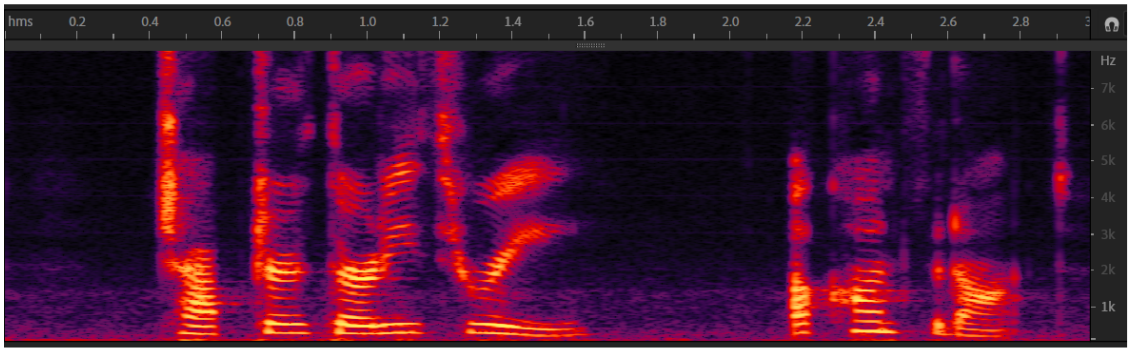


Figure 7: Coût (MSE) en fonction des epoch et boxplot des gains en RSB obtenus sur le test set pour les 3 architectures différentes

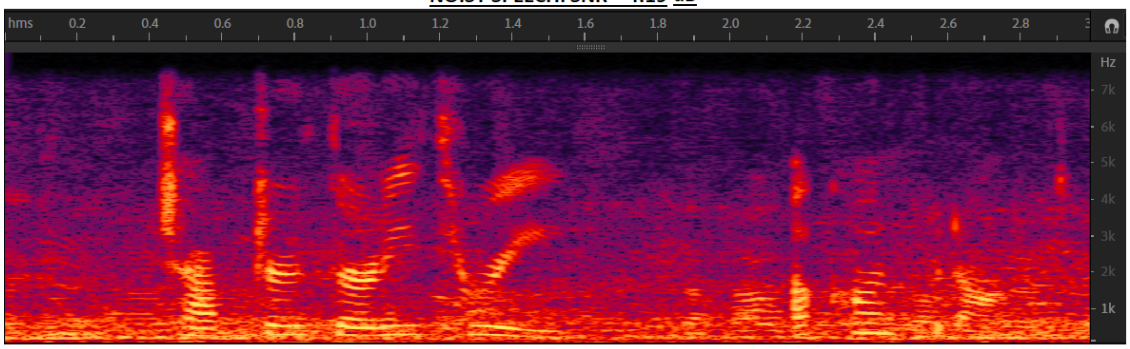
On peut observer figure 7, que le CNN obtient des meilleurs résultats que les autres architectures, cela s'explique par le fait que les architectures basées sur des convolutions sont plus propices à détecter des formes particulières, ainsi cela permet de faire la distinction entre du bruit et de la parole, notamment car la parole contient des fréquences fondamentales et des harmoniques, alors que le bruit n'a pas de forme particulière sur un spectrogramme. Le fait également que les CNN aient une structure hiérarchique, cela permet de capturer différents niveaux de complexité. Les CNN ont aussi moins de paramètres à entraîner que les autres architectures. Lors de ces expériences nous avons également essayé de conserver un nombre de paramètres à peu près égal (1 million environ). Mais c'est également un élément à prendre en compte lors de la comparaison des différentes architectures. Il est difficile d'augmenter le nombre de paramètres pour les FCN en comparaison aux LSTM et CNN.

Pour conclure sur ce projet, nous nous sommes rendus compte que les étapes de pré-processing sont importantes afin d'obtenir des bons résultats, en effet, les paramètres de STFT doivent être choisi de façon réfléchie, penser à normaliser les données et également réfléchir sur quelle unité il faudrait mettre les données (Amplitude, puissance, puissance en dB, etc...). Les hyperparamètres du modèle doivent également être correctement optimisés. Sur les hyperparamètres il n'y a pas encore d'intuition sur les bonnes valeurs, il faut tester différentes valeurs et choisir les meilleures. Nous avons vu également que les modèles tels que les LSTM conduisent à des meilleurs résultats que des FCN, notamment grâce à leur capacité de conserver de l'information pertinente sur les trames précédentes. Finalement nous avons vu que les CNN obtiennent des meilleurs résultats au final, car leur capacité à capturer différents niveaux de complexité. On peut voir page suivante, les spectrogrammes suite à un débruitage.

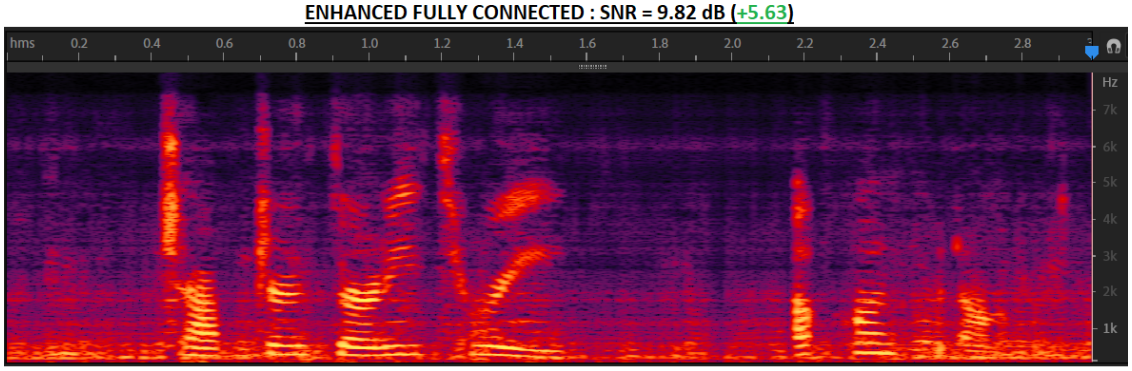
CLEAN SPEECH



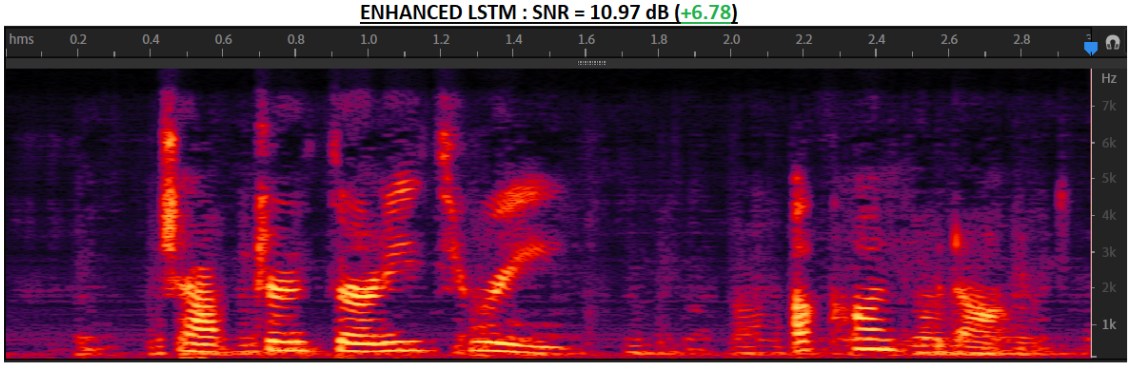
NOISY SPEECH: SNR = 4.19 dB



ENHANCED FULLY CONNECTED : SNR = 9.82 dB (+5.63)



ENHANCED LSTM : SNR = 10.97 dB (+6.78)



ENHANCED CNN : SNR = 11.48 dB (+7.29)

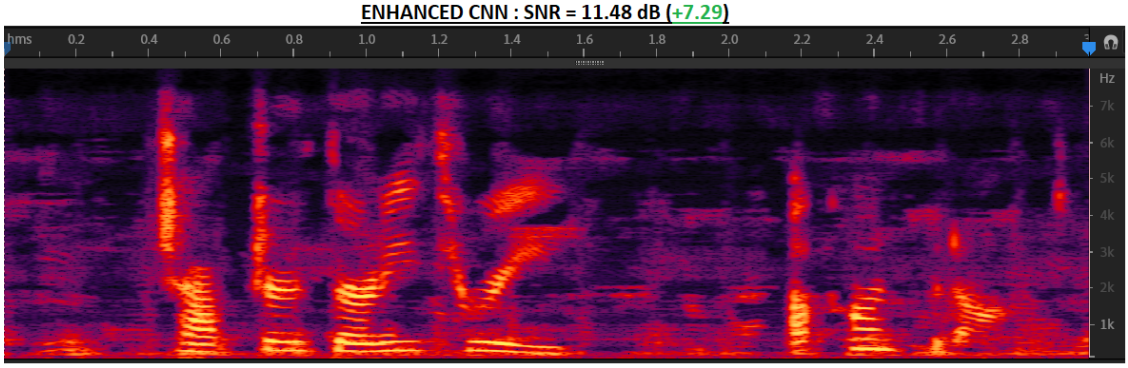


Figure 8 : Spectrogrammes d'une parole bruitée et débruitée par différentes méthodes