# Rapport Reconnaissance Vocale
# TensorFlow Speech Recognition Challenge

Margot Lacour, Paul-Hadrien Bourquin, Thomas Foltete
Adrien Pavao, Eléonore Bartenlian

Janvier 2018

## Contents

# 1 Introduction

Le but de ce projet était de faire le challenge Kaggle TensorFlow[1] Speech Recognition[2], c'est-à-dire créer un algorithme capable de reconnaître des commandes vocales simples du jeu de données Speech Commands.

Afin d'obtenir de bons résultats, nous avons tout d'abord analysé les données du problème et effectué diverses tâches de pré-processing dans le but d'affiner les performances des modèles appliqués par la suite sur ces données. Nous avons également cherché à afficher des représentations graphiques de ces données afin de nous aider à mieux comprendre à l'oeil nu ce qui distingue les mots appartenant aux différentes classes.

Dans ce rapport sont détaillés les choix effectués pour le pré-processing, les modèles implémentés et le tuning des hyper-paramètres effectués. Pour finir, nous regarderons les scores obtenus pour se faire une idée de la performance des modèles.

# 2 Les données

Le jeu de données comprend 65,000 fichiers audios d'une seconde de 30 mots courts, prononcés par des milliers de personnes différentes. La compétition consiste ensuite à les classer en 12 catégories : "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", "silence", "unknown". Les enregistrements bruts ne permettent pas de faire une classification facilement. Il est nécessaire de transformer et de retravailler les données.

Ainsi, la première action sur ces données à été de chercher à afficher l'onde et le spectrogramme de certains mots, ne serait-ce que pour avoir une idée plus concrète de la manière dont elles sont représentées.

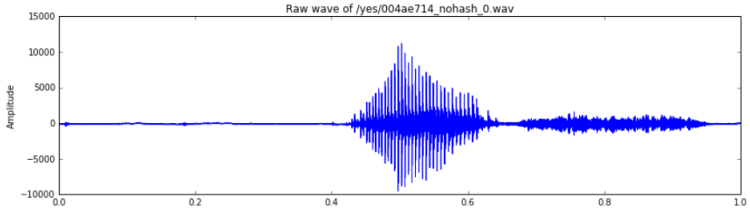


Figure 1: Onde d'un sample "yes"

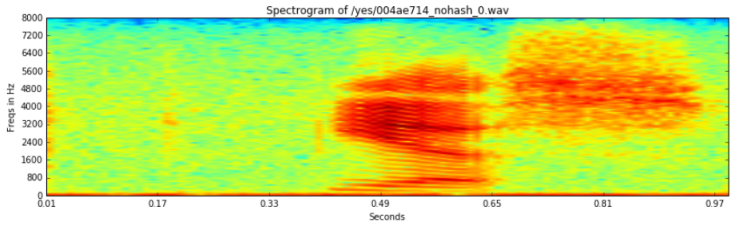


Figure 2: Spectrogramme du même sample "yes"

[1] https://www.tensorflow.org/

[2] https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

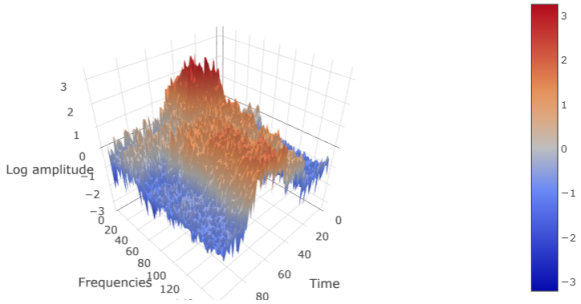


Figure 3: Spectrogramme 3D d'un sample "yes"

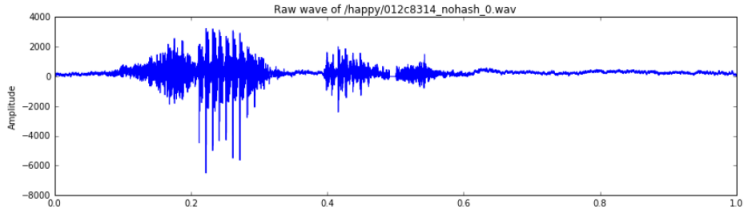


Figure 4: Onde d'un sample "happy"

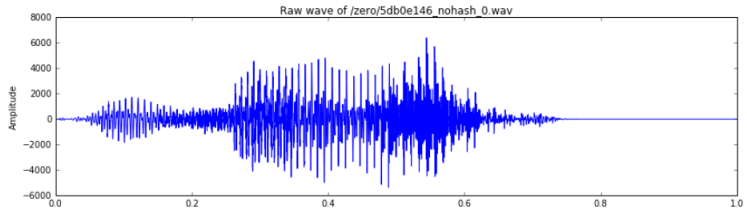


Figure 5: Onde d'un sample "zero"

On a définit une fonction qui calcule les spectogrammes. On prend les logarithmes des valeurs que l'on obtient pour des soucis de lisibilité de nos graphiques. D'après le théorème d'échantillonnage de Nyquist-Shannon[3], la fenêtre de fréquence est entre 0 et 8000 Hz. Si l'on décide d'utiliser les spectogrammes comme features d'apprentissage pour un réseau neuronal, on ne doit pas oublier de les normaliser.

On remarque qu'il est déjà possible de se faire une idée des intonations du mot prononcé en observant l'onde d'un mot. Par exemple, pour "Happy", on observera un fort pic au début pour le 'h' initial.

On remarque des silences avant et/ou après la prononciation des mots.

Nous avons également observé la répartition des classes, qui est ici relativement homogène comme le montre le graphique ci-dessous ; il n'y a pas de classe trop sur-représentée ou sous-représentée, sauf pour le background_noise, qui est une classe un peu à part.

[3] `https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_d%27%C3%A9chantillonnage`

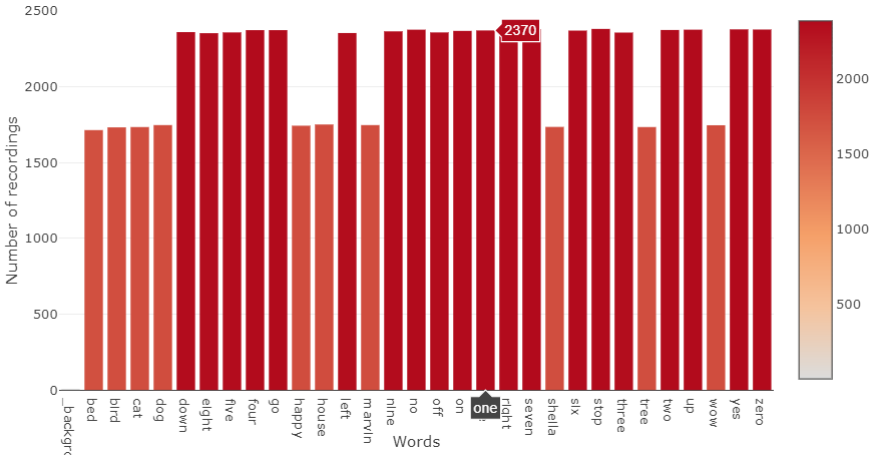


Figure 6: Nombre de mots par classe d'apprentissage

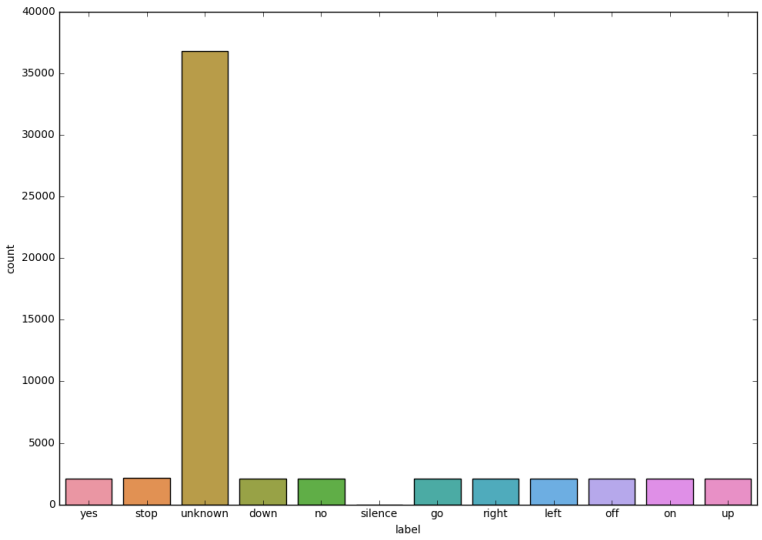


Figure 7: Distributions des classes

La métrique utilisée pour mesurer le taux de performance d'un modèle sur ces données est simplement le taux de précision des prédictions effectuées par ce modèle.

# 3 Pré-traitements

La partie de pré-traitement dans les systèmes de reconnaissance vocale est impérative et permet d'augmenter l'efficacité des étapes suivantes d'extractions d'attributs et de classification et améliore donc directement les performances.

Dans de nombreuses applications, le modèle doit être robuste à l'environnement acoustique. Le but du pré-processing est principalement d'obtenir une représentation du signal compacte (qui puisse donc être traité et rapidement) mais qui contienne néanmoins toutes les informations nécessaires pour pouvoir reconnaître la parole.

## 3.1 Sous-échantillonnage

Dans notre cas, le re-sampling va servir à réduire les dimensions.

Les données Kaggle sont échantillonnées avec une fréquence de 16k ; or, les fréquences parlées sont situées dans une bande plus étroite. Par exemple, au téléphone, les fréquences sont réduites à 8k et cela suffit à comprendre. Ainsi, nous avons pris le parti de re-sampler nos datasets à 8k.

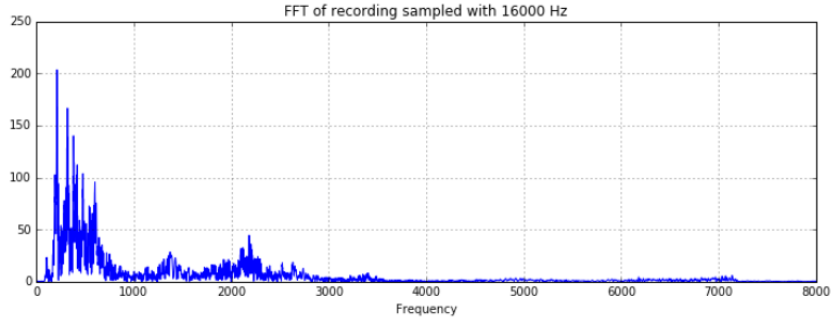


Figure 8: FFT du sample "Zero" vu plus haut, à 16000Hz

Comme on le voit sur la figure, les fréquences les plus courantes se trouvent dans la moitié gauche du graphe. Ainsi, on ne devrait normalement pas perdre d'information et cela va nous permettre de réduire considérablement le temps d'apprentissage. Même si l'on perd nécessairement des informations et cela peut faire la différence dans le cadre d'une compétition, cela sera rentable dans notre cas : nous pourrons entraîner le modèle sur plus d'époques que si nous avions gardé les samples à 16k puisque nous sommes limité dans notre matériel et en temps, et nous obtiendrons probablement des meilleurs résultats de cette manière.

Pour cela, on va commencer par calculer la FFT (transformée de Fourier Rapide)[1].

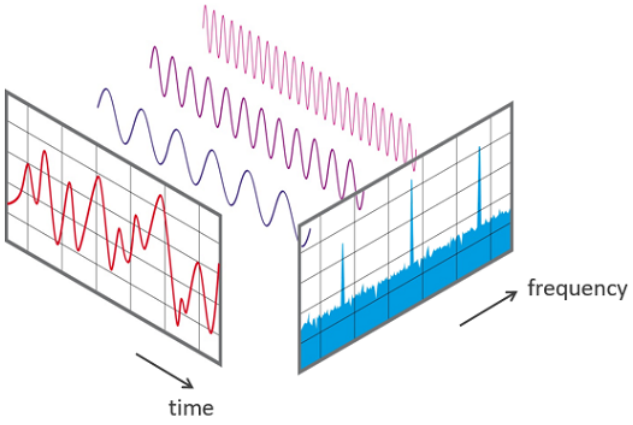


Figure 9: Signal dans le temps et en fréquence

C'est un algorithme couramment utilisé en traitement du signal. Son efficacité permet de réaliser des filtrages en modifiant le spectre et en utilisant la transformation inverse. Ainsi, après avoir calculé la FFT, nous allons enlever les fréquences qui ne nous intéressent pas (toutes celles au delà de 8000Hz) puis revenir au signal de base.

## 3.2 Fenêtrage

La parole est un signal non stationnaire qui dépend du temps. Cependant, on considère que la parole est constituée de phonèmes et donc pour la plupart de ces phonèmes les propriétés de la paroles sont

invariantes pour une petite période de temps (entre 5 et 100ms). On devrait pouvoir prendre des périodes courtes et pouvoir appliquer les méthodes de traitement du signal traditionnelles. Cependant, nous n'avons pas choisi d'explorer cette approche.

## 3.3 Suppression des silences

Même si les enregistrements de mots sont courts, il y a beaucoup de silence dedans. On a utilisé la méthode VAD pour enlever ces silences. Cela permet par exemple de réduire la taille de l'entraînement, accélérer la rapidité de la phase d'entraînement. Pour cela on coupe un morceau du début et de la fin du fichier.

## 3.4 Les spectrogrammes

On réalise la moyenne des spectrogrammes et des FFT par classe pour pouvoir en faire des caractéristiques pour l'apprentissage de notre modèle. Ci-dessous nous en avons affiché deux pour les mots "off" et "on". On peut voir que cela peut être un bon discriminant puisque l'on observe des différences entre classes. Cependant, ce n'est pas toujours le cas...

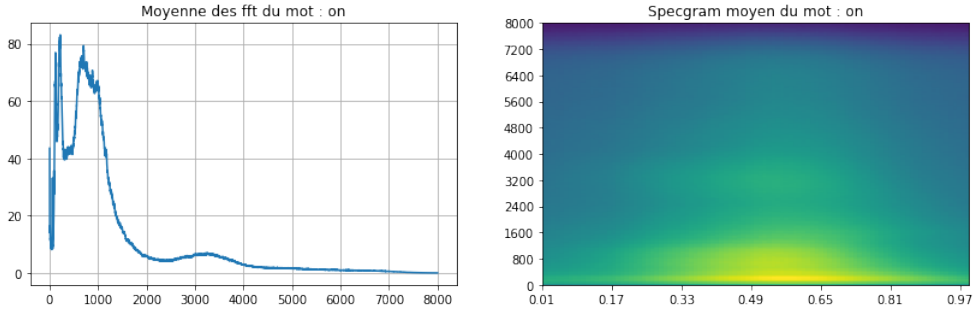


Figure 10: Moyenne FFT du mot "on"

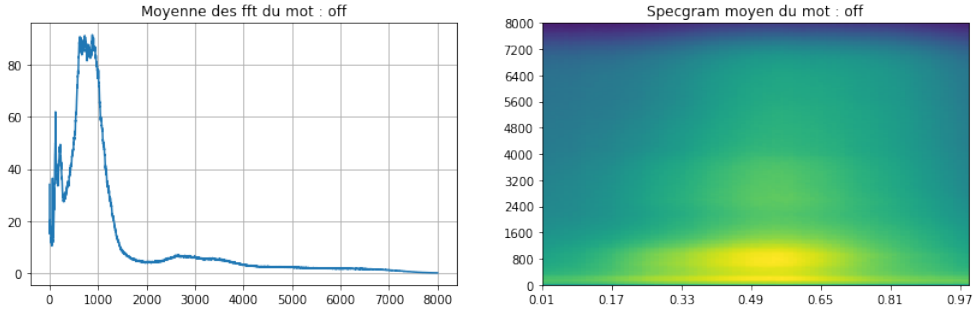


Figure 11: Moyenne FFT du mot "off"

On peut modéliser chaque FFT avec un mélange de Gaussiennes. Par contre, certaines moyennes de FFT ont l'air identiques : "stop" et "up" par exemple. Cependant, cela n'est pas trop grave car ces deux classes ont des spectrogrammes distinguables. Cela prouve la nécessité d'apprendre sur ces deux données.

## 3.5 Augmentation des données

L'augmentation des données est une stratégie commune adoptée dans le but d'augmenter la quantité de données d'entraînement, d'éviter l'overfitting et d'améliorer la robustesse des modèles.

Nous avons décidé de changer la vitesse des signaux afin de créer deux autres versions du signal original selon la méthode de Ko et al. [2] : une version ralentie de 10% et une accélérée de 10%. Cependant, nous n'avons pas réussi à améliorer nos résultats. En effet, le modèle s'entraînait trop lentement par rapport au modèle classique.

Une autre augmentation de données possible serait de bruiter légèrement les données de façon aléatoire.

# 4 Méthode Deep CNN

Les réseaux convolutionnels classiques ne capturent pas les caractéristiques temporelles des signaux audios. On va donc utiliser des réseaux récurrents profonds.

## 4.1 Modèle

Afin de façonner notre modèle, nous nous sommes plongés dans la littérature scientifique. Nous avons notamment étudié les architecture des réseaux de neurones profonds présentées dans les articles suivants : [3], [4], [5].

Quelques notions importantes pour la compréhension du modèle :

#### Couche de convolution

Un réseau de neurones à convolution (CNN) est un type de réseau de neurones artificiels acycliques. Ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations.

Lors d'un traitement convolutif les données sont découpée en petites zones : les tuiles. Chaque tuile sera traitée individuellement par un neurone artificiel. Tous les neurones ont les mêmes paramètres de réglage, légèrement décalé pour chaque champ récepteur. C'est cela que l'on appelle une convolution. Cette strate de neurones avec les mêmes paramètres est appelée "noyau de convolution".

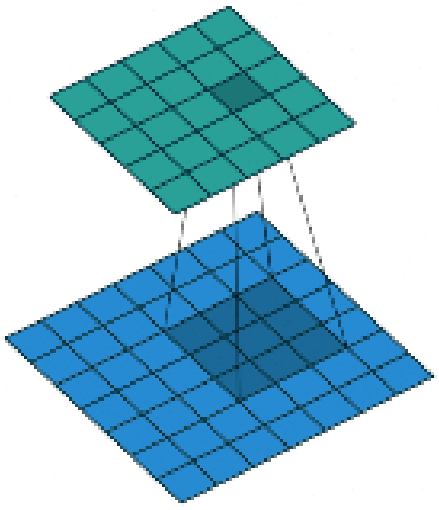

Figure 12: Schéma d'une convolution

Notre modèle comporte 5 couches de convolutions, de 16, 32, 64, 128 et 128 neurones respectivement. C'est donc un réseau profond convolutionel.

### Normalisation des batchs

Il s'agit d'une méthode de normalisation appliquée à chaque mini-batch durant la phase d'entraînement.

D'après l'article [6], cette normalisation des batchs améliore les performances du CNN.

Nous l'effectuons après chaque couche du réseau.

### Max pooling

Le pool maximum est un processus de discrétisation basé sur l'échantillonnage. L'objectif est de sous-échantillonner une représentation d'entrée (image, matrice de sortie de couche cachée, etc.), en réduisant sa dimension et permettant ainsi au modèle de faire des hypothèses sur les features dans les sous régions regroupées.

Cela sert en partie à éviter l'overfitting en fournissant une forme abstraite de la représentation, et cela minimise également le coût de calcul en réduisant le nombre de paramètres à apprendre.

Soit une matrice 4x4 représentant notre entrée initiale et un filtre 2x2 qu'on fait tourner sur notre entrée.

Pour chacune des régions représentées par le filtre, on prend le maximum de cette région et on créé une nouvelle matrice en sortie où chaque élément est le maximum d'une région de l'entrée originale :

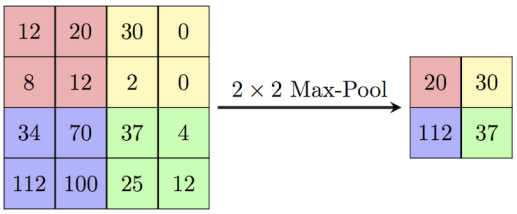


Figure 13: Max pooling

### Global average pooling

Au lieu d'adopter les couches de classification traditionnelles entièrement connectées dans le CNN, on utilise une couche de Global Average Pooling, puis le vecteur résultant est introduit dans la couche softmax [7].

### Dropout

La méthode du dropout consiste à désactiver des sorties de neurones aléatoirement avec une probabilité prédéfinie, pendant la phase d'apprentissage. Le dropout peut permettre une accélération de l'apprentissage et une réduction de l'overfitting. [8].

La technique du dropout est très courante dans les systèmes de reconnaissance de voix.

### Couches entièrement connectées

Les couches entièrement connectées sont des couches de réseaux neuronaux très courantes. Il s'agit des couches dont les neurones sont reliés à tous les neurones de la couche précédente. Ils sont notamment présent dans les réseaux feed-forward.

La couche fully connected que nous utilisons a pour fonction d'activation la fonction ReLU : c'est une couche de correction. Cette fonction augmente les propriétés non linéaires de la fonction de décision et de l'ensemble du réseau sans affecter les champs récepteurs de la couche de convolution. Cette fonction est préférable car elle rend le réseau neuronal plusieurs fois plus rapide[9], sans faire une différence significative à la généralisation de précision.

**Architecture du modèle**

Le tableau suivant résume les couches du CNN dans l'ordre. Les normalisations de batch et les max pooling ne sont pas précisés pour ne pas surcharger le schéma puisqu'il y en a à chaque couche.

| Couche | Taille | Information supplémentaire |
|---|---|---|
| Input shape | (257, 98, 2) | |
| Convolution | 16 | Kernel $3 \times 3$, Activation *elu* |
| Convolution | 32 | Kernel $3 \times 3$, Activation *elu* |
| Convolution | 64 | Kernel $3 \times 3$, Activation *elu* |
| Convolution | 128 | Kernel $3 \times 3$, Activation *elu* |
| Convolution | 128 | Kernel $1 \times 1$ |
| Global Average Pooling | | Filtre $2 \times 2$ |
| Fully connected | 256 | Activation ReLU |
| Dropout | | |
| Fully connected | 12 (output) | Activation softmax |

Figure 14: Explication de l'architecture du réseau

Voici les paramètres :

- Optimiseur : RMSprop
- Fonction de coût : Categorical cross-entropy
- Métrique : Précision

## 4.2 Implémentation

Pour l'implémentation, nous avons opté pour la librairie Python Keras. Il s'agit d'un librairie haut niveau permettant de réaliser aisément des réseaux neuronaux à l'architecture complexe.

Il y a quelques subtilités dans notre implémentation :

- L'utilisation de générateurs de mini-batch pour passer les données petit à petit car il y en a beaucoup.
- Callbacks : Sauvegarde des poids à chaque époque afin de pouvoir stopper l'apprentissage pouvant être très long, et baisse du taux d'apprentissage si le score ne s'améliore plus.
- 344 étapes par époque : on ne parcours pas toutes les données à chaque époque, mais $344 \times 64$ données. 64 est la taille des mini-batchs.

## 4.3 Résultats

Le réseau a été entraîné durant 50 époques.

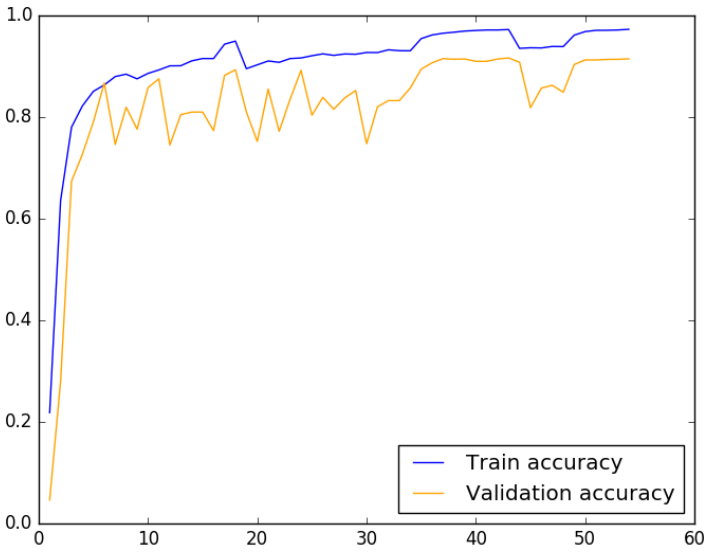


Figure 15: Précision sur l'ensemble d'entraînement et de validation au cours des 50 époques

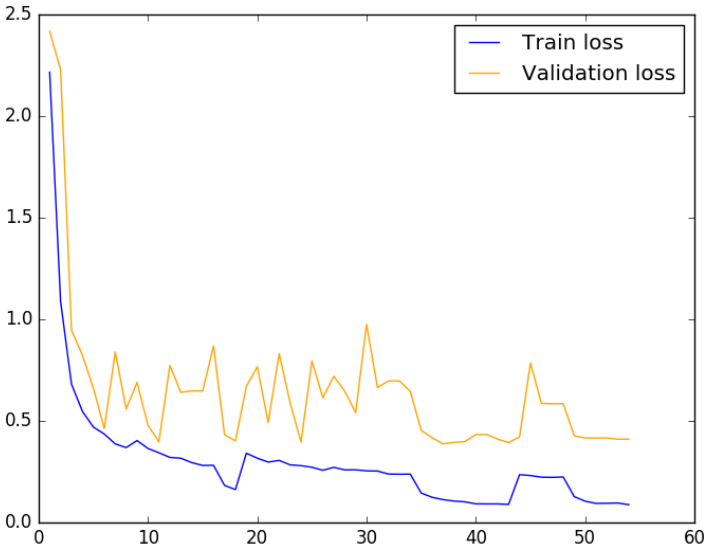


Figure 16: Fonction de coût sur l'ensemble d'entraînement et de validation au cours des 50 époques

On remarque une forte variance sur l'ensemble de validation. Il serait intéressant de baisser plus le taux d'apprentissage. Le fait de stopper et relancer le calcul par moment réinitialisait le taux d'apprentissage à sa valeur originale, gênant sa diminution progressive. Il est également possible que le modèle soit sujet à l'overfitting.

La précision obtenue lors d'une soumission Kaggle est de **0.73656**. 74% des exemples sont donc bien classifiés.

L'écart entre notre précision de validation et la précision sur l'ensemble de test de Kaggle indique qu'il faudrait faire une validation croisée afin de vérifier au mieux nos résultats avant les soumissions.

L'apprentissage du modèle a été contraint par une limitation technique : le temps de calcul. En effet,

n'ayant pas accès à des machines avec une grande puissance de calcul, l'entraînement de réseaux neuronaux profonds est difficile.

# 5 Discussion

Si l'on compare nos résultats obtenus au Leaderboard de Kaggle, on peut voir que les meilleurs modèles atteignent des taux de précision supérieurs à 91%.

La puissance de calcul à notre disposition a été un facteur particulièrement limitant. En effet, nous n'avons pas pu tester certaines architectures plus complexes et méthodes d'augmentation de données qui sont coûteuses en ressources.

Par ailleurs, les données étaient lourdes malgré les divers pré-traitements. Certains modèles n'ont donc pas pu être testés de manière efficace.

# 6 Améliorations possibles

Nous avons également pensé à de nombreuses autres méthodes et prétraitements que nous n'avons pas eu l'occasion de mettre en places. Ainsi, nous pourrions tester notre modèle neuronal avec plus d'époques. Nous pourrions également effectuer une recherche d'outliers dans les données pour les retirer afin qu'ils n'influencent pas négativement l'apprentissage. Il existe également d'autres méthodes d'augmentation des données comme l'ajout de bruit, et de nombreux pré-traitements à tester. Par exemple, on pourrait réduire les bruit dans les données : le but de cette étape est d'améliorer la qualité du signal en filtrant les bruits qui le corrompent ou en effectuant une restauration spectrale pour rendre la parole plus compréhensible.

Pour finir, nous pourrions également expérimenter d'autres structures de réseaux de neurones, peut-être plus efficace que celle que nous avons implémenté.

# Conclusion

Pour conclure, la reconnaissance de données vocales diffère des datasets sur lesquels nous sommes habitués à travailler. Cela nous a ainsi permis de découvrir de nombreuses méthodes de traitement des données que nous n'avions pas forcément mises en place auparavant.

Nous avons une bonne marge de progression concernant les performances de nos modèles. Ce projet nous a permis d'apprendre beaucoup de pré-traitements intéressants et de méthodes de Deep Learning, notamment concernant les réseaux de neurones convolutifs.

# References


[1] *Wikipedia article on FFT*. URL: https://en.wikipedia.org/wiki/Fast_Fourier_transform.

[2] Ko et al. "Audio Augmentation for Speech Recognition". In: *INTERSPEECH* (2015). URL: http://speak.clsp.jhu.edu/uploads/publications/papers/1050_pdf.pdf.

[3] William Chan et al. "Listen, Attend and Spell". In: *CoRR* abs/1508.01211 (2015). arXiv: 1508.01211. URL: http://arxiv.org/abs/1508.01211.

[4] Awni Y. Hannun et al. "Deep Speech: Scaling up end-to-end speech recognition". In: *CoRR* abs/1412.5567 (2014). arXiv: 1412.5567. URL: http://arxiv.org/abs/1412.5567.

[5] Ying Zhang et al. "Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks". In: *CoRR* abs/1701.02720 (2017). arXiv: 1701.02720. URL: http://arxiv.org/abs/1701.02720.

[6] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[7] Min Lin et al. "Network In Network". In: *arXiv:1312.4400v3* (2014). URL: https://arxiv.org/pdf/1312.4400.pdf.

[8] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research 15* (2014). URL: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf.

[9] Krizhevsky et al. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS'12* (2012). URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.