Keyce Informatique Et Intelligence Artificielle

COLLEGE DE PARIS

RAPPORT DE TRAVAIL

Intitule : Framework Pour Machine Learning

THEMES :

Reconnaissance Vocale Avec Tensorflow

Travail réaliser par

ROSNI FOMBEU

Sous la supervision de

Mr MINKOULOU ULRICH

Année Académique : 2024

Pour le compte du contrôle continu du second semestre

SOMMAIRE

LISTES DES FIGURES

INTRODUCTION GENERALE

La reconnaissance vocale est une technologie qui permet aux machines de comprendre et d'interpréter la parole humaine. Elle trouve des applications dans les systèmes d'assistance vocale, les commandes vocales, la transcription automatique, etc. Dans ce rapport, nous allons explorer comment créer un modèle de reconnaissance vocale à l'image de SIRI, GERVIS de Tony Stark dans Irone Man en utilisant TensorFlow, une bibliothèque d'apprentissage automatique open-source.

.

I. CONTEXTE D'ETUDE

La reconnaissance vocale, également connue sous le nom de reconnaissance automatique de la parole (ASR), est un domaine de l'intelligence artificielle qui vise à permettre aux machines de comprendre et d'interpréter le langage humain parlé. Elle consiste à convertir des signaux vocaux en texte, permettant ainsi aux ordinateurs de traiter et de comprendre les commandes vocales et les conversations.

Le développement de la reconnaissance vocale s'inscrit dans un large éventail de contextes et de domaines d'application, notamment :

- **Assistants personnels virtuels** : Les assistants vocaux comme Siri d'Apple, Google Assistant, Amazon Alexa et Microsoft Cortana utilisent la reconnaissance vocale pour répondre aux questions, exécuter des tâches, contrôler des appareils domestiques intelligents, et bien plus encore.
- **Transcription automatique** : La reconnaissance vocale est utilisée pour convertir des enregistrements audios en texte dans des applications telles que la transcription de réunions, d'interviews, de cours, de podcasts, etc.
- **Systèmes de dictée vocale :** Les logiciels de dictée vocale permettent aux utilisateurs de dicter du texte à leur ordinateur plutôt que de le taper, ce qui peut être utile pour les personnes ayant des difficultés à utiliser un clavier ou une souris.
- **Contrôle vocal de l'interface utilisateur :** La reconnaissance vocale peut être utilisée pour contrôler les interfaces utilisateur de divers dispositifs, tels que les téléphones, les voitures, les téléviseurs intelligents, etc., permettant aux utilisateurs de naviguer, de rechercher et d'interagir avec les appareils à l'aide de commandes vocales.
- **Applications médicales :** La reconnaissance vocale est utilisée dans le domaine médical pour la documentation des dossiers médicaux électroniques, les rapports d'opérations, la création de comptes rendus de consultations, etc.
- **Applications de sécurité :** La reconnaissance vocale peut être utilisée comme moyen d'authentification biométrique pour accéder à des appareils ou des systèmes sécurisés, offrant ainsi une sécurité renforcée par rapport aux mots de passe traditionnels.

- **Éducation et apprentissage en ligne :** Les systèmes de reconnaissance vocale peuvent être utilisés dans les environnements éducatifs pour évaluer la prononciation des apprenants de langues étrangères, pour permettre aux personnes malvoyantes d'accéder à du contenu écrit, ou pour fournir un retour d'information sur la lecture à voix haute.

En développant des systèmes de reconnaissance vocale plus précis et plus robustes, il est possible de rendre les interactions homme-machine plus naturelles, de faciliter l'accès à l'information pour les personnes ayant des capacités physiques ou sensorielles limitées, d'améliorer l'efficacité des tâches quotidiennes et professionnelles, et même d'ouvrir de nouvelles possibilités d'interaction avec la technologie dans des domaines tels que la santé, l'éducation, le divertissement et bien d'autres.

II-ETAPES DE REALISATION DU PROJET

1. Collecte des Données

- **Enregistrement Audio** : On peut nous même enregistrer nos propres échantillons audio mais vu la Quantité des données qu'il faut pour la réussite de cet exercice et le temps proposer nous n'allons pas opter pour cette option.

- **Google Speech Command :** Cette bibliothèque contient des enregistrements audio de 30 mots clés prononcés par des locuteurs divers tels que (zéro, one, two, yes, no, up, etc…). Elle est particulièrement adaptée à l'entraînement de modèles de reconnaissance de commandes vocales. La bibliothèque comprend environ 65 000 échantillons audio.

- **Ensembles de données :** Kaggle propose une vaste bibliothèque d'ensembles de données publics, couvrant de nombreux sujets tels que la santé, les finances, les transports, etc. Les utilisateurs peuvent explorer, télécharger et utiliser ces données pour leurs propres projets de data science et c'est le cas dans cette exercice nous allons devoirs télécharger le jeu de données contenant l'ensemble des données d'entrainement et de test suivant le schéma ci-dessous :

```
train/
├── audio/
│   ├── no/
│   │   ├── audio_file_1.wav
│   │   ├── audio_file_2.wav
│   │   └── ...
│   ├── yes/
│   │   ├── audio_file_1.wav
│   │   ├── audio_file_2.wav
│   │   └── ...
│   ├── one/
│   │   ├── audio_file_1.wav
│   │   ├── audio_file_2.wav
│   │   └── ...
│   ├── two/
│   │   ├── audio_file_1.wav
│   │   ├── audio_file_2.wav
│   │   └── ...
```

Figure 2 : donnée d'entrainement

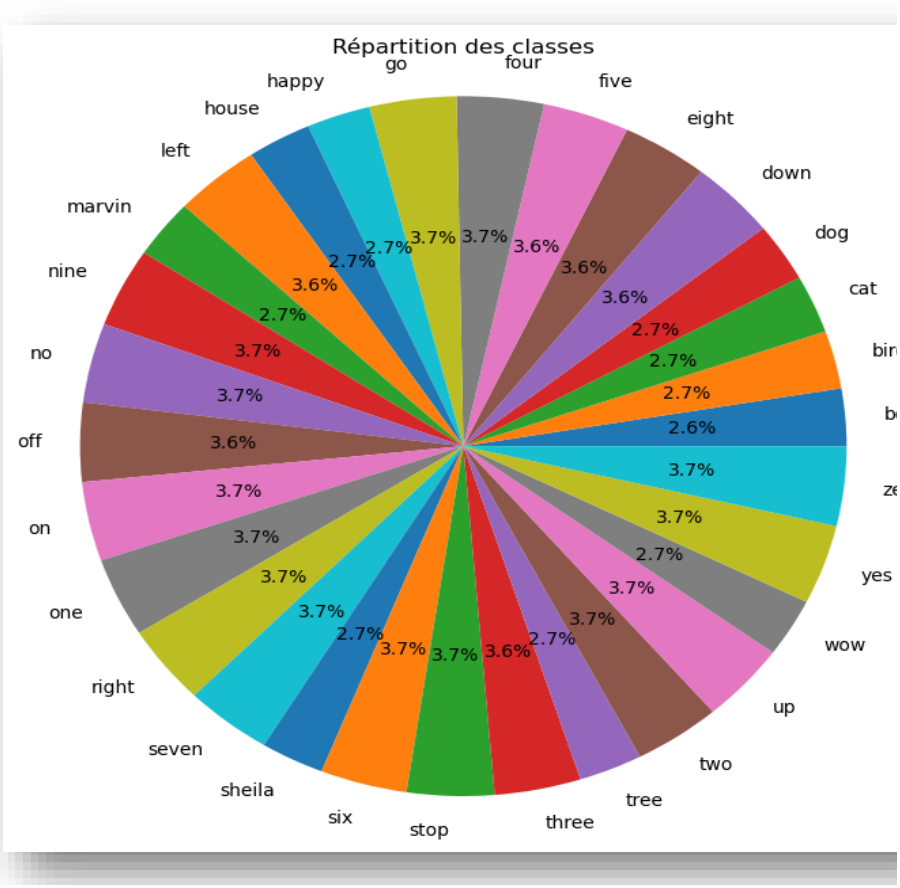
```
test/
├── audio_file_1.wav
├── audio_file_2.wav
├── audio_file_3.wav
├── audio_file_4.wav
├── audio_file_1.wav
├── audio_file_2.wav
├── audio_file_3.wav
├── audio_file_4.wav
├── audio_file_1.wav
├── audio_file_2.wav
├── audio_file_3.wav
├── audio_file_4.wav
├── audio_file_1.wav
├── audio_file_2.wav
├── audio_file_3.wav
├── audio_file_4.wav
```

Figure 1 :donnée de test

Le dossier audio contient les sous-dossiers dont les noms sont l'ensemble des commandes que nous allons traiter tout au long de notre devoir ;. Nous avons de l'autre cote un dossier test qui contient l'ensemble des fichiers audio tout mélanger qui doit nous servir lors de la phase test à noter qu'ils ne sont pas structurés comme dans le répertoire train car ici il sera question d'écouter un échantillon d'enregistrement et nous sortir la commande qui en ressort à l'intérieur.

2. Exploration et Visualisation des Données

Dans cette partie nous allons explorer l'ensemble de notre des données contenues dans le dossier train afin d'avoir une bonne compréhension sur nos données pour les manipuler facilement.



On a ici la répartition des différentes classes de notre dossier d'entrainement et comme le montre le résultat de la figure suivante on distingue au total 30 classes différentes pour notre cas d'étude

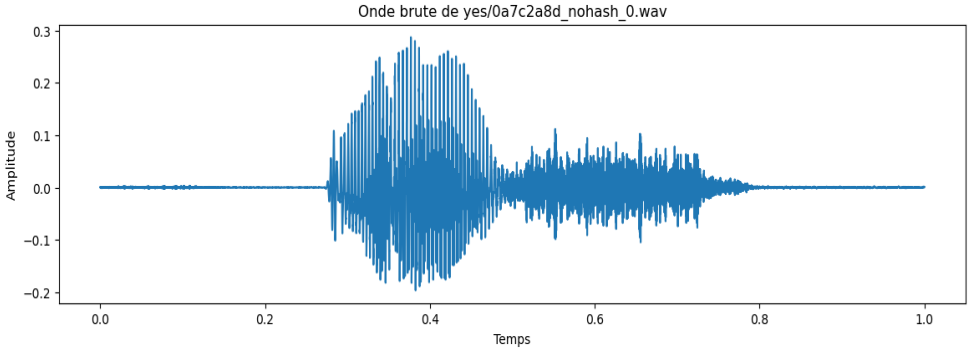


Figure 4 :visualisation des signal audio dans le temps

Ici on visualise le signal audio dans le temps afin de voir la durée sur laquelle un enregistrement s'étend pour déduire ensuite si une normalisation doit être effectuer ou permet de voir également si nos échantillons constituent des distorsions a des moments précis ou même des bruits qui pourrait affecter la qualité du signal du son.

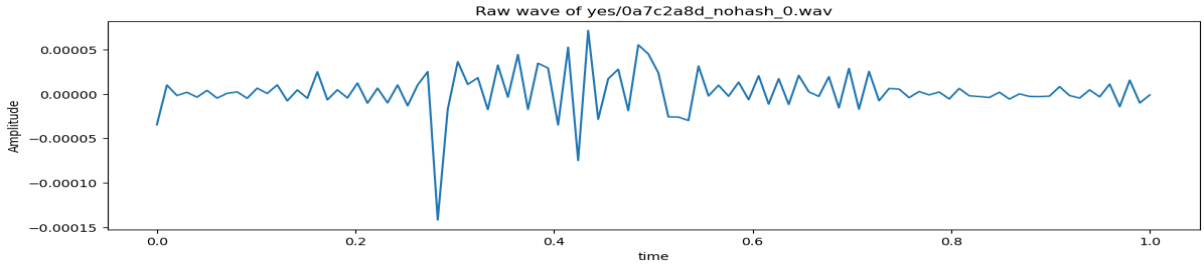


Figure 5 :visualisation détaillée

Cette visualisation détaillé est plus instructive et on voir sur ça que l'intervalle du temps [0.2,0.5] constitue la période ou la commande ressort et c'est le cas pour tous nos échantillons donc il y'aura pas de grands traitement à effectuer.

Taux d'échantillonnage

Regardons maintenant le taux d'échantillonnage des signaux audio.

Entrée [48]:
```
sample_rate=40000
samples, sample_rate = librosa.load(train_audio_path+'yes/0a7c2a8d_nohash_0.wav', sr = sample_rate)
ipd.Audio(samples, rate=sample_rate)
```
Out[48]: ▶ 0:00 / 0:01

Entrée [49]:
```
sample_rate=16000
samples, sample_rate = librosa.load(train_audio_path+'yes/0a7c2a8d_nohash_0.wav', sr = sample_rate)
ipd.Audio(samples, rate=sample_rate)
```
Out[49]: ▶ 0:00 / 0:01

Entrée [50]:
```
sample_rate=6000
samples, sample_rate = librosa.load(train_audio_path+'yes/0a7c2a8d_nohash_0.wav', sr = sample_rate)
ipd.Audio(samples, rate=sample_rate)
```
Out[50]: ▶ 0:00 / 0:01

Entrée [51]:
```
sample_rate=3000
samples, sample_rate = librosa.load(train_audio_path+'yes/0a7c2a8d_nohash_0.wav', sr = sample_rate)
ipd.Audio(samples, rate=sample_rate)
```
Out[51]: ▶ 0:00 / 0:01

Figure 6 :taux d'échantillonnage et differents song

En visualisant le taux d'échantillonnage a fréquence différente et en écoutant les différents song, on constate que les taux d'échantillonnage 16000 contient la partie du song qu'on recherche mais est un peu plus long et 6000 également mais la fin du song à ce taux ne ressort pas normalement ce qui fait que nous allons rester au bon milieu en choisissant le taux d'échantillonnage égale à 8000 pour échantillonner notre jeu de donne.

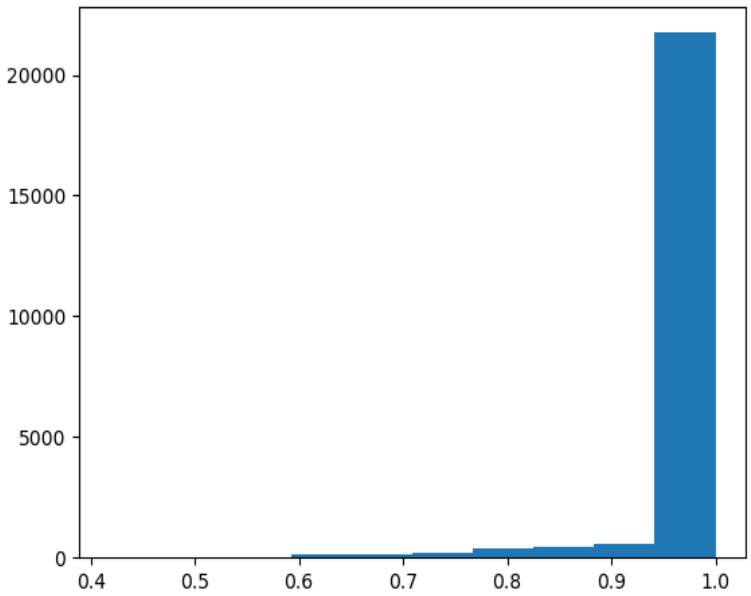


Figure 7 : différentes catégories d'enregistrement

Ici nous avons la durée des différents catégories d'enregistrement de notre jeu de donne et cela confirme également l'hypothèse selon laquelle la plupart de tous nos enregistrements étaient enregistrer sur un temps de 1seconde. Et on constate également que certains échantillons ont une durée inferieur a 1s dans la prochaine étape nous allons procéder au prétraitement des ondes audio, en effectuant un rechantillonage.

3. Prétraitement Des Ondes Audio

Dans la partie précédente sur l'exploration des données, nous avons constaté que la durée de quelques enregistrements est inférieure à 1 seconde et que le taux d'échantillonnage est trop élevé. Alors utilisons les étapes de prétraitement ci-dessous pour y remédier.

- Rééchantillonnage
- Suppression des commandes plus courtes de moins de 1 seconde

```
print(type(all_wave))
print(len(all_wave))
```

```
<class 'list'>
21312
```

```
print(len(all_wave))
print(all_wave[1].shape)
all_wave[1]
```

```
21312
(8000,)
array([0.00040267, 0.00062703, 0.00036664, ..., 0.00081659, 0.0004811 ,
       0.00071751], dtype=float32)
```

Convertir les étiquettes de sortie en encodage entier :

Apres Recheantillonage on constate qu'il ne nous reste plus 21312 enregistrement sur lesquels nous allons travailler et on voit également que nous avons définir le taux d'échantillonnage a 8000

Figure 8 : après recheantillonage

Dans la premier figure de notre devoir nous avons vu que nos différentes catégories étaient des chaines de caractère autrement dit des str () nous allons donc utiliser ses nom comme labels mais comme nous le savons bien le model que nous allons utiliser à la peine à traiter les chaine de caractère dans le processus de leurs traitement c'est pourquoi nous allons donc utiliser le OneHotEncoder pour transformer nos différents labels en des chiffres de 0 et 1. son fonctionnement consiste à mettre 1 a la place ou le label en str() convient c'est-à-dire que si on

dit labels par exemple sur la ligne de l'enregistrement si le label là-bas est one alors cette place prendra la valeur 1 et tous les autres labels seront remplacer par des 1. Nous l'avons préféré pour cette exercice car il s'agit ici du traitement des valeurs catégorielles et elle conserve l'information par rapport à son homologue LabelEncoder qui remplace jointement les labels par des valeurs entière continus

```
Entrée [62]: classes

Out[62]: ['down', 'go', 'left', 'no', 'off', 'on', 'right', 'stop', 'up', 'yes']

         Convertissons maintenant les étiquettes encodées en entiers en un vecteur one-hot, puisqu'il s'agit d'un problème de classification multi-classes :

Entrée [63]: from keras.utils import to_categorical
             y = to_categorical(y, num_classes=len(np.unique(y)))

Entrée [82]: len(y),y

Out[82]: (21312,
          array([[0., 0., 0., ..., 0., 0., 1.],
                 [0., 0., 0., ..., 0., 0., 1.],
                 [0., 0., 0., ..., 0., 0., 1.],
                 ...,
                 [0., 1., 0., ..., 0., 0., 0.],
                 [0., 1., 0., ..., 0., 0., 0.],
                 [0., 1., 0., ..., 0., 0., 0.]], dtype=float32))
```

Le schéma ci-dessus nous montre donc les changements appliquer et le fonctionnement appliquer de OneHotEncoder sur les 10 catégories que nous avons choisies pour poursuivre notre travail.

La prochaine étape de ce travail est de diviser les donnes en ensemble de donner d'entrainement et de validation

Avant de passer au découpage de nos donner essayons de faire un shape sur notre jeu de donne pour voir les dimensions qu'il possède. On verra que la dimension est de 2D (21312,8000) ce qui n'est pas bon pour la convolution que nous allons utiliser (conv1D) il faut donc appliquer un reshape sur notre jeu de donnes pour la rendre avec une dimension de 3. Après ça on pourra maintenant Appliquer la découpe de notre jeu de donne et Obtenir le résultat ci-dessous

Diviser en ensemble d'entraînement et ensemble de validation

Ensuite, nous entraînerons le modèle sur 80 % des données et validerons sur les 20 % restants :

```
[87]: X = all_wave.reshape(all_wave.shape[0], all_wave.shape[1], 1)
      X.shape
[87]: (21312, 8000, 1)

[185]: from sklearn.model_selection import train_test_split
       x_tr, x_val, y_tr, y_val = train_test_split(
           X,
           y,
           stratify=y,
           test_size=0.2,
           random_state=777,
           shuffle=True
       )
       x_tr.shape,x_val.shape,y_tr.shape,y_val.shape
185]: ((17049, 8000, 1), (4263, 8000, 1), (17049, 10), (4263, 10))
```

Figure 10 :diviser un ensemble d'entrainement et ensemble de validation

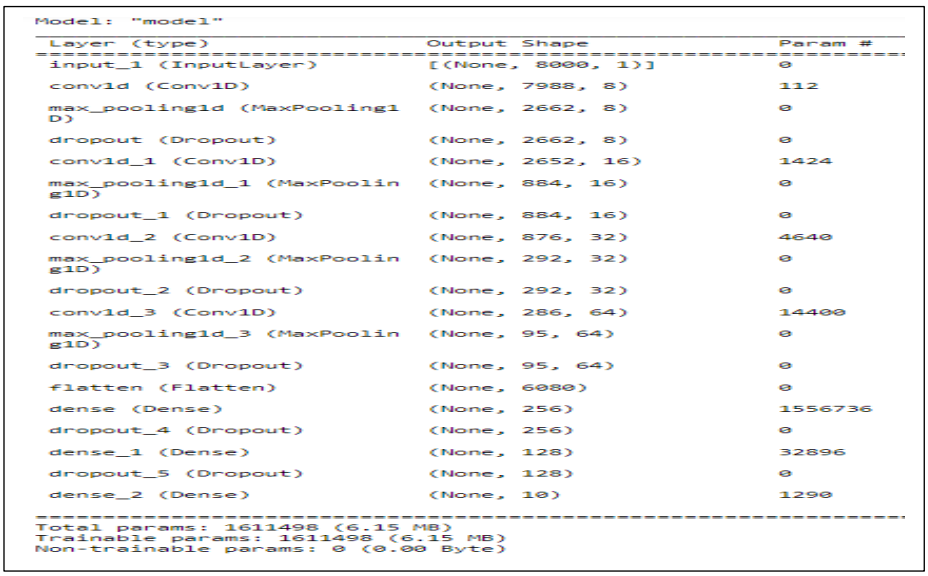
Architecture du Modèle pour ce Problème

Nous allons construire le modèle de conversion de la parole en texte en utilisant Conv1D. Conv1D est un type de réseau neuronal convolutif qui effectue la convolution le long d'une seule dimension.

Construction du Modèle

Implémentons le modèle en utilisant l'API fonctionnelle de Keras..Conv1D, Conv2D and Conv3D : https://xzz201920.medium.com/conv1d-conv2d-and-conv3d-8a59182c4d6

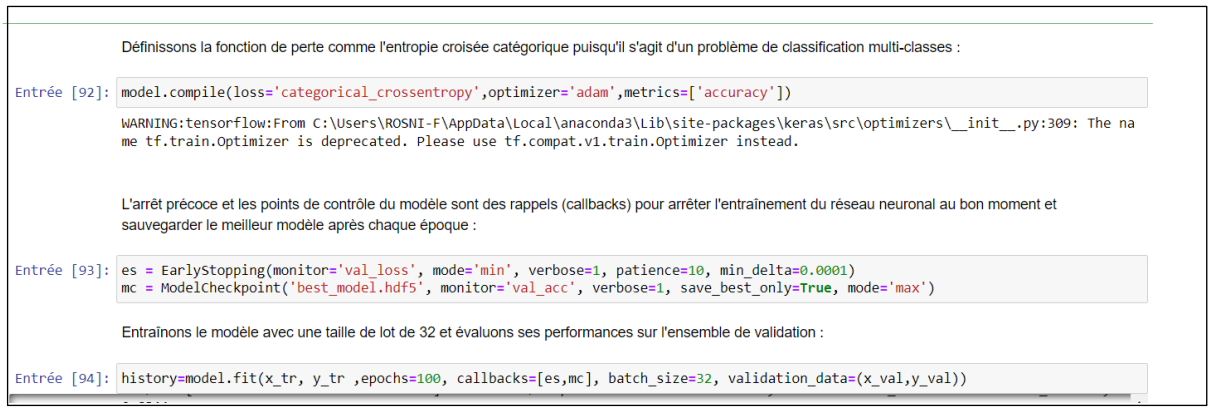
En gros cette étape consiste à utiliser un réseau neurone de convolution avec plusieurs couches d'autres enter qui vont prendre les différents images audio au format que nous avons définie lors du rechantillonage En suite les autres couches apprendrons à reconnaitre et diversifier l'ensemble des caractéristiques et en fin la couche de sortir vas donner avec une probabilité le nom de label reconnu dans le son en entier bien entendu après exécution ont donc cette sortir qui présente le nombre de couche d'entrer et de sortir avec les paramètres appropries



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 8000, 1)]         0
 conv1d (Conv1D)             (None, 7988, 8)           112
 max_pooling1d (MaxPooling1  (None, 2662, 8)           0
 D)
 dropout (Dropout)           (None, 2662, 8)           0
 conv1d_1 (Conv1D)           (None, 2652, 16)          1424
 max_pooling1d_1 (MaxPoolin  (None, 884, 16)           0
 g1D)
 dropout_1 (Dropout)         (None, 884, 16)           0
 conv1d_2 (Conv1D)           (None, 876, 32)           4640
 max_pooling1d_2 (MaxPoolin  (None, 292, 32)           0
 g1D)
 dropout_2 (Dropout)         (None, 292, 32)           0
 conv1d_3 (Conv1D)           (None, 286, 64)           14400
 max_pooling1d_3 (MaxPoolin  (None, 95, 64)            0
 g1D)
 dropout_3 (Dropout)         (None, 95, 64)            0
 flatten (Flatten)           (None, 6080)              0
 dense (Dense)               (None, 256)               1556736
 dropout_4 (Dropout)         (None, 256)               0
 dense_1 (Dense)             (None, 128)               32896
 dropout_5 (Dropout)         (None, 128)               0
 dense_2 (Dense)             (None, 10)                1290
=================================================================
Total params: 1611498 (6.15 MB)
Trainable params: 1611498 (6.15 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 11 :exécution du modèle

La prochaine étape consiste maintenant à entrainer notre model la capture ci-dessous est fournir avec les explications du choix de certains paramètres



Définissons la fonction de perte comme l'entropie croisée catégorique puisqu'il s'agit d'un problème de classification multi-classes :

Entrée [92]:
```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\ROSNI-F\AppData\Local\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309: The na
me tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

L'arrêt précoce et les points de contrôle du modèle sont des rappels (callbacks) pour arrêter l'entraînement du réseau neuronal au bon moment et sauvegarder le meilleur modèle après chaque époque :

Entrée [93]:
```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10, min_delta=0.0001)
mc = ModelCheckpoint('best_model.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
```

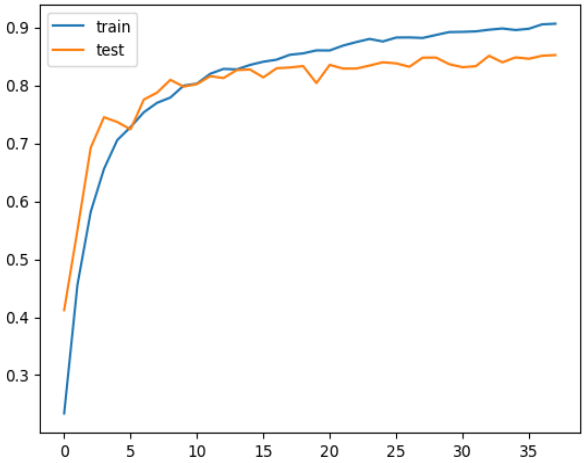
Entraînons le modèle avec une taille de lot de 32 et évaluons ses performances sur l'ensemble de validation :

Entrée [94]:
```
history=model.fit(x_tr, y_tr ,epochs=100, callbacks=[es,mc], batch_size=32, validation_data=(x_val,y_val))
```

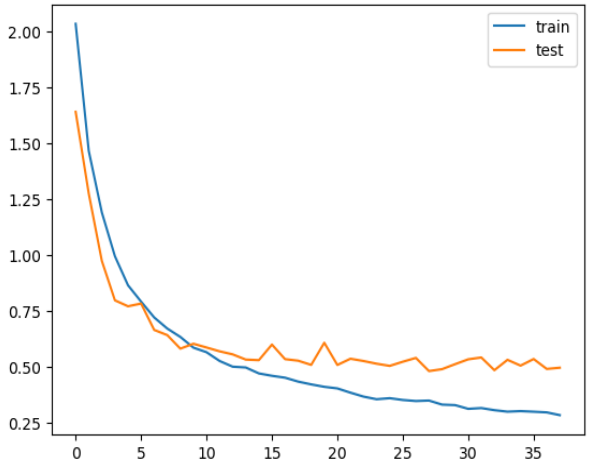
Apres exécution on aura donc le processus d'entrainement de notre model

Figure 12 :entrainement du modele

```
ly with val_acc available, skipping.
533/533 [==============================] - 46s 86ms/step - loss: 0.3006 - accuracy: 0.8952 - val_loss: 0.5042 - val_accuracy:
0.8480
Epoch 36/100
533/533 [==============================] - ETA: 0s - loss: 0.2980 - accuracy: 0.8974WARNING:tensorflow:Can save best model on
ly with val_acc available, skipping.
533/533 [==============================] - 37s 70ms/step - loss: 0.2980 - accuracy: 0.8974 - val_loss: 0.5338 - val_accuracy:
0.8456
Epoch 37/100
533/533 [==============================] - ETA: 0s - loss: 0.2951 - accuracy: 0.9050WARNING:tensorflow:Can save best model on
ly with val_acc available, skipping.
533/533 [==============================] - 39s 73ms/step - loss: 0.2951 - accuracy: 0.9050 - val_loss: 0.4894 - val_accuracy:
0.8508
Epoch 38/100
533/533 [==============================] - ETA: 0s - loss: 0.2826 - accuracy: 0.9061WARNING:tensorflow:Can save best model on
ly with val_acc available, skipping.
533/533 [==============================] - 38s 72ms/step - loss: 0.2826 - accuracy: 0.9061 - val_loss: 0.4948 - val_accuracy:
0.8520
Epoch 38: early stopping
```



Courbes Evolution de la précision pendant l'entrainement



Coubes Evolutif de la perte pendant l'entrainement

Figure 13 :évolution de la précision du jeu de donnée

Nous avons donc les graphes qui nous montre l'évolution de la précision(accuracy) du jeu de données d'entrainement et val_accuracy du jeu de donne de validation et pareil pour le loss et loss_val.

A cette étape il ne reste plus qu'à créer une fonction test sur lequel nous allons évaluer notre model d'entrainement pour notre cas nous avons utilisé dans un premier temps nos donnes de test et effectuer un random qui vas gérer un fichier.wav qui dois varier à chaque exécution de la cellule et en ressortir le label prédit et la probabilité que ce label appartient à la classe predite

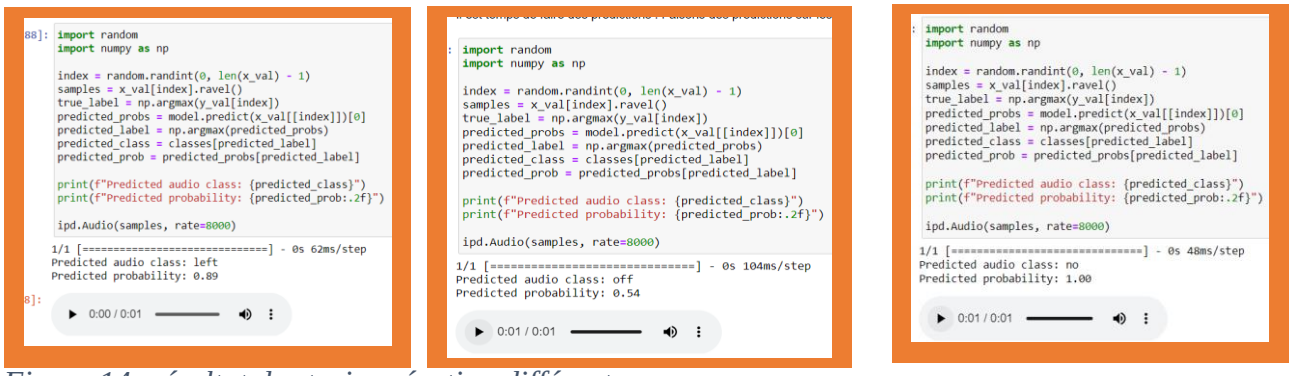


Figure 14 :résultat des trois exécution différentes

Nous avons donc le résultat de trois exécution différentes.

Une autre façon d'évaluer notre model a été de pouvoir faire nous même un enregistrement vocal et sauvegarder dans un répertoire au nom audio.wav et le charger dans notre fonction de prédiction pour voir comment ça peut fonctionner plus réellement et résultat n'était plutôt pas mal

Figure 15 : exemple de modèle entrainé

C'est donc ici que prend fin notre travail et il restera plus qu'à l'appliquer dans le cadre réel de notre quotidien et même en entreprise pour les systèmes utilisant les commandes vocales.

CONCLUSION

Dans ce projet de détection de commandes vocales, nous avons exploité la puissance des réseaux de neurones convolutionnels (CNN) et de TensorFlow pour créer un modèle capable de reconnaître et répondre efficacement à diverses commandes vocales. En transformant les échantillons audio en spectrogrammes et en les utilisant comme entrées pour notre CNN, nous avons réussi à entraîner un modèle performant avec une précision notable. Le pipeline de détection en temps réel permet une réponse instantanée aux commandes, démontrant ainsi l'efficacité de notre approche pour des applications pratiques telles que l'assistance personnelle et l'automatisation domestique. Malgré les défis liés à la variabilité des accents et des bruits de fond, notre modèle a montré une robustesse significative, et les futures améliorations pourraient encore augmenter sa précision et sa polyvalence. Ce projet illustre ainsi le potentiel des CNNs et de TensorFlow dans le domaine de la reconnaissance vocale, ouvrant la voie à des systèmes interactifs intelligents et évolutifs.