تمييز الصوت بالاعتماد على التعلم العميق

Speech Recognition Using Deep Learning

الدكتور: مهند عيسى

الطلاب: محمد سلامي، مايا فارس

٢٠٢٢/٦/١٥

□ ملخّص □

"Hey Google. What's the weather like today"

أو بالعربية: "مرحباً غوغل، ما هي حالة الطقس اليوم ؟"

سيبدو هذا مألوفا لأي شخص يمتلك هاتفا ذكيا في العقد الماضي. لا أستطيع أن أتذكر آخر مرة أخذت من الوقت لكتابة الاستعلام بأكمله على البحث جوجل. أنا ببساطة أسأل السؤال – وجوجل يحدد نمط الطقس بأكمله بالنسبة لي. إن التعرف على الكلام هو قدرة جهاز أو برنامج على تحديد الكلمات والعبارات باللغة المنطوقة وتحويلها لتنسيق قابل للقراءة بواسطة الآلة. ويتضمن التعرف على الكلام التقاط الموجات الصوتية وتحويلها إلى ملفات رقمية وتحويلها إلى وحدات لغة أساسية أو "phonemes صوتيات " وبناء لضمان التهجئة الصحيحة للكلمات التي تبدو متشابهة.

الكلمات المفتاحية: معالجة اللغة الطبيعية (NLP) الصوت، خلايا عصبونية، التعلم العميق، التعامل مع الإشارات الصوتية، البايثون، الذكاء الاصطناعي.

Github

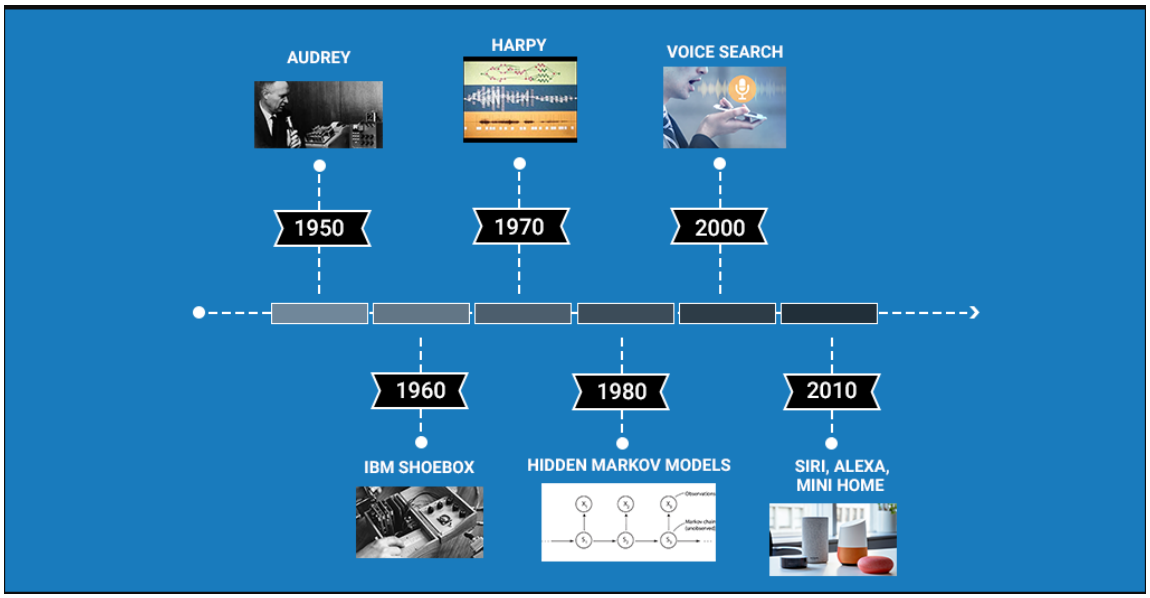
Mohamed-Salami/Sound-Recognition-Project: Sound Recognition depending on deep learning (github.com)

مقدمة:

تاريخ موجز للتعرف على الكلام عبر العقود:

يجب أن تكون على دراية بأنظمة التعرف على الكلام. إنها منتشرة في كل مكان هذه الأيام – من Siri من Apple إلى مساعد Google. هذه كلها تطورات جديدة على الرغم من التطورات السريعة في التكنولوجيا.

إن استكشاف التعرف على الكلام يعود إلى الخمسينيات من القرن الماضي؟ هذا صحيح – هذه الأنظمة موجودة منذ أكثر من ٥٠ عامًا! عن طريق هذا الجدول الزمني المصور ستفهم بسرعة كيف تطورت أنظمة التعرف على الكلام على مر العقود:



الشكل ١: خط زمني يوضح مراحل التعرف على الصوت.

- تم تطوير أول نظام للتعرف على الكلام AUDREY، في عام ١٩٥٢ من قبل مختبرات بيل. تم تصميم أودري للتعرف على الأرقام فقط.
- بعد ١٠ سنوات فقط، قدمت شركة IBM أول نظام للتعرف على الكلام IBM Shoebox، والذي كان قادرًا على التعرف على ١٦ كلمة بما في ذلك الأرقام.
- ساهمت وكالة مشاريع الأبحاث الدفاعية المتقدمة (DARPA) في برنامج يسمى Speech Understanding Research وأخيراً، تم تطوير Harpy الذي كان قادرًا على التعرف على ١٠١١ كلمة. لقد كان إنجازًا كبيرًا في ذلك الوقت.
- في الثمانينيات، تم تطبيق Hidden Markov Model (HMM) على نظام التعرف على الكلام. HMM هو نموذج إحصائي يستخدم لنمذجة المشكلات التي تتضمن معلومات متسلسلة. يتمتع بسجل حافل في العديد من تطبيقات العالم الحقيقي.
- في عام ٢٠٠١، قدمت Google تطبيق البحث الصوتي الذي سمح للمستخدمين بالبحث عن الاستعلامات من خلال التحدث إلى الجهاز. كان هذا هو أول تطبيق يدعم الصوت من

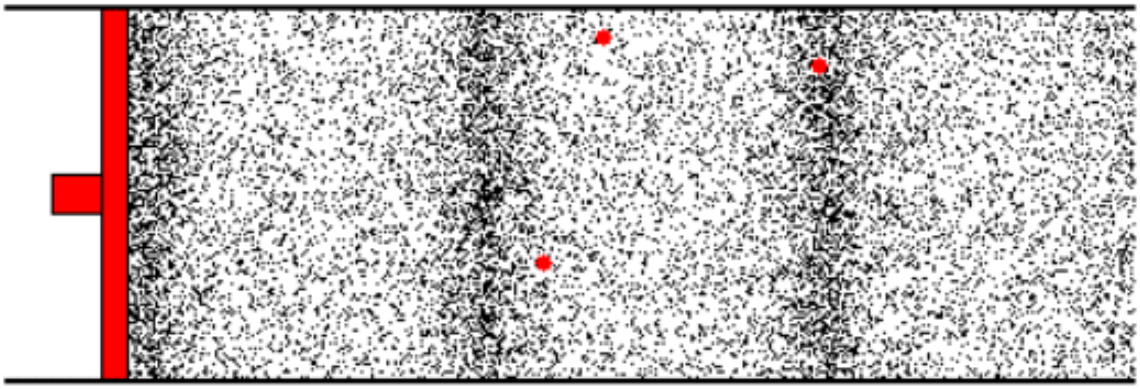
المستخدم مباشرة والذي حظي بشعبية كبيرة بين الناس. جعلت المحادثة بين الناس والآلات أسهل كثيرًا.

- تطبيقات Apple تطبيق Siri و Alexa من Amazon الذي قدم طريقة فورية وأسرع وأسهل للتفاعل مع أجهزة Apple من خلال استخدام صوتك فقط.

مقدمة في معالجة الإشارات:

الإشارة الصوتية

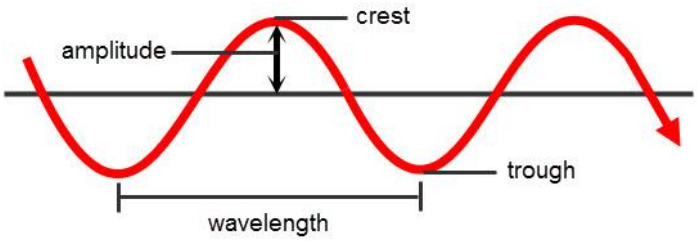
أي جسم يهتز ينتج موجات صوتية. إذا تساءلنا يومًا كيف يمكننا سماع صوت شخص ما؟ إنه بسبب الموجات الصوتية. عندما يهتز جسم ما، تتأرجح جزيئات الهواء ذهابًا وإيابًا من وضع الراحة وتنقل طاقتها إلى الجزيئات المجاورة. ينتج عن هذا انتقال الطاقة من جزيء إلى آخر ينتج بدوره موجة صوتية.



الشكل ٢: إنضغاطات الإشارة الصوتية.

بارمترات الإشارة الصوتية:

المطال: يشير المطال إلى أقصى إزاحة لجزيئات الهواء من موضع السكون. القمة والقاع: القمة هي أعلى نقطة في الموجة بينما القاع هو أدنى نقطة. طول الموجة: تُعرف المسافة بين قمتين أو قاع متتاليين بالطول الموجي. الدور: تشكل حركة واحدة كاملة صعودًا وهبوطًا للإشارة دورة. التردد: يشير التردد إلى مدى سرعة تغير الإشارة خلال فترة زمنية.



الشكل ٣: بارامترات الإشارة الصوتية

لدينا أنواع مختلفة من الإشارات، وهي الرقمية والتشابهية. الرقمية هي تمثيل منفصل للإشارة على مدى فترة زمنية. هنا، يوجد عدد محدود من العينات بين أي فترتين زمنيتين. التشابهية هي تمثيل مستمر للإشارة على مدى فترة زمنية. في الإشارة التناظرية، يوجد عدد لا حصر له من العينات بين أي فترتين زمنيتين، مثل الإشارة الصوتية.

أهمية البحث وأهدافه:

نظراً للدور البارز الذي يلعبه الذكاء الاصطناعي في تكنولوجيا اليوم، من المهم أن نعرف كيفية بناء الشبكات العصبونية بشكلها الكامل التي تقوم بالتعرف على الصوت باستخدام التدريب والتعلم العميق بدايةً من التقاط الأمواج الصوتية ومعالجتها مما يمكن الشبكة العصبونية من التعامل معها والتدرب من خلالها لبناء نموذج حي من الذكاء الاصطناعي المعتمد على التعلم العميق .

طرائق البحث ومواده:

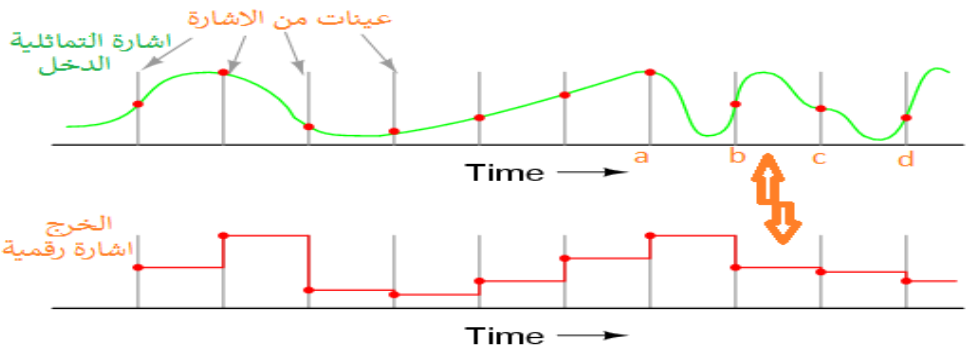
أخذ عينات الإشارة

الإشارة الصوتية هي تمثيل مستمر للسعة لأنها تتغير بمرور الوقت.

الإشارات التناظرية هي استنزاف للذاكرة لأنها تحتوي على عدد لا حصر له من العينات ومعالجتها تتطلب الكثير من الناحية الحسابية. لذلك ، نحتاج إلى تقنية لتحويل الإشارات التناظرية إلى إشارات رقمية حتى نتمكن من العمل معها بسهولة.

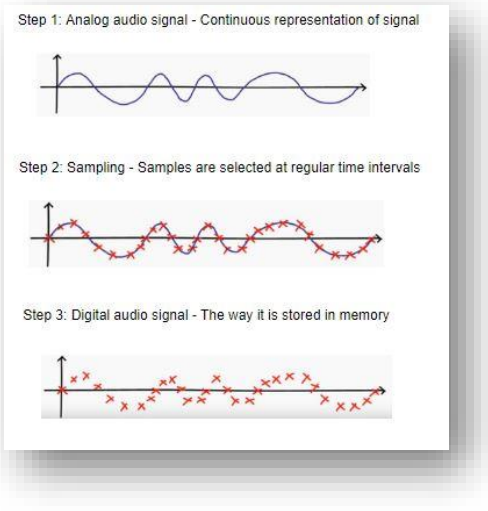
أخذ عينات الإشارة هو عملية تحويل إشارة تناظرية إلى إشارة رقمية عن طريق اختيار عدد معين من العينات في الثانية من الإشارة التناظرية.

يتم تعريف معدل أخذ العينات أو تكرار أخذ العينات على أنه عدد العينات المختارة في الثانية.



تقنيات استخراج الميزات المختلفة من إشارة صوتية

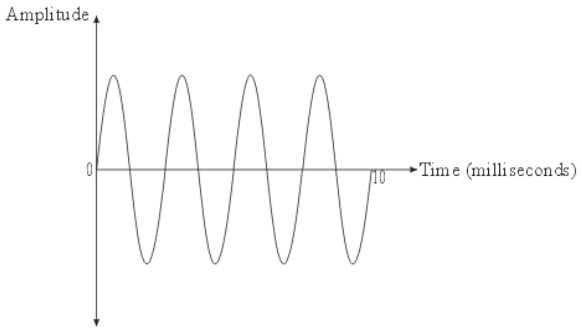
تتمثل الخطوة الأولى في التعرف على الكلام في استخراج الميزات من الإشارة الصوتية التي سنقوم بإدخالها إلى نموذجنا لاحقًا. الآن، سنعرض الطرق المختلفة لاستخراج الميزات من الإشارة الصوتية:



الشكل ٤: طريقة أخذ العينات

المجال الزمني:

هنا، يتم تمثيل الإشارة الصوتية من خلال المطال كتابع للزمن. بكلمات بسيطة، إنها ربط بين المطال والزمن. الميزات هي المطالات (السعات) التي يتم تسجيلها على فترات زمنية مختلفة.



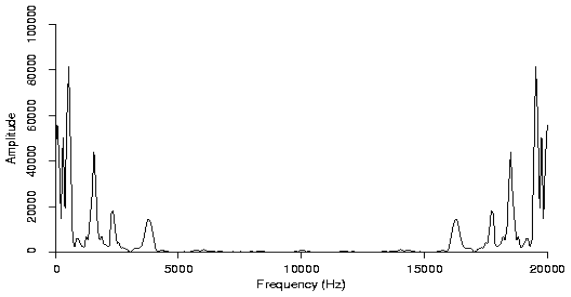
الشكل ٥: المجال الزمني.

يتمثل الحد من تحليل المجال الزمني في أنه يتجاهل تمامًا المعلومات المتعلقة بمعدل الإشارة التي يعالجها تحليل مجال التردد. لذلك دعونا نناقش ذلك في القسم التالي.

المجال الترددي:

في مجال التردد، يتم تمثيل الإشارة الصوتية بالمطال كتابع للتردد. الميزات هي المطالات المسجلة بترددات مختلفة.

يتمثل الحد من تحليل مجال التردد هذا في أنه يتجاهل تمامًا ترتيب أو تسلسل الإشارة التي يتم تناولها من خلال تحليل المجال الزمني.

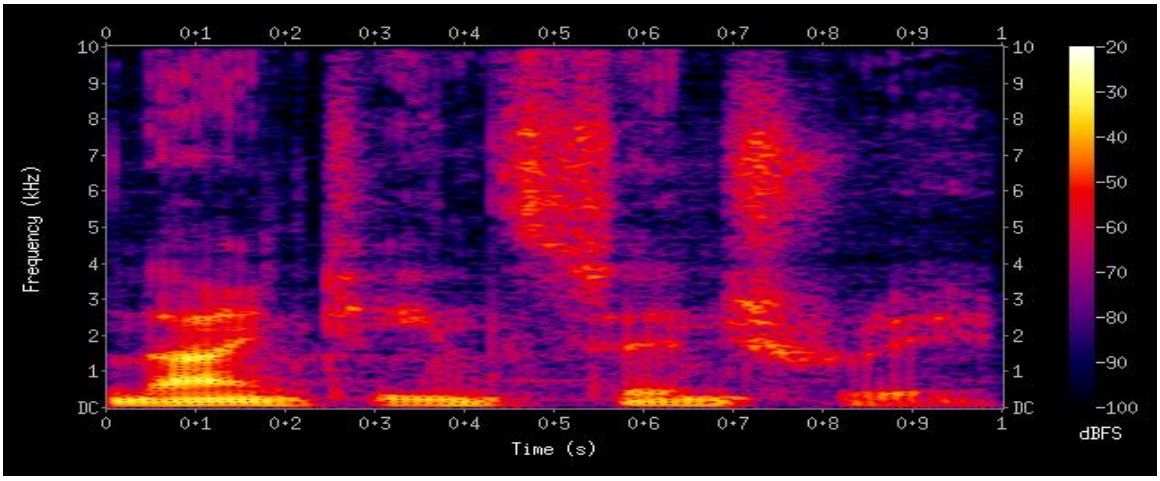


الشكل ٦: المجال الترددي.

تذكير: يتجاهل تحليل المجال الزمني تمامًا مكون التردد بينما لا يولي تحليل مجال التردد أي اهتمام لمكون الوقت. يمكننا الحصول على الترددات المعتمدة على الوقت بمساعدة مخطط طيفي(spectrogram).

مخطط طيفي:

إنه تمثيل ثنائي الأبعاد بين الوقت والتردد حيث تمثل كل نقطة في الرسم سعة تردد معين في وقت معين من حيث كثافة اللون. بعبارات بسيطة، يعد المخطط الطيفي نطاقًا واسعًا (نطاقًا واسعًا من الألوان) للترددات لأنها تختلف بمرور الوقت

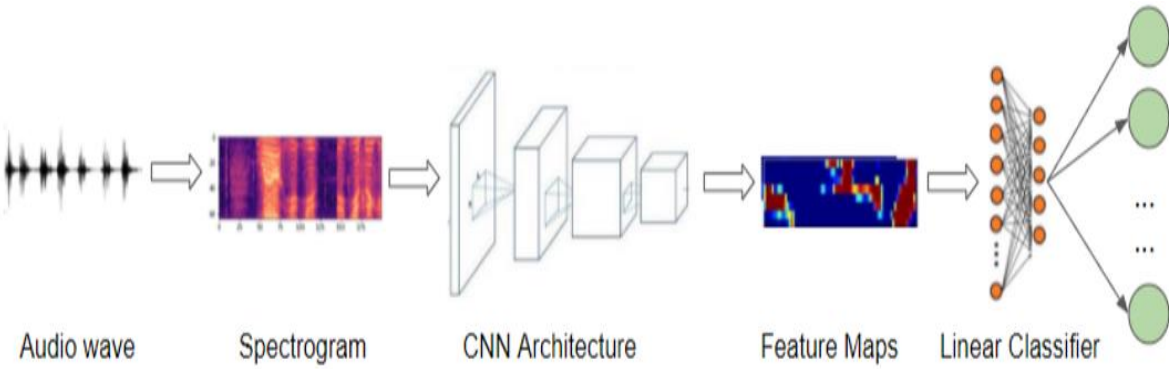


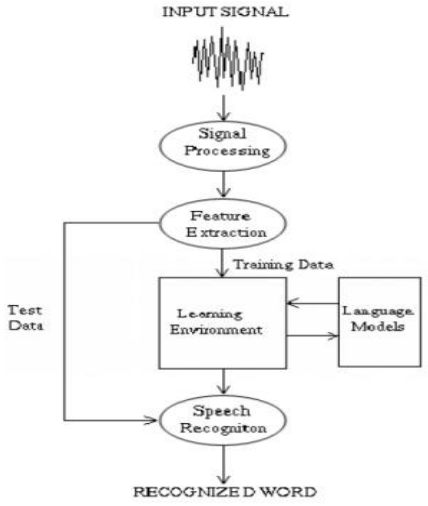
الشكل ٧: مخطط طيفي.

بناء الشبكة العصبونية والتعامل معها:

طريقة الشبكة العصبونية في التدرب والتعرف على الكلام تتوضح في الصورة التالية:

بداية من استقبال الصوت ومعالجة الإشارة واستخراج الميزات منها (الترددية والزمنية والمطالية) ثم إدخالها الى بيئة التعلم.





الشكل ٨: طريقة التدريب في الشبكة العصبونية

الشكل ٩: خوارزميات التعامل مع الصوت واستخراج الميزات.

الشبكات العصبية:

السماة الأساسية للشبكات العصبية:

١. تتكون من عدد كبير من وحدات المعالجة.
٢. يتم انشاء وصلات بين كل وحدة ومعالجة وأخرى (fully connected)
٣. يتم حساب وزن كل وصلة والتعامل معها على أساس الوزن(الكلفة)
٤. اجراء التدريب عن طرق التعامل مع الوصلات وكلفتها واختيار الأنسب لها .

وحدات المعالجة :

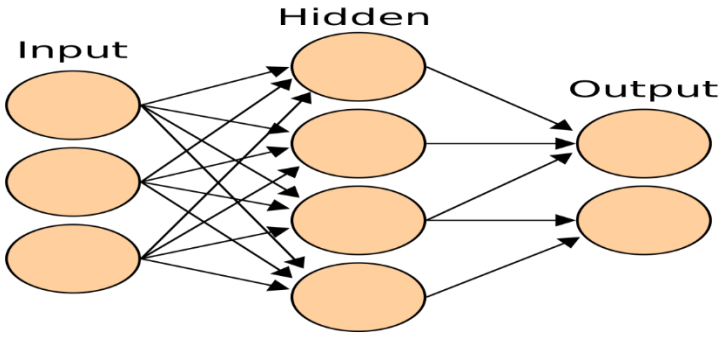
تحتوي الشبكة العصبية على عدد هائل من وحدات المعالجة, مثل الخلايا العصبية في الدماغ حيث تعمل كل هذه الوحدات في نفس الوقت بشكل متوازٍ, حيث تقم بكل العمليات الحسابية في النظام من دون وجود أي معالج يقوم بالإشراف عليها.

في كل لحظة زمنية , تقوم كل وحدة معالجة بحساب تابع رقمي للمدخلات المحلية الخاصة بها ,

وتبث النتيجة وتسمى قيمة التنشيط الى الوحدات المجاورة لها .

يتم تقسيم وحدات المعالجة في الشبكة الى:

- وحدات معالجة للادخال تستقبل البيانات من البيئة المحيطة لها (المعلومات الحسية الأولية) .
- وحدات معالجة مخفية التي تتعامل مع البيانات داخلياً.
- وحدات معالجة للإخراج والتي تمثل القرارت واشارات التحكم .



الشكل ١٠: بنية الشبكة العصبونية

الاتصالات:

يتم تنظيم الوحدات في الشبكة في هيكل معين بواسطة مجموعة من الأوزان، كخطوط في مخطط. لكل وزن قيمة حقيقية، تتراوح عادةً من – الحد حتى +الحد، على الرغم من أن النطاق يكون محدو ًدا في بعض الأحيان .

تصف قيمة أو قوة الوزن مقدار تأثير الوحدة على جارتها , حيث يتسبب الوزن الموجب في إثارة وحدة أخرى، بينما يتسبب الوزن السالب في منع وحدة أخرى.

الأوزان تكون عادة أحادية الاتجاه من وحدات الإدخال باتجاه وحدات الإخراج ولكنها قد تكون ثنائية الاتجاه خاصة عندما يكون هناك تمييز بين وحدات اللإدخال وحدات الإخراج.

إن قيم كل الأوزان تحدد مسبقا عن طريق التدريب و العمليات الحسابية في الشبكة مع أي نمط إدخال وبالتالي فإن الأوزان تمثل الذاكرة أو المعرفة للشبكة.

قد تتغير الأوزان نتيجة للتدريب، ولكنها تميل إلى التغيير ببطء، لأن المعرفة المتراكمة تتغير ببطء.

الحساب:

يبدأ الحساب دائمًا بتقديم نمط إدخال للشبكة، أو تثبيت نمط تنشيط على وحدات الإدخال ثم يتم حساب

تنشيط جميع الوحدات المتبقية إما بشكل متزامن(كل ذلك في نظام متوازٍ)أو بشكل غير متزامن(واحد في كل مرة، إما بترتيب عشوائي أو ترتيب طبيعي)، كما هو الحال في الشبكات غير الهيكلية، تسمى هذه العملية نشر التنشيط؛ في الشبكات متعددة الطبقات تسمى إعادة التوجيه ,

و الانتشار مع تقدمه من طبقة الإدخال إلى طبقة الإخراج.

التدريب:

يعني تدريب الشبكة، بمعناها العام، تكييف اتصالاتها بحيث تعرض الشبكة السلوك الحسابي المرغوب فيه لكل أنماط المدخلات .وعادة ما تتضمن العملية تعديل الأوزان ؛ ولكنها تتضمن أحيانًا أيضًا تعديل الهيكل الفعلي للشبكة، أي إضافة أو حذف اتصالات من الشبكة .

أي يعتبر تعديل الوزن أكثر عمومية من تعديل الهيكل، حيث يمكن للشبكة ذات الاتصالات الوفيرة أن تتعلم تعيين أي من أوزانها إلى صفر، والذي له نفس التأثير الذي يؤدي إلى حذف هذه الأوزان .ومع ذلك، يمكن أن تحسن التغيرات الهيكلية كلاً من التعميم وسرعة التعلم، وذلك من خلال تقييد فئة الوظائف التي يمكن للشبكة تعلمها

- يمكن التحكم في هذا عن طريق تعديل معدل التعلم.
- نوع شبكات التغذية التي تستخدم التعلم الخاضع للإشراف.

النتائج والمناقشة:

تنفيذ نموذج تحويل الكلام إلى نص في لغة بايثون:

تم تنفيذ النموذج العملي **على منصة collab** يمكن الاطلاع على الرابط
https://colab.research.google.com/drive/1Zj2dfq54TyDo1e-s0xEzJme5FVzTCSPK?usp=sharing

- نقوم باستيراد المكتبات:

أولاً ، قم باستيراد جميع المكتبات الضرورية إلى دفتر ملاحظاتنا. LibROSA و SciPy هما مكتبات Python المستخدمة لمعالجة الإشارات الصوتية.

```
1  import os
2  import librosa   #for audio processing
3  import IPython.display as ipd
4  import matplotlib.pyplot as plt
5  import numpy as np
6  from scipy.io import wavfile #for audio processing
7  import warnings
8  warnings.filterwarnings("ignore")
```

modules.py hosted with ♥ by GitHub view raw

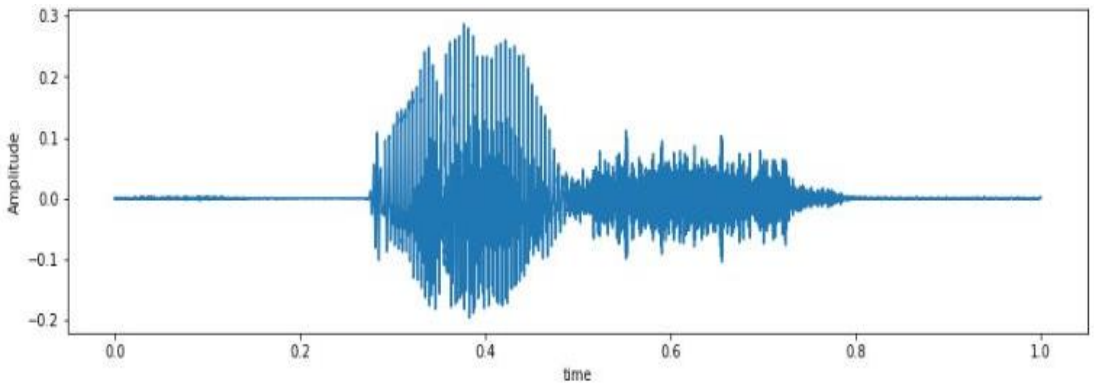
- استكشاف البيانات :

يساعدنا استكشاف البيانات والتصور على فهم البيانات وكذلك خطوات المعالجة المسبقة بطريقة أفضل.

تصور الإشارة الصوتية في مجال السلاسل الزمنية:

الآن ، سنتصور الإشارة الصوتية في نطاق السلاسل الزمنية:

```
train_audio_path = '../input/tensorflow-speech-recognition-challenge/train/audio/'
samples, sample_rate = librosa.load(train_audio_path+'yes/0a7c2a8d_nohash_0.wav', sr = 16000)
fig = plt.figure(figsize=(14, 8))
ax1 = fig.add_subplot(211)
ax1.set_title('Raw wave of ' + '../input/train/audio/yes/0a7c2a8d_nohash_0.wav')
ax1.set_xlabel('time')
ax1.set_ylabel('Amplitude')
ax1.plot(np.linspace(0, sample_rate/len(samples), sample_rate), samples)
```



- معدل أخذ العينات:

دعونا الآن نلقي نظرة على معدل أخذ العينات للإشارات الصوتية:

```
ipd.Audio(samples, rate=sample_rate)
print(sample_rate)
```

- إعادة أخذ العينات :

مما سبق يمكننا أن نفهم أن معدل أخذ العينات للإشارة هو ١٦٠٠٠ هرتز. سوف نعيد أخذ عينات منه إلى ٨٠٠٠ هرتز لأن معظم الترددات المتعلقة بالكلام موجودة عند ٨٠٠٠ هرتز:

```
samples = librosa.resample(samples, sample_rate, 8000)
ipd.Audio(samples, rate=8000)
```

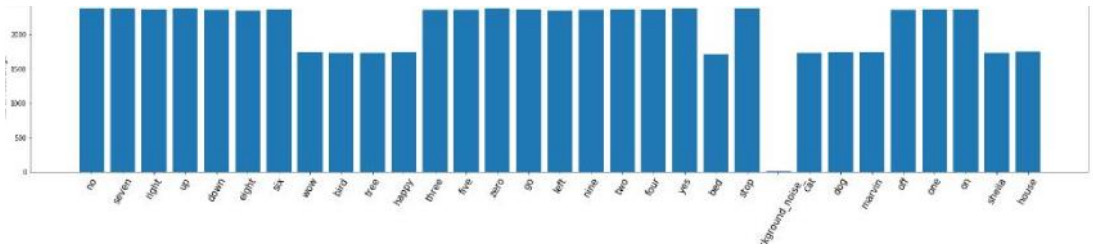
الآن ، سنفهم عدد التسجيلات لكل أمر صوتي:

```
labels=os.listdir(train_audio_path)

#find count of each label and plot bar graph
no_of_recordings=[]
for label in labels:
    waves = [f for f in os.listdir(train_audio_path + '/'+ label) if f.endswith('.wav')]
    no_of_recordings.append(len(waves))

#plot
plt.figure(figsize=(30,5))
index = np.arange(len(labels))
plt.bar(index, no_of_recordings)
plt.xlabel('Commands', fontsize=12)
plt.ylabel('No of recordings', fontsize=12)
plt.xticks(index, labels, fontsize=15, rotation=60)
plt.title('No. of recordings for each command')
plt.show()

labels=["yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go"]
```



Table

A Brie
throu

Introc

Differ
for an

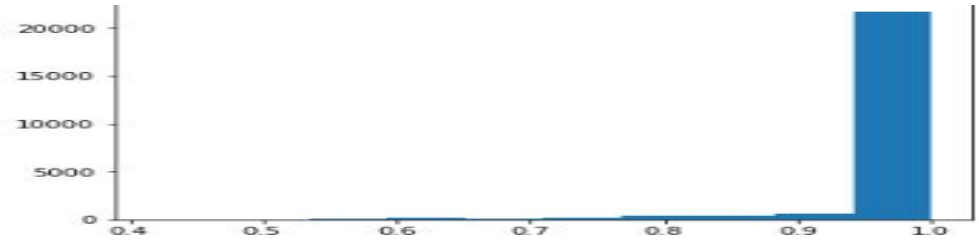
Under

- مدة التسجيلات:

بالنظرة على توزيع مدة التسجيلات

```
duration_of_recordings=[]
for label in labels:
    waves = [f for f in os.listdir(train_audio_path + '/'+ label) if f.endswith('.wav')]
    for wav in waves:
        sample_rate, samples = wavfile.read(train_audio_path + '/' + label + '/' + wav)
        duration_of_recordings.append(float(len(samples)/sample_rate))

plt.hist(np.array(duration_of_recordings))
```



- المعالجة المسبقة للموجات الصوتية:

في جزء استكشاف البيانات سابقًا ، رأينا أن مدة التسجيلات قليلة أقل من ثانية واحدة وأن معدل أخذ العينات مرتفع للغاية. لذا ، سنقرأ الموجات الصوتية ونستخدم خطوات المعالجة أدناه للتعامل مع هذا:

فيما يلي الخطوتان اللتان سنتبعهما:

١. إعادة أخذ العينات.
٢. إزالة الأوامر الأقصر التي تقل عن ثانية واحدة.

سنحدد خطوات المعالجة المسبقة:

```
train_audio_path = '../input/tensorflow-speech-recognition-challenge/train/audio/'

all_wave = []
all_label = []
for label in labels:
    print(label)
    waves = [f for f in os.listdir(train_audio_path + '/'+ label) if f.endswith('.wav')]
    for wav in waves:
        samples, sample_rate = librosa.load(train_audio_path + '/' + label + '/' + wav, sr = 16000)
        samples = librosa.resample(samples, sample_rate, 8000)
        if(len(samples)== 8000) :
            all_wave.append(samples)
            all_label.append(label)
```

تحويل تسميات الإخراج إلى عدد صحيح مشفر:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y=le.fit_transform(all_label)
classes= list(le.classes_)
```

الآن ، قم بتحويل الملصقات المشفرة ذات العدد الصحيح إلى متجه واحد ساخن نظرًا لأنها مشكلة متعددة التصنيفات:

```
from keras.utils import np_utils
y=np_utils.to_categorical(y, num_classes=len(labels))
```

أعد تشكيل المصفوفة ثنائية الأبعاد إلى ثلاثية الأبعاد نظرًا لأن الإدخال إلى conv1d يجب أن يكون مصفوفة ثلاثية الأبعاد:

```
all_wave = np.array(all_wave).reshape(-1,8000,1)
```

- تقسيم ملفات التدريب الى مجموعتين ال (Train Set)&(Validation Set) :

بعد ذلك ، سنقوم بتدريب النموذج على ٨٠٪ من البيانات والتحقق من صحة ٢٠٪ المتبقية:

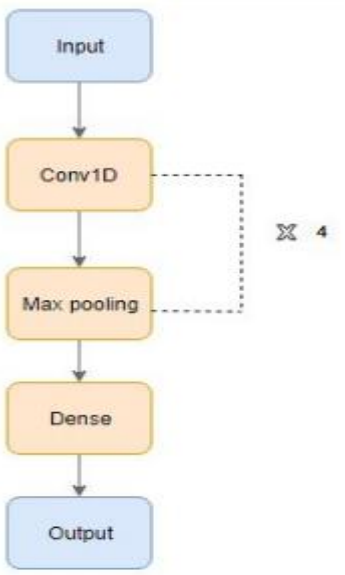
```
from sklearn.model_selection import train_test_split
x_tr, x_val, y_tr, y_val = train_test_split(np.array(all_wave),np.array(y),stratify=y,test_size
```

- نموذج معماري لهذه المشكلة.

سنبني نموذج تحويل الكلام إلى نص باستخدام conv1d

Conv1d عبارة عن شبكة عصبونية تقوم بالالتواء (التدريب و التنفيذ) على بُعد واحد فقط.

وبنية النموذج Conv1d :



- بناء نموذج:

دعونا ننفذ النموذج باستخدام واجهة برمجة تطبيقات Keras الوظيفية:

```
from keras.layers import Dense, Dropout, Flatten, Conv1D, Input, MaxPooling1D
from keras.models import Model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import backend as K
K.clear_session()

inputs = Input(shape=(8000,1))

#First Conv1D layer
conv = Conv1D(8,13, padding='valid', activation='relu', strides=1)(inputs)
conv = MaxPooling1D(3)(conv)
conv = Dropout(0.3)(conv)

#Second Conv1D layer
conv = Conv1D(16, 11, padding='valid', activation='relu', strides=1)(conv)
conv = MaxPooling1D(3)(conv)
conv = Dropout(0.3)(conv)

#Third Conv1D layer
conv = Conv1D(32, 9, padding='valid', activation='relu', strides=1)(conv)
conv = MaxPooling1D(3)(conv)
```

نحدد وظيفة الخسارة لتكون إنتروبيا قاطعة لأنها مشكلة متعددة التصنيفات:

```
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
conv = Conv1D(64, 7, padding='valid', activation='relu', strides=1)(conv)
conv = MaxPooling1D(3)(conv)
conv = Dropout(0.3)(conv)

#Flatten layer
conv = Flatten()(conv)

#Dense Layer 1
conv = Dense(256, activation='relu')(conv)
conv = Dropout(0.3)(conv)

#Dense Layer 2
conv = Dense(128, activation='relu')(conv)
conv = Dropout(0.3)(conv)

outputs = Dense(len(labels), activation='softmax')(conv)

model = Model(inputs, outputs)
model.summary()
```

التوقف المبكر ونقاط التفتيش النموذجية هي عمليات رد الاتصال لإيقاف تدريب الشبكة العصبية في الوقت المناسب وحفظ أفضل نموذج بعد كل فترة:

```
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')
```

دعونا نقوم بتدريب النموذج على حجم دفعة ٣٢ وتقييم الأداء على مجموعة الانتظار:

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

- خريطة التدريب:

سنعتمد على الصور مرة أخرى لفهم أداء النموذج خلال فترة زمنية:

تم تنفيذ التدريب عن طريق منصة COLLAB وأخذ صور توضيحية ل تحسن ACCURANCY مع الزمن

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

Epoch 1/100
1457/1457 [==============================] - ETA: 0s - loss: 2.8552 - accuracy: 0.1618WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 25s 9ms/step - loss: 2.8552 - accuracy: 0.1618 - val_loss: 2.0500 - val_accuracy: 0.3562
Epoch 2/100
1453/1457 [============================>.] - ETA: 0s - loss: 2.0341 - accuracy: 0.3638WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 2.0337 - accuracy: 0.3639 - val_loss: 1.6185 - val_accuracy: 0.4982
Epoch 3/100
1454/1457 [============================>.] - ETA: 0s - loss: 1.7345 - accuracy: 0.4581WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 1.7344 - accuracy: 0.4581 - val_loss: 1.3898 - val_accuracy: 0.5855
Epoch 4/100
1453/1457 [============================>.] - ETA: 0s - loss: 1.5211 - accuracy: 0.5272WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 1.5207 - accuracy: 0.5273 - val_loss: 1.2169 - val_accuracy: 0.6373
Epoch 5/100
1454/1457 [============================>.] - ETA: 0s - loss: 1.3808 - accuracy: 0.5712WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 1.3810 - accuracy: 0.5711 - val_loss: 1.0821 - val_accuracy: 0.6777
Epoch 6/100
1455/1457 [============================>.] - ETA: 0s - loss: 1.2858 - accuracy: 0.6003WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 1.2854 - accuracy: 0.6004 - val_loss: 1.1176 - val_accuracy: 0.6570
Epoch 7/100
1454/1457 [============================>.] - ETA: 0s - loss: 1.2089 - accuracy: 0.6281WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 1.2087 - accuracy: 0.6281 - val_loss: 0.9877 - val_accuracy: 0.7045
Epoch 8/100
1453/1457 [============================>.] - ETA: 0s - loss: 1.1481 - accuracy: 0.6425WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 1.1478 - accuracy: 0.6426 - val_loss: 0.9663 - val_accuracy: 0.7097
Epoch 9/100
1455/1457 [============================>.] - ETA: 0s - loss: 1.1014 - accuracy: 0.6613WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 1.1018 - accuracy: 0.6612 - val_loss: 0.8570 - val_accuracy: 0.7411
Epoch 10/100
1452/1457 [============================>.] - ETA: 0s - loss: 1.0626 - accuracy: 0.6735WARNING:tensorflow:Can save best model only with val_acc available, skipping.
```

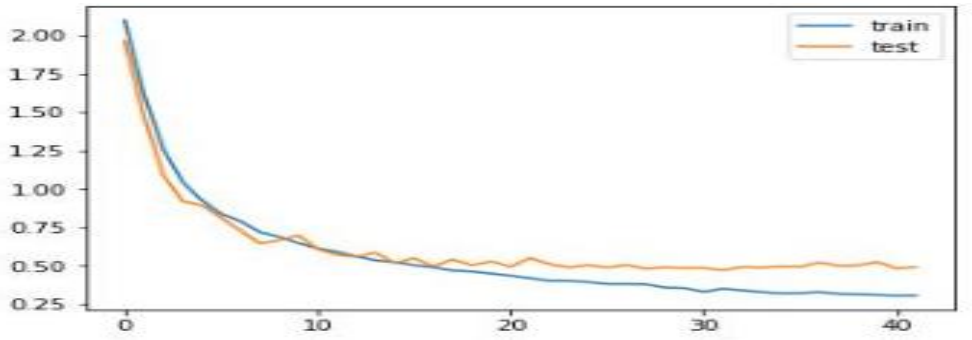
```
history=model.fit(x_tr, y_tr ,epochs=100, callbacks=[es,mc], batch_size=32, validation_data=(x_val,y_val))
```

```
Epoch 1/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.6843 - accuracy: 0.7910WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6848 - accuracy: 0.7908 - val_loss: 0.6197 - val_accuracy: 0.8212
Epoch 2/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.6746 - accuracy: 0.7931WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 12s 9ms/step - loss: 0.6748 - accuracy: 0.7931 - val_loss: 0.6357 - val_accuracy: 0.8162
Epoch 3/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.6774 - accuracy: 0.7918WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6773 - accuracy: 0.7919 - val_loss: 0.6189 - val_accuracy: 0.8176
Epoch 4/100
1451/1457 [============================>.] - ETA: 0s - loss: 0.6664 - accuracy: 0.7962WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6666 - accuracy: 0.7962 - val_loss: 0.6134 - val_accuracy: 0.8194
Epoch 5/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.6609 - accuracy: 0.7980WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.6608 - accuracy: 0.7981 - val_loss: 0.6113 - val_accuracy: 0.8189
Epoch 6/100
1456/1457 [============================>.] - ETA: 0s - loss: 0.6381 - accuracy: 0.8033WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6380 - accuracy: 0.8033 - val_loss: 0.5998 - val_accuracy: 0.8243
Epoch 7/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.6529 - accuracy: 0.8008WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6533 - accuracy: 0.8007 - val_loss: 0.6400 - val_accuracy: 0.8141
Epoch 8/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.6505 - accuracy: 0.8024WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6501 - accuracy: 0.8025 - val_loss: 0.6630 - val_accuracy: 0.8045
Epoch 9/100
1451/1457 [============================>.] - ETA: 0s - loss: 0.6426 - accuracy: 0.8050WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6424 - accuracy: 0.8050 - val_loss: 0.5846 - val_accuracy: 0.8289
Epoch 10/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.6320 - accuracy: 0.8056WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6326 - accuracy: 0.8055 - val_loss: 0.6593 - val_accuracy: 0.8082
Epoch 11/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.6381 - accuracy: 0.8080WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6381 - accuracy: 0.8080 - val_loss: 0.6218 - val_accuracy: 0.8156
Epoch 12/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.6264 - accuracy: 0.8086WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.6266 - accuracy: 0.8085 - val_loss: 0.6199 - val_accuracy: 0.8185
```

```
[29] 1455/1457 [============================>.] - ETA: 0s - loss: 0.5692 - accuracy: 0.8260WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5691 - accuracy: 0.8260 - val_loss: 0.6004 - val_accuracy: 0.8234
Epoch 2/100
1452/1457 [============================>.] - ETA: 0s - loss: 0.5783 - accuracy: 0.8235WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5781 - accuracy: 0.8235 - val_loss: 0.5946 - val_accuracy: 0.8267
Epoch 3/100
1457/1457 [==============================] - ETA: 0s - loss: 0.5644 - accuracy: 0.8292WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5644 - accuracy: 0.8292 - val_loss: 0.5981 - val_accuracy: 0.8269
Epoch 4/100
1456/1457 [============================>.] - ETA: 0s - loss: 0.5737 - accuracy: 0.8251WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5737 - accuracy: 0.8251 - val_loss: 0.6186 - val_accuracy: 0.8182
Epoch 5/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.5678 - accuracy: 0.8284WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5678 - accuracy: 0.8283 - val_loss: 0.5989 - val_accuracy: 0.8251
Epoch 6/100
1454/1457 [============================>.] - ETA: 0s - loss: 0.5716 - accuracy: 0.8268WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5715 - accuracy: 0.8267 - val_loss: 0.5938 - val_accuracy: 0.8254
Epoch 7/100
1457/1457 [==============================] - ETA: 0s - loss: 0.5586 - accuracy: 0.8285WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5586 - accuracy: 0.8285 - val_loss: 0.5827 - val_accuracy: 0.8275
Epoch 8/100
1457/1457 [==============================] - ETA: 0s - loss: 0.5583 - accuracy: 0.8288WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 15s 10ms/step - loss: 0.5583 - accuracy: 0.8288 - val_loss: 0.5747 - val_accuracy: 0.8320
Epoch 9/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.5553 - accuracy: 0.8301WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5554 - accuracy: 0.8301 - val_loss: 0.6475 - val_accuracy: 0.8083
Epoch 10/100
1451/1457 [============================>.] - ETA: 0s - loss: 0.5543 - accuracy: 0.8318WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5538 - accuracy: 0.8318 - val_loss: 0.5972 - val_accuracy: 0.8247
Epoch 11/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.5609 - accuracy: 0.8291WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5610 - accuracy: 0.8291 - val_loss: 0.6222 - val_accuracy: 0.8213
Epoch 12/100
1455/1457 [============================>.] - ETA: 0s - loss: 0.5555 - accuracy: 0.8302WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5553 - accuracy: 0.8303 - val_loss: 0.6133 - val_accuracy: 0.8240
Epoch 13/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.5569 - accuracy: 0.8304WARNING:tensorflow:Can save best model only with val_acc available, skipping.
```

```
9] Epoch 8/100
1457/1457 [==============================] - ETA: 0s - loss: 0.5583 - accuracy: 0.8288WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 15s 10ms/step - loss: 0.5583 - accuracy: 0.8288 - val_loss: 0.5747 - val_accuracy: 0.8320
Epoch 9/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.5553 - accuracy: 0.8301WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5554 - accuracy: 0.8301 - val_loss: 0.6475 - val_accuracy: 0.8083
Epoch 10/100
1451/1457 [============================>.] - ETA: 0s - loss: 0.5543 - accuracy: 0.8318WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5538 - accuracy: 0.8318 - val_loss: 0.5972 - val_accuracy: 0.8247
Epoch 11/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.5609 - accuracy: 0.8291WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5610 - accuracy: 0.8291 - val_loss: 0.6222 - val_accuracy: 0.8213
Epoch 12/100
1455/1457 [============================>.] - ETA: 0s - loss: 0.5555 - accuracy: 0.8302WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5553 - accuracy: 0.8303 - val_loss: 0.6133 - val_accuracy: 0.8240
Epoch 13/100
1453/1457 [============================>.] - ETA: 0s - loss: 0.5569 - accuracy: 0.8304WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5573 - accuracy: 0.8302 - val_loss: 0.5929 - val_accuracy: 0.8276
Epoch 14/100
1451/1457 [============================>.] - ETA: 0s - loss: 0.5507 - accuracy: 0.8318WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5511 - accuracy: 0.8318 - val_loss: 0.6025 - val_accuracy: 0.8273
Epoch 15/100
1451/1457 [============================>.] - ETA: 0s - loss: 0.5446 - accuracy: 0.8349WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5450 - accuracy: 0.8349 - val_loss: 0.6134 - val_accuracy: 0.8204
Epoch 16/100
1454/1457 [============================>.] - ETA: 0s - loss: 0.5546 - accuracy: 0.8309WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5544 - accuracy: 0.8309 - val_loss: 0.6527 - val_accuracy: 0.8110
Epoch 17/100
1457/1457 [==============================] - ETA: 0s - loss: 0.5487 - accuracy: 0.8331WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 13s 9ms/step - loss: 0.5487 - accuracy: 0.8331 - val_loss: 0.6470 - val_accuracy: 0.8148
Epoch 18/100
1455/1457 [============================>.] - ETA: 0s - loss: 0.5421 - accuracy: 0.8349WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1457/1457 [==============================] - 14s 10ms/step - loss: 0.5421 - accuracy: 0.8350 - val_loss: 0.6350 - val_accuracy: 0.8160
Epoch 18: early stopping
```

```
1  from matplotlib import pyplot
2  pyplot.plot(history.history['loss'], label='train')
3  pyplot.plot(history.history['val_loss'], label='test')
4  pyplot.legend() pyplot.show()
```



- تحميل أفضل نموذج :

```
from keras.models import load_model
model=load_model('best_model.hdf5')
```

- حدد تابع الذي يتنبأ بالنص للصوت المحدد:

```
1  def predict(audio):
2      prob=model.predict(audio.reshape(1,8000,1))
3      index=np.argmax(prob[0])
4      return classes[index]
```

- وقت التنبؤ! سنقوم بإنشاء تنبؤات بشأن بيانات التحقق من الصحة Validation Data

```
import random
index=random.randint(0,len(x_val)-1)
samples=x_val[index].ravel()
print("Audio:",classes[np.argmax(y_val[index])])
ipd.Audio(samples, rate=8000)
print("Text:",predict(samples))
```

في هذا الكود نطلب من المستخدم بتسجيل الأوامر الصوتية.

```
import sounddevice as sd
import soundfile as sf

samplerate = 16000
duration = 1 # seconds
filename = 'yes.wav'
print("start")
mydata = sd.rec(int(samplerate * duration), samplerate=samplerate,
    channels=1, blocking=True)
print("end")
sd.wait()
```

الآن سنقرأ الأمر الصوتي المحفوظ ونحوله إلى نص:

```
os.listdir('../input/voice-commands/prateek_voice_v2')
filepath='../input/voice-commands/prateek_voice_v2'

#reading the voice commands
samples, sample_rate = librosa.load(filepath + '/' + 'stop.wav', sr = 16000)
samples = librosa.resample(samples, sample_rate, 8000)
ipd.Audio(samples,rate=8000)

predict(samples)
```

الاستنتاجات والتوصيات:

هذه صورة مصغرة للأشياء التي يمكننا القيام بها بالتعلم العميق.

- من الممكن تطوير واجهات المستخدم القائمة على الكلام وذلك بالأعتماد على التعلم العميق .
- يمكن الحصول على دقة أكبر في التعرف على الكلمات وذلك من خلال التعديل على بناء الشبكة العصبونية بزيادة الطبقات المخفية العاملة في الشبكة .
- يمكن تضمين مزيد من التحكم في النظام/الأوامر من خلال العمل على خوازميات استخراج صوت محسنة

References

- GitHub - aravindpai/Speech-Recognition: This repository contains the code for the speech recognition in python
- Deploying a Keras Flower Classification Model - Analytics Vidhya
- Signal Processing | Building Speech to Text Model in Python (analyticsvidhya.com)