Voice

When you bang a drum its skin vibrates. The harder you bang, the bigger the vibrations. The vibrating drum skin causes nearby air particles to vibrate, which in turn causes other nearby air particles to vibrate. These vibrating particles make up a sound wave.

Sound waves move at a speed of 343 m/s in the air and even faster in liquids and solids. The waves transport energy from the sound source, such as a drum, to the environment. When vibrating air particles force your ear drum to vibrate, your ear detects sound waves. The sound gets louder as the vibrations get bigger.

Sound wave

Artificial Intelligence

Artificial intelligence (AI) is the simulation of human intelligence in robots that have been trained to think and act like humans. The phrase can also refer to any machine that demonstrates human-like characteristics like learning and problem-solving.

The ability of artificial intelligence to rationalize and execute actions that have the best likelihood of reaching a certain goal is its ideal feature. Machine learning is a subset of artificial intelligence that refers to the idea that computer systems can learn from and adapt to new data without the need for human intervention.

Deep learning techniques allow for this autonomous learning by absorbing large volumes of unstructured data including text, photos, and video.

The majority of people immediately think of robots when they hear the words artificial intelligence. That's because big-budget movies and literature have human-like computers wreaking devastation on the planet. Nothing, however, could be further from the truth.

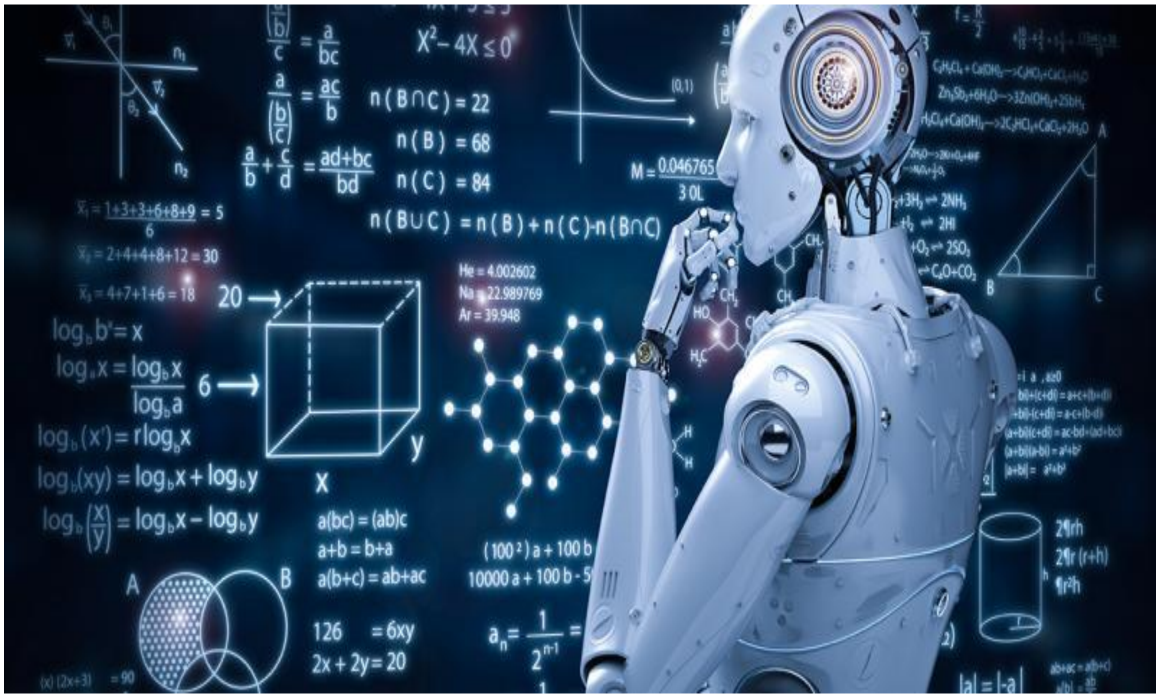
Artificial intelligence is founded on the idea that human intelligence may be characterized in such a way that a machine can simply duplicate it and carry out activities ranging from the most basic to the most complicated. Artificial intelligence's goals include simulating human cognitive processes. To the extent that they can be concretely characterized, researchers and developers in the field are making unexpectedly rapid progress in simulating tasks such as learning, reasoning, and perception. Some predict that in the near future, innovators will be able to create systems that can learn and reason about any subject faster than humans can. Others, on the other hand, remain suspicious, claiming that all cognitive activity is loaded with value judgments based on human experience.

As technology progresses, earlier artificial intelligence criteria become obsolete. Machines that calculate basic calculations or recognise text using optical character recognition, for example, are no longer called artificial intelligence because these functions are now regarded standard computer functions.

AI is constantly improving to benefit a wide range of sectors. A multidisciplinary approach based on mathematics, computer science, linguistics, psychology, and other disciplines is used to wire machines.

Machine learning

The premise of machine learning is that a computer programme can learn and adapt to new data without the need for human involvement. Machine learning is a branch of artificial intelligence (AI) that maintains the accuracy of a computer's built-in algorithms.



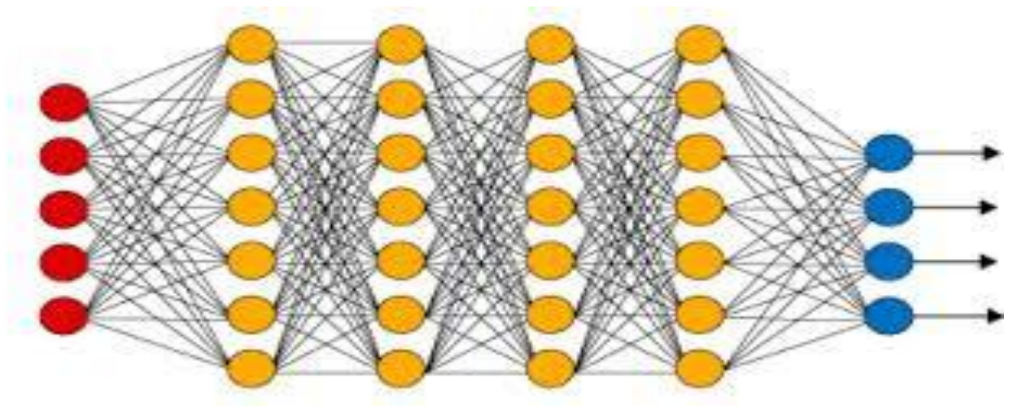
Various areas of the business are dealing with massive amounts of data in various formats gathered from various sources. Because of the advancement of technology, particularly increased processing capabilities and cloud storage, vast amounts of data, known as big data, are becoming more readily available and accessible. Companies and governments recognise the enormous insights that can be obtained from analysing big data, but often lack the resources and time to go through its vast amounts of data.

As a result, various businesses are employing artificial intelligence technologies to gather, process, communicate, and exchange important information from data sets. Machine learning is one type of AI that is increasingly being used for massive data processing.

Machine learning data applications are created using a complicated algorithm or source code embedded in the machine or computer. This programming code constructs a model that recognises data and makes predictions based on that data. For its decision-making process, the model leverages parameters integrated into the algorithm to build patterns. When new or extra data becomes available, the algorithm modifies the parameters automatically to see whether there is a pattern shift. The model, on the other hand, should not be altered.

Deep learning

Deep learning is an artificial intelligence (AI) function that mimics the human brain's processing of data and pattern creation in order to make decisions. Deep learning is an artificial intelligence subset of machine learning that uses neural networks to learn unsupervised from unstructured or unlabeled data. Deep neural learning or deep neural network are other terms for the same thing.



Deep learning has progressed in lockstep with the digital era, which has resulted in an avalanche of data in all formats and from all corners of the globe. Big data is gathered from a variety of sources, including social media, internet search engines, e-commerce platforms, and online theatres. This massive volume of data is easily accessible and can be shared via fintech tools such as cloud computing.

However, because the data is typically unstructured, it could take decades for humans to analyse and extract useful information. Companies are increasingly using AI systems for automated support as they see the enormous potential that can be realised by unlocking this wealth of data.

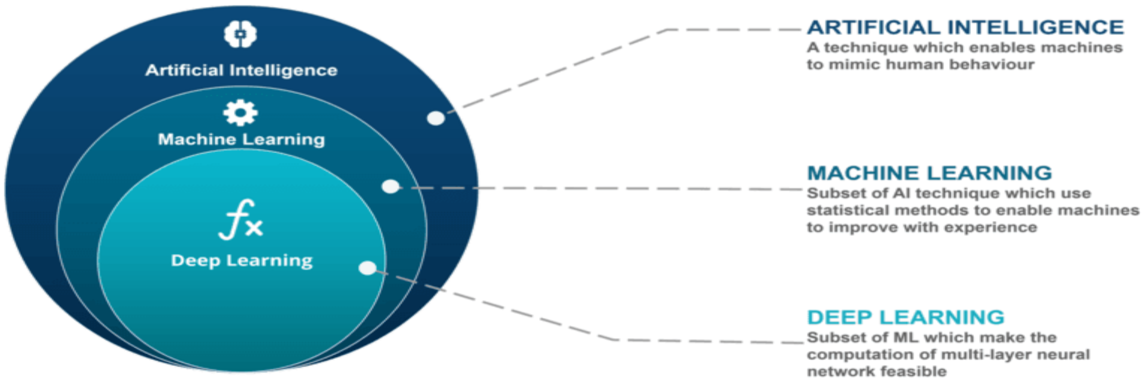
Machine Learning vs. Deep Learning

Machine learning, a self-adaptive algorithm that gains increasingly better analysis of patterns with experience or freshly acquired data, is one of the most common AI techniques used for processing massive data.

Machine learning tools could be used by a digital payments corporation to detect the presence or potential for fraud in its system. A computational algorithm implemented into a computer model will process all transactions that take place on the digital platform, look for patterns in the data set, and flag any anomalies found by the pattern.

Machines may now handle data in a nonlinear manner thanks to the hierarchical function of deep learning systems.

Deep learning, a subset of machine learning, employs artificial neural networks at a hierarchical level to carry out machine learning. The artificial neural networks are constructed in the same way as the human brain, with neuron nodes connected in a web-like fashion. The hierarchical function of deep learning systems allows machines to process data in a nonlinear manner, whereas standard programmes develop analyses using data in a linear manner.


Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

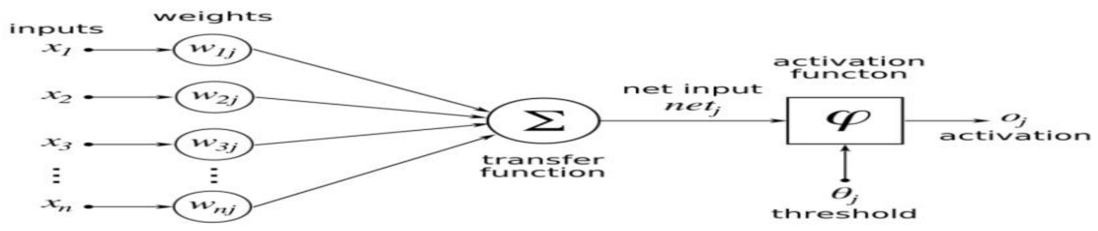
Convolutional neural networks (CNN)

Computer scientists have been trying to construct computers that can understand visual input since the 1950s, when artificial intelligence was in its infancy. Over the next few decades, the field of computer vision progressed at a slow but steady pace. When a group of academics from the University of Toronto built an AI model that outperformed the top image recognition algorithms by a considerable margin in 2012, computer vision took a quantum leap.

The AI system, which became known as AlexNet (named after its main creator, Alex Krizhevsky), won the 2012 ImageNet computer vision contest with an amazing 85 percent accuracy. The runner-up scored a modest 74 percent on the test.

A convolutional neural network (CNN), a specific sort of artificial neural network that essentially resembles the human vision system, was at the heart of the AlexNet. CNNs have grown increasingly important in computer vision applications in recent years. Here's everything you need to know about CNN's history and operation.

Multiple layers of artificial neurons make up convolutional neural networks. Artificial neurons are mathematical functions that calculate the weighted sum of various inputs and output an activation value, similar to their biological counterparts.



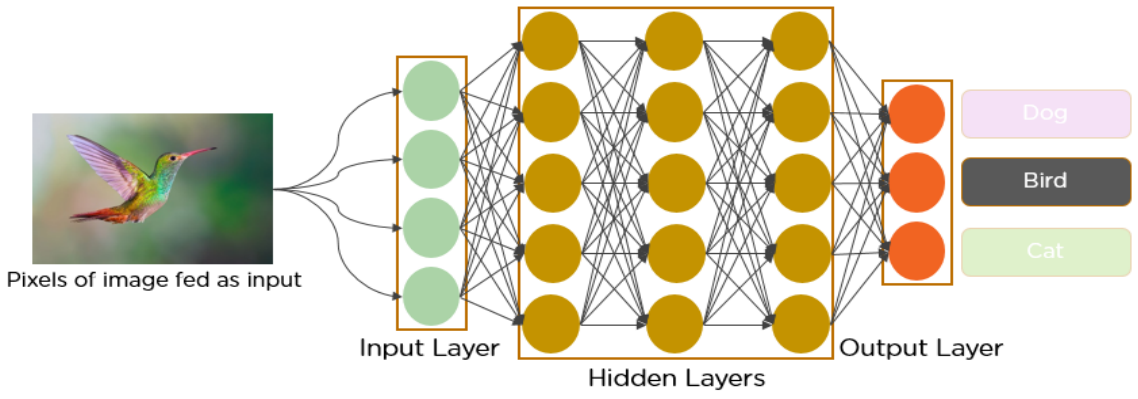
Each neuron's action is determined by its weights. The artificial neurons in a CNN pick out numerous visual properties when fed with pixel values.

Each of the layers in a ConvNet generates various activation maps when you feed it an image. Activation maps highlight the image's most important characteristics. Each neuron takes a pixel patch as input, multiplies its colour values by its weights, adds them together, and runs them through the activation function.

Basic features such as horizontal, vertical, and diagonal edges are usually detected by the CNN's first (or bottom) layer. The first layer's output is sent into the next layer, which extracts more complicated features like corners and edge combinations. The layers recognise higher-level features such as objects, faces, and more as you progress further into the convolutional neural network.

Convolution is the process of multiplying pixel values by weights and summing them (hence the name convolutional neural network). A CNN is typically made up of numerous convolution layers, but it can also include other elements. A classification layer is the final layer of a CNN, and it takes the output of the final convolution layer as input (remember, the higher convolution layers detect complex objects).

The classification layer generates a series of confidence ratings (numbers between 0 and 1) based on the activation map of the final convolution layer, which indicate how likely the image is to belong to a "class." For example, if a ConvNet detects cats, dogs, and horses, the final layer's output is the likelihood that the input image contains any of those animals.



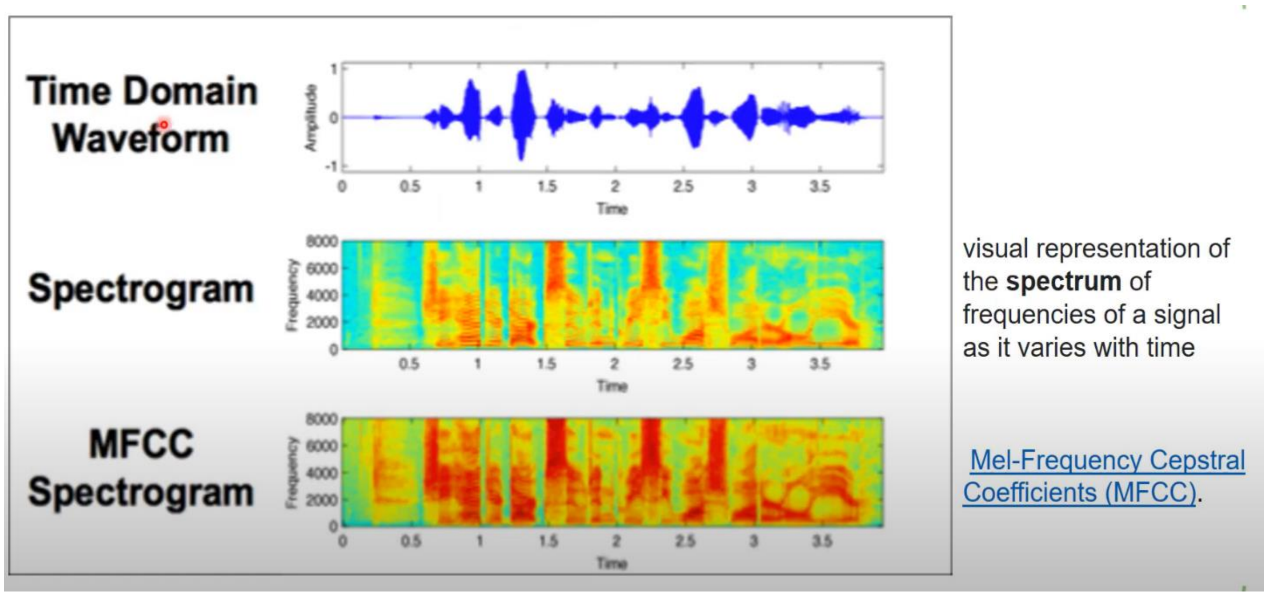
Principle of speech recognition

Ai can’t deal with the sound waves so if we need to recognize the voice we will convert it to the wave form and draw it .

But in this case we will have a problem all waves in time domain are very similar to each other .

So we have a frequency domain we can use it but we have another problem in this case if we have some noise the word which you say will be another word in the model and the number of frequency are very huge .

What if we draw every number of this frequency by a specific color so we can use Mel frequency cepstral coefficient (Mfcc) .



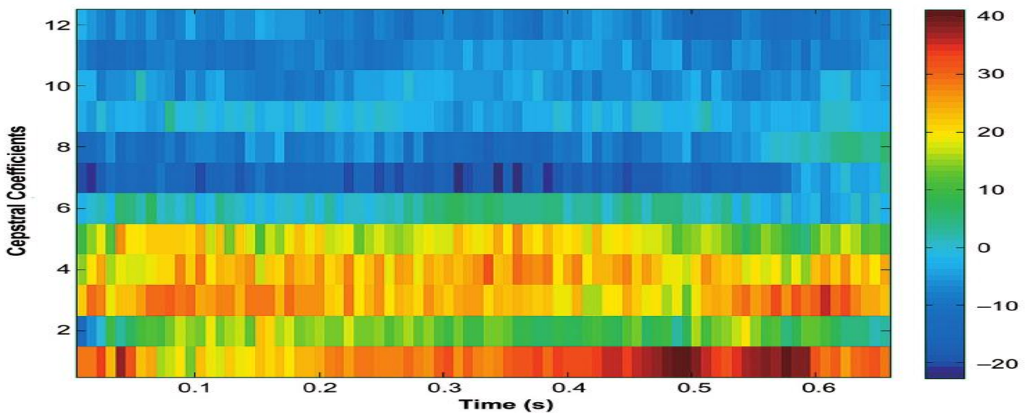
visual representation of the **spectrum** of frequencies of a signal as it varies with time

Mel-Frequency Cepstral Coefficients (MFCC).

Mel frequency cepstral coefficient (MFCC)

This is the most important thing to make us recognize the speech . because this tool make us to convert the sound wave to image contain all the frequency and every specific frequency have a specific color .

Before we make a model to recognize the speech we need a lot o data to give it to the model to train by it .



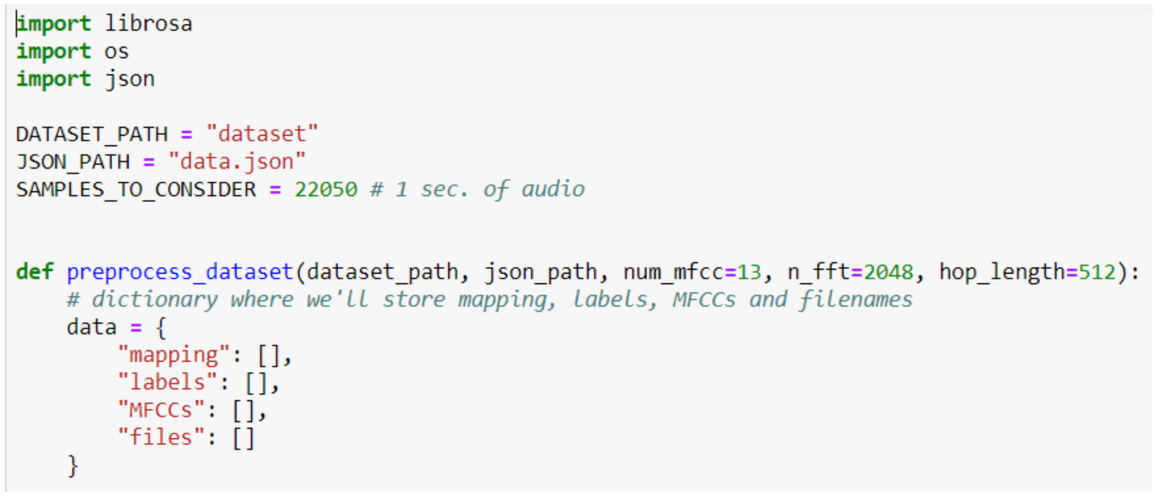
Preparing the data

This the first step to build the speech recognition model .

In this step we need some requirement

1- Python 3.8
2- Json library
3- Librosa library
4- Os library

Import libraries and define the dictionary



```
import librosa
import os
import json

DATASET_PATH = "dataset"
JSON_PATH = "data.json"
SAMPLES_TO_CONSIDER = 22050 # 1 sec. of audio


def preprocess_dataset(dataset_path, json_path, num_mfcc=13, n_fft=2048, hop_length=512):
    # dictionary where we'll store mapping, labels, MFCCs and filenames
    data = {
        "mapping": [],
        "labels": [],
        "MFCCs": [],
        "files": []
    }
```

To make a good accuracy you want to have a lot of data so in this model we will predict the English numbers from 0 to 9 .

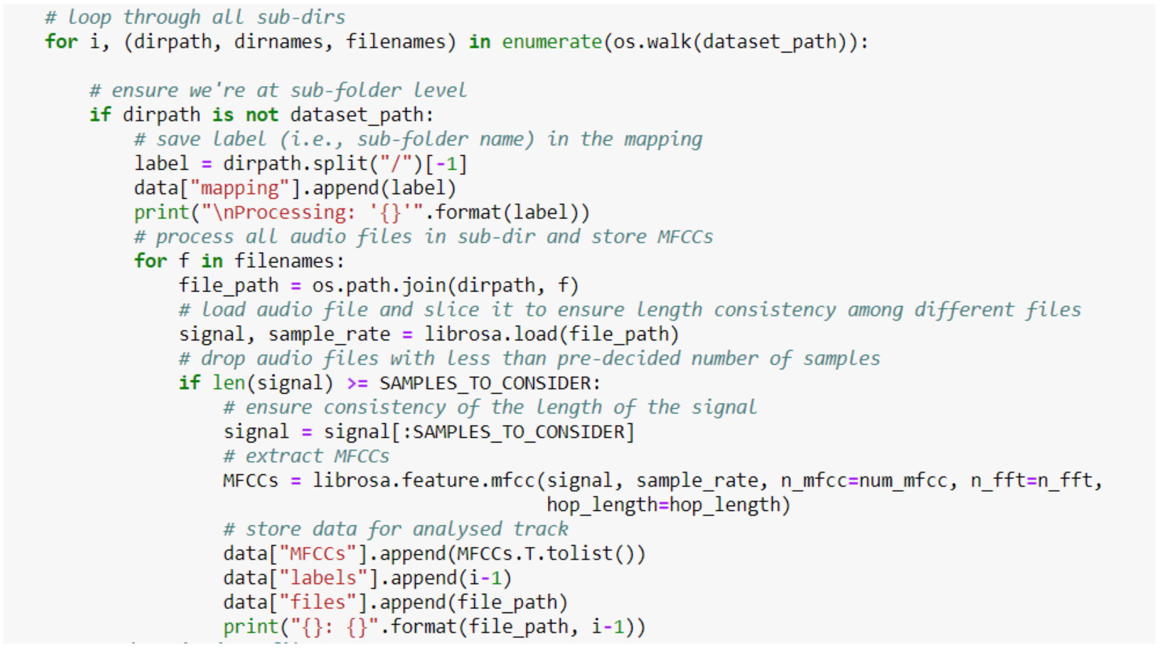
So we collect 2850 audio file for each number to train the model by it .

To use the library which help you in this code we will import it first .

We should read the whole audio file for all number so we use os library for that .

We need to convert every audio file to MFCC form to make it easy for CNN take some features from every audio file to train the model by it so librosa will help us for that .

Read all data audio file and extract the MFCCs from it and store it .



```
# loop through all sub-dirs
for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):

    # ensure we're at sub-folder level
    if dirpath is not dataset_path:
        # save label (i.e., sub-folder name) in the mapping
        label = dirpath.split("/")[-1]
        data["mapping"].append(label)
        print("\nProcessing: '{}'".format(label))
        # process all audio files in sub-dir and store MFCCs
        for f in filenames:
            file_path = os.path.join(dirpath, f)
            # load audio file and slice it to ensure length consistency among different files
            signal, sample_rate = librosa.load(file_path)
            # drop audio files with less than pre-decided number of samples
            if len(signal) >= SAMPLES_TO_CONSIDER:
                # ensure consistency of the length of the signal
                signal = signal[:SAMPLES_TO_CONSIDER]
                # extract MFCCs
                MFCCs = librosa.feature.mfcc(signal, sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
                                             hop_length=hop_length)
                # store data for analysed track
                data["MFCCs"].append(MFCCs.T.tolist())
                data["labels"].append(i-1)
                data["files"].append(file_path)
                print("{}: {}".format(file_path, i-1))
```

Now we will read all audio file and convert it to MFCC form .

Now we ready for preparing the data .

We will read all audio file and convert it to MFCC form and we will make a dictionary that contain (mapping , labels , MFCCs , files)

Mapping contain the name of the directory which have the 2800 audio file so in this case mapping have 10 word from zero to nine .

Labels contain number that pointing to every number . in this case the first number pointing to zero and so on .

MFCCS pointing to the number of MFCC for each audio file .

File pointing to the name file of every number.

We will load the whole audio file by librosa and convert it to MFCC and after we do that we will save this dictionary in json file .

When we load the audio file and convert it we give it some arguments

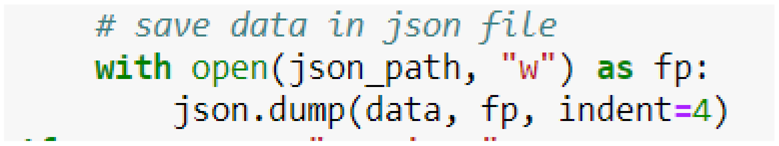
Num_Mfcc → Number of coefficients to extract .

N_fft → Interval we consider to apply FFT. Measured in # of samples.

hop_length → Sliding window for FFT. Measured in # of samples .

in this data we will load the audio file that contain 1 second of voice only so SAMPLES_TO_CONSIDER = 22050 this number mean 1 sec of speech

.store the data in a json file

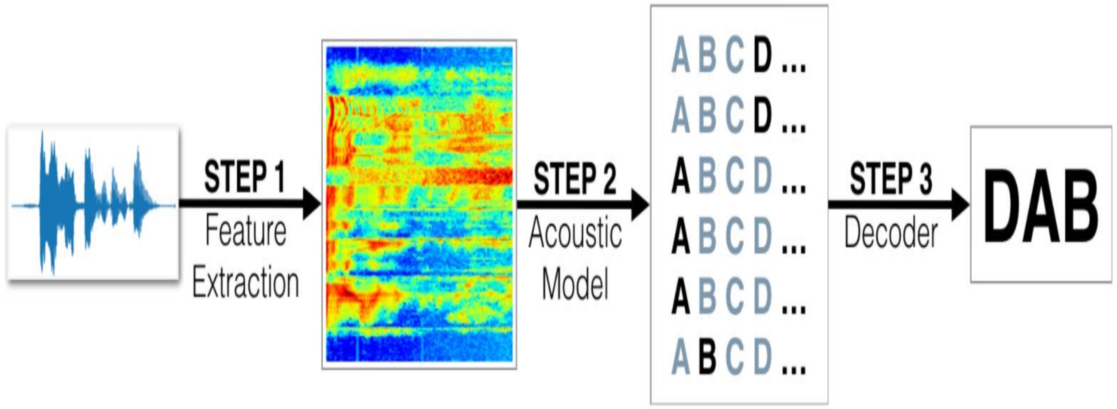


```
    # save data in json file
    with open(json_path, "w") as fp:
        json.dump(data, fp, indent=4)
```

we need to save all this data in a json file because this type is make the read data is very easy to deal with it so we use the third library json library .

now we are ready to build a model by our own data which we preparing above .

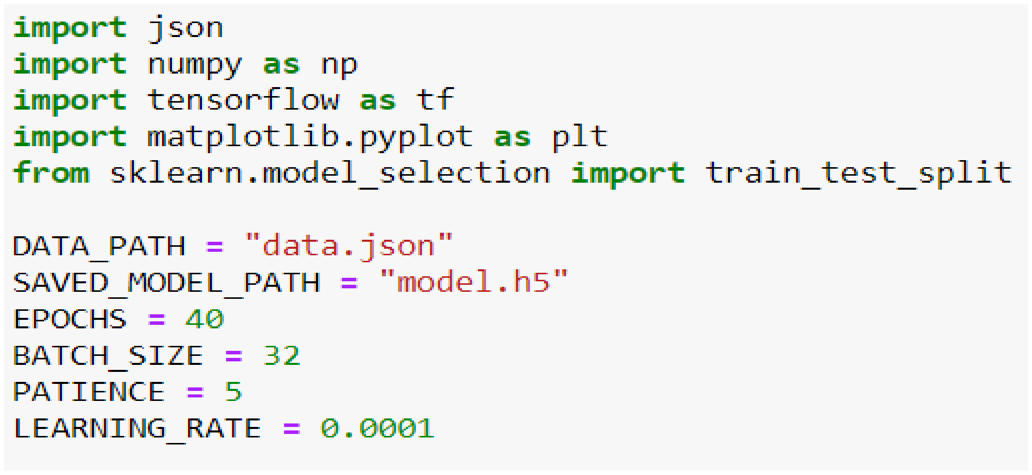
Building CNN model



The requirements in the 2nd step are

1- json library
2- Tensorflow library (try to install it by anaconda because this library will make some problem if you try to install it by the terminal by pip so I recommend to download anaconda and use this command ~ conda install tensorflow , and you can do it with the other library by this command ~ conda install lib_name)
3- Numpy library
4- matplotlib.pyplot library
5- from sklearn.model_selection import train_test_split

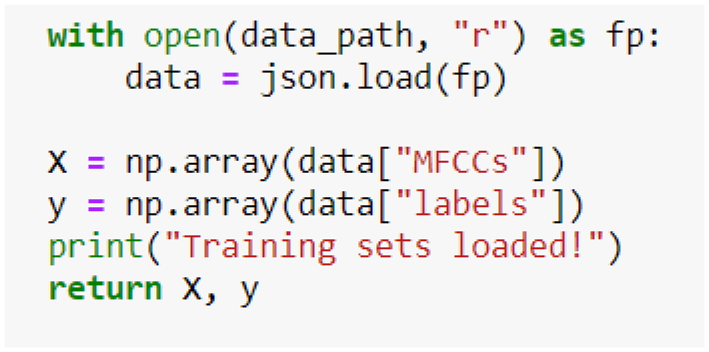
import libraries and define variables



```
import json
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

DATA_PATH = "data.json"
SAVED_MODEL_PATH = "model.h5"
EPOCHS = 40
BATCH_SIZE = 32
PATIENCE = 5
LEARNING_RATE = 0.0001
```

I tried to use vosk library to build a model but I found it very difficult to install and caused different problems so I used these libraries .

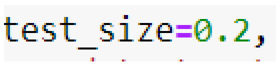
Read data from json file and store it



```
with open(data_path, "r") as fp:
    data = json.load(fp)

X = np.array(data["MFCCs"])
y = np.array(data["labels"])
print("Training sets loaded!")
return X, y
```

We start by reading the json file that we made in the 1st step .

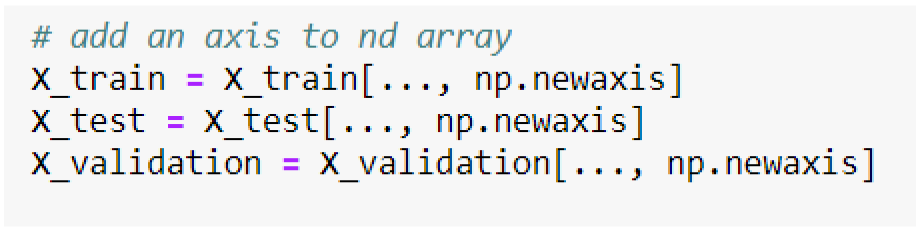
And store the MFCC and Labels in x and y variables .



```
test_size=0.2,
```

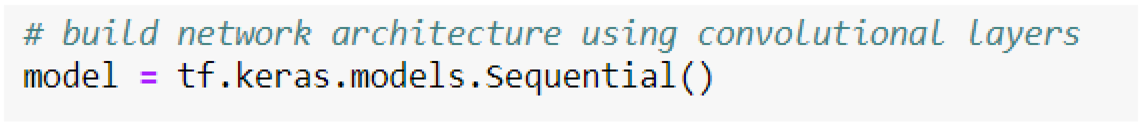
20% of data we will use it to tset the model and the other 80% we will use it to train the model .

Add new axis



```
# add an axis to nd array
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]
X_validation = X_validation[..., np.newaxis]
```

X_train is 2d array and we should convert it to 3d array so we use numpy to do this step and we put the third layer by 1 to make the image gray and we do this step to the x_test and x_validation .



```
# build network architecture using convolutional layers
model = tf.keras.models.Sequential()
```

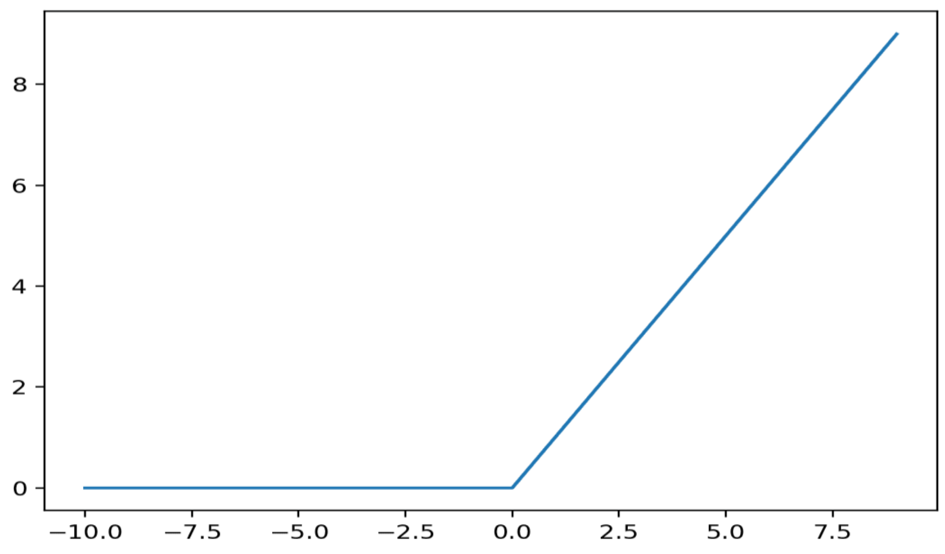
this is line build a model that we try to make .

make a 3 convolution layers

```
# 1st conv layer
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=input_shape,
                                 kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D((3, 3), strides=(2,2), padding='same'))

# 2nd conv layer
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                                 kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D((3, 3), strides=(2,2), padding='same'))

# 3rd conv layer
model.add(tf.keras.layers.Conv2D(32, (2, 2), activation='relu',
                                 kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D((2, 2), strides=(2,2), padding='same'))
```

after that we make a number of convolution layers in this model we make 3 layers but we can make more or less number if we want . it's depend on the accuracy so we can say number of layers is a hyperparameter .

for each layer we out the 1st number which pointing to number of filters we use so in the first layer we use 64 filter and we use activation function " relu " and this is relu function f(x)=max(0,x)



activation function

activation functions give out the final value given out from a neuron. So, an activation function is basically just a simple function that transforms its inputs into outputs that have a certain range. There are various types of activation functions that perform this task in a different manner, For example, the sigmoid activation function takes input and maps the resulting values in between 0 to 1.

One of the reasons that this function is added into an artificial neural network in order to help the network learn complex patterns in the data. These functions introduce nonlinear real-world properties to artificial neural networks. Basically, in a simple neural network, x is defined as inputs, w weights, and we pass f (x) that is the value passed to the output of the network. This will then be the final output or the input of another layer.

If the activation function is not applied, the output signal becomes a simple linear function. A neural network without activation function will act as a linear regression with limited learning power. But we also want our neural network to learn non-linear states as we give it complex real-world information such as image, video, text, and sound.

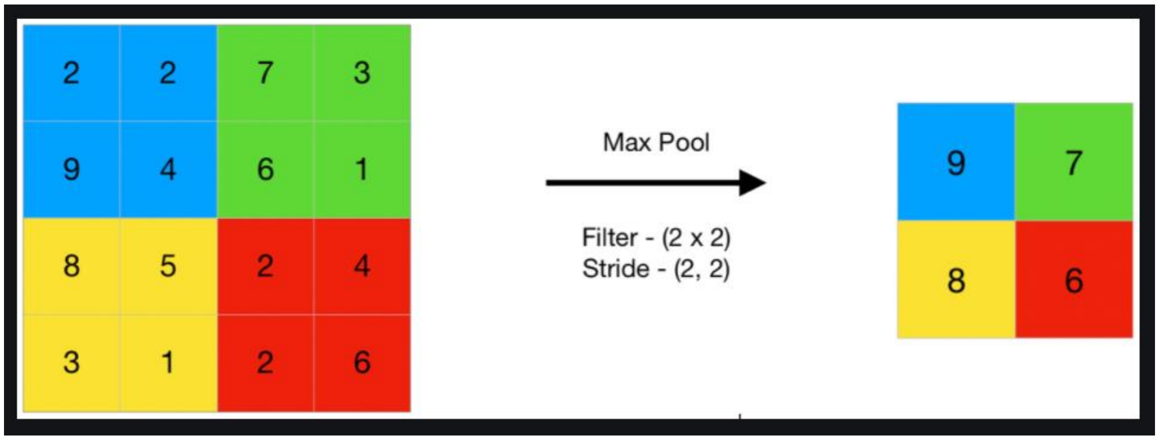
Max pooling

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

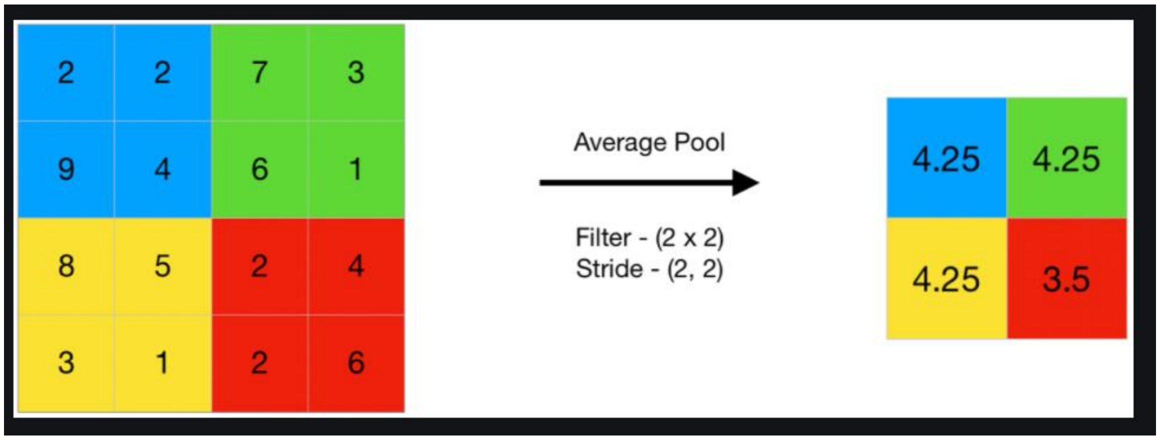
Types of Pooling Layers

1- Max pooling (recommended in many models)
is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



2- Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

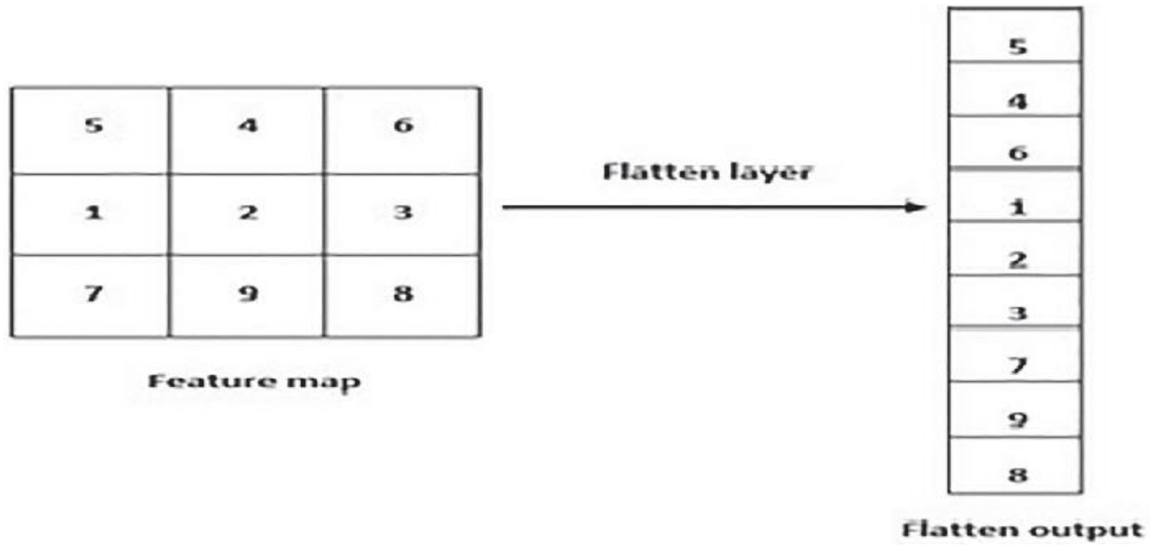


In the 2nd convolution layer we do the same steps but we reduce the number of filters to 32 .

In the 3rd convolution layer we do the same steps but we make the kernel of maxpooling (2 , 2) .

Flatten the output

```
# flatten output and feed into dense layer
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64, activation='relu'))
tf.keras.layers.Dropout(0.3)
```



Now we will make a flatting step to make the output reshape in 1D array to make it a input to the last neurons to give us the nearby output .

Dropout used to reduce the number of the neurons which we don't need because there are not important so that make the model make processing faster

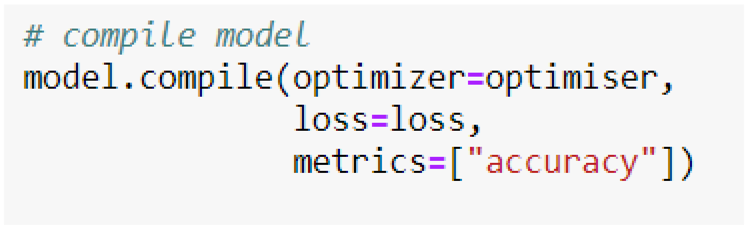
Softmax the output layer

```
# softmax output layer
model.add(tf.keras.layers.Dense(10, activation='softmax'))

optimiser = tf.optimizers.Adam(learning_rate=learning_rate)
```

.

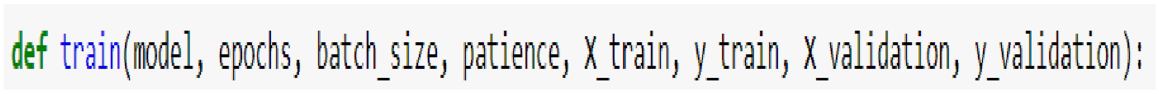
to reduce the processing of model again we need to make a softmax activation to make the model faster and faster .

optimizer help model to detect the final value of input we user the common optimizer " adam " .



```
# compile model
model.compile(optimizer=optimiser,
              loss=loss,
              metrics=["accuracy"])
```

after that we should compile the model .

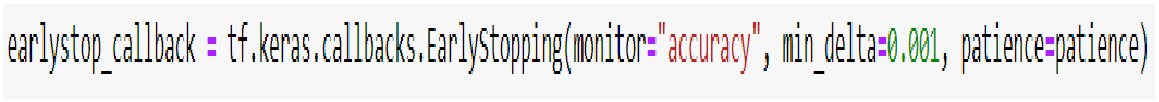


```
def train(model, epochs, batch_size, patience, X_train, y_train, X_validation, y_validation):
```

we use a variable that called epochs that mean the number of the model will see the whole data and that variable is a hyperparameter and in this model we put it 40 .

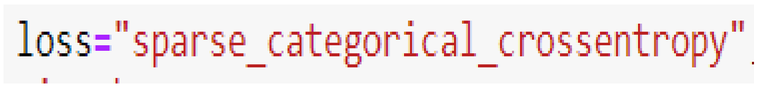
batch_size → Samples per batch

patience → Num epochs to wait before early stop, if there isn't an improvement on accuracy



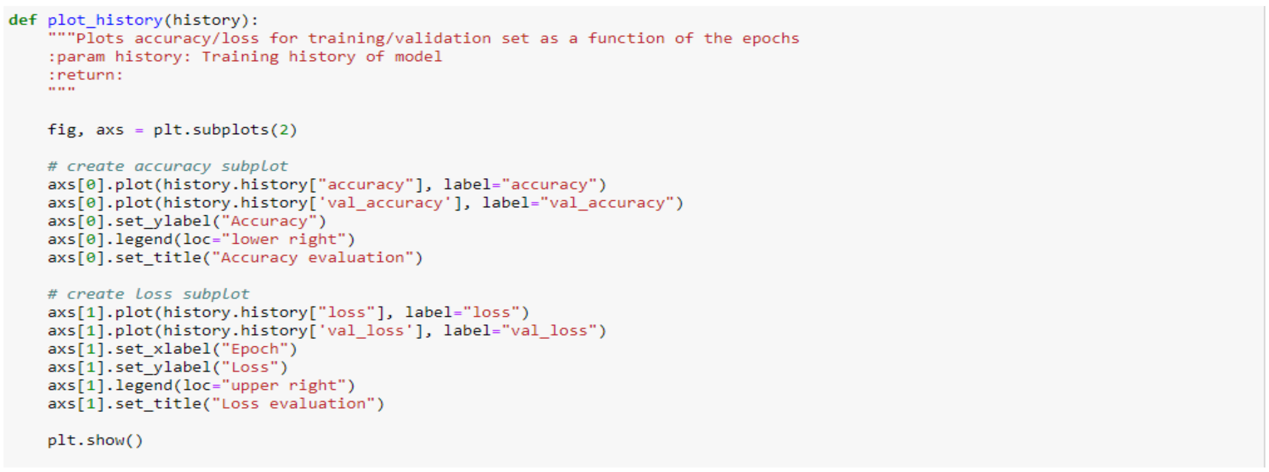
```
earlystop_callback = tf.keras.callbacks.EarlyStopping(monitor="accuracy", min_delta=0.001, patience=patience)
```

to avoid the processing doesn't give us a improvement in the accuracy we use Earlystopping when the change in delta loss is 0.001 and in this model we use "sparse_categorical_crossentropy"

as a loss function 

```
loss="sparse_categorical_crossentropy"
```

plot model summary

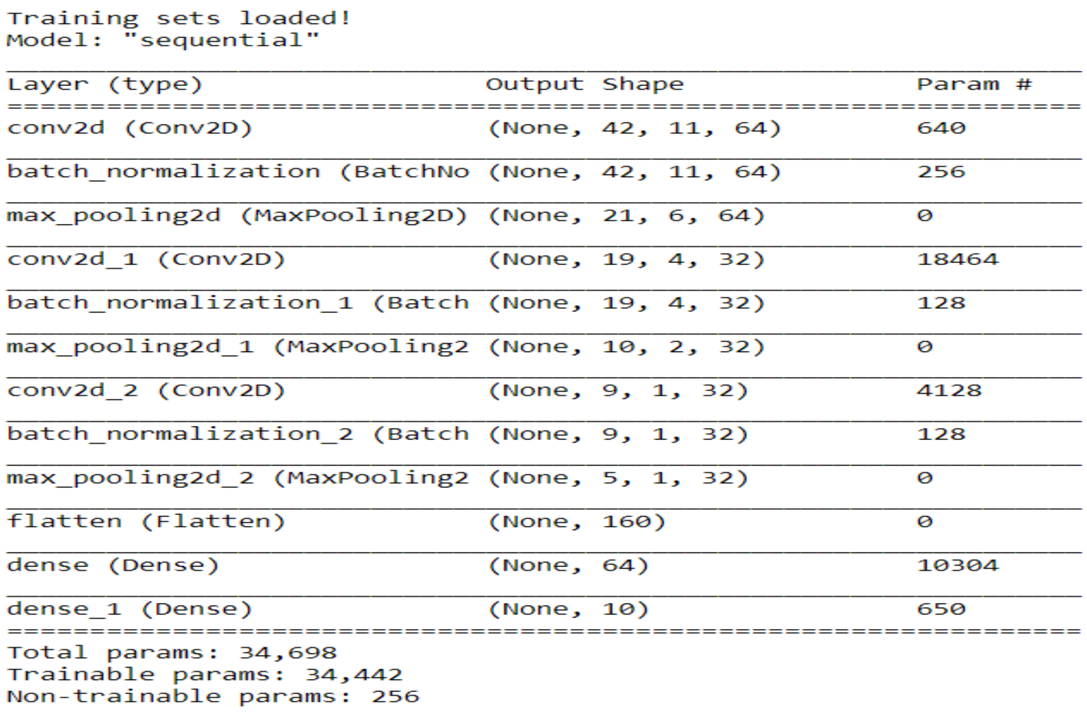


```python
def plot_history(history):
    """Plots accuracy/loss for training/validation set as a function of the epochs
    :param history: Training history of model
    :return:
    """

    fig, axs = plt.subplots(2)

    # create accuracy subplot
    axs[0].plot(history.history["accuracy"], label="accuracy")
    axs[0].plot(history.history['val_accuracy'], label="val_accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy evaluation")

    # create loss subplot
    axs[1].plot(history.history["loss"], label="loss")
    axs[1].plot(history.history['val_loss'], label="val_loss")
    axs[1].set_xlabel("Epoch")
    axs[1].set_ylabel("Loss")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Loss evaluation")

    plt.show()
```

This part of code help us to see the summary of the model and the accuracy and the loss in this model .

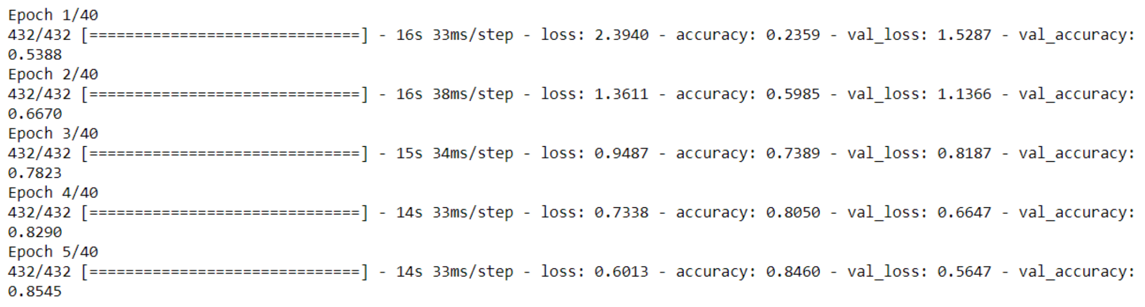
Now we finish the train of model so we save it by the extension of .h5 file and when we predict we will call the h5 model .

Summary of the model



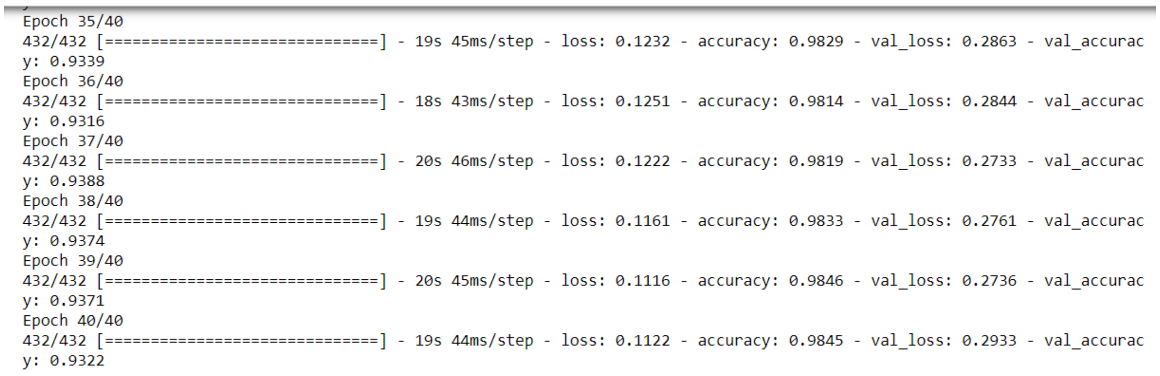
```
Training sets loaded!
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 42, 11, 64)        640
_________________________________________________________________
batch_normalization (BatchNo (None, 42, 11, 64)        256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 21, 6, 64)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 19, 4, 32)         18464
_________________________________________________________________
batch_normalization_1 (Batch (None, 19, 4, 32)         128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 2, 32)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 1, 32)          4128
_________________________________________________________________
batch_normalization_2 (Batch (None, 9, 1, 32)          128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 1, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 160)               0
_________________________________________________________________
dense (Dense)                (None, 64)                10304
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 34,698
Trainable params: 34,442
Non-trainable params: 256
```

First 5 epochs



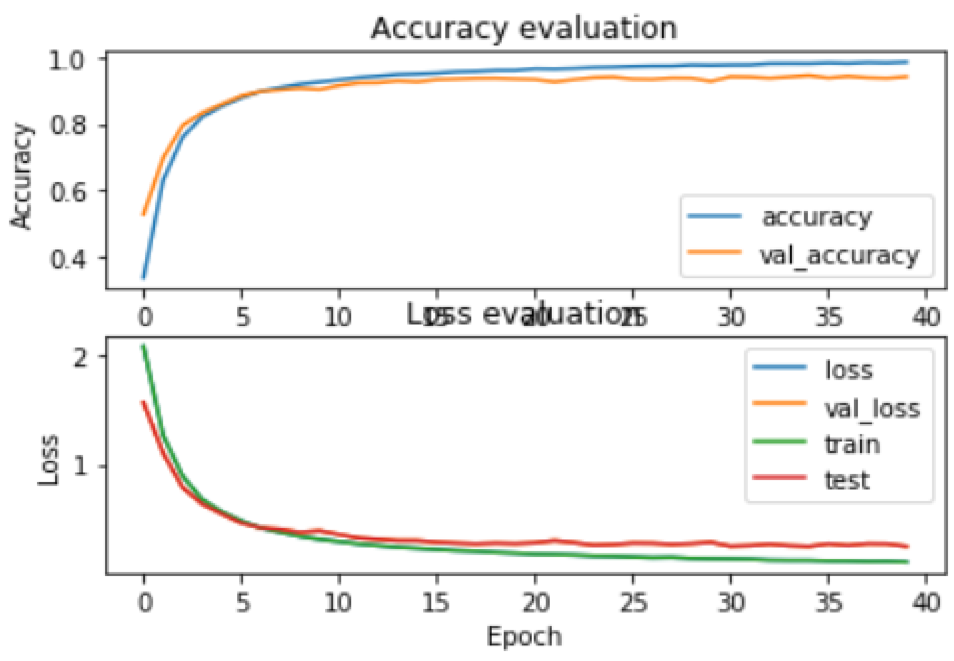
```
Epoch 1/40
432/432 [==============================] - 16s 33ms/step - loss: 2.3940 - accuracy: 0.2359 - val_loss: 1.5287 - val_accuracy:
0.5388
Epoch 2/40
432/432 [==============================] - 16s 38ms/step - loss: 1.3611 - accuracy: 0.5985 - val_loss: 1.1366 - val_accuracy:
0.6670
Epoch 3/40
432/432 [==============================] - 15s 34ms/step - loss: 0.9487 - accuracy: 0.7389 - val_loss: 0.8187 - val_accuracy:
0.7823
Epoch 4/40
432/432 [==============================] - 14s 33ms/step - loss: 0.7338 - accuracy: 0.8050 - val_loss: 0.6647 - val_accuracy:
0.8290
Epoch 5/40
432/432 [==============================] - 14s 33ms/step - loss: 0.6013 - accuracy: 0.8460 - val_loss: 0.5647 - val_accuracy:
0.8545
```

The last 5 epochs



```
Epoch 35/40
432/432 [==============================] - 19s 45ms/step - loss: 0.1232 - accuracy: 0.9829 - val_loss: 0.2863 - val_accurac
y: 0.9339
Epoch 36/40
432/432 [==============================] - 18s 43ms/step - loss: 0.1251 - accuracy: 0.9814 - val_loss: 0.2844 - val_accurac
y: 0.9316
Epoch 37/40
432/432 [==============================] - 20s 46ms/step - loss: 0.1222 - accuracy: 0.9819 - val_loss: 0.2733 - val_accurac
y: 0.9388
Epoch 38/40
432/432 [==============================] - 19s 44ms/step - loss: 0.1161 - accuracy: 0.9833 - val_loss: 0.2761 - val_accurac
y: 0.9374
Epoch 39/40
432/432 [==============================] - 20s 45ms/step - loss: 0.1116 - accuracy: 0.9846 - val_loss: 0.2736 - val_accurac
y: 0.9371
Epoch 40/40
432/432 [==============================] - 19s 44ms/step - loss: 0.1122 - accuracy: 0.9845 - val_loss: 0.2933 - val_accurac
y: 0.9322
```

Accuracy and loss evaluation



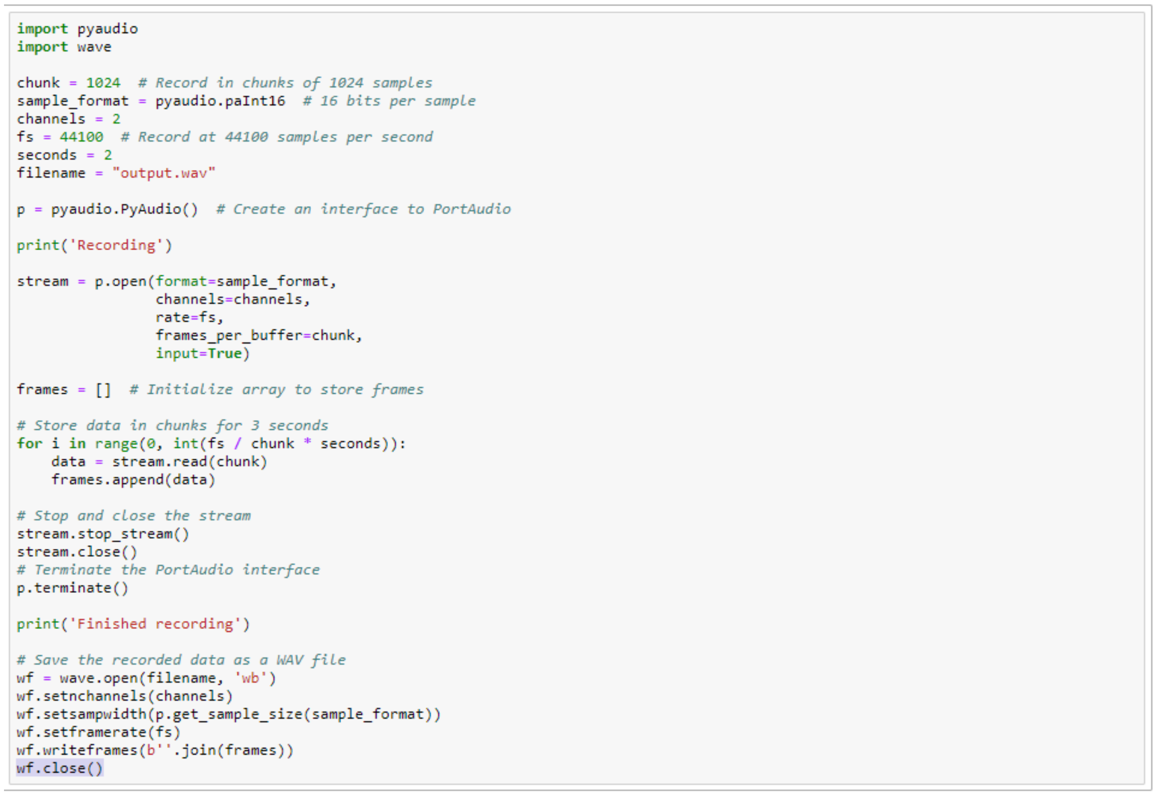
Recording

We need to record our voice and make a prediction for this audio file .

The requirement for this step

1- Pyaudio library
2- Wave library

Pyaudio library make the recording is very easy



```python
import pyaudio
import wave

chunk = 1024  # Record in chunks of 1024 samples
sample_format = pyaudio.paInt16  # 16 bits per sample
channels = 2
fs = 44100  # Record at 44100 samples per second
seconds = 2
filename = "output.wav"

p = pyaudio.PyAudio()  # Create an interface to PortAudio

print('Recording')

stream = p.open(format=sample_format,
                channels=channels,
                rate=fs,
                frames_per_buffer=chunk,
                input=True)

frames = []  # Initialize array to store frames

# Store data in chunks for 3 seconds
for i in range(0, int(fs / chunk * seconds)):
    data = stream.read(chunk)
    frames.append(data)

# Stop and close the stream
stream.stop_stream()
stream.close()
# Terminate the PortAudio interface
p.terminate()

print('Finished recording')

# Save the recorded data as a WAV file
wf = wave.open(filename, 'wb')
wf.setnchannels(channels)
wf.setsampwidth(p.get_sample_size(sample_format))
wf.setframerate(fs)
wf.writeframes(b''.join(frames))
wf.close()
```

We record the voice for only 2 second because in our model we use a data contain a 1 second voice for each audio file so in this we record for 2 second and take the voice in middle of this audio file to avoid the noise and the silence .

After we record the voice we save it by output.wave file

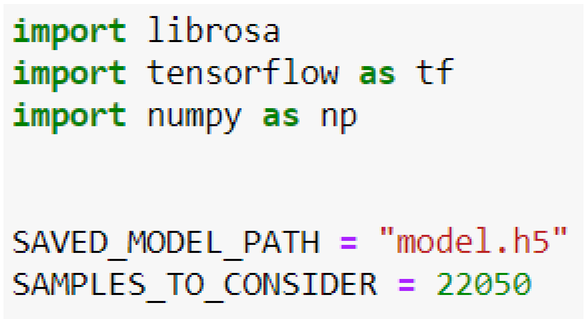
Prediction

Finally we reach to the final step in this model now we can record a voice and we have a model that contain data cam make us to detect the word that we say

The requirement of this step

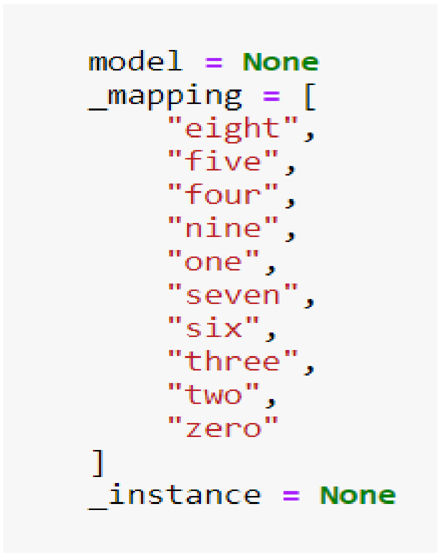
1- Librosa library
2- Tensorflow library
3- numpy library

import libraries and define variables



```
import librosa
import tensorflow as tf
import numpy as np


SAVED_MODEL_PATH = "model.h5"
SAMPLES_TO_CONSIDER = 22050
```

first step is to load the mapping that we are made in the first step of this mode .



```
model = None
_mapping = [
    "eight",
    "five",
    "four",
    "nine",
    "one",
    "seven",
    "six",
    "three",
    "two",
    "zero"
]
_instance = None
```

```python
# load audio file
signal, sample_rate = librosa.load(file_path)

if len(signal) >= SAMPLES_TO_CONSIDER:
    # ensure consistency of the length of the signal
    signal = signal[:SAMPLES_TO_CONSIDER]

    # extract MFCCs
    MFCCs = librosa.feature.mfcc(signal, sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
                                 hop_length=hop_length)
return MFCCs.T
```

and then we take the path of the audio file that we save in the 3rd step

and load it by librosa library and get the MFCC for it .

and make it a 3d array by putting new axis that make it a gray image

by using numpy .

before that we should ensure that audio file is 1 sec and if it large than 1 second we take the middle 1 second of this audio file .

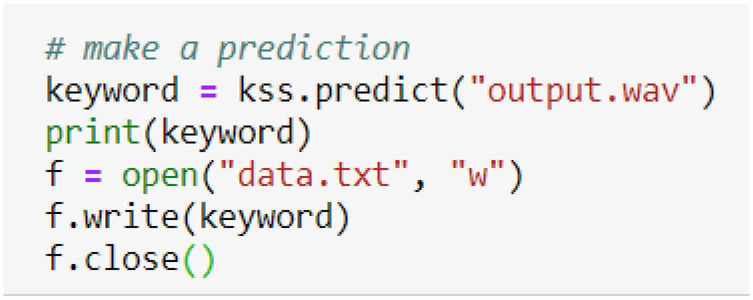
```python
# extract MFCC
MFCCs = self.preprocess(file_path)

# we need a 4-dim array to feed to the model for prediction: (# samples, # time steps, # coefficients, 1)
MFCCs = MFCCs[np.newaxis, ..., np.newaxis]

# get the predicted label
predictions = self.model.predict(MFCCs)
predicted_index = np.argmax(predictions)
predicted_keyword = self._mapping[predicted_index]
return predicted_keyword
```

then we send this MFCC data to the model so the model will compare the data which sent with the data which stored in the model .

after that the model will get the mapping word that compatible with the data of MFCC and put it in keyword variable .

```python
# ensure an instance is created only the first time the factory function is called
if _Keyword_Spotting_Service._instance is None:
    _Keyword_Spotting_Service._instance = _Keyword_Spotting_Service()
    _Keyword_Spotting_Service.model = tf.keras.models.load_model(SAVED_MODEL_PATH)
return _Keyword_Spotting_Service._instance
```

in the end we write the word which the model predicted in a file because if anyone need to use this word as a speech command service he will read it from the file and do what he want .

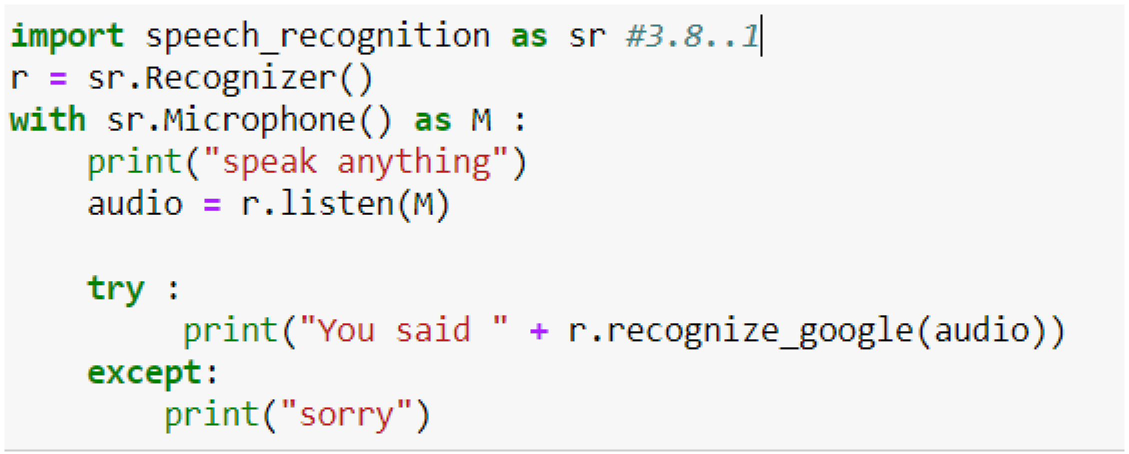


```
# make a prediction
keyword = kss.predict("output.wav")
print(keyword)
f = open("data.txt", "w")
f.write(keyword)
f.close()
```

This is the offline model if we haven't a network .

Online model

if you can provide a network in your application you can use a google assistant model by using this code



```python
import speech_recognition as sr #3.8..1
r = sr.Recognizer()
with sr.Microphone() as M :
    print("speak anything")
    audio = r.listen(M)

    try :
         print("You said " + r.recognize_google(audio))
    except:
        print("sorry")
```

You need to install the speech_recognition library . then this code is recording your voice and send it to google assistant model and this model will know the whole words that you talk and send it back in a string form .

This is the online solution for speech recognition problem

Version of libraries

Offline model :

tensorflow	2.4.1
pyaudio	0.2.11
numpy	1.15.
matplotlib	2.2.3
keras	2.4.3

Online model

speech_recognition	3.8..1