٢,١,١,٣ TensorFlow

TensorFlow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

TensorFlow was developed by the Google Brain team for internal Google use in research and production.

For signal processing there are some tools used for editing audio data :

Audacity : Audacity can be used for post-processing of all types of audio, including effects such as normalization, trimming, and fading in and out.

And some python modules :

K librosa NumPy IP[y]: IPython

pandas glob seaborn matplotlib

Crying translator

Inspiring theory : many Pediatricians talk about babies' crying that it differs by the difference of crying reason.

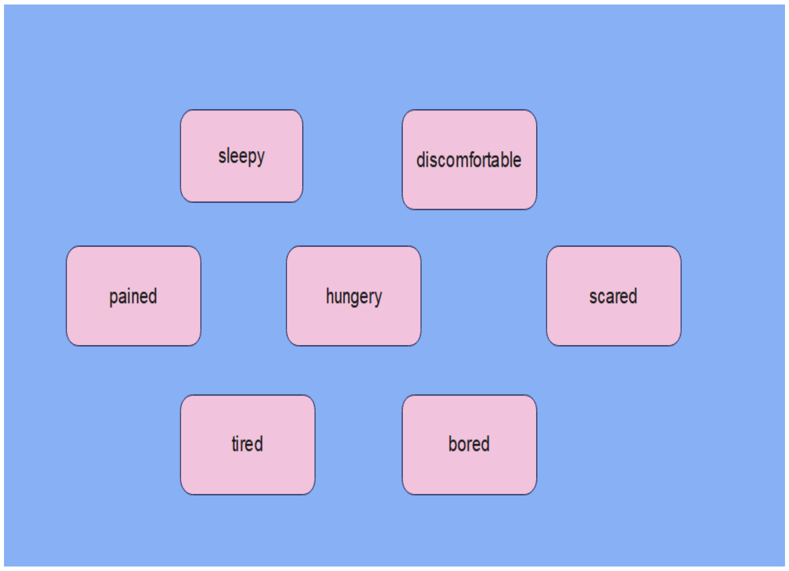
Based on these theories, we thought that it is possible to design a deep learning model capable of recognizing these patterns and predicting the cause of crying.

The process started by asking some questions :

I. What are the main reasons for baby crying?
II. What is needed data?

III. What are the cases that I should consider?

So beginning with main reasons :



And we may need data about each instance like baby's age and gender:

The age is distributed in five phases

(١st → from birth to ٤th month,

٢nd → from ٤th month to ٧th,

٣rd → from ٧th to the ١٠th,

٤th → from ١٠th to the ١٧th,

٥th → from ١٧th to the ٢nd year)

gender : male or female.

Cases should be considered :

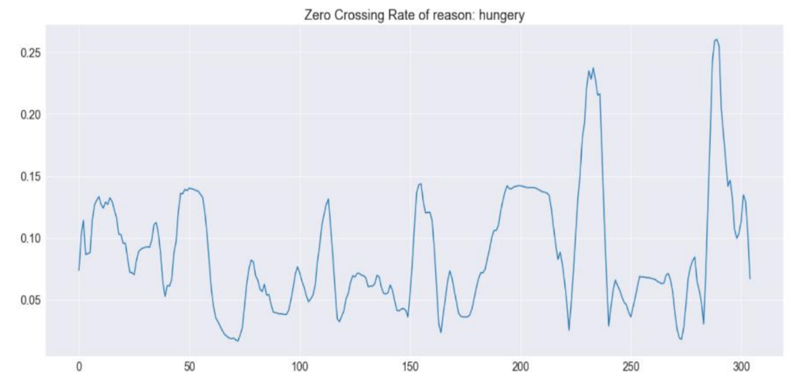
- If, for any reason, the model received not crying audio file, it have not predict the reason, the prediction should be "not crying". So, there will be a model that predict if the audio is crying or not then another model predict the reason of crying.
- While the babies grow their needs be more complicated so more data needed about them, it's related with data quantity. But there was no solution because of the lack of data.

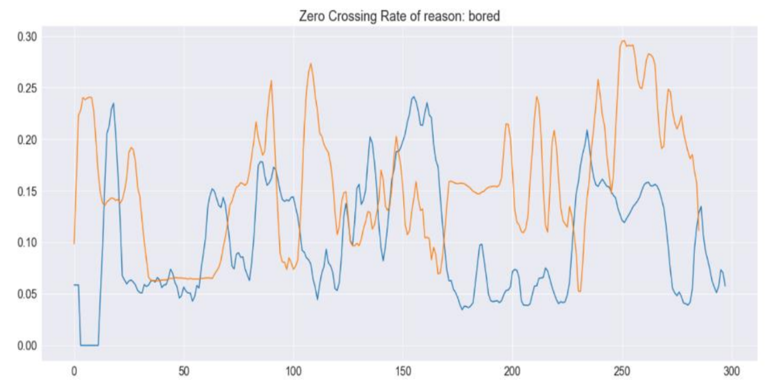
The steps followed :

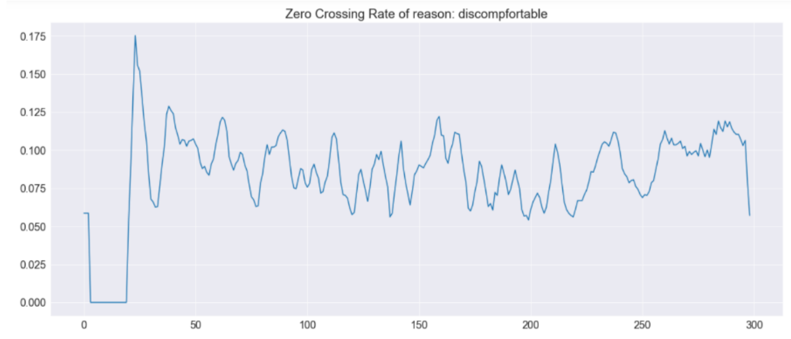
- Data gathering.
- data quality insurance and labeling the rest of the data (unsupervised & semi-supervised learning)
- data preprocessing
- models architecting
- training the models
- model serialization
- model validation

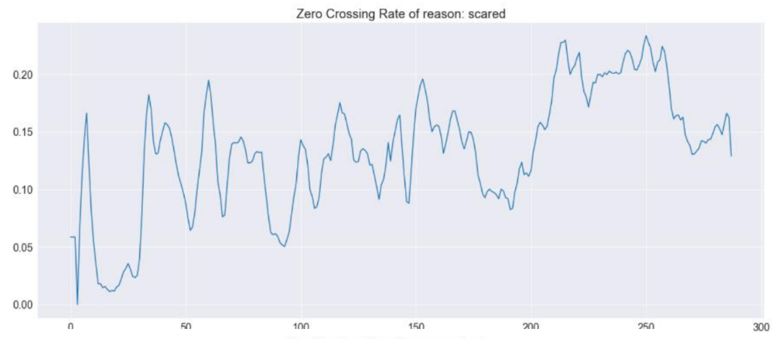
we gathered data manually from sound websites (AI sounds, free sounds…etc. and then took the main tags related to each sound file and put them in CSV file. for this reason we must ensure that this data is well labeled and then label the rest of the instances that was without the needed tags.

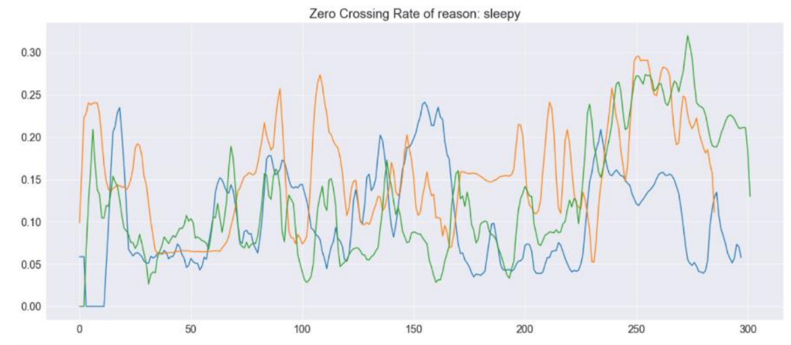
١) Get the Zero crossing rate for each reason

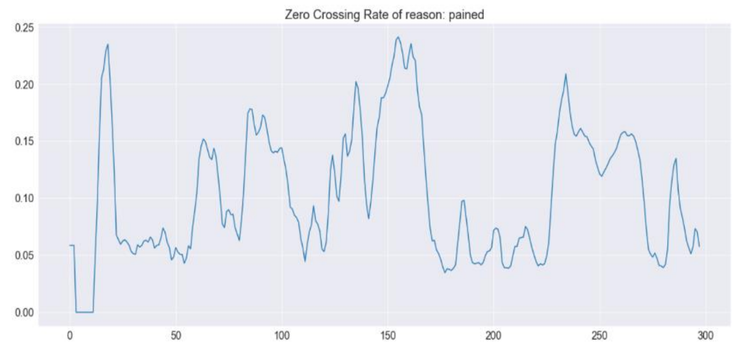


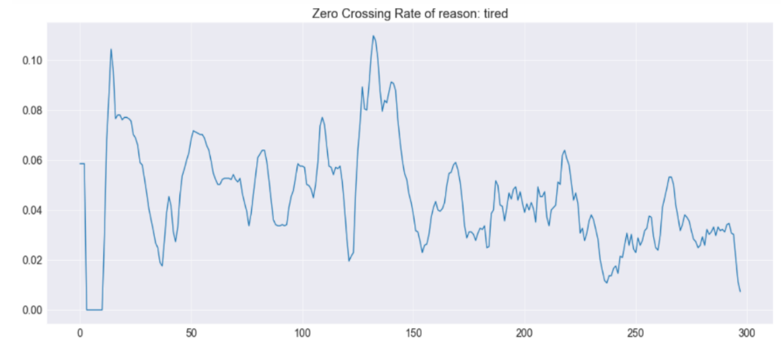




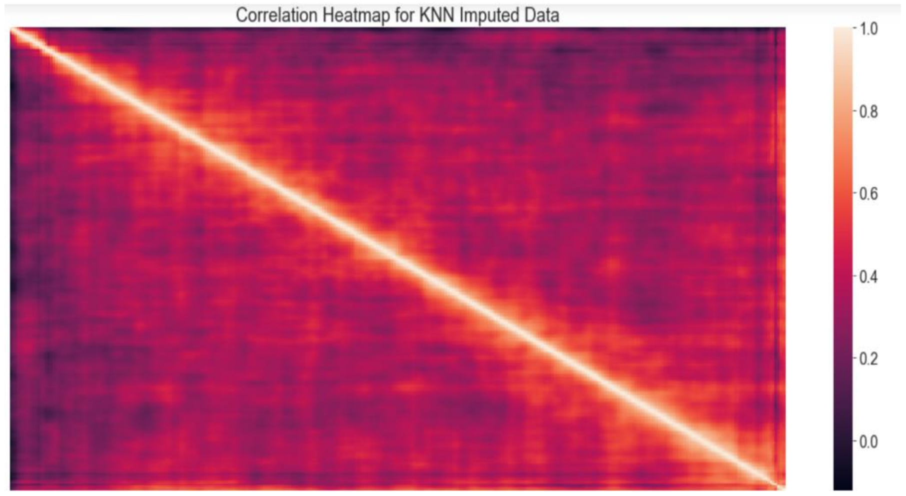






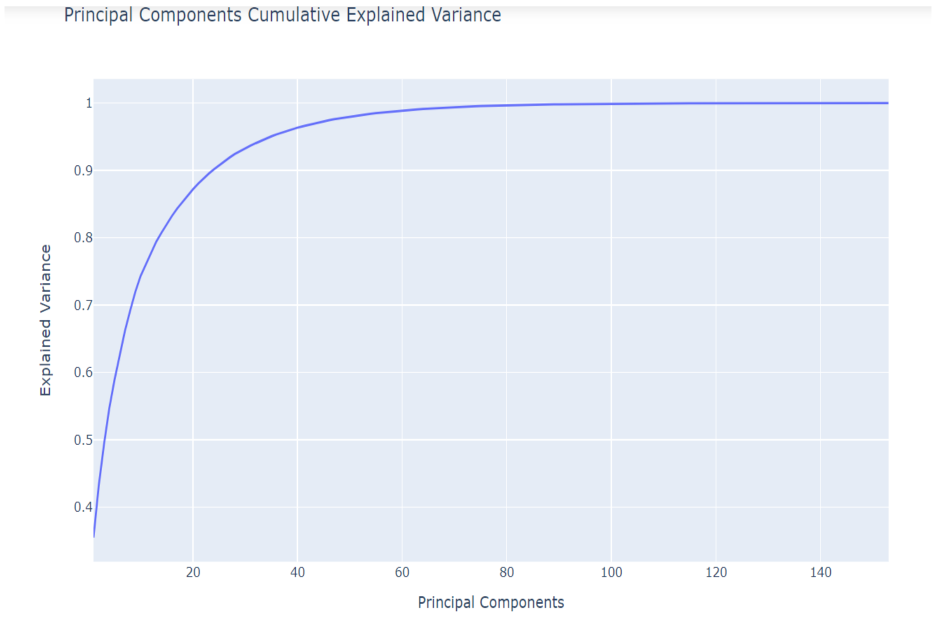


So, starting with this rate some algorithms are functioned as KNN and Kmeans to see the the correlation and visualize the clusters this step helped a lot to see the outliers so that was important for data quality and data quantity by labeling new instances.

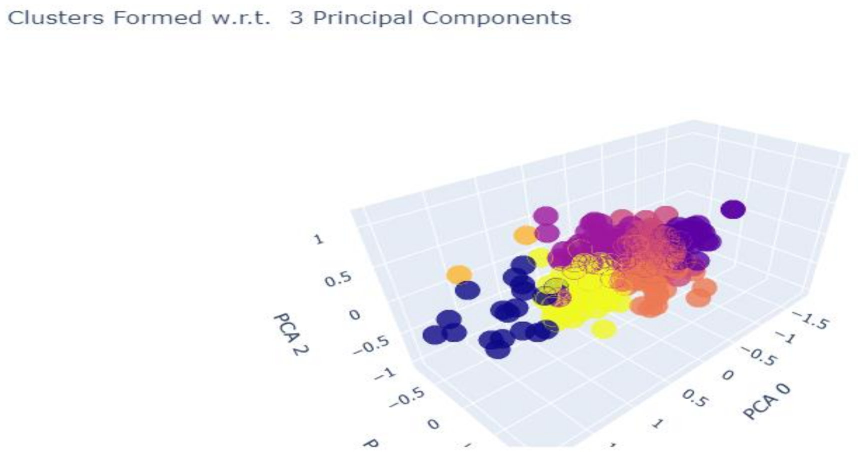


So, the data is imputed and we can see if "feature multicollinearity" is existing or no by heatmap

"Multicollinearity refers to the high correlation between two or more explanatory variables"



And this step helped a lot in doing principal component analysis.



Unsupervised Modelling

Since we want to identify the naturally occurring patterns within the data we first need to address the varying time length issue using dimensionality reductio

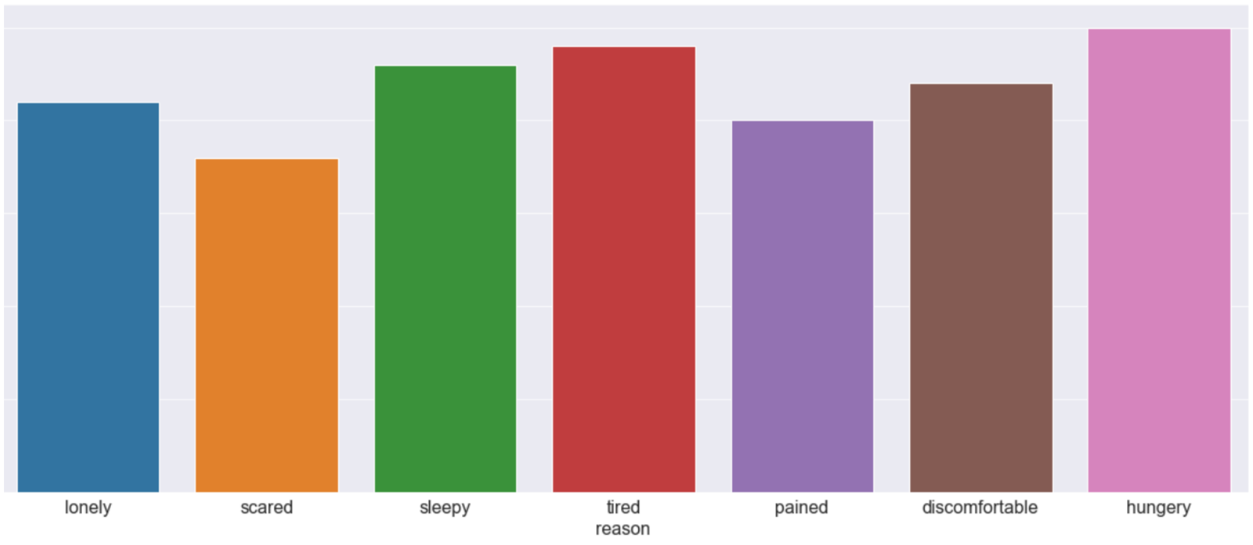
K-Means Clustering

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

With the reduced dimensions where the variance explanation is fixed we can then move on to cluster our data

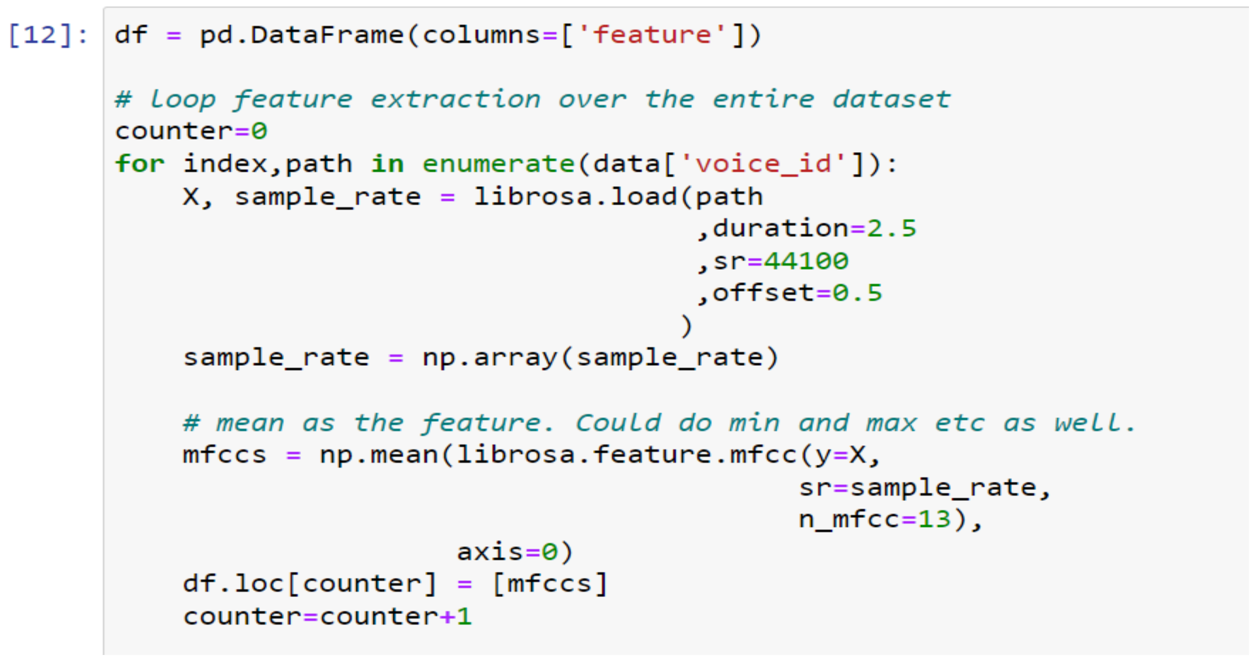
***is the training data is representative enough**

Before dealing with data there was another condition (proportionality of classes) it was the fear of Nonrepresentative Training Data. So starting with visualizing the number of instances the represent each class :



So, it’s clear that the numbers of instances that represent each class are convergent.

audio preprocessing :



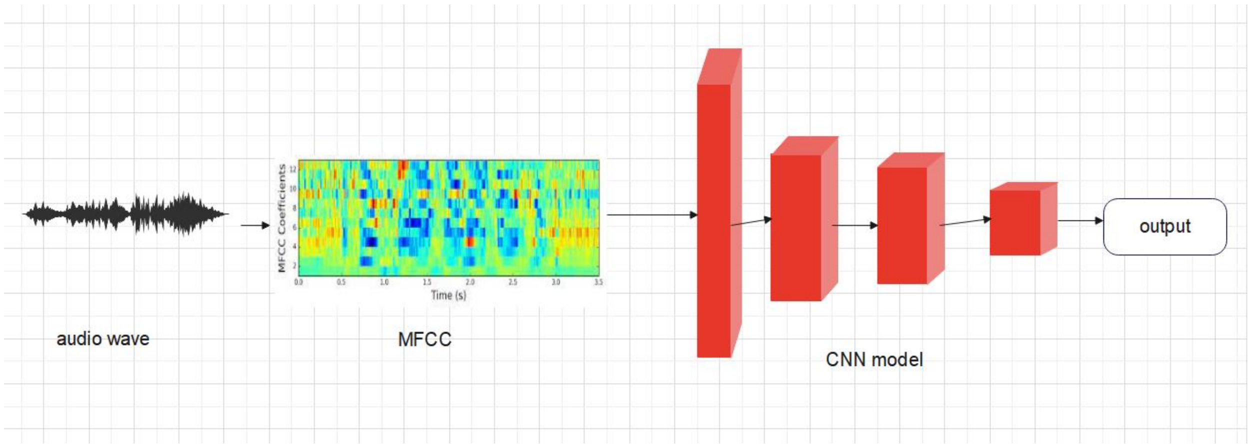
```
[12]: df = pd.DataFrame(columns=['feature'])

      # Loop feature extraction over the entire dataset
      counter=0
      for index,path in enumerate(data['voice_id']):
          X, sample_rate = librosa.load(path
                                        ,duration=2.5
                                        ,sr=44100
                                        ,offset=0.5
                                       )
          sample_rate = np.array(sample_rate)

          # mean as the feature. Could do min and max etc as well.
          mfccs = np.mean(librosa.feature.mfcc(y=X,
                                               sr=sample_rate,
                                               n_mfcc=13),
                          axis=0)
          df.loc[counter] = [mfccs]
          counter=counter+1
```

We started with extracting MFCCs features from each audio and In order to optimize space and memory, we're going to read each audio file, extract its mean across all MFCC bands by time, and just keep the extracted features, dropping the entire audio file data.

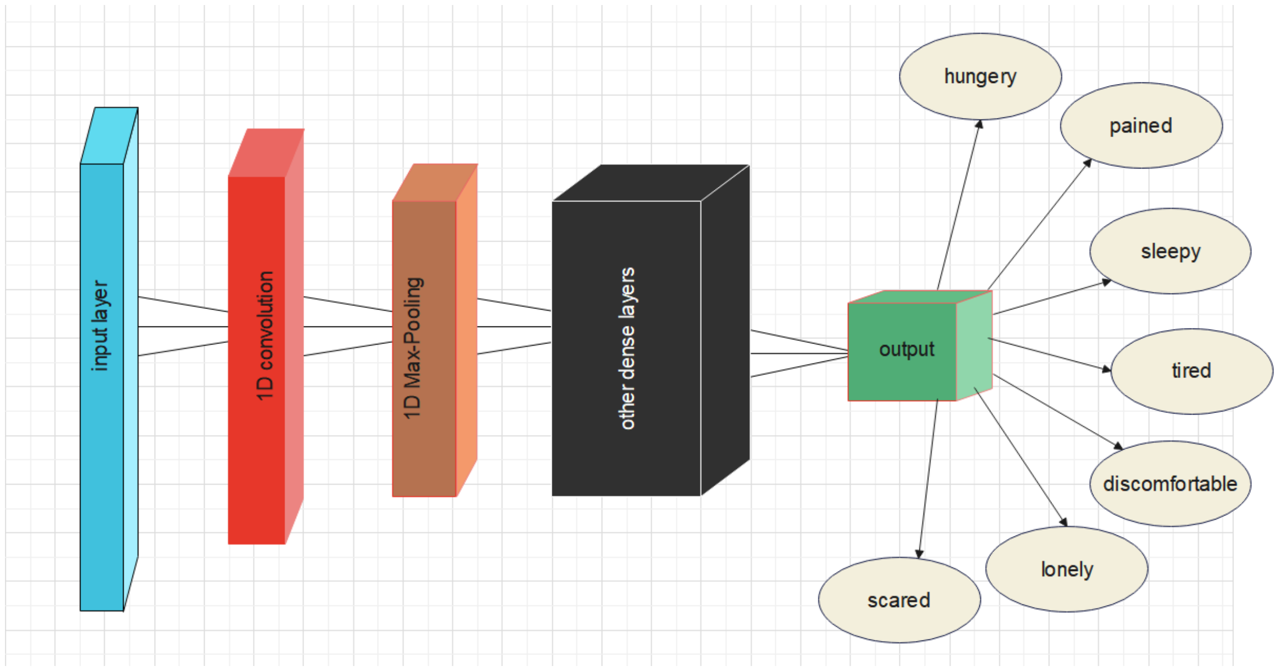
```
[34]: # Split between train and test
      X_train, X_test, y_train, y_test = train_test_split(df.drop(['voice_id','target'],axis=1)
                                                          , df.target
                                                          , test_size=0.25
                                                          , shuffle=True
                                                          , random_state=42
                                                         )
```

Then splitting training set and testing set and finally encoding labels by “one-hot encoding”.

model architecture :

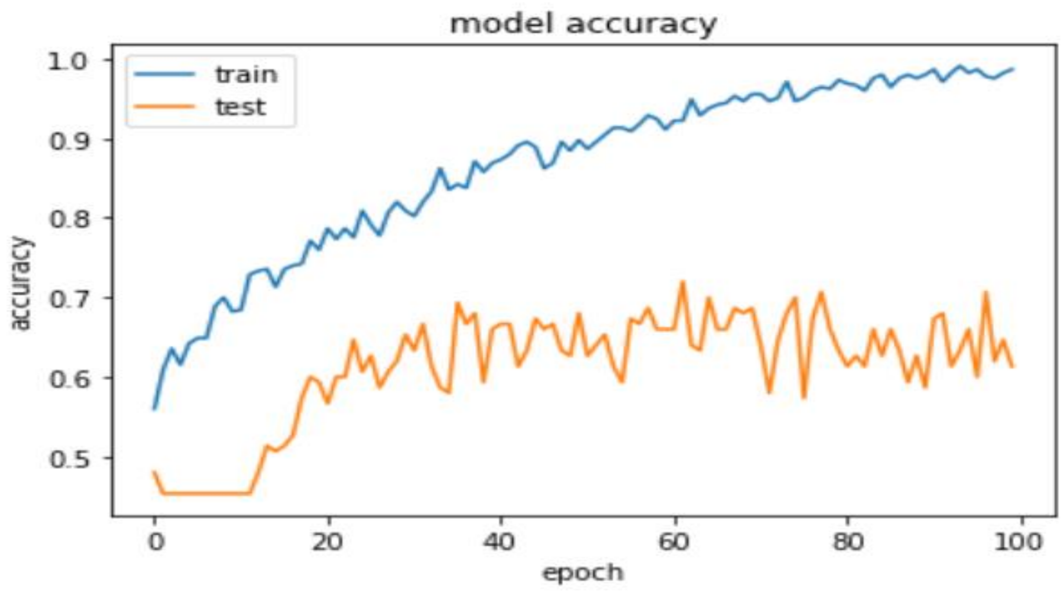


zooming in the CNN model :



Model optimization :

```
In [39]: opt = tf.keras.optimizers.RMSprop(learning_rate=0.00001, decay=1e-6)#lr=0.00001, decay=1e-6
         model.compile(loss='categorical_crossentropy', optimizer=opt,metrics=['accuracy'])
```



Model serialization :

MODEL SERIALIZATION

```
# save model and weights
model_name = 'crying_translator_model.h5'
save_dir = os.path.join(os.getcwd(), 'saved_models') # creates a folder in current working directory (CWD)

if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)   # adds file in folder
model.save(model_path)
print('Save model and weights at %s' % model_path)

model_json = model.to_json()
with open('model_json.json', 'w') as json_file:
    json_file.write(model_json)
```

MODEL VALIDATION

```
# Loading json and model architecture
json_file = open('model_json.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# load weights into new model
loaded_model.load_weights('saved_models/crying_translator_model.h5')
print('Model is loaded from disk')

# keras optimiser
opt = tf.keras.optimizers.RMSprop(learning_rate=0.00001, decay=1e-6)
loaded_model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
score = loaded_model.evaluate(X_test, y_test, verbose = 0)
```