Emotion recognition from audio

Urszula Wasowska, Maciej Kapitan

February 2024

1 Motivation

Our project is about model that is able to recognize emotions from speech. Our motivation was to check how neural networks can handle with audio data, and also, we wanted to check if there is a mathematical method for recognising emotions. We as humans are able to get emotional sense of speech, but we cannot explain how we do that. It's nearly impossible to write step by step algorithm which could analyze audio data and by using some ifs, loops etc. say what emotion speaker has. For problem like that, neural network are great solutions. They are very complicated structures which are able to emulate tasks that we think we do using our intuition. But now, when machine learning is rapidly growing, it turns out that for many such tasks there can be algorithm. The systems based on our work may be useful for example with telemarketing - nowadays in many companies there are chatbots or callbots. It allows to save time and money for hiring tele-consultants. We think that it may be very useful for companies to recognize customers emotions and react to them - for example for customers that are angry, there can be less calls or maybe more attractive offers.

2 Dataset

The dataset contains audio files from two speakers. The audio are 1,2-seconds long. Speakers tell the same sentence e.g. "Say the word bar", but with different emotions in their voice. There are seven emotions:

- angry
- disgust
- fear
- happy
- neutral
- sad
- pleasant surprised

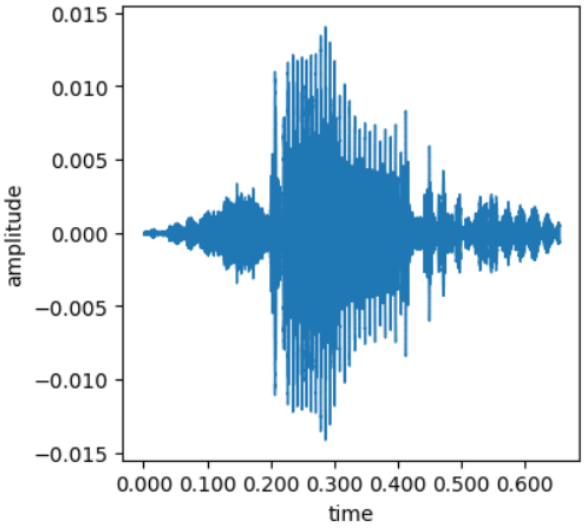
Audio of each category are stored in separate directory. Files are in wave format

3 Audio file format

Our audio files are stored in .wav format. After loading it with librosa package we got two values:

- First value (y) is very long array - each value means amplitude of sound in certain moment
- Second value (sr) is sample rate - it informs how many waves per second our sound has

Having this data we can plot a waveplot:



But this kind of data is not very informative. The sequence of amplitudes is extremely long and it would be vary hard even form powerful recurrent neural network to handle with them. But there is a method for transforming this data into more readable form - Fourier transform. Having timeseries of amplitudes, using Fourier transform we could rewrite it as a mapping of different frequencies into different amplitudes.

Fourier series is mathematical method which enables to rewrite periodic functions into sum of sines and cosines with different frequencies. A constant sound is a timeseries of amplitudes with given frequency. With Fourier transform we can transfer this timeseries into sum of sines and cosines of different frequencies and amplitudes;

$$S(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos\left(\frac{2n\pi}{T} x\right) + b_n \sin\left(\frac{2n\pi}{T} x\right) \right)$$

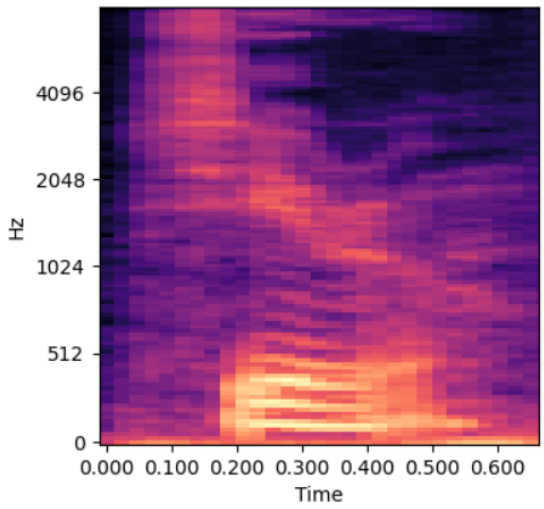
Where a and b are amplitudes and number which multiply T is period. Using calculus we can count these amplitudes for different frequencies:

$$a_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \cos\left(\frac{2n\pi}{T} x\right) dx, \quad n = 0, 1, 2, \ldots,$$

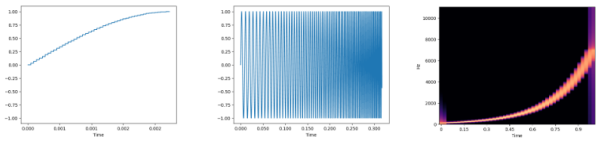
$$\mathrm{b}_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \sin\left(\frac{2n\pi}{T} x\right) dx, \quad n = 0, 1, 2, \ldots,$$

Having that we can transform timeseries of amplitudes into plot where on the x-axis we have different frequencies and on y-axis - amplitudes. This algorithm is implemented in the python library called librosa. It requires a lot of time to prove this equation and assign correct amplitudes for frequencies, so we won't be describing this in detail, but we wanted to give some intuition what is the base of preprocessing sounds.

Bu what if our sound isn't just one, constant sound, but for exapmle speach or music? Then we can divide timeseries for shorter fragments and apply Fourier transform for each one, so the number of timesteps is reduced but for every timestep, instead of amplitude, we have set of different frequencies with assigned amplitude. To show this on plot we can use spectogram plot. X-axis is time, y-axis is frequency and brighter colour mean bigger amplitude. As we can see, the frequencies in the y axis are in logarithmic scale - because this is a way that we hear sounds - we can easier differentiate frequencies 512 and 612 than 1012 and 1112.



A good example to show these transformations is chirp sound from librosa package. It is a sound with increasing frequecy.



3.1 MFCC

In our project we transform sound from wave format into The Mel-Frequency Cepstral Coefficients (MFCC), which is improvement of mel spectogram,

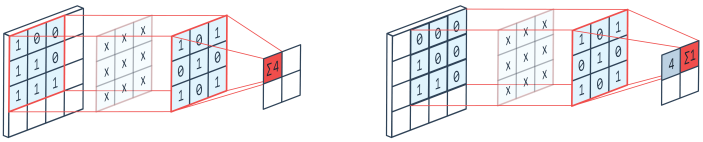
Here's a brief characterization of the individual steps in the MFCC extraction process:

- Preemphasis: Initially, the audio signal undergoes a preemphasis process, which involves amplifying higher frequencies to reduce the impact of noise.
- Frame Division: The audio signal is divided into short frames, and each frame is analyzed separately. This step is crucial to assume that signal features are constant over a short period.
- Transformations: Subsequently, each frame undergoes transformations, such as FFT (Fast Fourier Transform), to obtain a representation of the signal in the frequency domain.
- Mel Filterbank: In this phase, a Mel filterbank is applied, designed based on the human auditory system. This ensures a better representation of features that are more consistent with human sound perception.
- Logarithmization: The results from the Mel filterbank are logarithmized to adapt to human hearing, which operates in a logarithmic manner concerning sound intensity.
- Cosine Transform: The next step involves applying a cosine transform (Cepstral Transform) to obtain cepstral coefficients.
- MFCC Extraction: Finally, a few initial cepstral coefficients are selected as MFCC, representing the audio signal.

In this picture on axis x we have different frames on which our sound is splitted, and in the y axis we have different frequencies. The brighter colour means the brighter amplitude of given frequency in certain moment.

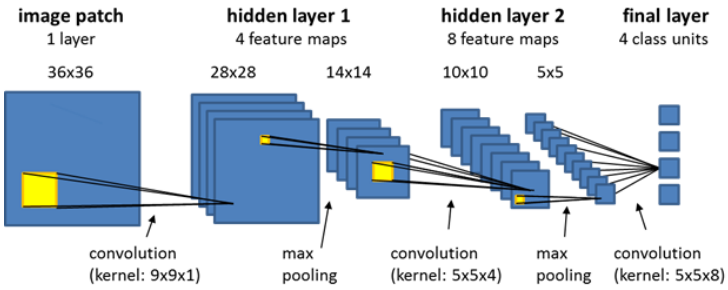
4 Methods

- Convolutional layer - convolutional layers are special type of neural network layers that handle very well with images or sequence data. They contains different filters which we can define as channels. Each filter has fixed size . These filter walk through input data and make matrix multiplications with each fragment. When one filter has walked through whole input, we got one channel - and for every filer we have differen output image



Source: data science stack exchange

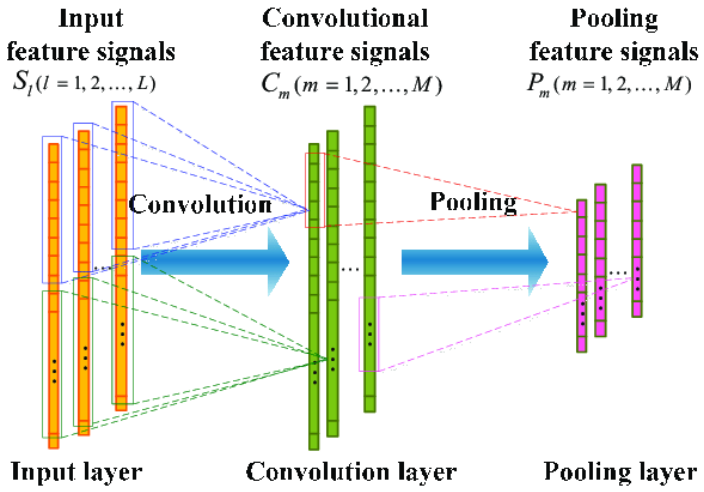
In the image above we can see how single filter works with input data. In this picture, filter of size 3x3 for 2d data iterates through input image and multiply pixel values by its weights, adds them and as a result it return one pixel. As we can see, convolutional operation reduces image size, but it can be handled by adding padding pixels around image.



Source: Trimble recognition help

In this picture we can see how the whole convolutional layer looks like. It genrates many channels from sigle picture. MaxPooling layer is a layer for reducing size of an image - it's for example 2x2 filter which iterates through image and choose max pixel from each square - as a result we got image with reduced size. After this operations, when sizes of result images are small enough, we can flatten input into single vector and user normal dense layers.

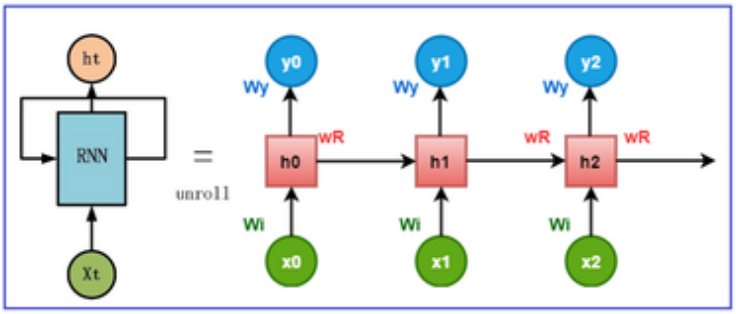
In our example we use 1d convolutional layers which we can see in the below image



Źródło: Research gate

- LSTM (Long Short-Term Memory) Layer - is an example of a recurrent layer in a neural network. Recurrent layers differ from regular layers in that they are adept at processing sequences, such as audio in this case. In recurrent layers, the input for the i-th feature is the output of the (i-1)-th feature and the i-th signal from the previous layer.

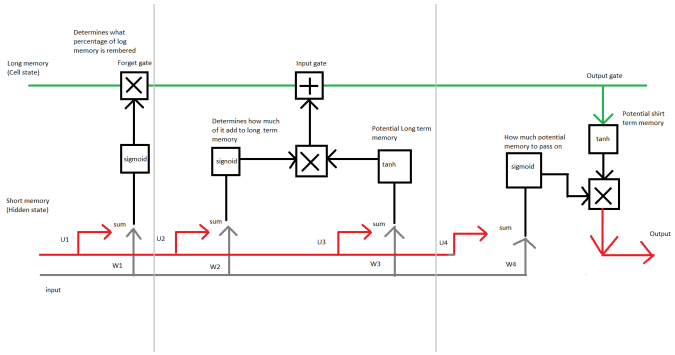
  This means that during the calculation of activation, the activation from the previous layer and the previous feature is taken into account. Therefore, in the process of processing a sentence, the order of words becomes crucial.



Źródło: Dot net tutorial

In LSTM layers, improvements have been introduced compared to regular recurrent networks. An LSTM network addresses the issue of vanishing or exploding gradients by having two information channels - one responsible for long-term memory and the other for short-term memory. The information passed between neurons is controlled by special gates that decide whether a particular signal from the previous neurons should be passed on or forgotten. This allows them to effectively convey important

information over longer periods, which is crucial for processing long-term dependencies in sequential data.



- Flatten Layer - this layer is designed to flatten the data. Since each sound frame, after coming is a vector, and a whole sound is a matrix, the entire dataset becomes three-dimensional. The flatten layer transforms matrices representing sentences into vectors.
- Dropout Layer - this layer prevents overfitting, one of the most common issues during the training of neural networks. In brief, overfitting occurs when the network's parameters are too precisely tailored to the training data, causing the model to perform poorly on test data. The dropout layer addresses this problem by randomly deactivating some neurons during the learning process. This action prevents situations where weights in certain neurons become excessively large or extremely small. Consequently, the algorithm doesn't overly rely on a subset of neurons but instead has to depend on the entire network.
- Batch Normalization - The aim of usage Batch Normalization is to change distribution of inputs to more normal. All machine learning algorithms work the best with known distributions, e.g. normal. Batch normalization is responsible for standarizing the data from each batch and after that, it applies a linear transformation to scale and shift the data. Batch normalization helps stabilize the optimization process, reduce internal covariate shift, and improves gradient flow, leading to faster convergence and better generalization.
- Dense layer - for dense layers we use relu ($max(0, x)$) activation function - these function is considered to be the best activation for hidden layers. Its goal is to break linearity of calculations - because if activation would be linear, then the output would be just linear function of input. That allows to break symmetry and find some non linear dependences.
- Activation Layer - the activation layer computes the value of the softmax funcitons which return probabilities of inputs belonging to one of 7 classess.

4.1 Optimizers

We used the popular Adam optimizer for learning optimization. Optimizers in neural networks enable faster and more efficient training of the model.

A common problem in neural networks is that for different data batches, gradients of individual features may differ from each other. Therefore, changing the gradient in one batch may decrease the error, while in another batch, it could increase it. Optimizers, in the process of updating weights, also consider the change in weights in previous steps, making the learning process more 'smooth'.

Adam optimizer is considered to be the best algorithm - it connect momentum optimizer and RSM prop. For both of these algorithms the base of their work is to calculate exponentially weighted average from previous gradients, so that current weight update is some mixture of current gradient and previous average. The main goal of these operations is to change trajectory of sliding gradient so it could reach global minimum.

We also use callback ReduceLROnPlateau which is responsible for decreasing learining rate by certain factor, only if validation accuracy (or some other metric) hasn't improved for some fixed number of epochs.

5 Experiment

We starts from iterating through directory with audio files. Sounds from different categories are stored in different folders. We add images path and their category for data array. Next we calculate mfcc features for every image - because we need to have mean and std for our dataset in order to standarize it. Because loading all mfcc features into array is memory expensive, we calculates running mean and running std which can return results similar to normal mean and std when the data is big enough. Then we split dataset into train, val and test. Next we are gonna create generator for splitted datasets. Each generator returns batch of data with given size and index. First element of batch size are mfcc features of audio files and second are one hot encoded labels. Shape of data from batch is (batch_size, n_frames, n_mfcc_features) That means, every audio files is splitted into n_frames frames (longer sequences are truncated and shorter are padded with zeros) and for every frame we got n_mfcc_features features.

5.1 Training

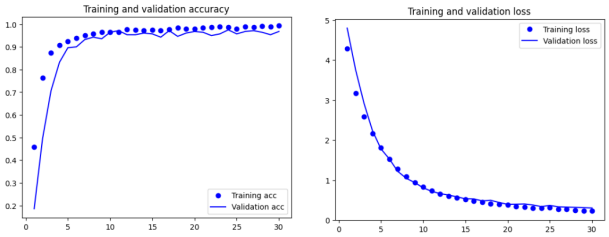
We tested two models - one using CNN (convolutional neural network) and second using LSTM (Long-short term memory).

Fist model contains Conv1d layers - we use one dimension because voice message is a sequence, another frequencies and from some other literature and tasks we suppose that it can handle better that conv2d layers. With conv1d we appy filters in second axis of dimension (batch_size, n_frames, n_mfcc_features) - so we move with filter through time axis and mfcc_features are channels (like rgb in images). We also use batch normalization which gave us better accuracy. As regularization method we use dropout and L1 L2 regularization.

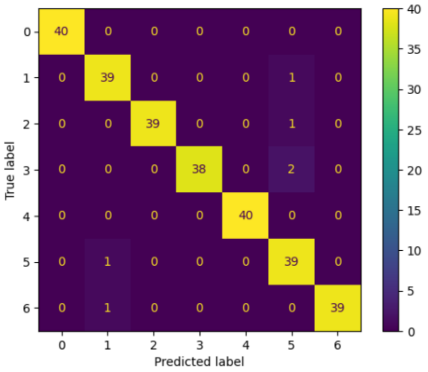
```
train_datagen = sd.train_gen()
val_datagen = sd.val_gen()

model = models.Sequential()
model.add(layers.Conv1D(32, 3, activation='relu', padding="same"))
model.add(layers.MaxPooling1D(pool_size=(2,), strides=None, padding="same"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1D(32, 3, activation='relu', padding="same"))
model.add(layers.MaxPooling1D(pool_size=(2,), strides=None, padding="same"))
model.add(layers.BatchNormalization())
model.add(layers.Flatten())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(128, activation='relu', kernel_regularizer=regularizers.L1L2(l1=1e-3, l2=1e-3)))
model.add(layers.Dense(64, activation='relu', kernel_regularizer=regularizers.L1L2(l1=1e-3, l2=1e-3)))
model.add(layers.Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(train_datagen, validation_data=val_datagen, verbose=True, shuffle=False,
                    epochs=30, steps_per_epoch=train_datagen.n_batches, validation_steps=val_datagen.n_batches,
                    callbacks=callbacks_list)
```

The results of our algorithm are shown in the picture below:



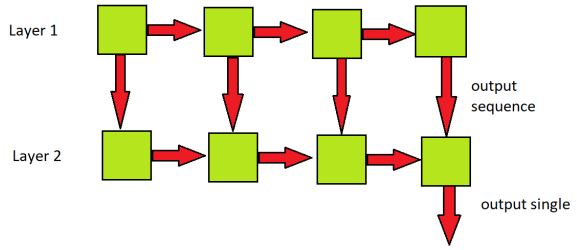
As we can see, the model is learning very fast, and there is no overfitting - both training and validation loss are decreasing the same way. We also have evaluated our model with test data and plotted confusion matrix:



The test accuracy is around 97% - we think it's not so bad, as we have 7 categories and only 280 samples from test data.

5.2 Recurrent Neural Network

Our recurrent model contains two LSTM layers - one is returning sequences - so that means that result is not only from final neuron but from all previous neurons. The second returns only one item from sequence

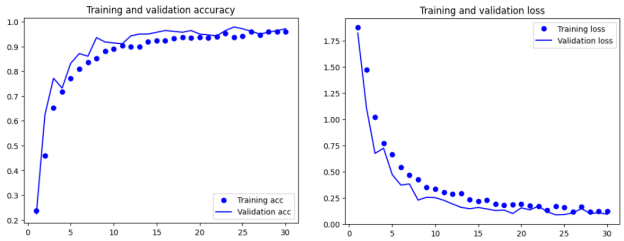


In LSTM layer there is also some dropout - it works the same as normal Dropout layer, but it implemented between LSTM neurons. Our model is in the below picture:

```
train_datagen = sd.train_gen()
val_datagen = sd.val_gen()

model = models.Sequential()
model.add(layers.LSTM(64, recurrent_dropout=0.2, dropout=0.1, return_sequences=True))
model.add(layers.LSTM(32))
model.add(layers.Flatten())
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(train_datagen, validation_data=val_datagen, verbose=True, shuffle=False,
                    epochs=30, steps_per_epoch=train_datagen.n_batches, validation_steps=val_datagen.n_batches)
```

We have got following results:

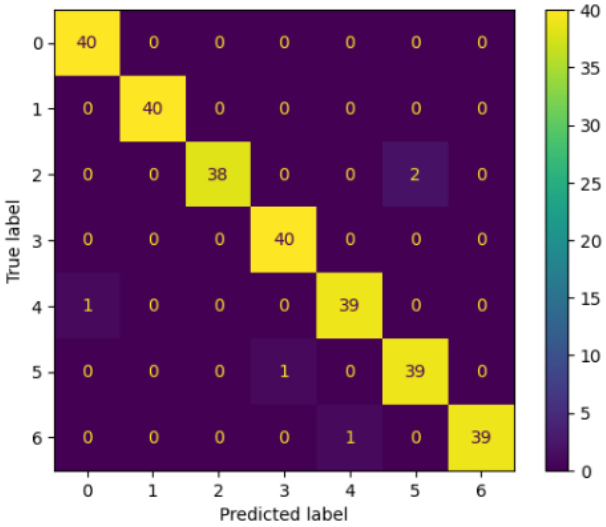


And test accuracy for that model was around 94% so it's slightly worse that with CNN.

5.3 Combination of CNN and LSTM

Lastly, we've decided to join this two methods and try combination of CNN and LSTM. We have got the best test accuracy - 0.9821

```
model = models.Sequential()
model.add(layers.Conv1D(32, 3, activation='relu', padding="same"))
model.add(layers.MaxPooling1D(pool_size=(2,), strides=None, padding="same"))
model.add(layers.BatchNormalization())
model.add(layers.LSTM(64, recurrent_dropout=0.2, dropout=0.1))
model.add(layers.Flatten())
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(64, activation='relu', kernel_regularizer=regularizers.L1L2(l1=5e-3, l2=1e-3)))
model.add(layers.Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(train_datagen, validation_data=val_datagen, verbose=True, shuffle=False,
                    epochs=30, steps_per_epoch=train_datagen.n_batches, validation_steps=val_datagen.n_batches,
                    callbacks=[model_checkpoint_callback])
```



6 Conclusion

Our conclusion is that CNN neural networks can handle slightly better with that type of input data - but on the other hand, LSTM neural network hasn't required much regularization - that means, these model can easily find significant patterns inputs.

Second, we think most important conclusion is that training neural network to achieve very good accuracy takes much time - this is because big neural nets are so complicated that we cannot explain their way of work. Sometimes even theory with literature doesn't work well with input data - in developing very good model we have to rely on our intuition and of course - on computational power of our laptops and time.

We have also tested our model with my own voice. The result was suprisingly correct!