Speech Emotion Recognition using CNN

Shubhankar Mangesh Borse | GtID 903331341 | sborse3@gatech.edu

1. Abstract

Emotion recognition from speech is one of the wide areas in research off late. This project aims at constructing a robust model for Speech Emotion Recognition. The secondary goal of the project is to analyse successes and failures while building this model and give some insight into how to build such a model in the process. The proposed method was implemented and then improved by changing different aspects of the model. The improvements are highlighted along the way and finally, the model gives accuracy of 70.8% on the test set.

2. Introduction

Most of the actions taken by humans can be classified with their emotions. In turn, human emotions are an integral part of their daily interactions. Hence, classifying these emotions becomes an important task. It is also necessary to recognize, interpret and respond to a particular human's emotional state.

Speech is one of the primary ways to express human emotions, and in most cases, recognizing the tone of their speech is sufficient to classify how they're feeling at any given time. Hence, this project works with human speech as a sufficient means to classify the subject's emotions.

Some of the applications of speech emotion recognition are in call centres, where the call automatically switches to a human operator if the client shows irritation while speaking to the machine. However, there's a deeper reason to why I'm interested in this project. I would like to run the algorithm I develop on speech samples of human subjects who come from not-so-fortunate backgrounds and try to figure out how they actually feel.

Emotions in speech are hard to classify. The boundaries of classification are quite blurred. This is very different from conventional classification problems, which have well-defined classes. When it comes to emotions, a lot of the classes are generally highly correlated. For example, a person can be 'happy' and 'excited' at the same time. They're two different classes of emotions, but it would be hard to differentiate between both of them.

The question which now arises is, how do we characterize speech, in such a way that we incorporate all of its aspects which would give us emotional information [1]. The characteristics of the speech sample are the descriptor features, which are broadly computed in two stages [2]. The first stage talks about the short-time features, which are also called Low Level Descriptors (LLD's). These descriptors are basically characteristics of the particular phoneme (such as formants, etc). The next stage to feature extraction basically aggregate LLD features over time and find their statistics. These statistics give us High Level Descriptors and are called High Level Statistical Features (HSF's).

Once the features are extracted, we need to apply some dimension reduction techniques on the features to obtain a compact representation [3] and perform a machine learning algorithm which can classify the sentences based on our input classes.

3. Literature Review

There are various ways of building a model to classify the different emotions [4]. The methods depend highly on how the literature perceives emotions. Initially, the features used for speech characterization were hand-crafted features used by the traditional approach [5]. These features weren't as good as the LLD's and HSF's, as shown in [2].

The literature on emotion detection talks about an ongoing debate on Low Level vs High Level descriptors, and which one out of both of them give us more information to classify emotions. In [2], the authors at Microsoft Research Group say that emotional information lies in the temporal data, and hence the HSF's should be given more importance. Hence, they use RNN's to build the classifier.

Recently, there has been growing interest to apply deep learning to automatically learn useful features from emotional speech data. The authors in [6] used a Deep Neural Network (DNN) on top of traditional utterance-level statistical features to improve the recognition accuracy compared to conventional classifiers such as Support Vector Machines (SVM). The work in [7] used deep feed-forward and recurrent neural networks (RNN) at the frame level to learn the short-term acoustic features, followed by traditional mapping to a sentence-level representation using extreme learning machines (ELM). In some other cases, the authors used both convolutional and recurrent layers to learn the mapping directly from time domain speech signals to the continuous-valued circumplex model space of emotion.

4. Hypothesis

RNN's are used to capture temporal information in the speech sequence. This is done by adding feedback to the system. Most of the systems use bidirectional RNN's to classify speech, which makes one of the main advantages of RNN's (real-time computation) obsolete.

To capture temporal information, we require the property of "statistical invariance in time". This means, the weights of the same word are shared in time. Incidentally, this is one of the strong suits of CNN's. The idea is that each layer in the CNN would be sensitive to a different aspect of a spoken sequence, no matter where it would lie in the sequence.

Thus, the primary hypothesis of the project is that the CNN would effectively be able to capture both non-temporal and temporal information in a spoken sequence, such that it would create a model which is robust enough to classify emotions effectively.

5. Implementation

The project is built in python. I simulated it on python 3.6.2 and used the LibRosa library for audio analysis and feature extraction. The Machine Learning model was built on keras framework, and managing labels and datasets is done using pandas.

The steps followed are shown by the flowchart in Figure 1. These are explained in detail subsequently.

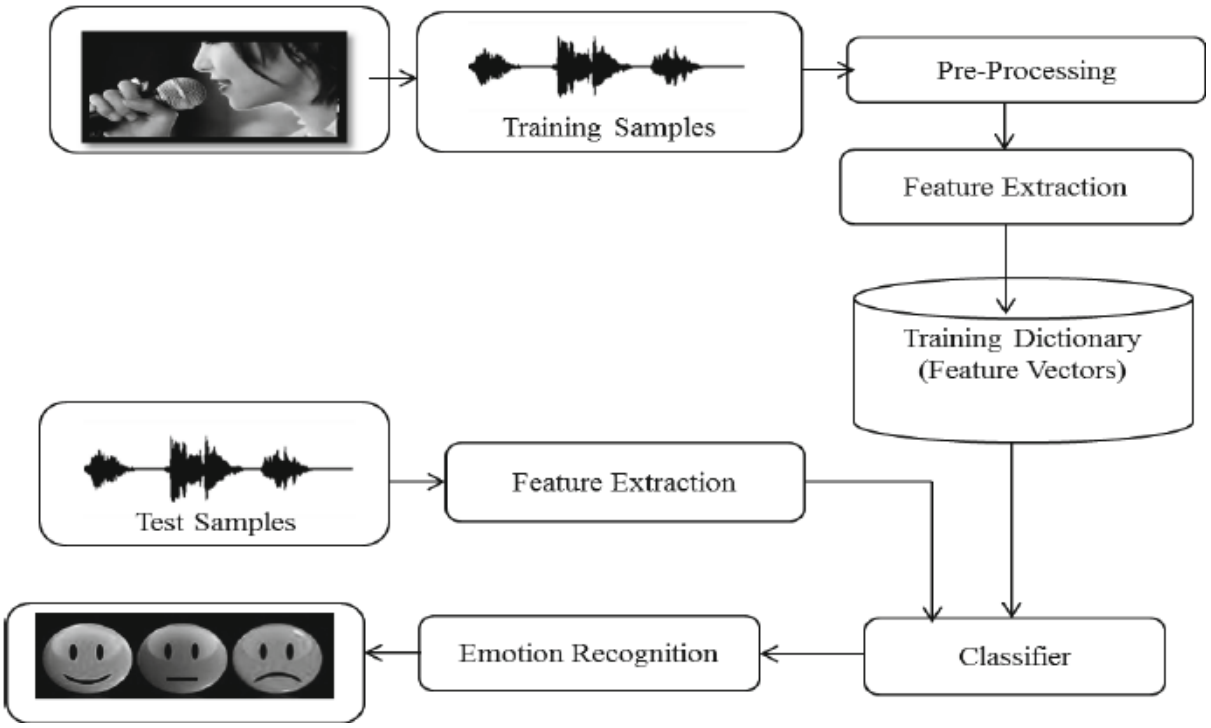

Figure 1: Flowchart for Emotion Recognition

5.1. Dataset Acquisition

To eliminate any sort of bias in the system, I have merged two datasets to form a large database. These datasets are both available in open source.

1) Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) [8]: The dataset has around 1500 audio file input from 24 actors. 12 of them are male and 12 are female. These actors record short audios in 8 different emotions. Audio files are named in such a way that the 7th character is indicative of the different emotions that they represent.
2) Surrey Audio Visual Expressed Emotion (SAVEE) [9]: The dataset contains of 500 audio files recorded for 4 different male actors. The labels are extracted using the first two characters of the file name, which correspond to the different emotions.

I aimed to classify 5 different emotions: Calm, Happy, Sad, Angry and Fearful. Hence, the classifier is built for 5 class labels. Total number of samples comes to 1200 once this is done.

5.2. Information extraction and parsing labels

All voice recordings from both datasets are stored in one directory, and the names of each file are stored in a list. Their labels are then retrieved using the information provided in the previous section, and then files corresponding to those labels are read. To make the inputs consistent, the first 3 seconds of samples are extracted from these files. The files have speech segments sampled at 44.1 kHz.

5.3. Windowing and Feature Extraction

I used the LibRosa library in python for extracting features from the speech segments obtained before. The 3 second segment was windowed using a 20ms Kaiser window [10]. The Kaiser window aims to increase the ratio of the main lobe to side lobe in it's spectrum. The window representation, as defined in it's paper is shown in figure 2.

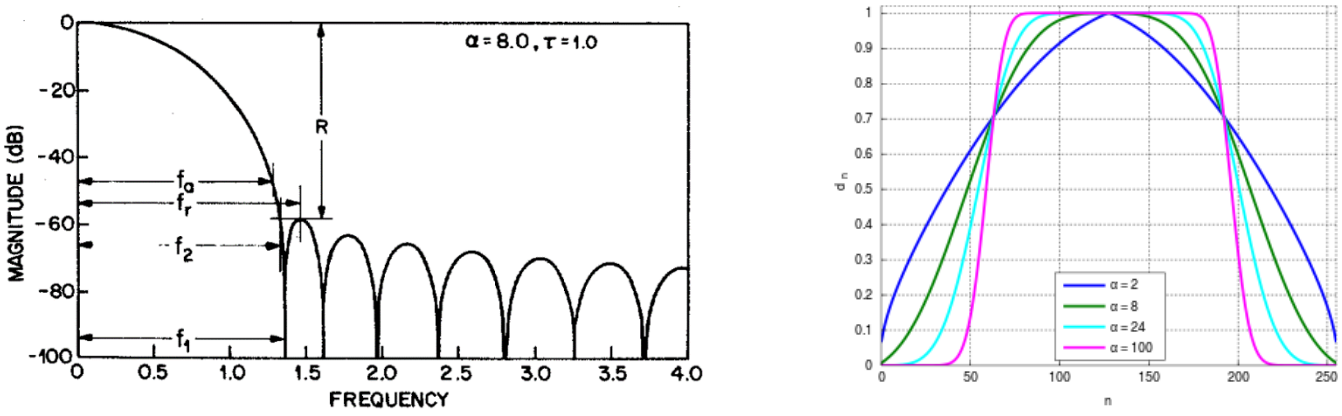

Figure 2: The Kaiser Window in Frequency and Time

As seen from the figures, we use this sort of architecture (smooth edges) to remove the ringing effects which are produced by box windows, due to their sharp edges. On the other hand, we want the window to have a lowpass structure and have a large main lobe compared to side lobes in the frequency domain, so that most of the information in the frame is passed. The width of main lobe is given by $4\pi\alpha/M$, where M is length of the window.

Mel Frequency Cepstral Coefficients: The mel-frequency cepstrum [11] is highly effective in audio recognition and in modelling the subjective pitch and frequency content of audio signals. Mel scale is an approximated logarithmic scale of frequency. The Mel-scale is calculated as:

$$Mel(f) = 2595log_{10}(1 + \frac{f}{700}) \quad (1)$$

Mel scale has a constant mel-frequency interval and covers the frequency range of 0 Hz - 20050 Hz. The Mel-Frequency Cepstral Coefficients (MFCCs) are computed from the FFT power coefficients which are filtered by a triangular band pass filter bank. The filter bank consists of 12 triangular filters. The Mel-frequency coefficients are calculated as:

$$C_n = \sqrt{\frac{2}{k}} \sum_{k=1}^{K} \log(S_k) \cos\left[\frac{n(k-0.5)\pi}{k}\right], \quad n = 1,2,\dots,N \quad (2)$$

Here, S_k represents the filterbank output, and N is the total number of samples in the window. For our application, I've chosen N = 20.

Spectral Centroid: The spectrum of speech sample is characterized by its spectral centroid. This is basically a collection of the center of mass of each short time spectrum. Each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame.

Something to note here, is that a CNN makes good use of a sequence of data. Hence, the coefficients and centroids would be good features as long as they are fed to the network in order. In fact, they would be way more effective individually, rather than being appended to each other horizontally. This is shown in the results section below.

5.4. Pre-Processing

It is important that the features all range between a standard set of values, and hence they're normalized using the normalize function present in the scipy library. It is also necessary to fill the NaN values using either interpolation or zeros. I tried both and the results were better by using interpolation.

The code uses pandas library to handle datasets. The interpolation and filling of NaN values is taken care of using the functions present in this library. Once this is done, we divide the data into train and test splits.

5.5. The Network Architecture

The keras library is used to implement the network architecture which is desired. The final architecture is shown in Figure 4.

The basic intuition which went behind the proposed architecture was, as we go further in the network, the size of input array should keep condensing in such a way that the output layer should be a k-dimensional array, which is representative of the k classes. The different types of layers which help us create such an architecture are as follows:

Convolutional Layer: This runs a given number of convolution maps over the sequence, and each map produces outputs sensitive to different 'features' in the sequence. Hence, this layer is a self-trained feature extractor.

Activation Function: We don't want the network output to be a linear combination of weights, as linear models rarely capture sequences well. Hence, we use an activation function to induce a non-linearity in the model. This model uses Rectifiers (ReLU) as an activation function.

Maxpooling Layer: Sticking to the first intuition I proposed about CNN's, we know that the size of arrays we are dealing with needs to reduce with time. Maxpooling layers basically take the strongest points in a neighbourhood and pass them on to the next layer.

Dropout Layer: Too many parameters in a network would mean that they align in such a way that they give extremely high accuracy (around 99%) on the training set. This gives a bad accuracy during testing. Hence, we add dropout layers to our network, which basically drop neurons randomly during training. This helps generalize the training set, so that we don't overfit and get a good testing accuracy.

Densely Populated Layer: The penultimate layer of our network needs to be a densely populated layer, which connects all its inputs to all the outputs. The outputs generally map to each of classes. The intuition here is that the information regarding class label can lie anywhere in the sequence.

Softmax Layer: Densely populated layers spit out numbers. To convert these numbers to class probabilities, we need to map them from 0-1 and add up to 1. Softmax function does the trick here, and hence we use a Softmax layer.

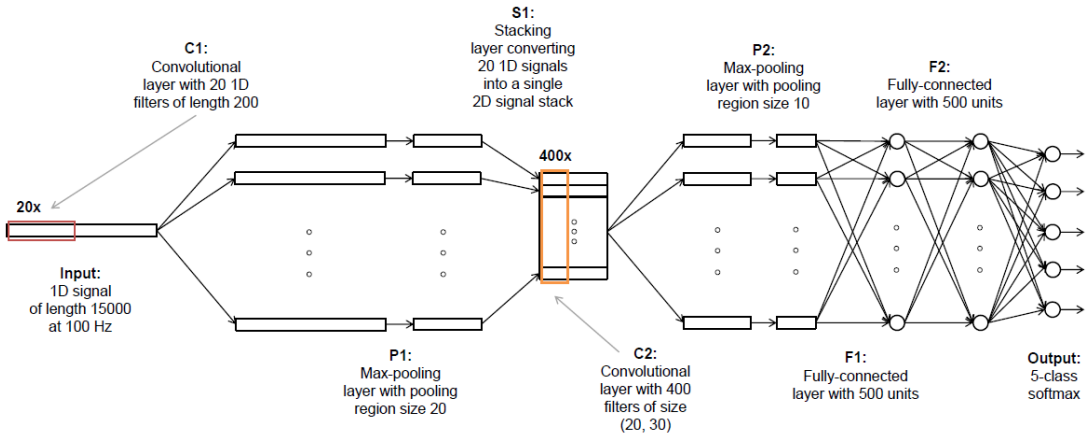

Figure 3: Basic 1-D CNN structure

To put things into perspective, the basic structure of our CNN needs to have a sequence of Convolutional and Activation layers. In between this sequence, we add Maxpooling in order to reduce array size.

An important thing I learnt is that increasing the depth as you move forward in the system architecture works wonders. After failing to achieve a good accuracy on the first run, I

changed the initial structure of the network in such a way that the depth (number of convolution maps) increases slowly as the network progresses. This is seen in Figure 4.

When I noticed that the testing accuracy was turning out to be really high and testing would converge at an average rate, I added dropout layers to parts of the network, as seen in Figure 4.

5.6. Training the network: Cross Entropy Loss

Once we define the network architecture, we need to train the network in such a way that it shifts its weights towards becoming an optimal classifier. We do this by defining a loss function to test the 'goodness' of our network, and then optimize it to shift weights to classify emotions better. This loss function finds out how far apart the current network output is from the desired output, and then shifts the network weights in the direction of negative gradient of the loss. The loss function defined over the network is categorical cross entropy. Let's dig into this function a bit deeper.

Considering that we have 5 classes. Hence, $k = 0,1,2,3,4,5$. We have a speech feature vector X as input to our model, and Y is the target label of X. Now Y is a binary vector of 5 dimensions. For example, if the label is 'angry' and this represents class k=3, we have:

$Y = [0\ 0\ 1\ 0\ 0]^T$.

Our network spits out some estimate of Y. Let's call this $\hat{Y}$. Hence, the cross entropy is defined in Equation 3.

$$L(Y, \hat{Y}) = \frac{1}{K}\sum_{k=1}^{K} Y.(log_2\hat{Y}) \quad (3)$$

Here, $K = 5$ is the total number of classes.

This provides a distance metric between the target label and expected label, and we need to basically minimize this distance. This is done by Stochastic Gradient Descent [12] to obtain the optimum set of parameters Θ of the network.

6. Results

The implementation explained in section 5 was followed step by step. I had 1200 input speech segments, which are a combination of two datasets. The task is to predict 5 different classes: Happy, Sad, Angry, Calm and Fearful. The speech segments were processed using 20ms Kaiser windows and features were extracted using 20 coefficients of Mfcc's. After this, there were 5200 features for each speech segment. These features were pre-processed, and inputs were split into train and test splits by randomly splitting them in the ratio of 8:2. The final split was 953 for training and 247 for testing.

The summary of my model is shown in Figure 4. The model was trained over a cross entropy loss using stochastic gradient descent. The train splits were passed to the model to train for 1010 epochs, and the training and testing accuracy was measured after every epoch. The

accuracy for training was 94.7% while that for testing was reported as 70.8%, and it is visible in Figure 5. This is better than the average accuracy for human classifiers, which is 66.5% on the databases selected. The loss function for training and testing is plotted in Figure 5.

Layer (type)	Output Shape	Param #
conv1d_15 (Conv1D)	(None, 5200, 32)	192
activation_17 (Activation)	(None, 5200, 32)	0
conv1d_16 (Conv1D)	(None, 5200, 32)	10272
activation_18 (Activation)	(None, 5200, 32)	0
dropout_11 (Dropout)	(None, 5200, 32)	0
max_pooling1d_7 (MaxPooling1	(None, 1300, 32)	0
conv1d_17 (Conv1D)	(None, 1300, 64)	10304
activation_19 (Activation)	(None, 1300, 64)	0
dropout_12 (Dropout)	(None, 1300, 64)	0
conv1d_18 (Conv1D)	(None, 1300, 64)	41024
activation_20 (Activation)	(None, 1300, 64)	0
conv1d_19 (Conv1D)	(None, 1300, 128)	41088
activation_21 (Activation)	(None, 1300, 128)	0
dropout_13 (Dropout)	(None, 1300, 128)	0
max_pooling1d_8 (MaxPooling1	(None, 325, 128)	0
conv1d_20 (Conv1D)	(None, 325, 128)	163968
activation_22 (Activation)	(None, 325, 128)	0
dropout_14 (Dropout)	(None, 325, 128)	0
max_pooling1d_9 (MaxPooling1	(None, 40, 128)	0
conv1d_21 (Conv1D)	(None, 40, 128)	245888
activation_23 (Activation)	(None, 40, 128)	0
dropout_15 (Dropout)	(None, 40, 128)	0
flatten_3 (Flatten)	(None, 5120)	0
dense_3 (Dense)	(None, 5)	25605
activation_24 (Activation)	(None, 5)	0

Total params: 538,341
Trainable params: 538,341
Non-trainable params: 0

Figure 4: Final CNN Architecture used

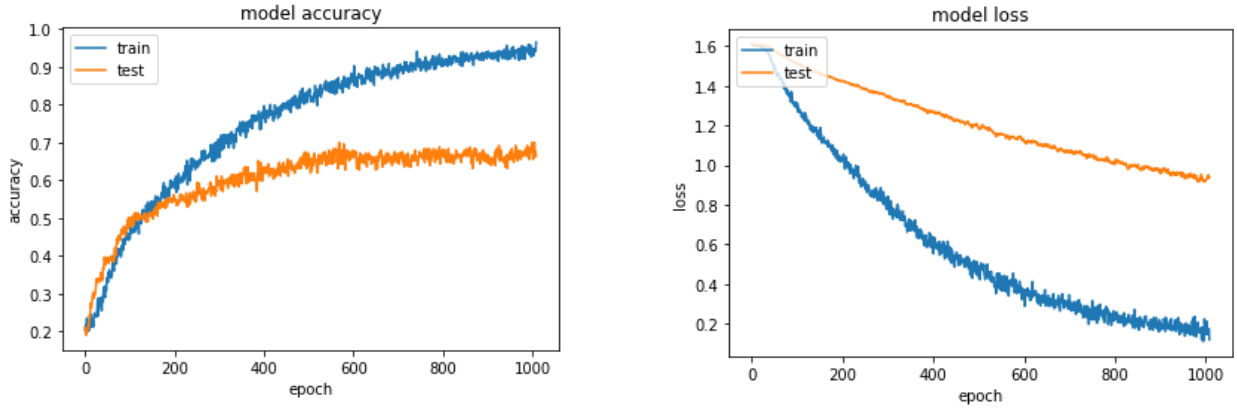

Figure 5: Accuracy of the Model and Loss function

The results are promising as the model performs decently well even though I have merged two datasets. The misclassifications are explainable in most cases. For example, some samples which were labelled as calm were actually sad. The confusion Matrix for the testing set of samples is given in figure 6.

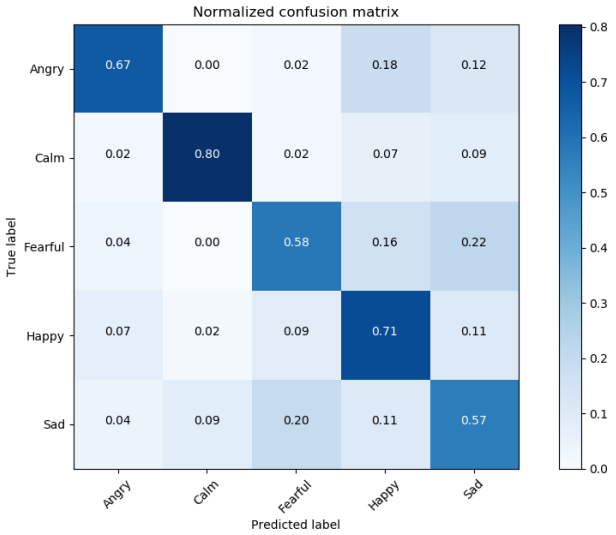

Figure 6: Confusion Matrix of Total set of Samples

While constructing this robust model, I had made some errors in judgement. The importance of different aspects in the model architecture and feature extraction steps are highlighted by these errors in the sections below.

6.1. Significance of Features

Initially, the model was built using a set of spectral centroids and mean of mfcc's appended to each other. The accuracy for such a system was at max 43%. This is clearly visible in Figure 7. The issue with appending an aggregation of features over time in a CNN is that the appending process loses temporal structure. What we can do to mitigate this is, maybe increase the depth of the input feature vector, so the features are not appended but 'stacked' on top of one another. I dint choose this option, but I increased my feature size by using the actual value of all Mfcc coefficients instead of their mean over time. This gave a higher accuracy as displayed in Figure 5.

6.2. Importance of choosing the right Model Architecture

The model I started with had Maxpooling and convolutional layers, but each of these layers had a set number of feature maps. This gave a poor performance (Around 60% accuracy). The architecture was then changed for the better so that the number of feature maps increase as we go deeper in the network. The importance of depth is clearly signified by this result.

As we increase depth in networks, we get closer and closer to modelling the human brain. More depth in convolutional layer means more feature maps. Each map is robust to a particular feature of the speech segment. This would give a better accuracy than using a shallow network but blindly increasing the number of its parameters.

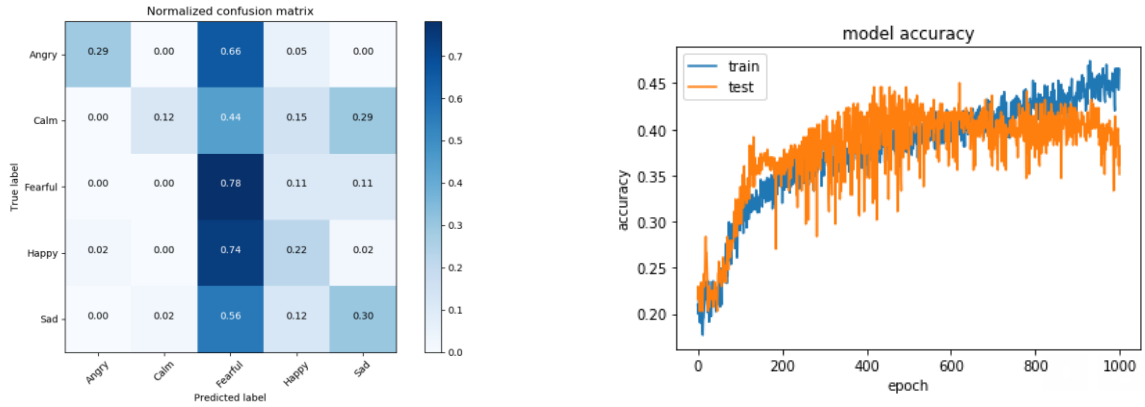

Figure 7: High error as an effect of poor choice of feature extraction

6.3. Importance of Regularization

One of my test runs gave an accuracy of 41.2% on testing, but the training accuracy was way higher (99.9%). If this case arises, it means we have too many parameters for the number of samples, and hence our model is overfitting by memorizing arbitrary representations of the training set. The 'learning' part of it reduces. The confusion matrix for this model is shown in Figure 8.

While overfitting, most of our parameters in the model are not tweaked even slightly from the initial conditions. To combat this, we regularize the model in such a way that it will randomly drop nodes while training, so that the model generalizes well using even fewer set of parameters, and there's not too much reliance on a single parameter.

This is done by adding dropout layers to the model. Hence, the model accuracy went up by 30% after dropout layers were added.

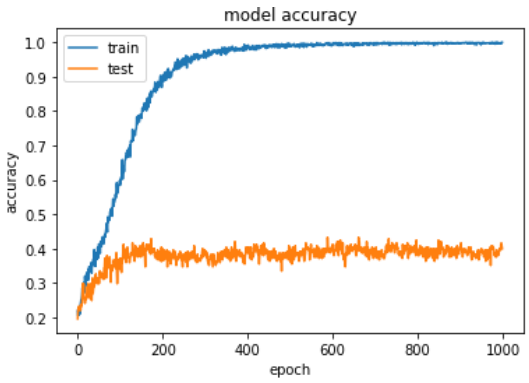

Figure 8: Effect of overfitting

7. Conclusion and Future Work

A robust CNN model was built to classify emotions in speech with a decent accuracy of slightly over 70%. Moreover, the CNN architecture and various aspects of the Deep Learning framework such as significance of features and regularization is also reported by explaining my errors in judgement along the way.

The current model performs well compared to most of the methods available online, which give accuracy of 40-65% [2] on various datasets. However, there are a few papers which report accuracy of 84-95% [13] on the SAVEE dataset which has been used here.

Hence, there is a lot of room for improvement. The feature extraction could be done by 'stacking' different features over one another instead of appending them. This might give us more information and keep the temporal structure intact. The model could also be improved further by optimizing on its number of layers.

8. References

[1] Marie Tahon and Laurence Devillers, "Towards a small set of robust acoustic features for emotion recognition: challenges," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 24, no. 1, pp. 16–28, 2016.

[2] S. Mirsamadi, E. Barsoum and C. Zhang, "Automatic speech emotion recognition using recurrent neural networks with local attention," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, 2017, pp. 2227-2231.

[3] Bj¨orn Schuller, Dejan Arsic, Frank Wallhoff, Gerhard Rigoll, et al., "Emotion recognition in the noise applying large acoustic feature sets," Speech Prosody, Dresden, pp. 276–289, 2006.

[4] Shashidhar G Koolagudi and K Sreenivasa Rao, "Emotion recognition from speech: a review," International journal of speech technology, vol.15, no.2, pp.99–117, 2012

[5] Nwe, Tin Lay, Say Wei Foo, and Liyanage C. De Silva. "Speech emotion recognition using hidden Markov models." Speech communication 41.4 (2003): 603-623.

[6] Andr´e Stuhlsatz, Christine Meyer, Florian Eyben, Thomas Zielke, G¨unter Meier, and Bjorn Schuller, "Deep neural networks for acoustic emotion recognition: raising the benchmarks," in 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2011, pp. 5688–5691.

[7] Kun Han, Dong Yu, and Ivan Tashev, "Speech emotion recognition using deep neural network and extreme learning machine.," in Interspeech, 2014, pp. 223–227.

[8] Livingstone, S. R., & Russo, F. A. (in press). The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. *PloS one*.

[9] http://kahlan.eps.surrey.ac.uk/savee/

[10] Kaiser, James F.; Schafer, Ronald W. (1980). "On the use of the I_0-sinh window for spectrum analysis". *IEEE Transactions on Acoustics, Speech, and Signal Processing*. **28**: 105–107. doi:10.1109/TASSP.1980.1163349

[11] *Min Xu; et al. (2004). "HMM-based audio keyword generation". In Kiyoharu Aizawa; Yuichi Nakamura; Shin'ichi Satoh. Advances in Multimedia Information Processing – PCM 2004: 5th Pacific Rim Conference on Multimedia. Springer. ISBN 3-540-23985-5. Archived from the original on 2007-05-10.*

[12] *Ferguson, Thomas S. (1982). "An inconsistent maximum likelihood estimate". Journal of the American Statistical Association. **77** (380): 831–834.*

[13] Lijiang Chen, Xia Mao, Yuli Xue, Lee Lung Cheng, “Speech emotion recognition: Features and classification models”, *Digital Signal Processing, Volume 22, Issue 6, 2012,* Pages 1154-1160, ISSN 1051-2004, https://doi.org/10.1016/j.dsp.2012.05.007.