**Neural Network for Sound Classification -Finger snapping**

**Introduction:**

Have you ever you ever seen magicians or wizards controlling their environment at their own will? Have you ever been astonished how they can light candles when they clap or snap? Well that is today possible through the use of Neural Networks. In this project we will design, build, and test a program that records sounds from a microphone and detects its type from predefined classes that we trained the model on.

**Preparation:**

For this project, we will need to have Tensorflow (gpu version) and keras installed. This tutorial by Ankit [Bhatia](https://medium.com/@ab9.bhatia/set-up-gpu-accelerated-tensorflow-keras-on-windows-10-with-anaconda-e71bfa9506d1) is helpful. I have also used the following libraries for various purposes:

* Scipy, soundfile, sounddevice, librosa (sound manipulation)
* Pandas (data management)
* Matplotlib (plotting)
* python\_speech\_features (sound cleaning)
* pickle (data storage)

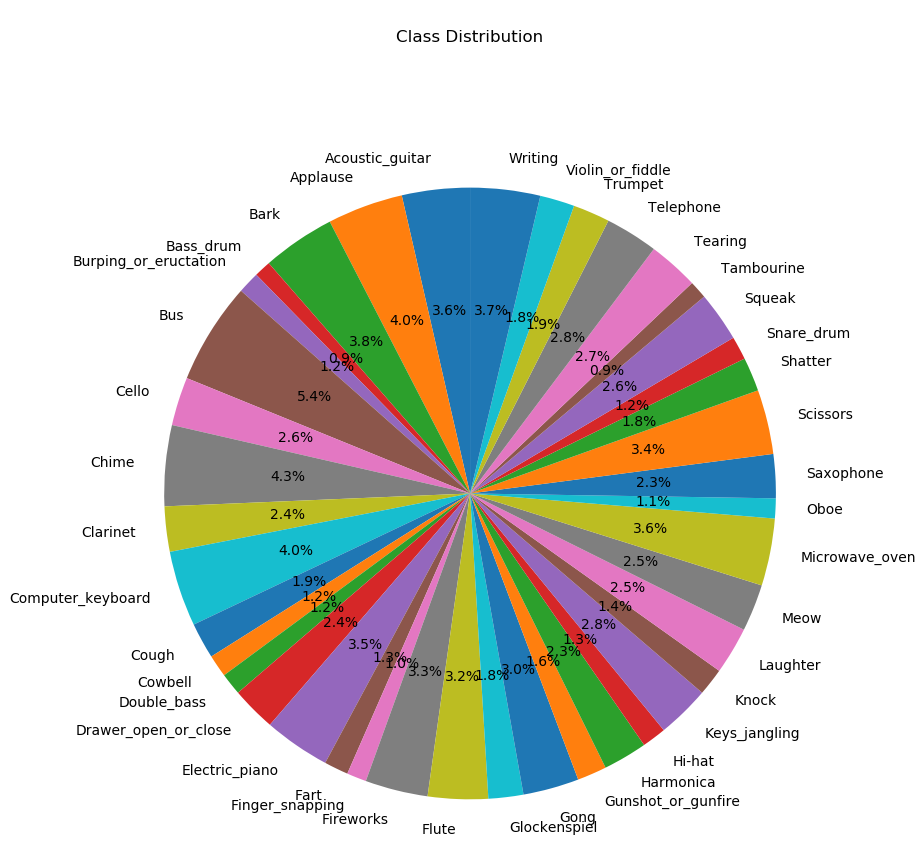
**The Data:**

The bulk of our data comes from Kaggle. It is from a dataset called Freesound General-Purpose Audio Tagging Challenge which was launched in 2018 as a competition to make a sound classifier. If you want to access and download the data set use the following link: <https://www.kaggle.com/c/freesound-audio-tagging/data>

After the competition ended, the data was updated later to include 9.5k samples unequally distributed among 41 categories. The minimum number of audio samples per category in the train set is 94, and the maximum 300. The duration of the audio samples ranges from 300ms to 30s due to the diversity of the sound categories and the preferences of Freedsound users when recording sounds. The total duration of the train set is roughly 18h.

Out of the ~9.5k samples from the train set, ~3.7k have manually verified ground truth annotations and ~5.8k have non-verified annotations. The non-verified annotations of the train set have a quality estimate of at least\*\* 65-70% in each category.

Due to the lack of sample of finger snaps in the dataset compared to other classes, I have recorded an additional 110 samples. These sample are also useful from the training because it uses the same microphone for the evaluation. (sound quality for evaluation is different from training which might affect the results)

The initial distribution of our data is as follow

**The average length of each class is as follow:**

Label mean length

Applause 11.632867

Acoustic\_guitar 10.430200

Bark 11.196485

Bass\_drum 2.569667

Burping\_or\_eructation 3.353810

Bus 15.634862

Cello 7.453267

Chime 12.405391

Clarinet 6.947467

Computer\_keyboard 11.602857

Cough 5.524774

Cowbell 3.437906

Double\_bass 3.373467

Drawer\_open\_or\_close 6.853544

Electric\_piano 10.188400

Fart 3.720800

Finger\_snapping 3.018632

Fireworks 9.638067

Flute 9.237400

Glockenspiel 5.357660

Gong 8.595822

Gunshot\_or\_gunfire 4.520952

Harmonica 6.777697

Hi-hat 3.718067

Keys\_jangling 8.121871

Knock 4.210538

Laughter 7.266800

Meow 7.236000

Microwave\_oven 10.330685

Oboe 3.067023

Saxophone 6.745200

Scissors 9.913474

Shatter 5.215400

Snare\_drum 3.587000

Squeak 7.645800

Tambourine 2.754480

Tearing 7.746933

Telephone 8.117500

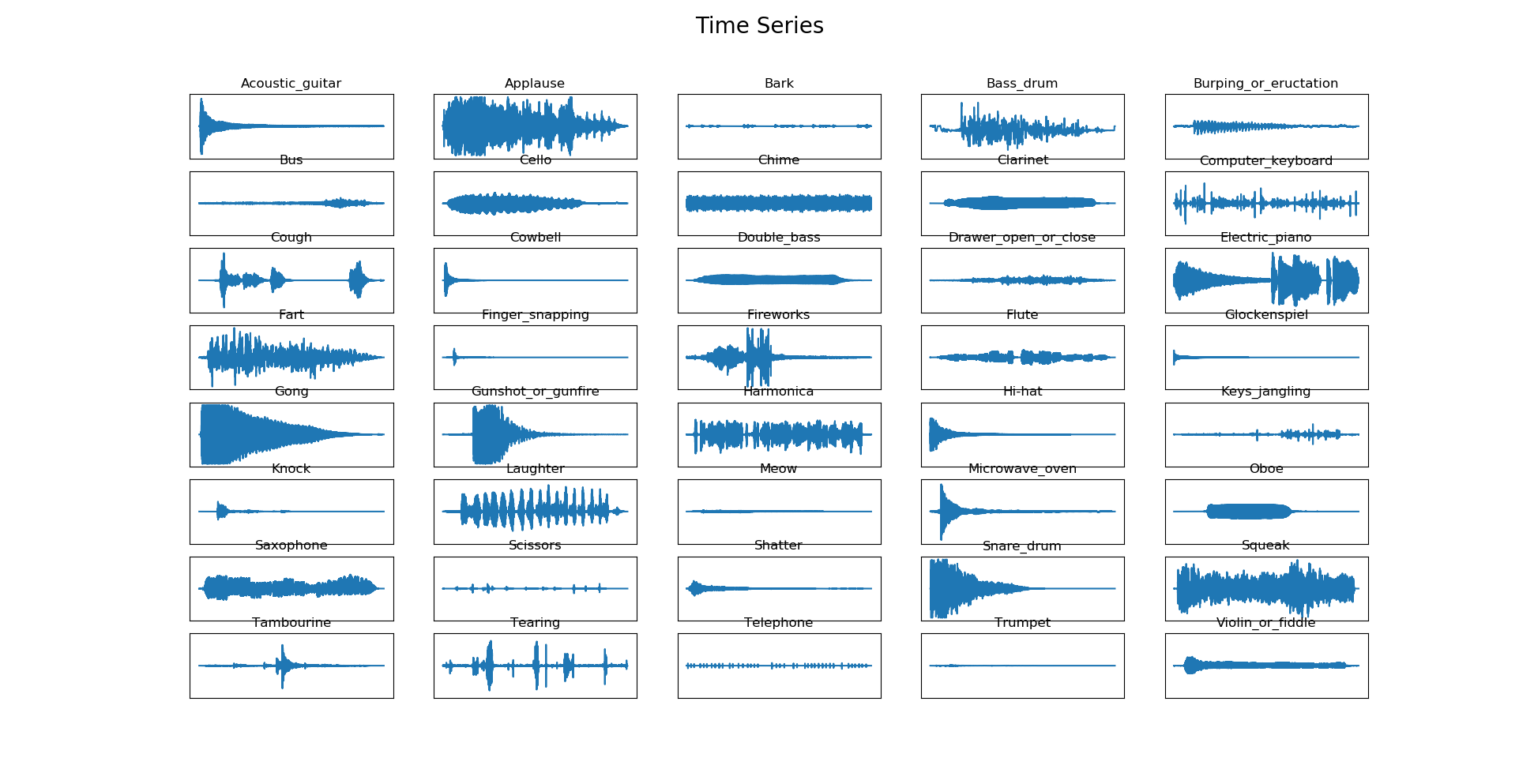
Trumpet 5.667200

Violin\_or\_fiddle 5.325800

Writing 10.738148

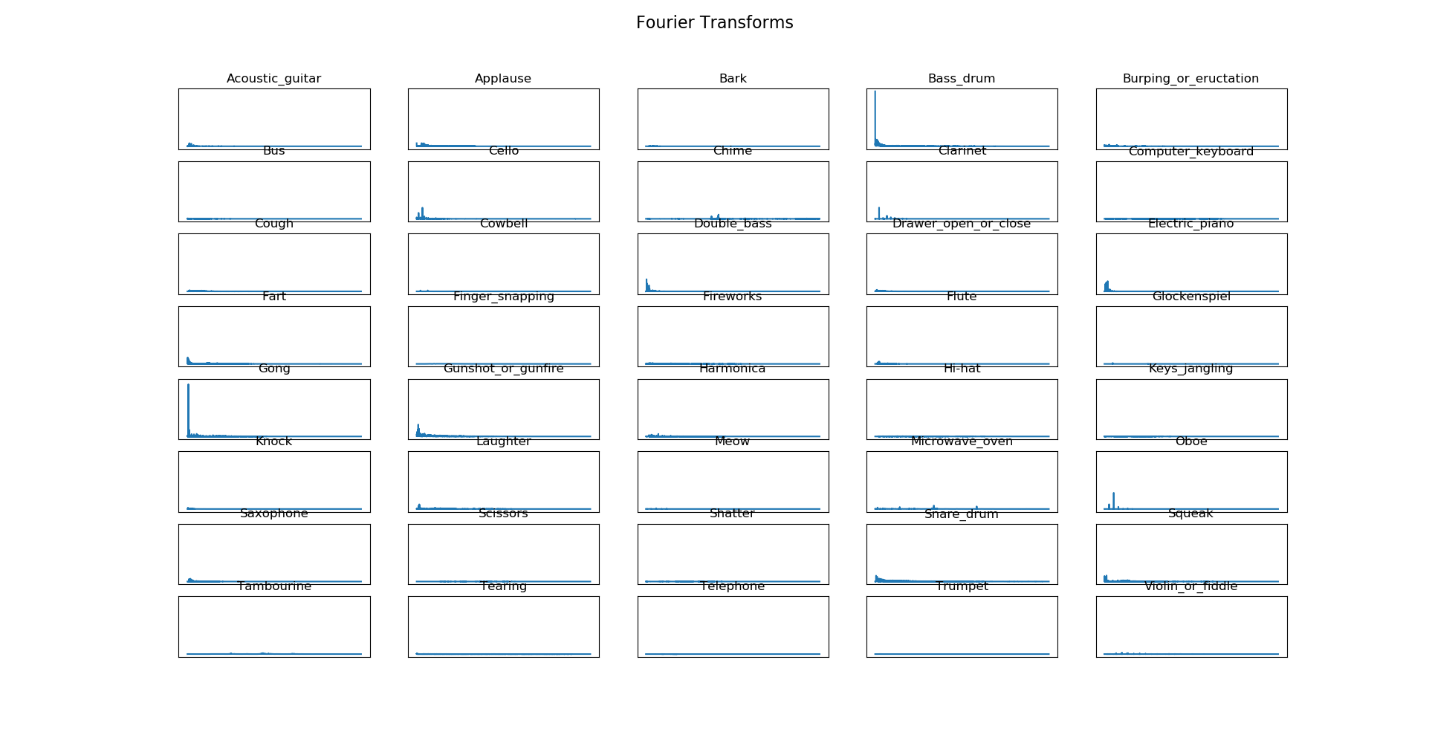
**Data processing:**

In order to train our Neural Network, we have to prepare and clean the data that will be fed. It is really hard to train a model using the raw sound files as the computer finds it too complicated to a classification on.

At first our different sounds would look like this:

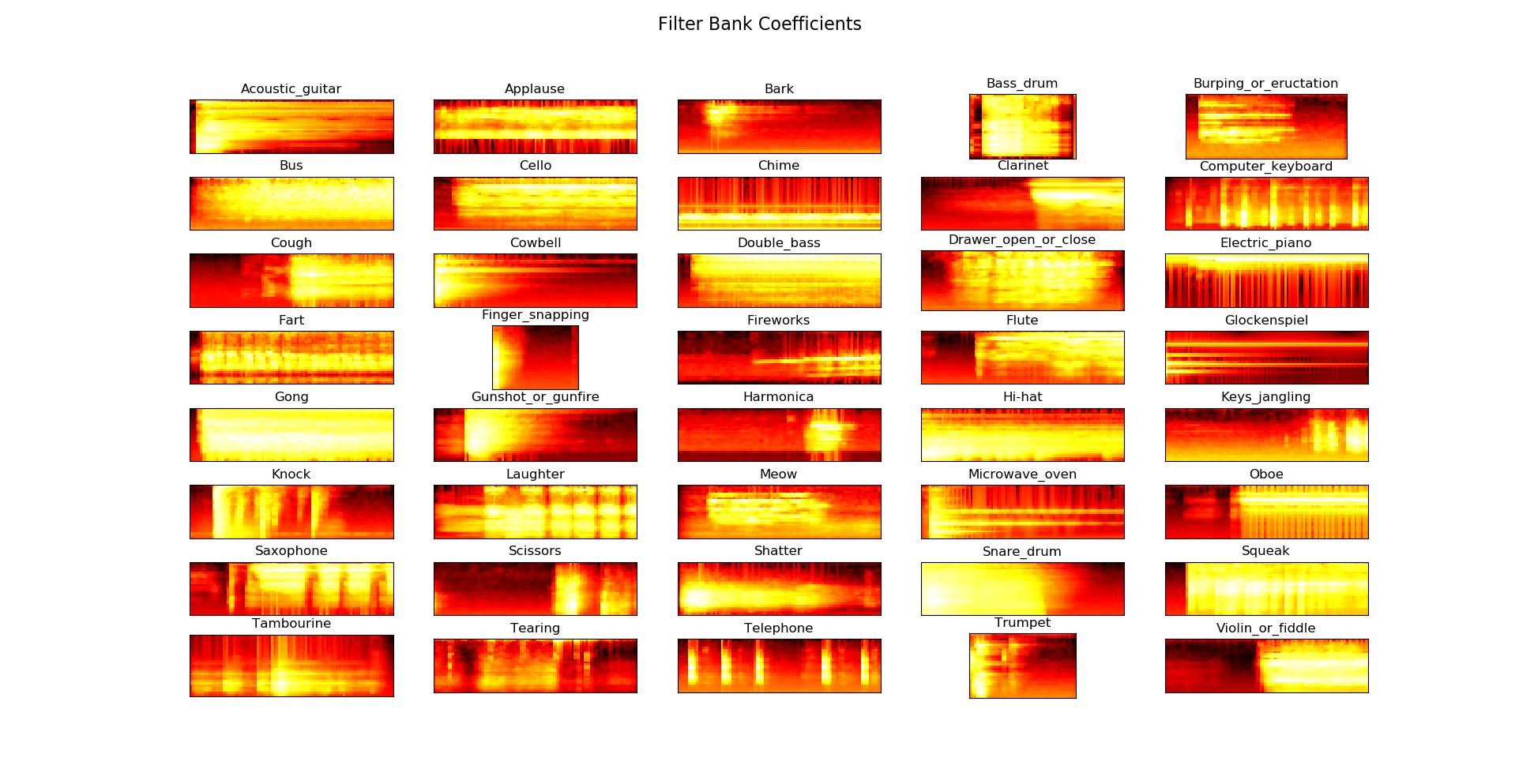
As you can see it is also really hard to discern between different sound classes even for humans!

Our first intuition would be to look at the Fourier transform which models the amplitude in function of the frequency.

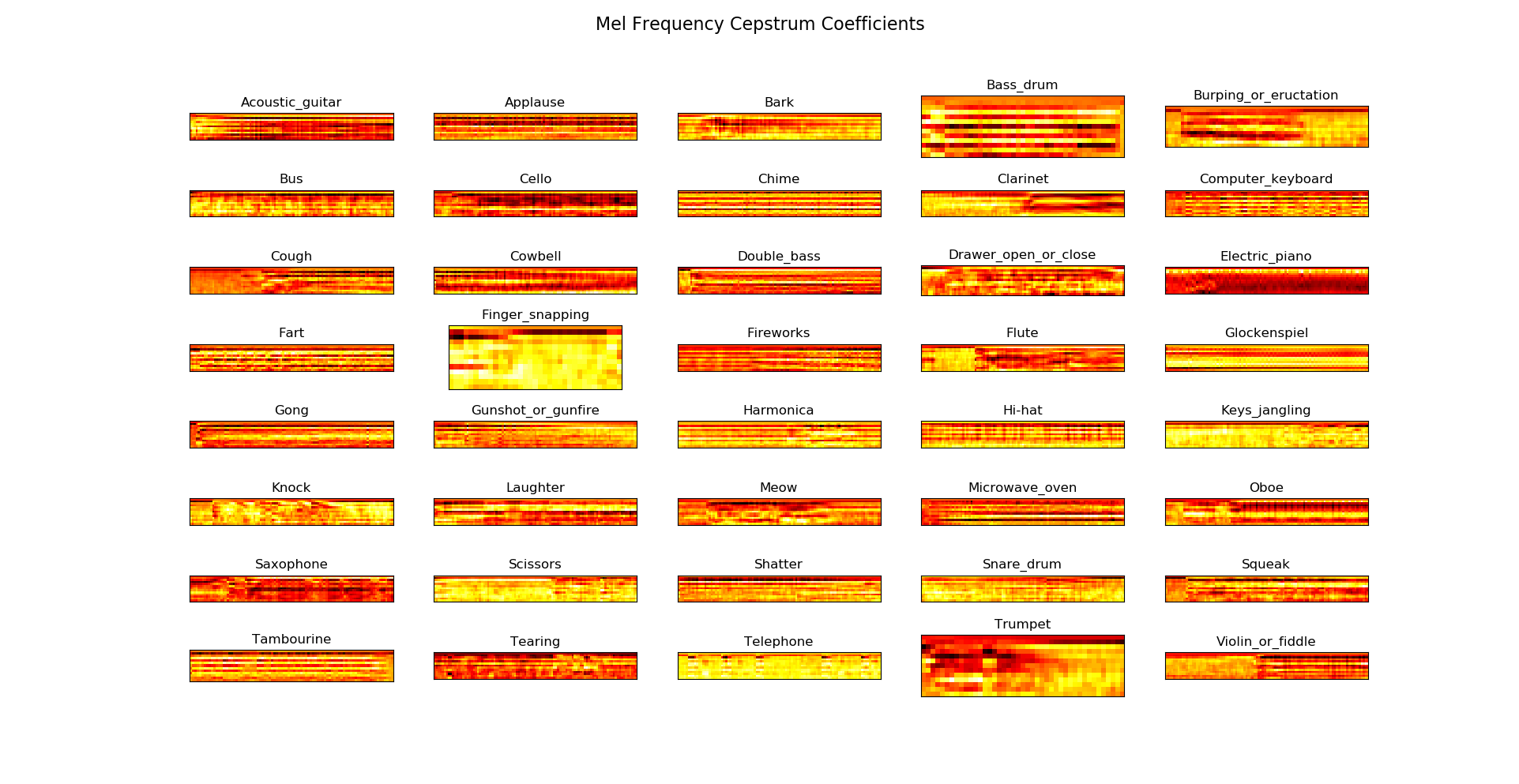


Because most of the frequencies are low and the amplitude are not high as well, most of the sounds are similar.

The next step is to convert thee sounds into spectrograms (Time vs frequency) using Filter bank coefficients (info [here](https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html)). Filter banks aim to mimic the human perception of sounds which is nonlinear.



The results are a lot better than what we had in our previous methods. Yet there is a lot of data in these spectrograms. A sensible thing to do is to compress these into more informative spectrogram using MFCCs ( Mel Frequency Cepstrum Coefficients) – more info [here](http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/).



The result highlights the differences between the classes (more focus on where the sound happens in terms of Time vs frequencies)

Python speech features library has already an implemented function for MFCC that takes as an input the signal and returns the MFCC array that represent the previous graph. We use this function on the set of our data to create our feature matrix (input).

Our next step is to make our input normalized. The data we have has different lengths and different values. Therefore, we decided to sample randomly from our sound files 10mn od data, run it through our mfcc function. We also normalize the values depending on the highest and lowest values of the inputs. Finally, we reshape our data to match the Neural Network architecture. For instance, if we are using a convolutional neural network such as AlexNet. We reshape our data as following:

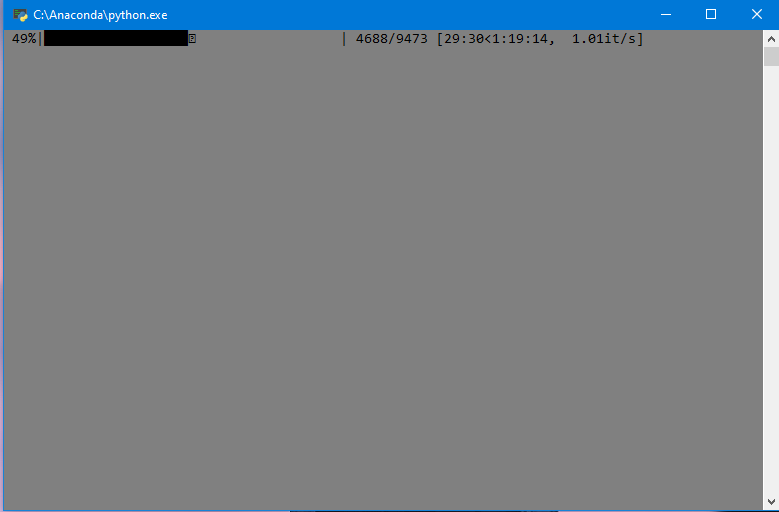
x.reshape(x.shape[0], x.shape[1], x.shape[2], 1)

where x.shape[0] is the number of samples

x.shape[1] the length of the input

x.shape[2] different value for different frequency banks

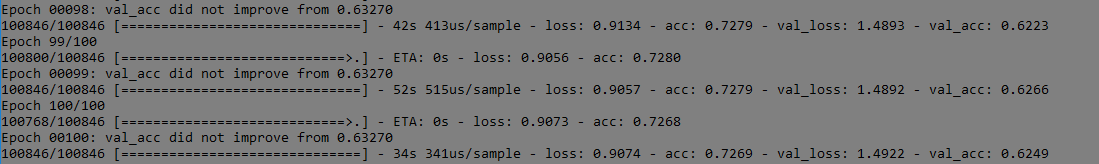
1 🡪 depth of input for the pooling and filtering in our conv Net



Building random features (sets of inputs for the net)

**Model Building and training:**

After preparing our data, It is time now to build our model. For this project I have tried 4 different Network architectures.

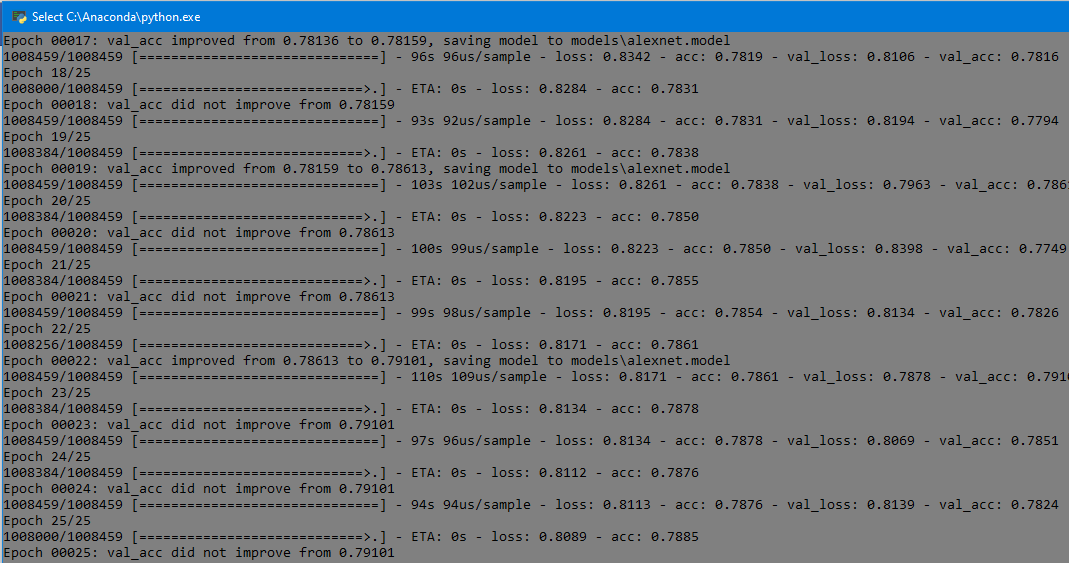
The first was a custom Convolutional Neural Network. The reason behind this choice is to try to detect features specific to the sound classes from the MFCC data. Because, every sound is a little bit different, a convolutional architecture in theory should work well. 

-result of training of custom Conv net-

The second is a Recurrent neural network that tries to take advantage of the time dependency in sounds. However, comparing the training results of the two choices, the Convolutions Net is more accurate.

Naturally, we thought of trying a recurrent conv net to take advantage of both architectures. However, the results were not any better than the Simple recurrent net.

Finally, we tested the training with an AlexNet model which is based on a convolutional net. This model gave us the best result with ~80% accuracy after 80 epochs.



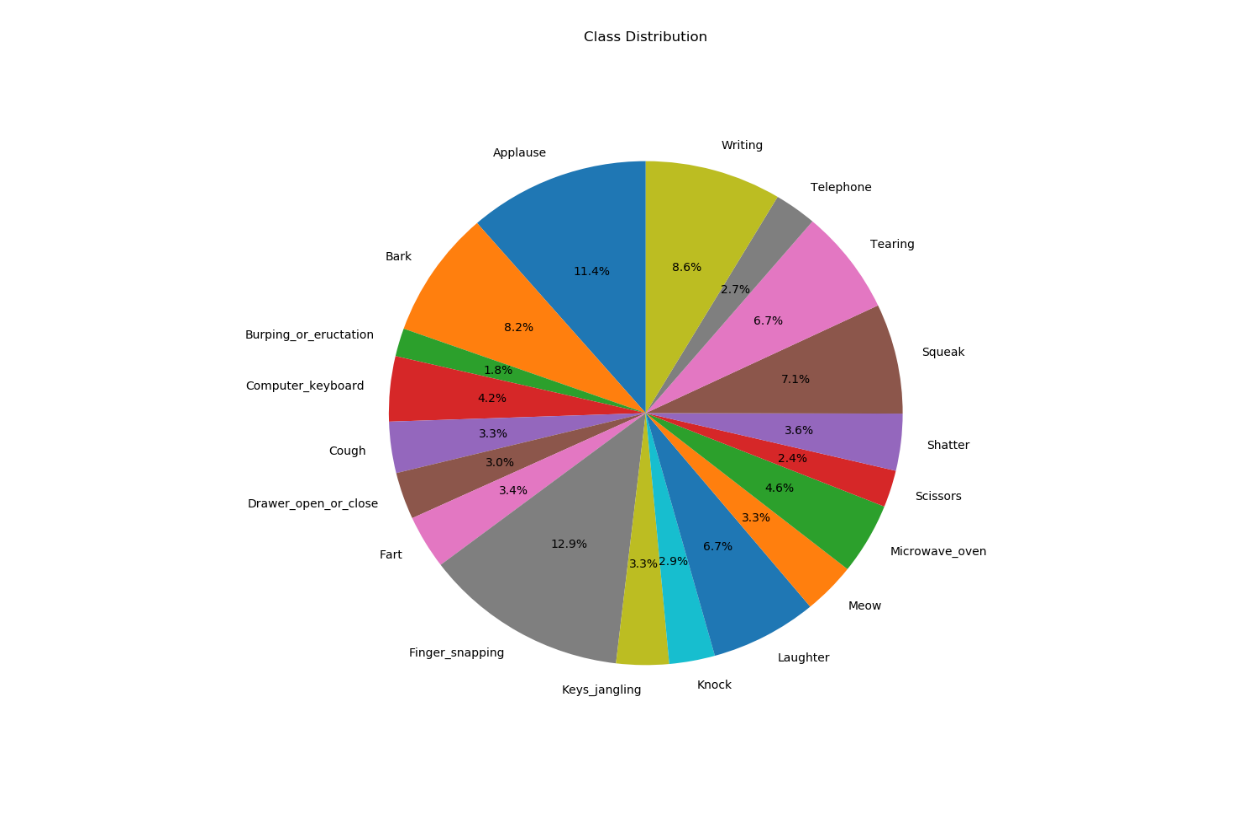
-training of AlexNet Model -

**Evaluation:**

For evaluations we record from our laptop microphone sound samples every one second in an open loop. For every sample, we use an envelope to clean the sound (keep data beyond a certain threshold). After that, we cut our data into 10ms chunks that we feed into our Model to evaluate. We also use the accuracy of the evaluation to make an estimate guess of the class of the sound throughout the 1 second.

After training, our model seems to be doing a good job detecting certain type of sounds but not specifically finger snaps. This might be due to the lack of snapping sounds or the different sound quality. Our model can detect reliably coughs, laughs, and whistles (classified as wind instruments). However, it confuses finger snaps as barks or shatters which might be understandable as those sounds are short and loud.

We tried multiple attempts to rectify this problem:

* Added more snap sounds to our training data.
* we increased the sample time from 10ms to 25ms to capture more data that will help the model detect the sounds correctly. Even though this increased our training and validation accuracy to 90%, the evaluation accuracy did not seem to improve.
* We increased the sampling rate from 16Khz to 44100Khz 🡪 worse results
* Made the network train on binary choices --- snap or no snap 🡪 it became really good at not detecting snaps.
* Trimmed down the class number from 41 to 20 that are most common within houses or offices. -🡪 even better improvement detecting coughs, laughs, microwave sounds, burping, and barking.