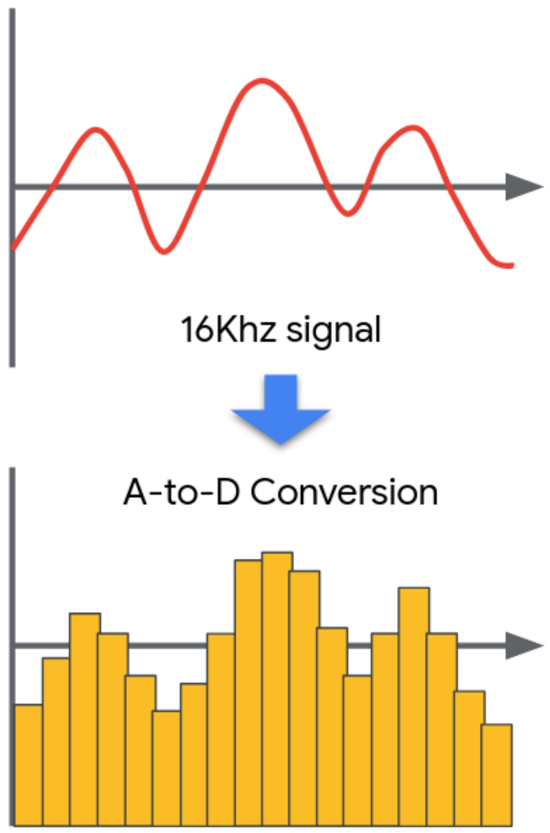
Spectrograms and MFCCs

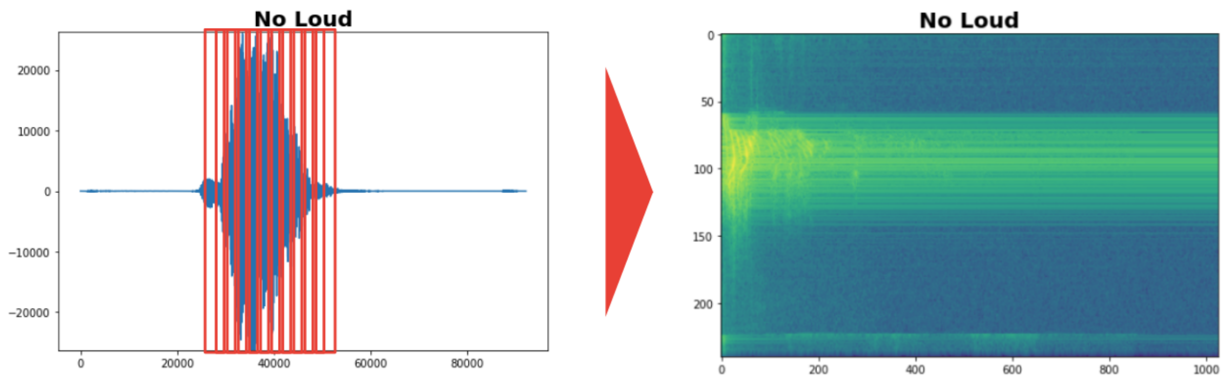
For Keyword Spotting, raw analog audio signal is first transformed into a digital signal.



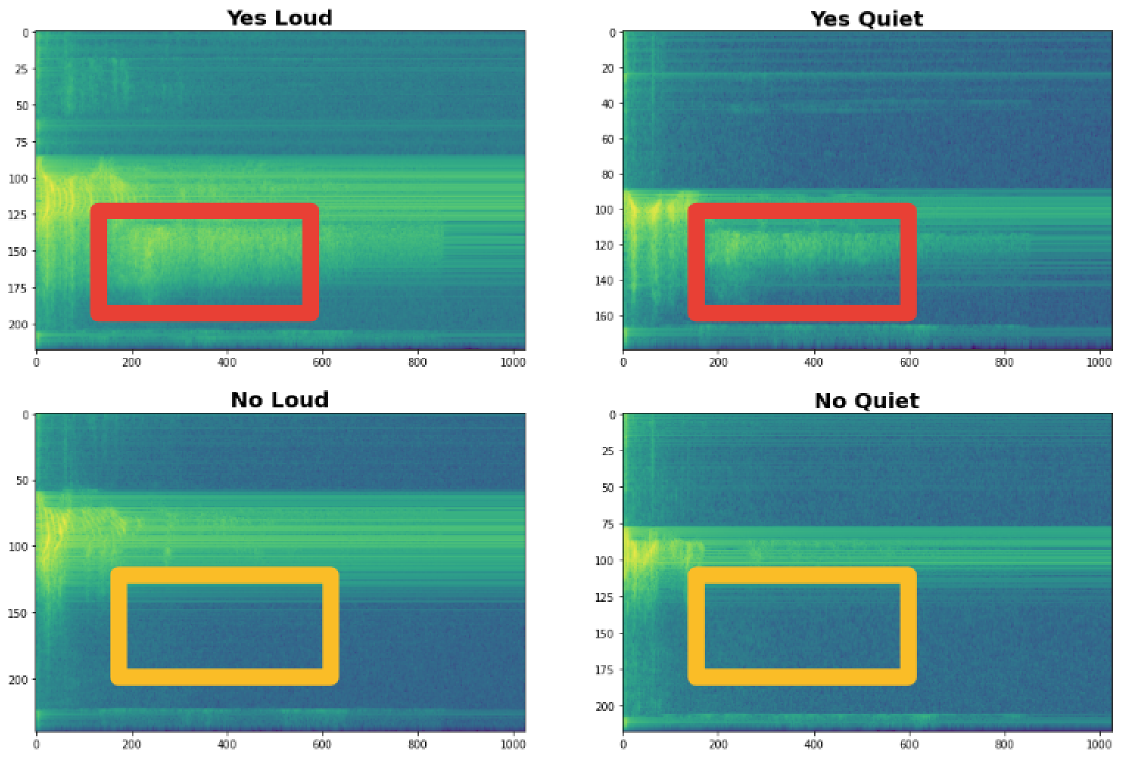
Then the digital signal in the **time domain** (time on the x-axis and amplitude on the y-axis) is transformed into the **frequency domain** (frequency on the x-axis and amplitude on the y-axis) through the use of a **fourier transform**. If you'd like to dive deeper into understanding how the fourier transform works we suggest you watch this fantastic video on the 3Brown1Blue youtube channel.

Of course for the purposes of TinyML we need to worry about how we can compute this transformation efficiently -- both in the memory needed to store the program, and in the memory and latency needed for the computation. The solution is the use of an approximate **fast fourier transform (FFT)** of which there are multiple possible implementations. TensorflowLite Micro uses the KISS_FFT library, which includes more documentation about the design choices made by the library to be compact in memory on their GitHub page.

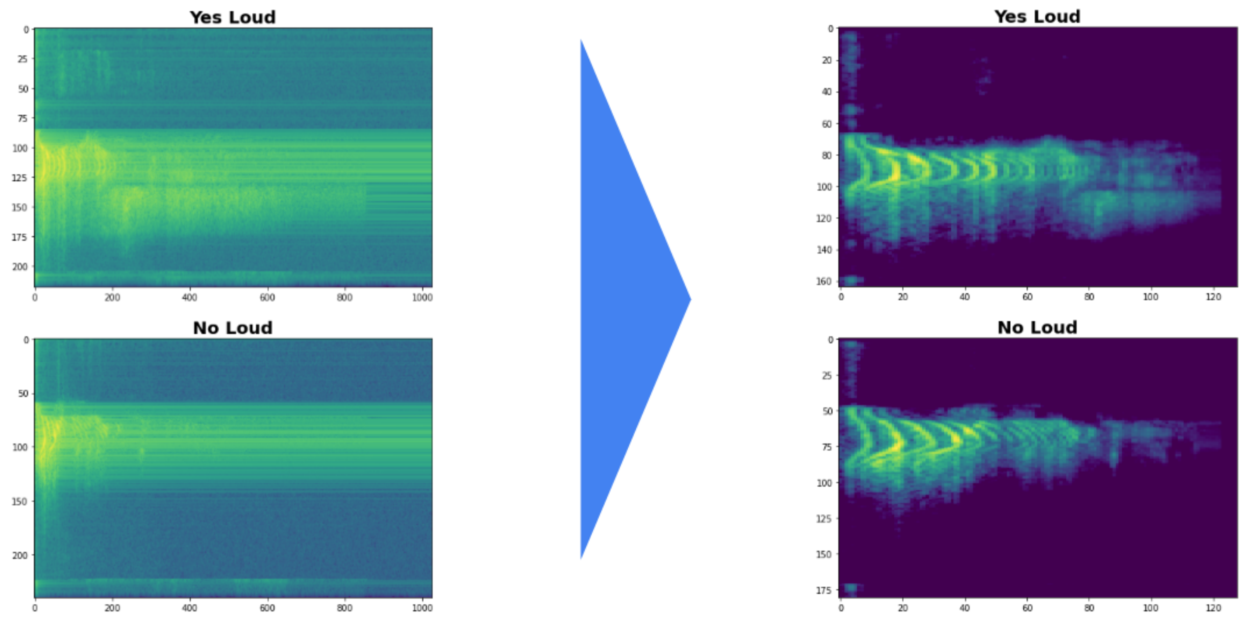
By applying the FFT through a sliding window procedure we can analyze the intensity (in color) of each of the frequencies over time. This is a **spectrogram.**



A spectrogram is a better input for a deep learning system than the standard input audio signal as we have already extracted the frequencies for the neural network and so it doesn't need to learn how to do that on its own. Even visually, it is easier to see the difference between the "yes" and "no" spectrograms.



Also, the human ear can tell low frequency signals apart easier than high frequency signals according to the Mel Scale. We can therefore create a **Mel Filter Bank** to take advantage of this phenomenon and group together larger clusters of higher frequency signals to provide more useful data to our machine learning algorithm. The output of this process is a **Mel Frequency Cepstral Coefficient (MFCC)**. For more information on how this works here are three great posts: post1, post2, post3.



Of course there are many different ways we could solve this learning problem and it is likely that very large neural networks would perform quite well on this task without this preprocessing. However, for the case of TinyML this preprocessing is vital and drastically improves the final model performance without taking up a lot of memory or latency!