# Audio AI: isolating vocals from stereo music using Convolutional Neural Networks

hacking music towards the democratization of derivative content

Ale Koretzky
Feb 4, 2019 · 15 min read ★

What if we could go back to 1965, knock on Abbey Road Studios' front door holding an 'All Access' badge, and have the privilege of listening to those signature Lennon-McCartney harmonies A-Capella? Our input here is a medium quality mp3 of *We Can Work it Out* by The Beatles. The top track is the input mix and the bottom track, the isolated vocals coming out of our model.

*linear or convolutive-* process, have been mixed with other signals. The field has many practical applications including but not limited to speech denoising and enhancement, music remixing, spatial audio, remastering, etc. In the context of music production, it is sometimes referred to as *unmixing* or *demixing*. There's a good amount of resources on the subject, going from ICA-based *-blind-* Source Separation, to semi-supervised Non-negative Matrix Factorization techniques, to more recent neural network-based approaches. For a nice walkthrough on the first two, you can check out these tutorial mini-series from CCRMA, which I found very useful back in the day.

**But before jumping into design stuff.. a liiittle bit of Applied Machine Learning philosophy…**

As someone who's been working in signal & image processing for a while and prior to the '*deep-learning-solves-it-all'* boom, I will introduce the solution as a **Feature Engineering** journey and show you **why, for this particular problem**, an artificial neural network ends up being the best approach. Why? Very often I find people writing things like:

*"with deep learning you don't have to worry about feature engineering anymore; it does it for you"*

or worst…

*"the difference between machine learning and deep learning* < let me stop you right there… Deep Learning is still Machine Learning! > *is that in ML you do the feature extraction and in deep learning it happens automatically inside the network".*

These generalizations, probably coming from the fact that DNNs can be pretty effective at learning good latent spaces, are just wrong. It frustrates me to see recent grads and practitioners being sold on the above misconceptions and going for the '*deep-learning-solves-it-all'* approach as something you just throw a bunch of raw data at (yes, even after doing some pre-processing you can still be a sinner :)), and expect things to just work as desired. In the real world, where your data is not as simple, clean and pretty as the MNIST dataset and where you have to care about things like real-time, memory and so on, these misconceptions can leave you stuck in experimentation mode for a very long time.

> Feature Engineering not only remains a very important discipline when designing artificial neural networks; in most cases and just like with any other ML technique, it separates production-ready solutions from failing or underperforming experiments. A deep understanding of your data and its domain can still get you very far...

. . .

## From A to Z

Ok, now that I'm done preaching, let's get into what you came for! Just like with every other data problem that I've worked on in my career, I'll begin by asking the question ***"how does the data look like"?***. Let's take a look at the following fragment of singing voice from an original studio recording.

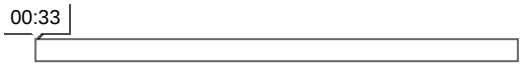


'One Last Time' studio vocals, Ariana Grande

time. We could extract things such as envelopes, RMS values, zero-crossing rate, etc, but these *features* are *too primitive* and not discriminative enough for helping us solve the problem. If we want to extract vocal content from a mix we should somehow **expose the structure of human speech,** to begin with**.** Luckily, the Short-Time Fourier Transform (STFT) comes to the rescue.

**Studio vocals, Ariana Grande - STFT**

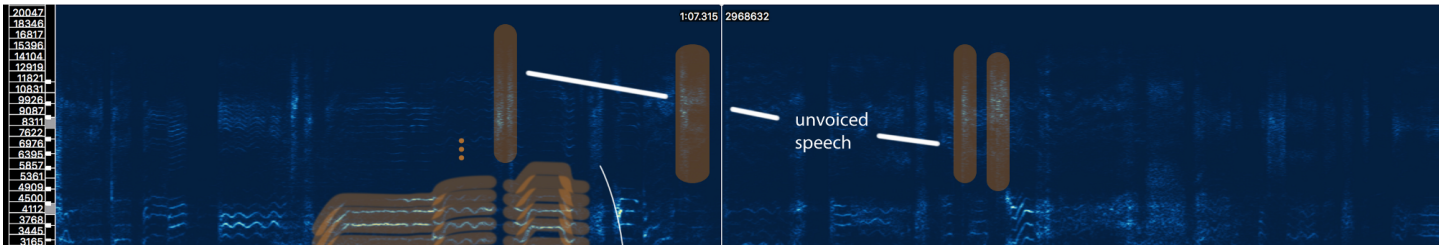
from **Alejandro Koretzky**

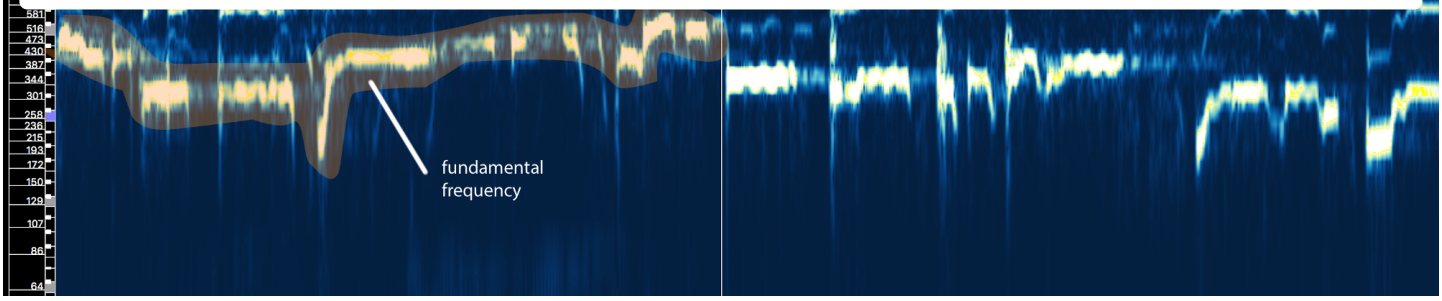



STFT magnitude spectrum - window size = 2048, overlap = 75%, log-frequency [Sonic Visualizer]

Although I love Speech Processing and I would definitely enjoy going through *source-filter modeling, cepstrum, quefrencies, LPC, MFCC* and so on, I'll skip all that stuff and focus on the core elements related to our problem, so that the article is digestible by as many people as possible and not exclusively by the Audio Signal Processing / Speech community.

So, what does the structure of human speech tell us?






Well, there are 3 main elements that we can identify here:

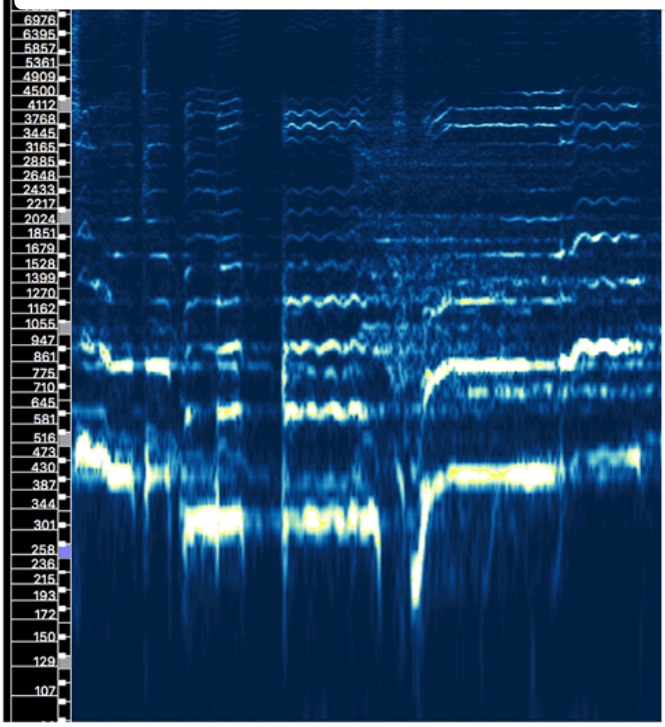
- A **fundamental frequency** (f0), determined by the frequency of vibration of our vocal cords. In this case, Ariana is singing in the 300–500 Hz range.
- A number of **harmonics** above f0, following a similar *shape* or pattern. These harmonics happen at integer multiples of f0.
- **unvoiced** speech, which includes consonants like *'t', 'p', 'k'*, 's' (which are not produced by the vibration of our vocal cords), breaths, etc. They manifest as short bursts in the high-frequency region.

## A first shot using a rule-based approach

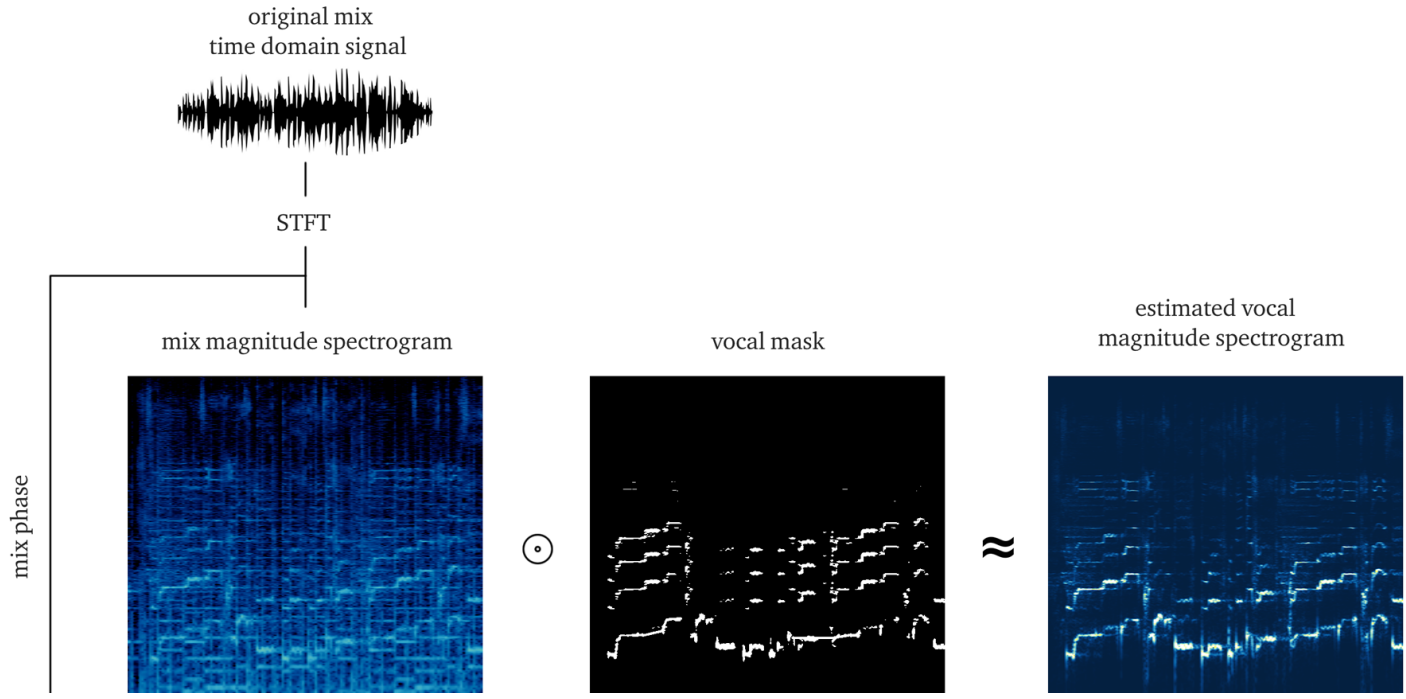
Let's forget for a second about this thing called Machine Learning. Based on our knowledge about the data, can we come up with a method to extract our vocals? Let me give it a try…

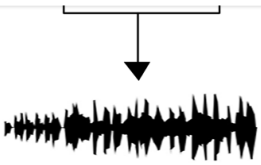
***Naive* vocal isolation V1.0:**

1. Identify vocal sections. There's a lot going on within a mix. We want to focus on the sections that actually contain vocal content and ignore the rest.
2. Distinguish between voiced and unvoiced sections. As we saw, voiced speech looks very different from unvoiced speech, therefore they probably need different treatment.
3. Estimate the frequency of the fundamental over time.
4. Based on the output of 3, apply some sort of mask to capture the harmonic content.
5. Do something else about the unvoiced sections…




If we do a decent job, the output of this process should be a *soft* or *binary mask* that, when applied (element-wise multiplication) to the magnitude STFT of the mix, gives us an approximate reconstruction of the magnitude STFT of the vocals. From there, we then combine this vocal STFT estimate with the phase information of the original mix, compute an inverse STFT, and obtain the time-domain signal of the reconstructed vocals.





estimated vocal
time domain signal

Doing this from scratch is already a lot of work. But for the sake of the demonstration, we're going to use an implementation of the pYIN algorithm. Even though it's meant for solving step 3, with the right constraints it takes care of 1 and 2 pretty decently, while tracking the vocal fundamental even in the presence of music. The example below contains the output from this approach, without addressing the unvoiced sections.

**Ariana Grande, extracted vocals, harmonic mask**

from **Alejandro Koretzky**


Well…? It sort of did the trick but the quality of the recovered vocal is not there yet. Maybe with additional time, energy and budget we can improve this method and get it to a better place.

Now let me ask you…

What happens when you have **multiple vocals**, which is definitely the case in at least 50% of today's professionally produced tracks?

What happens when the vocals have been processed with **reverberation, delays** and

from **Alejandro Koretzky**

00:37

Are you feeling the pain already…? I am.

Very soon, ad-hoc methods like the one described above become a house of cards. The problem is just too complex. There are too many rules, too many exceptions to the rules and too many varying conditions (effects and different mixing settings). The multi-step approach also implies that errors in one step propagate issues to the step that comes after. Improving each step would be very costly, it would require a large number of iterations to get right and last but not least, we would probably end up with a computationally expensive pipeline, something that by itself can be a deal-breaker.

**These are the kind of scenarios where we need to start thinking of a more *end-to-end* approach and let ML figure out -PART- of the underlying processes and operations required to solve the problem. However, we are not throwing the towel when it comes to feature engineering and you'll see why.**

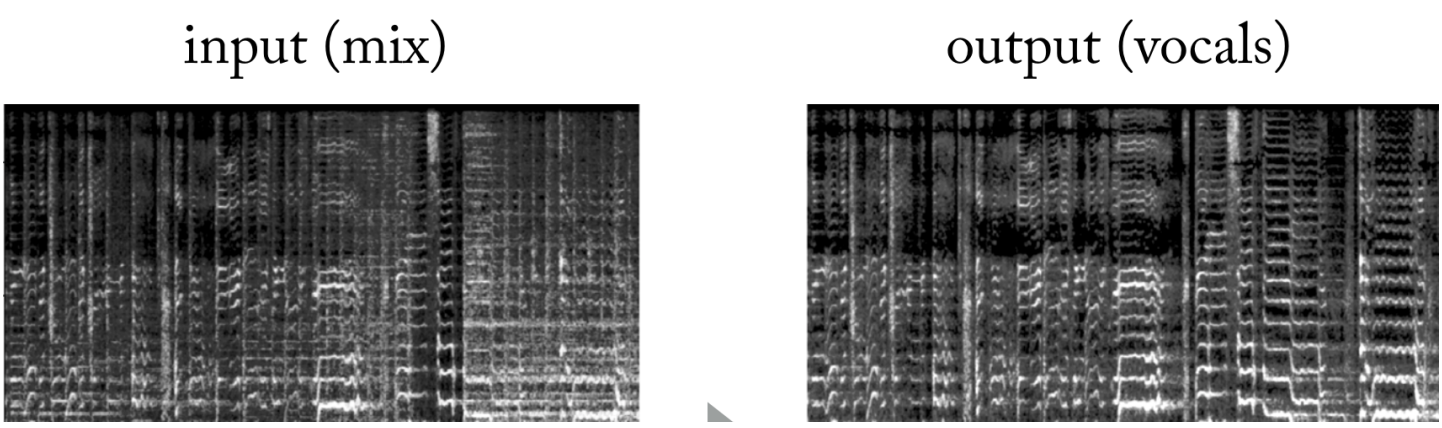
## The hypothesis: we can use a CNN as a transfer function that maps mixes into vocals

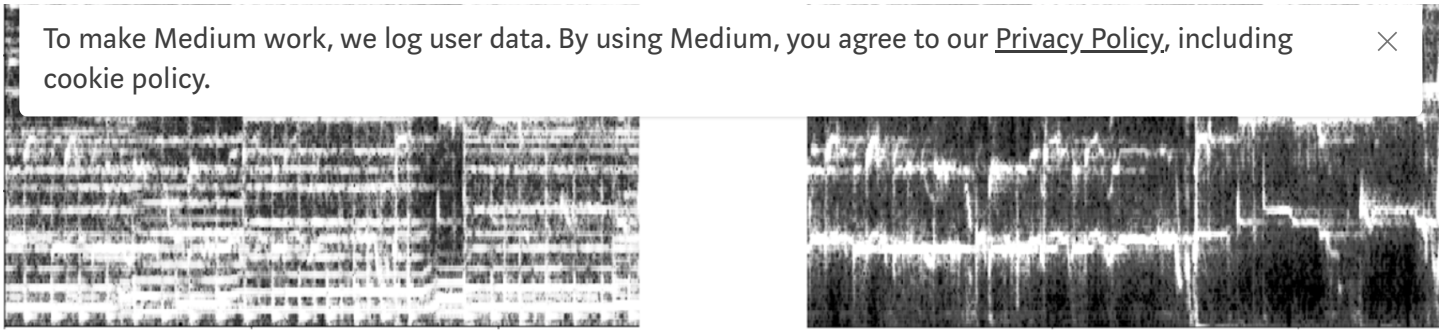
Inspired by the achievements with CNNs on natural images, why not apply the same reasoning here?

input output

CNNs have been successful at tasks such as image colorization, deblurring and super-resolution.

At the end of the day, we know we can represent an audio signal 'as an image' using the Short-Time Fourier Transform right? Even though these *audio images* don't follow the statistical distribution of natural images, they still expose spatial patterns (in the time vs frequency space) that we should be able to learn from.





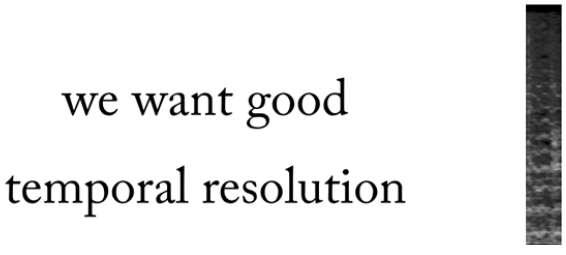
Mix: you can see the kick drum and baseline at the bottom, and some of the synths in the middle getting mixed with the vocals. On the right, the corresponding vocals-only

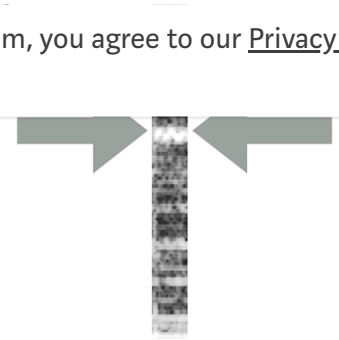
At the time, validating this experiment was a costly endeavor, because obtaining or generating the training data required was already a big investment. One of the practices I always try to implement in applied research is to first **identify a simpler problem that validates the same principles as the original one,** but that does not require as much work. This allows you to keep your hypotheses smaller, iterate faster and pivot with minimum impact when things don't work as expected.

An implied condition for the original solution to work is that a **CNN must be capable of understanding the structure of human speech**. A simpler problem can then be: *given a mix fragment, let's see if a CNN can classify these fragments as containing vocal content or not*. We are looking at a music-robust Vocal Activity Detector (VAD), implemented as a binary classifier.

## Designing our feature space

We know that audio signals such as music and human speech embed temporal dependencies. In simpler terms, nothing happens in isolation at a given time-frame. If I want to know whether a given section of audio contains human speech or not, I should probably look at the neighbor regions as well. That *temporal context* can give me good information about what's going on in the region of interest. At the same time, we want to perform our classification in very small time increments, so that we can capture the human voice with the highest temporal resolution possible.

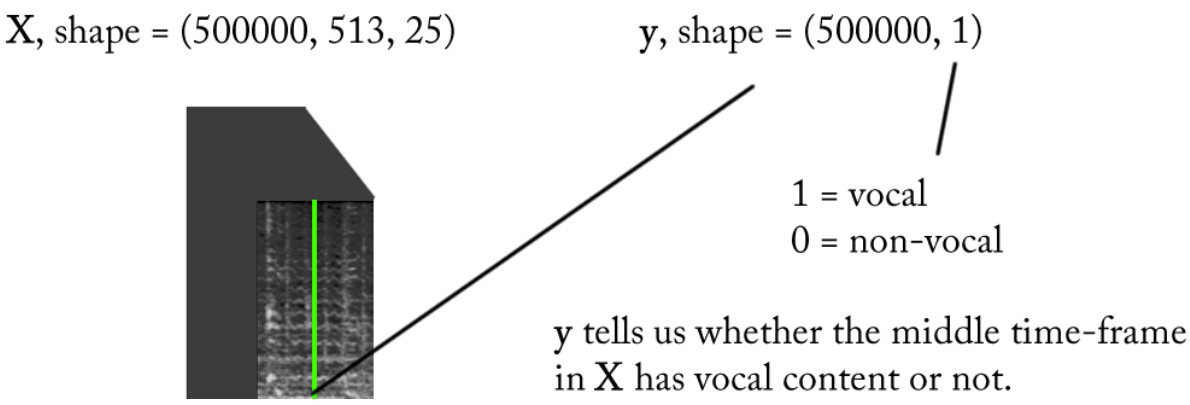


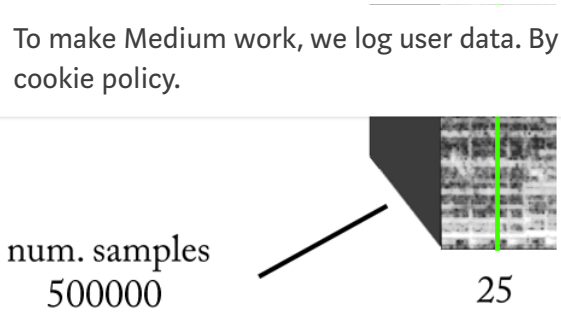


Let's do some numbers…

- Sampling rate (fs): 22050 Hz (we are downsampling from 44100 to 22050)
- STFT design: window size = 1024, hop size = 256, Mel scale interpolation for perceptual weighting. Since our input data is *real,* we can work with one half of the STFT (the why is out of the scope of this post…) while keeping the DC component (not a requirement), giving us 513 frequency bins.
- Target classification resolution: a single STFT frame (~11.6 milliseconds = 256 / 22050)
- Target temporal context: ~300 milliseconds = 25 STFT frames.
- Target number of training examples: 0.5M
- Assuming we are using a sliding window with a stride of 1 STFT time-frame to generate our training data, we need around 1.6 hours of labeled audio to generate our 0.5M data samples. [if you'd like to know more details about generating the actual dataset, please feel free to ask in the comments]

With the above requirements, the input/output data to our binary classifier looks like this:



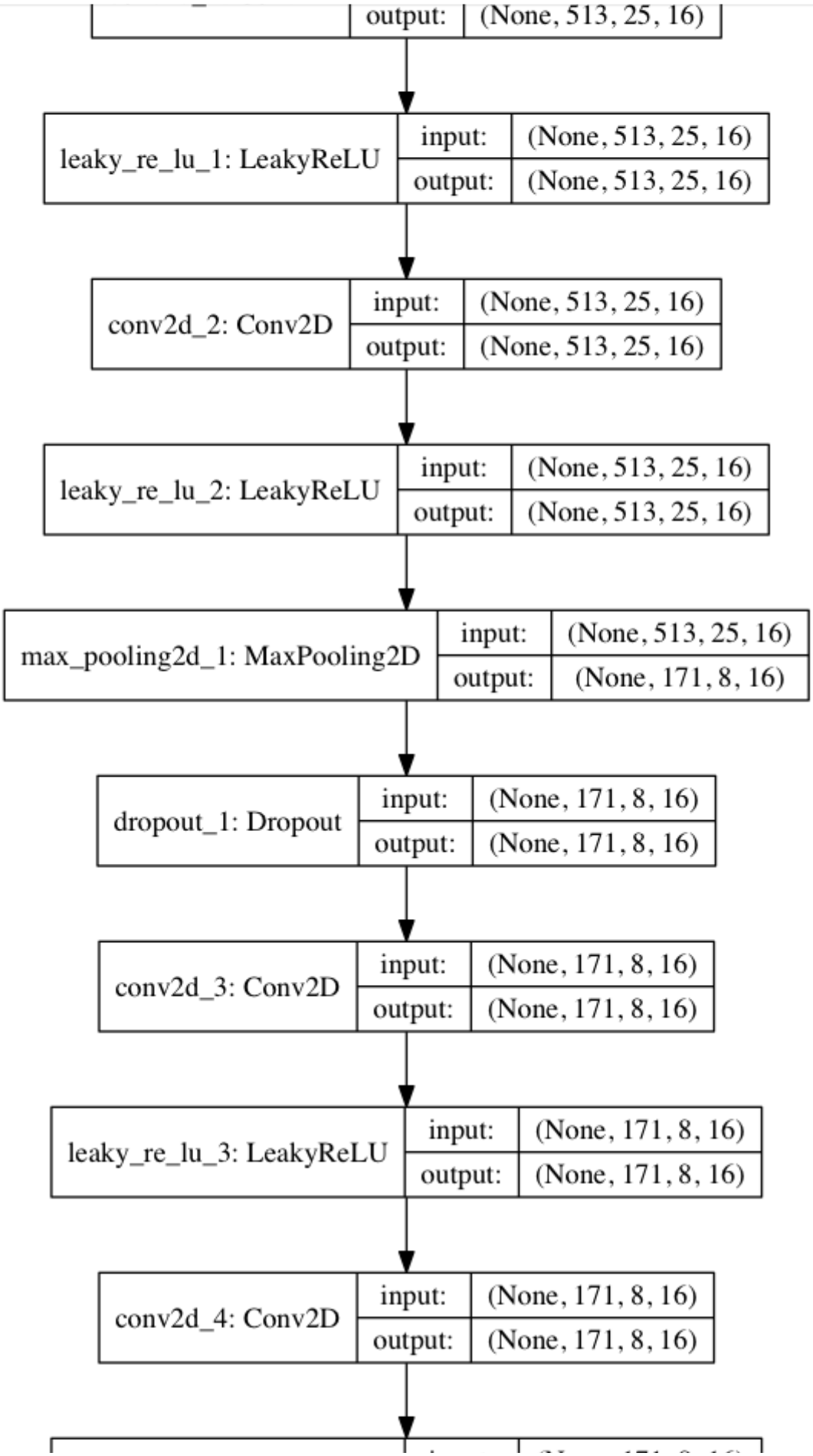


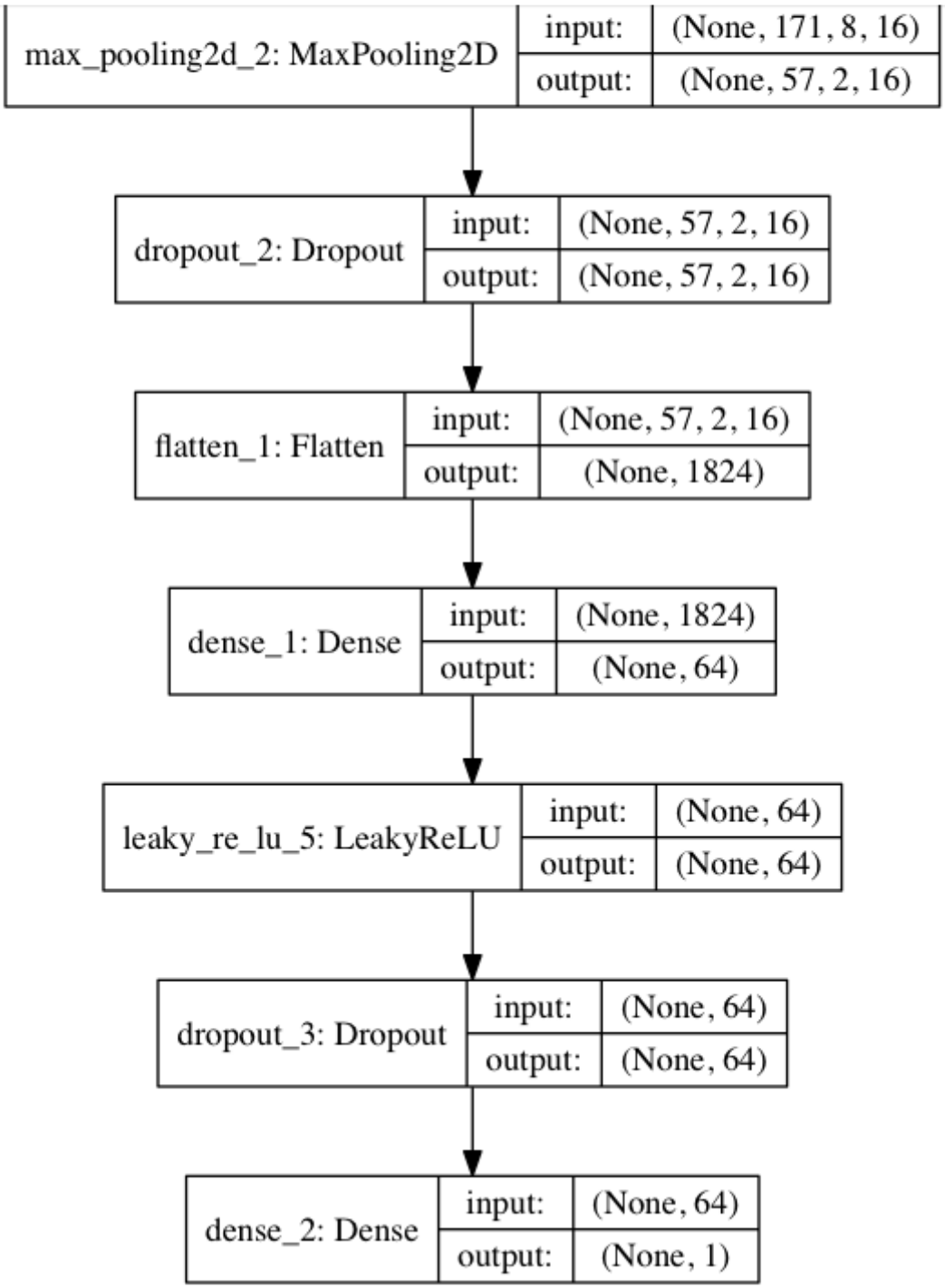

## The model

Using Keras, we can build a small CNN model to validate our hypothesis.

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D,
MaxPooling2D
from keras.optimizers import SGD
from keras.layers.advanced_activations import LeakyReLU

model = Sequential()
model.add(Conv2D(16, (3,3), padding='same', input_shape=(513, 25,
1)))
model.add(LeakyReLU())
model.add(Conv2D(16, (3,3), padding='same'))
model.add(LeakyReLU())
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Dropout(0.25))

model.add(Conv2D(16, (3,3), padding='same'))
model.add(LeakyReLU())
model.add(Conv2D(16, (3,3), padding='same'))
model.add(LeakyReLU())
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64))
model.add(LeakyReLU())
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss=keras.losses.binary_crossentropy, optimizer=sgd,
metrics=['accuracy'])
```

| | | |
|---|---|---|
| conv2d_1_input: InputLayer | input: | (None, 513, 25, 1) |
| | output: | (None, 513, 25, 1) |

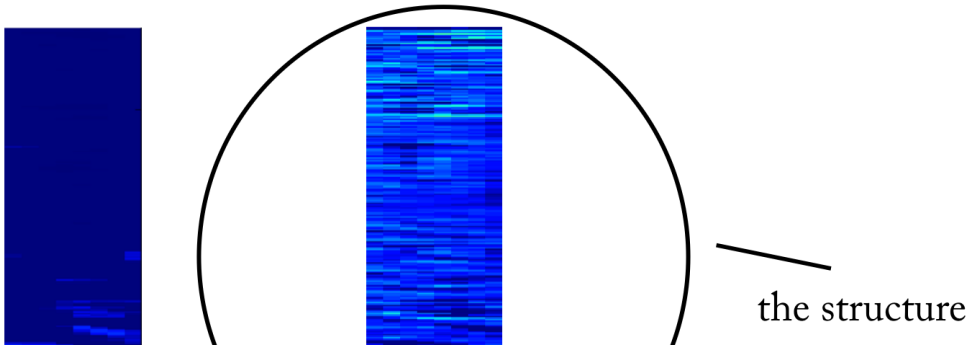


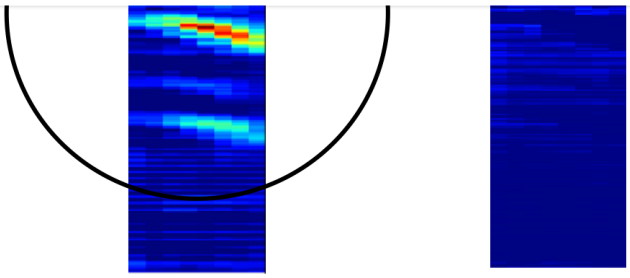



| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_1 (Conv2D) | (None, 513, 25, 16) | 160 |
| leaky_re_lu_1 (LeakyReLU) | (None, 513, 25, 16) | 0 |
| conv2d_2 (Conv2D) | (None, 513, 25, 16) | 2320 |
| leaky_re_lu_2 (LeakyReLU) | (None, 513, 25, 16) | 0 |

```
conv2d_3 (Conv2D)            (None, 171, 8, 16)        2320
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 171, 8, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 171, 8, 16)        2320
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 171, 8, 16)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 57, 2, 16)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 57, 2, 16)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1824)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                116800
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 123,985
Trainable params: 123,985
Non-trainable params: 0
_________________________________________________________________
```

With an 80/20 train-test split and after ~50 epochs we reach ~**97% test accuracy**, which means there's enough evidence that our CNN model can discriminate between music sections containing vocal content and music sections without vocal content. By inspecting some of the feature maps coming out of our 4th convolutional layer, it looks like our network has optimized its kernels to perform 2 tasks: filtering out music and filtering out vocals…





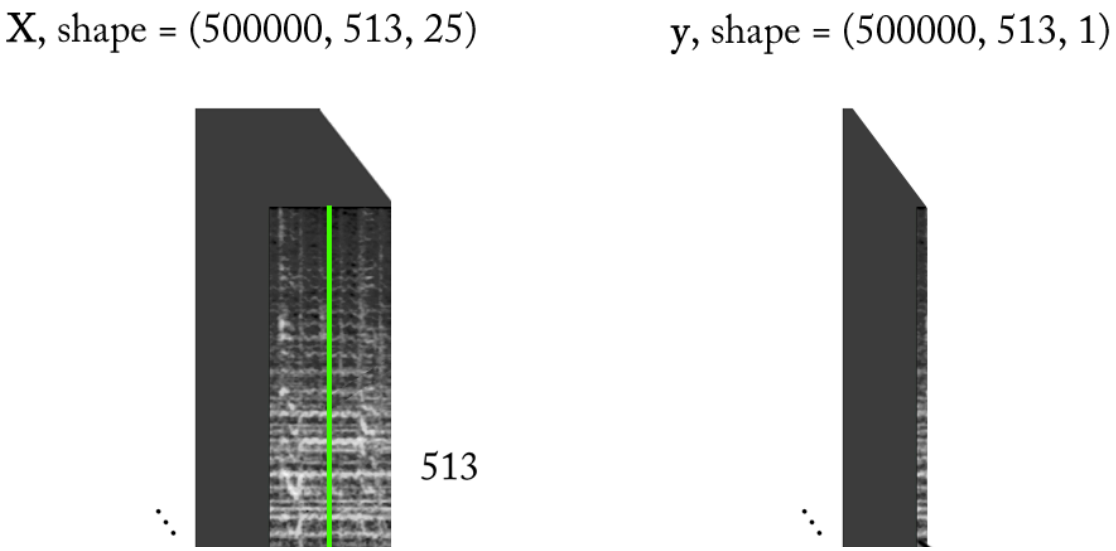
Sample feature maps at the output of the 4th conv. layer. Apparently, the output on the left is the result of a combination of kernel operations that try to preserve vocal content while ignoring music. The high values resemble the harmonic structure of human speech. The feature map on the right seems to be the result of the opposite task.

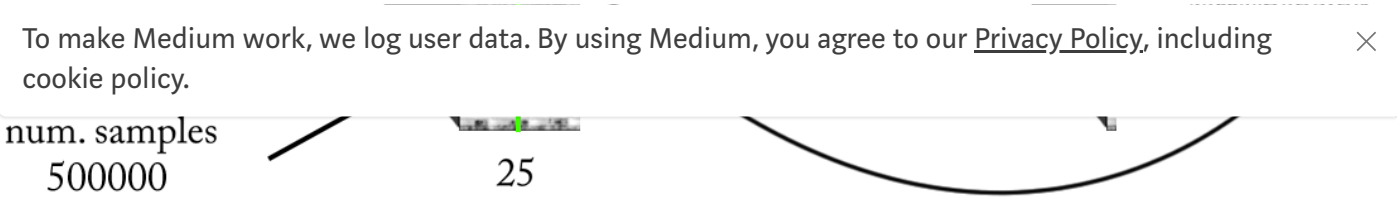
## From VAD to Source Separation

Now that we've validated this simpler classification problem, how do we go from detecting vocal activity in music all the way to isolating vocals from music? Well, rescuing some ideas from our *naive* method described at the beginning, we still want to somehow end up with an estimate of the vocal's magnitude spectrogram. This now becomes a regression problem. What we want to do is, given a time-frame from the STFT of the mix (with enough temporal context), estimate the corresponding vocal time-frame's magnitude spectrum.

**What about the training set? (you might be asking yourself at this point)**

oh Lord… that was something. I'm gonna address this at the end of the article so that we don't switch contexts yet!






If our model learns well, during inference, all we need to do is implement a simple sliding window over the STFT of the mix. After each prediction, we move our window to the right by 1 time-frame, predict the next vocal frame and concatenate it with the previous prediction. In regards to the model, we can start by using the same model we used for VAD as a baseline and by making some changes (output shape is now (513,1), linear activation at the output, MSE as loss function), we can begin our training.

**Don't claim victory yet…**

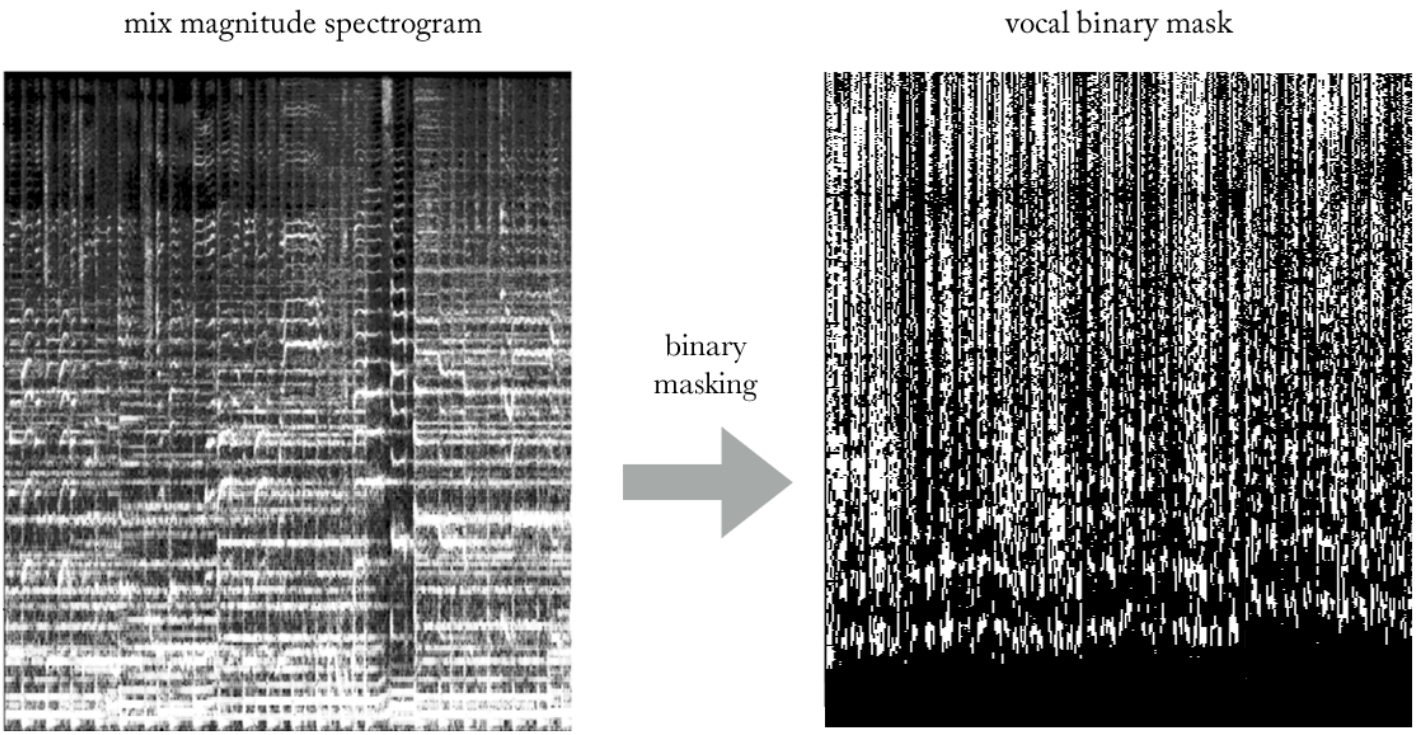
Although the above input/output representation makes sense, after training our vocal separation model several times, with varying parameters and data normalizations, the results are not there yet. It seems like **we are asking for too much…**

We went from a binary classifier to trying to do *regression* on a 513-dimensional vector. Although the network learns the task to a degree, after reconstructing the vocal's time domain signal, there are obvious artifacts and interferences from other sources. Even after adding more layers and increasing the number of model parameters, the results don't change much.

So then the question became: **can we trick the network into *thinking* it is solving a simpler problem and still achieve the desired results?**

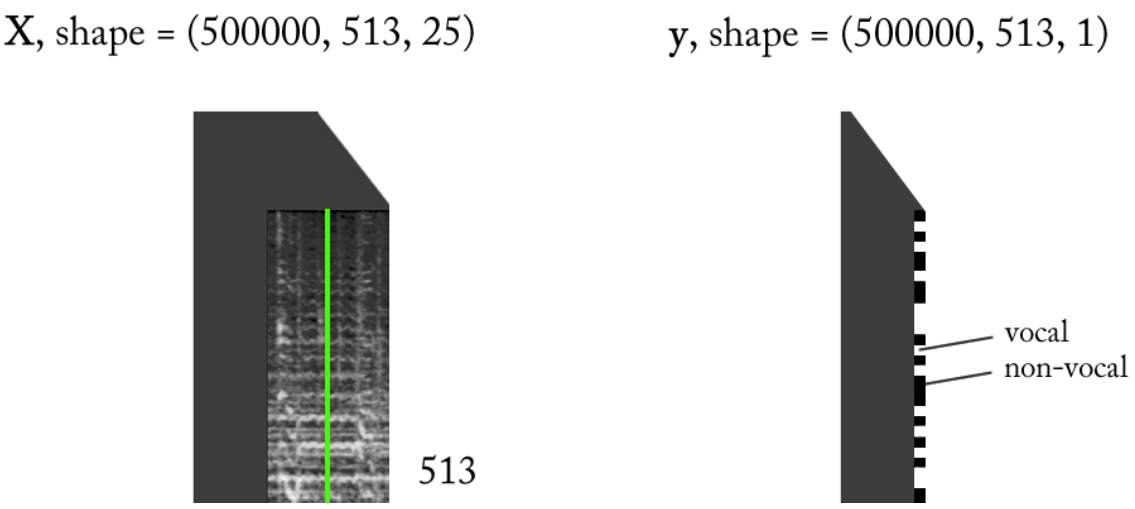
What if instead of trying to estimate the vocal's magnitude STFT, we trained the network to learn a *binary mask* that, when applied to the STFT of the mix, gives us a simplified but **perceptually-acceptable-upon-reconstruction** estimate of the vocal's magnitude spectrogram?

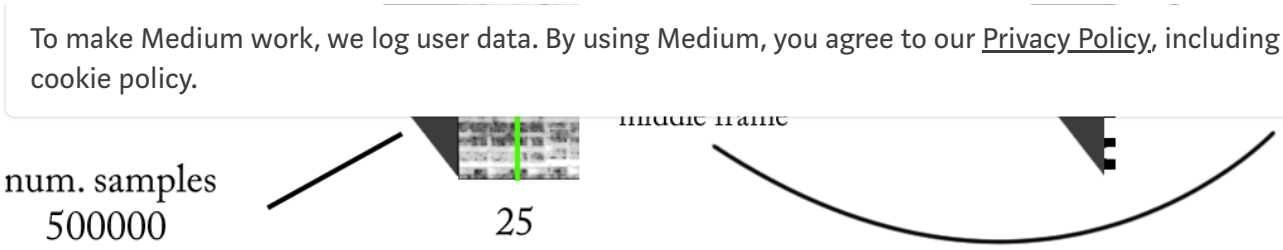
By experimenting with different heuristics, we came up with a relatively simple (and definitely unorthodox from a Signal Processing perspective) way to extract singing voice from mixes using binary masks. Without going too much into the details, we are going to think of the output as a binary *image* where, a value of '1' indicates **predominant presence of vocal content** at a given frequency and timeframe location, and a value of '0' indicates predominant presence of music at the given location.



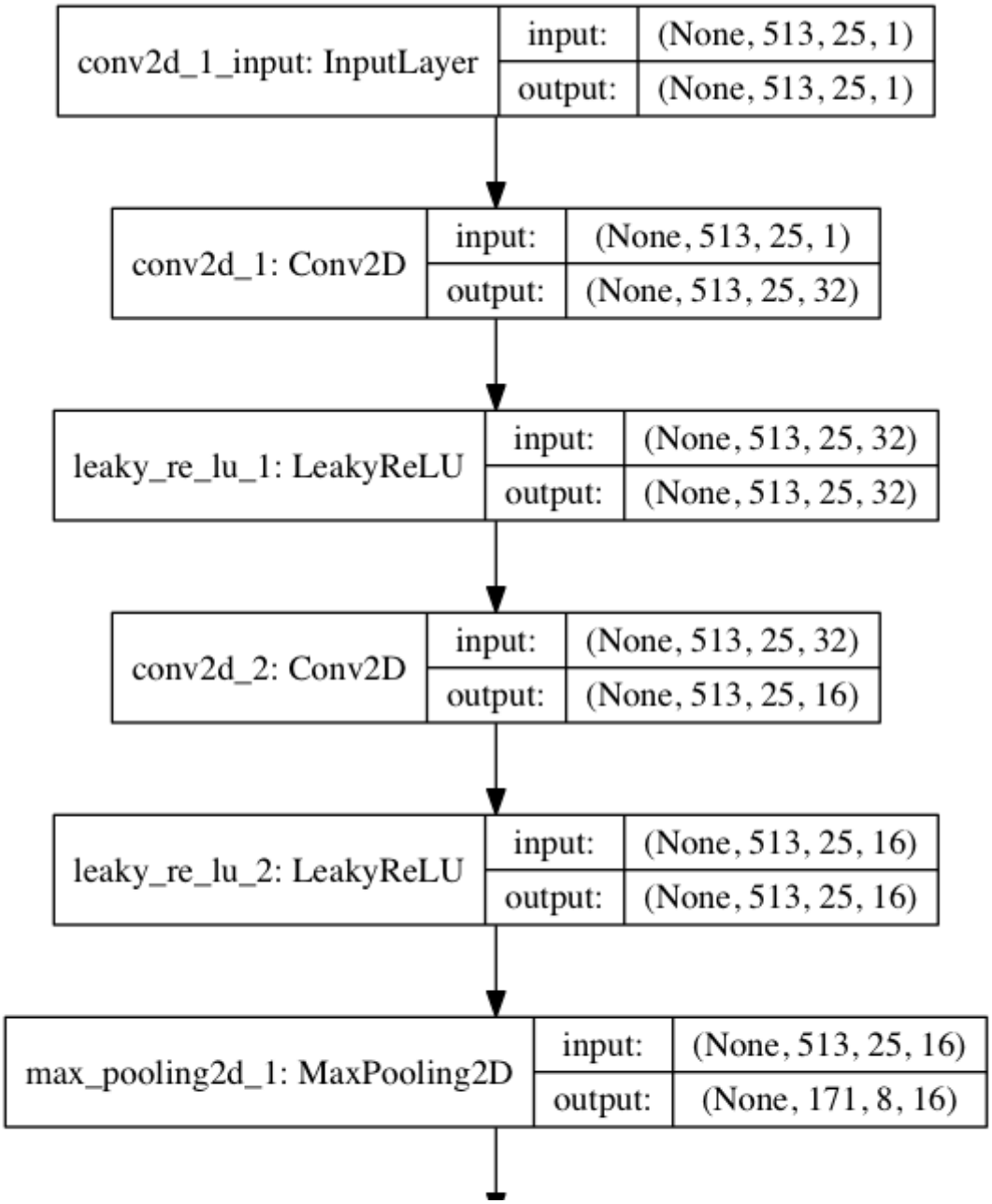

Binary masking (or TF masking) techniques have been around for decades and have been proven very effective in specific audio source separation and classification problems. However, most available techniques have been optimized for speech enhancement / recognition applications and fall short when it comes to real-world data such as professionally-produced & mastered music. Our binarization method was specifically designed for this scenario.

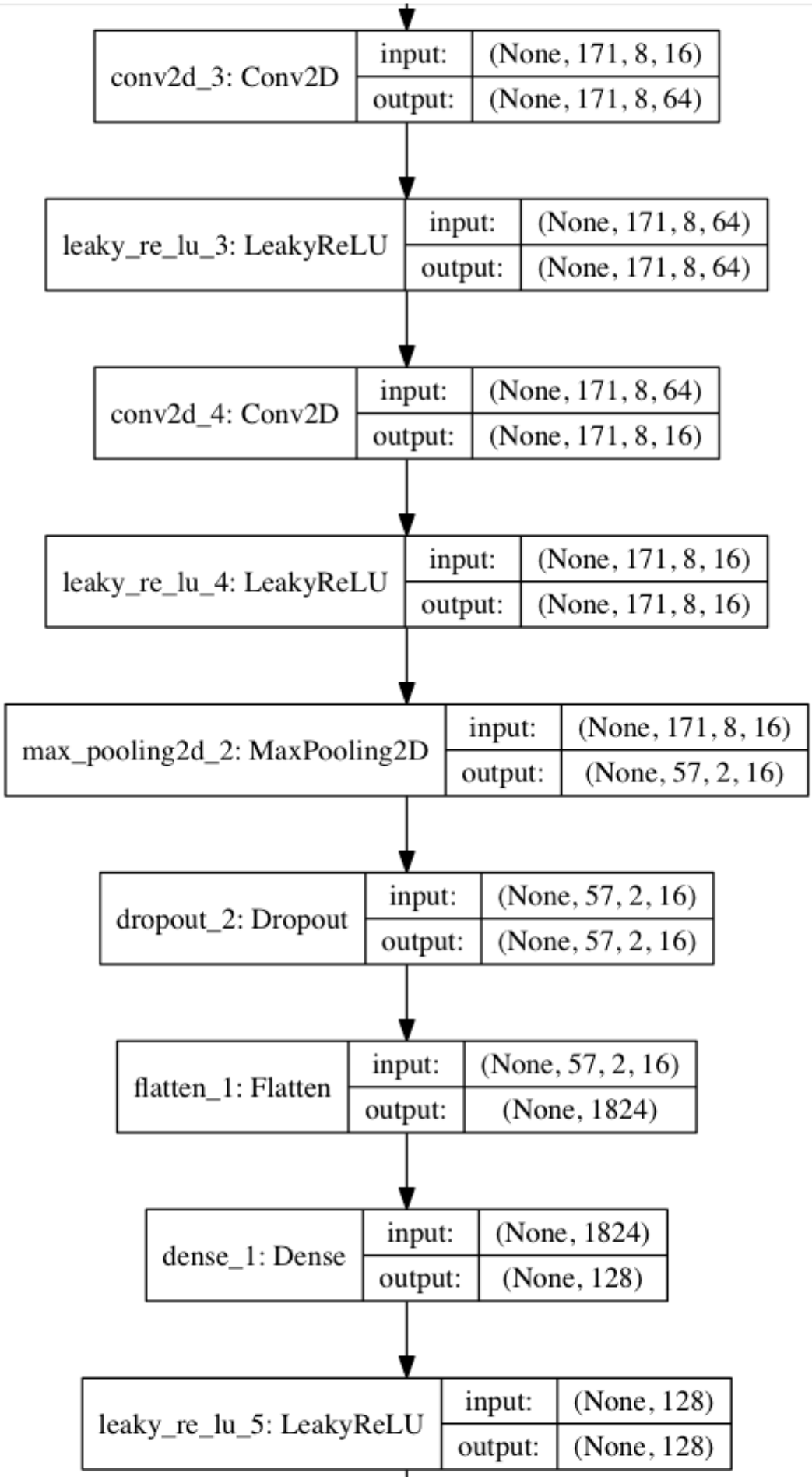
Our problem now becomes some sort of regression-classification hybrid. We are asking the model to “classify pixels” at the output as vocal or non-vocal, although conceptually (and also in terms of the loss function used -MSE- ), the task is still a regression one.

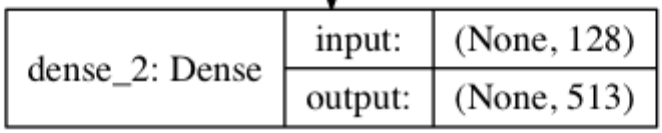





Although the distinction might not seem relevant to some, it actually makes a huge difference in the model's ability to learn the assigned task, the second one being way more simple and constrained. At the same time, it allows us to keep our model relatively small in terms of number of parameters considering the complexity of the task, something highly desired for real-time operation, which was a design requirement in this case. After some minor tweaks, the final model looks like this.



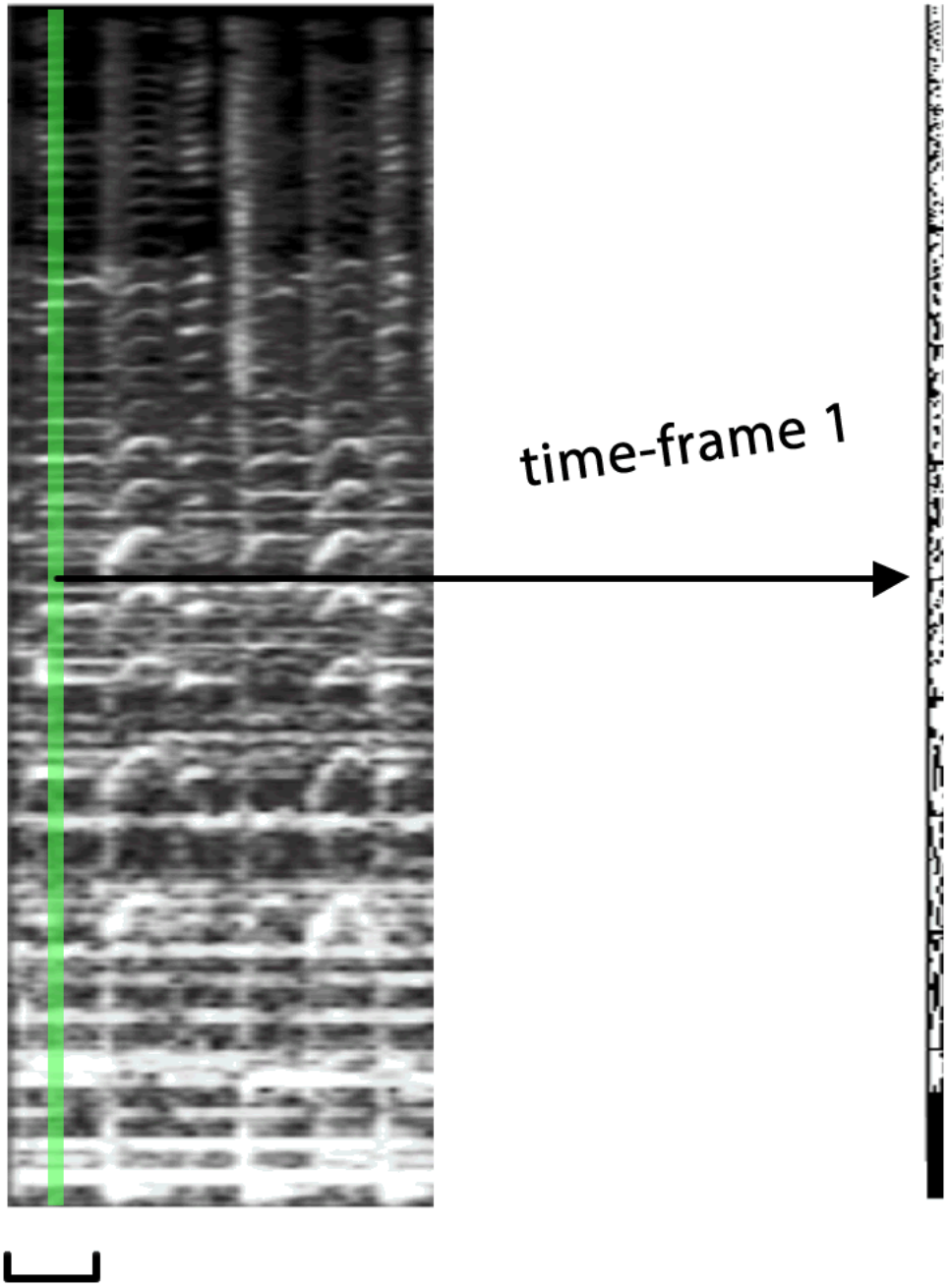





| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_1 (Conv2D) | (None, 513, 25, 32) | 320 |
| leaky_re_lu_1 (LeakyReLU) | (None, 513, 25, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 513, 25, 16) | 4624 |
| leaky_re_lu_2 (LeakyReLU) | (None, 513, 25, 16) | 0 |
| max_pooling2d_1 (MaxPooling2 | (None, 171, 8, 16) | 0 |
| dropout_1 (Dropout) | (None, 171, 8, 16) | 0 |
| conv2d_3 (Conv2D) | (None, 171, 8, 64) | 9280 |
| leaky_re_lu_3 (LeakyReLU) | (None, 171, 8, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 171, 8, 16) | 9232 |
| leaky_re_lu_4 (LeakyReLU) | (None, 171, 8, 16) | 0 |
| max_pooling2d_2 (MaxPooling2 | (None, 57, 2, 16) | 0 |
| dropout_2 (Dropout) | (None, 57, 2, 16) | 0 |
| flatten_1 (Flatten) | (None, 1824) | 0 |
| dense_1 (Dense) | (None, 128) | 233600 |
| leaky_re_lu_5 (LeakyReLU) | (None, 128) | 0 |
| dropout_3 (Dropout) | (None, 128) | 0 |
| dense_2 (Dense) | (None, 513) | 66177 |

Total params: 323,233
Trainable params: 323,233

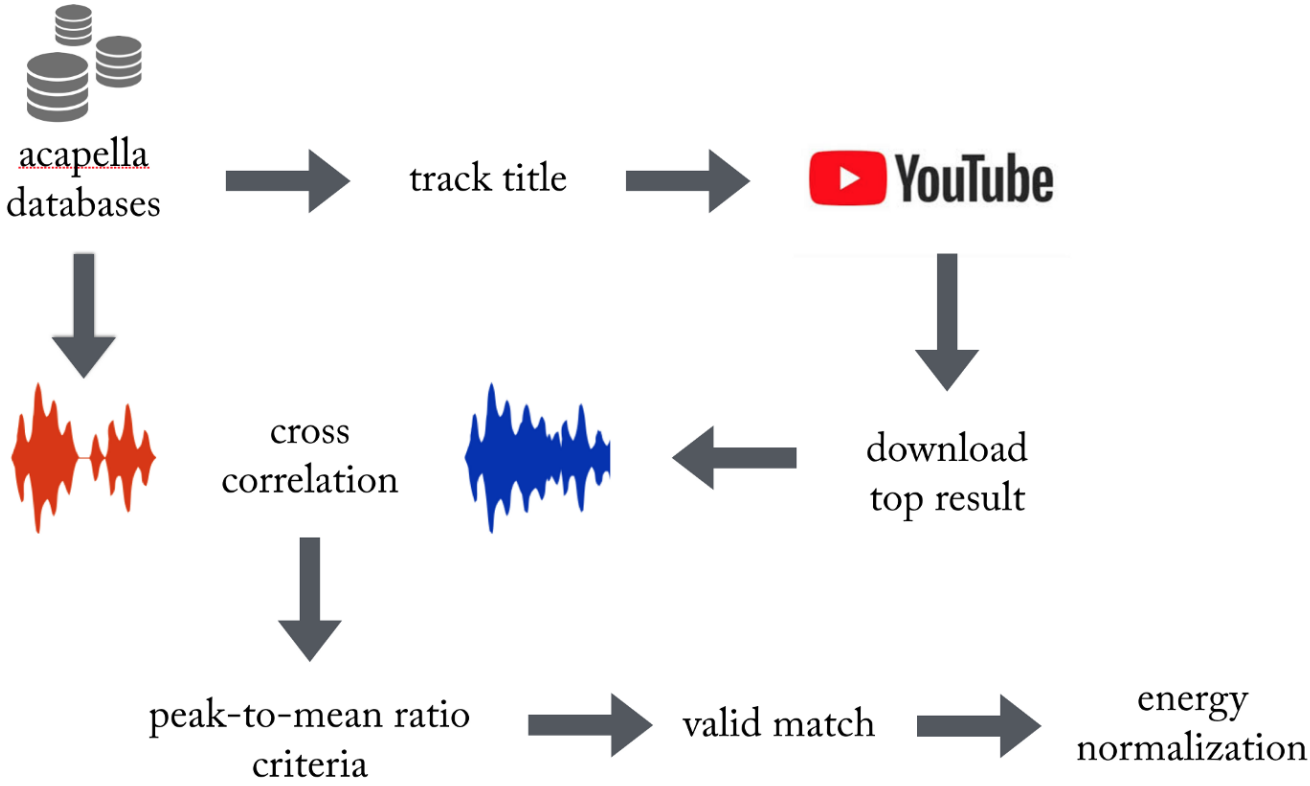
that we do, we are predicting a single timeframe of the vocals' binary mask. Again, by implementing a simple sliding window with a stride of one timeframe, we keep estimating and concatenating consecutive timeframes, which end up making up the whole vocal binary mask.



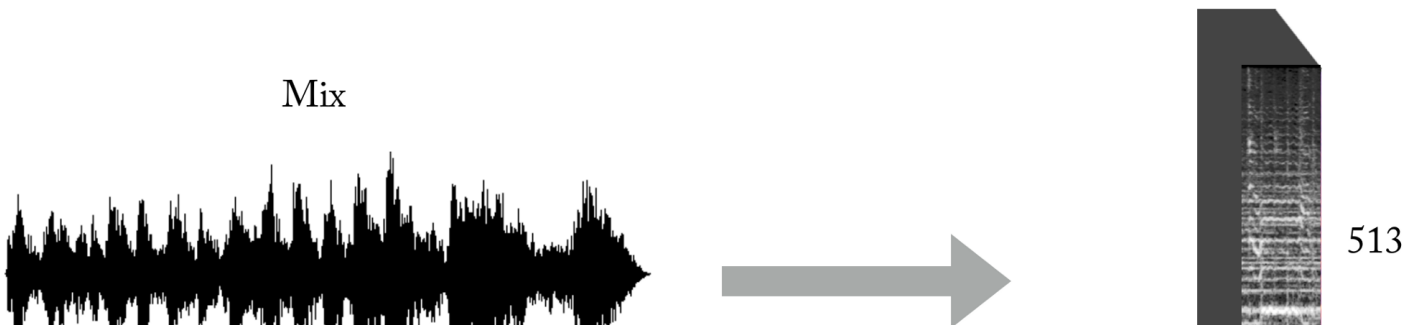

## Creating the training set

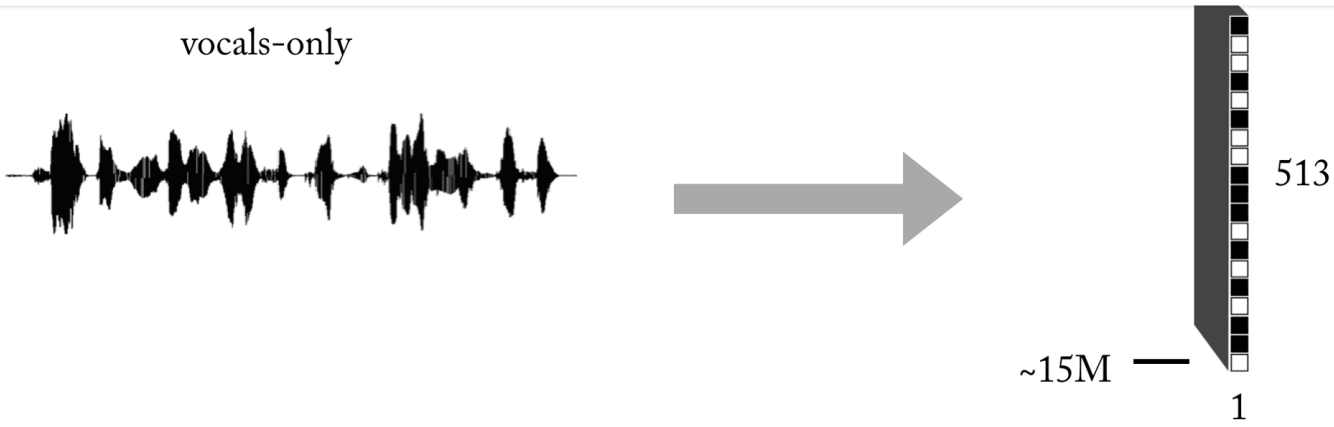
As you know, one of the biggest pain points in supervised Machine Learning (leave aside all those toy examples with available datasets out there) is having the right data

normalized vocal tracks. There's more than one way to build this dataset and here we used a combination of strategies, ranging from manually creating mix <> vocal pairs with some acapellas found online, to finding RockBand stems, to web-scraping Youtube. Just to give you an idea of what part of this time-consuming and painful process looked like, our "dataset project" involved creating a tool to automatically build mix <> vocal pairs as illustrated below:




We knew we needed a good amount of data for the network to learn the transfer function needed to map mixes into vocals. Our final dataset consisted of around 15M examples of ~300-millisecond fragments of mixes and their corresponding vocal binary masks.






### Pipeline architecture

Building a Machine Learning model for a given task is only part of the deal. In production environments, we need to think about software architecture, pipelines, and optimization strategies, especially when we're dealing with real-time.

For this particular problem the reconstruction into the time-domain can be done all at once after predicting a full vocal binary mask (offline mode) or, more interestingly, as part of a multithreaded pipeline where we acquire, process, reconstruct and playback in small segments, allowing this to be streaming-friendly, and even capable of delivering real-time deconstruction on music that's being recorded on the fly, with minimum latency. Given this is a whole topic on its own, I'm going to leave it for another article focused on **real-time ML pipelines**…

## I think I've covered enough so why don't we listen to a couple more examples?

### Daft Punk — Get Lucky (Studio)

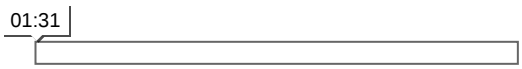
'Daft Punk - Get Lucky', vocal isolation from stereo using Convolutional Neural Networks

from Alejandro Koretzky

## Adele — Set Fire to the Rain (live recording!)

**'Adele - Set Fire to the Rain', vocal isolation from stereo using Convolutional Neural Networks**

from **Alejandro Koretzky**




Notice how at the very beginning our model extracts the crowd's screaming as vocal content :). In this case we have some additional interference from other sources. This being a live recording it kinda makes sense for this extracted vocal not to be as high quality as the previous ones.

## Ok, so there's 'one last thing' ...

## Given this works for vocals why not apply it to other instruments...?

This article is extensive enough already but given you've made it this far I thought you deserved to see one last demo. With the exact same reasoning for extracting vocal content, we can try to split a stereo track into STEMs (drums, bassline, vocals, others) by making some modifications to our model and of course, by having the appropriate training set :). If you are interested in the technical details for this extension, just leave me some comments. I will consider writing a 'part 2' for the Stereo-to-Stems deconstruction case when time allows!

*** **UPDATE 06/2019*****

Part 2, Stereo-to-Stems deconstruction is **here!**

**'Daft Punk - Get Lucky' STEMs isolation from stereo using Convolutional Neural Networks**

from **Alejandro Koretzky**

Thanks for reading and don't hesitate in leaving questions. I will keep writing articles on **Audio AI** so stay tuned! As a final remark, you can see that the actual CNN model we ended up building is not that special. The success of this work has been driven by focusing on the **Feature Engineering** aspect and by implementing a lean process for hypotheses validations, something I'll be writing about in the near future!

ps: shoutouts to Naveen Rajashekharappa and Karthiek Reddy Bokka for their contributions to this work!

Machine Learning Signal Processing Audio Music Towards Data Science