Assignment - 1

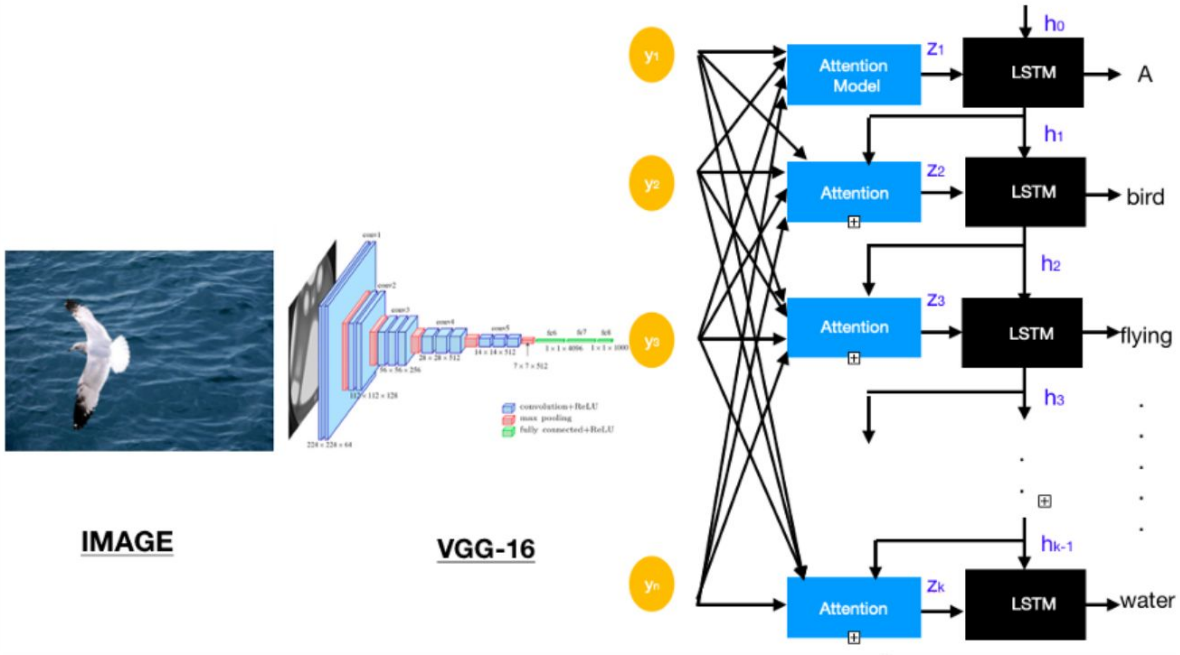
1) Image Caption Generation:

Generating captions of an image using deep learning is a very fun type of project. It shows computers can understand the image through captions. For a caption model, it not only needs to find which objects are contained in the image and also needs to be able to express their relationships in a natural language such as English.

Image Caption Generation with Attention Mechanism

Concept of Attention Mechanism:

The image is first divided into n parts, and we compute with a Convolutional Neural Network (CNN) representations of each part $h_1,...,h_n$. When the RNN is generating a new word, the attention mechanism is focusing on the relevant part of the image, so the decoder only uses specific parts of the image.



We can recognize the figure of the “classic” model for image captioning, but with a new layer of attention model. What is happening when we want to

predict the new word of the caption? If we have predicted i words, the hidden state of the LSTM is hi. We select the « relevant » part of the image by using hi as the context. Then, the output of the attention model z_i, which is the representation of the image filtered such that only the relevant parts of the image remain, is used as an input for the LSTM. Then, the LSTM predicts a new word and returns a new hidden state h_{i+1}.

For example, there is an image of a man is throwing a frisbee.

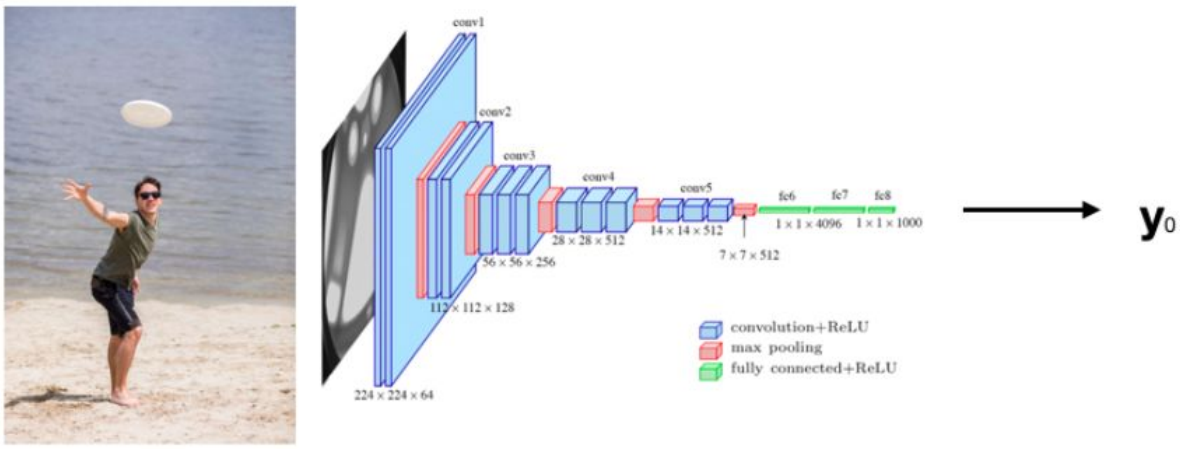
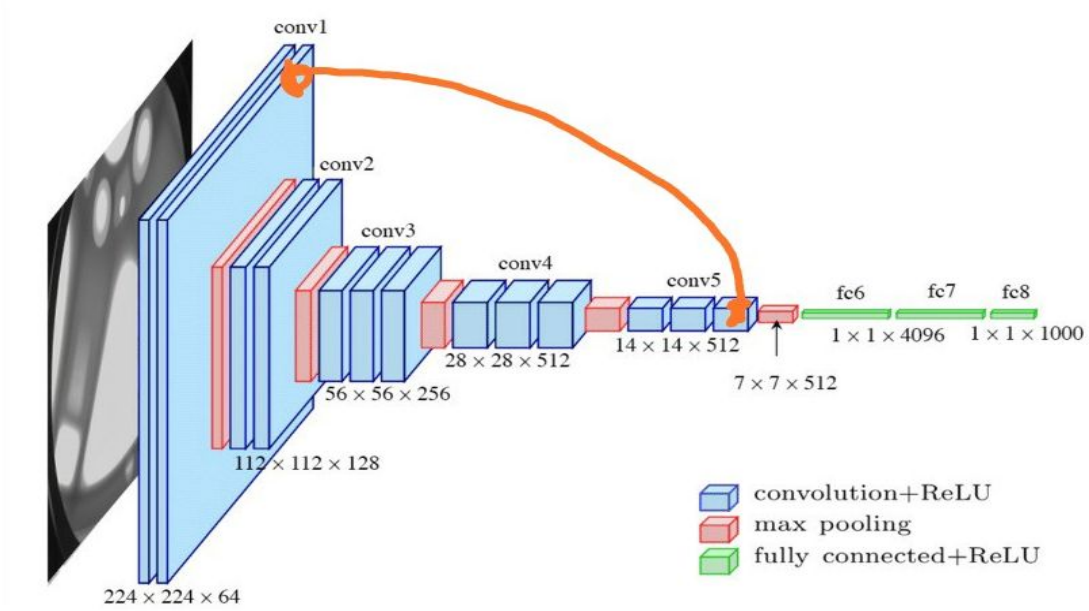


IMAGE **VGG-16 Encoder**

So, when I say the word ‘man’ that means we need to focus only on the man in the image, and when I say the word ‘throwing’ then we have to focus on his hand in the image. Similarly, when we say ‘frisbee’ we have to focus only on the frisbee in the image. This means ‘man’, ‘throwing’, and ‘frisbee’ come from different pixels in the image. But the VGG-16 representation we used does not contain any location information in it.

But every location of convolution layers corresponds to some location of the image as shown below.



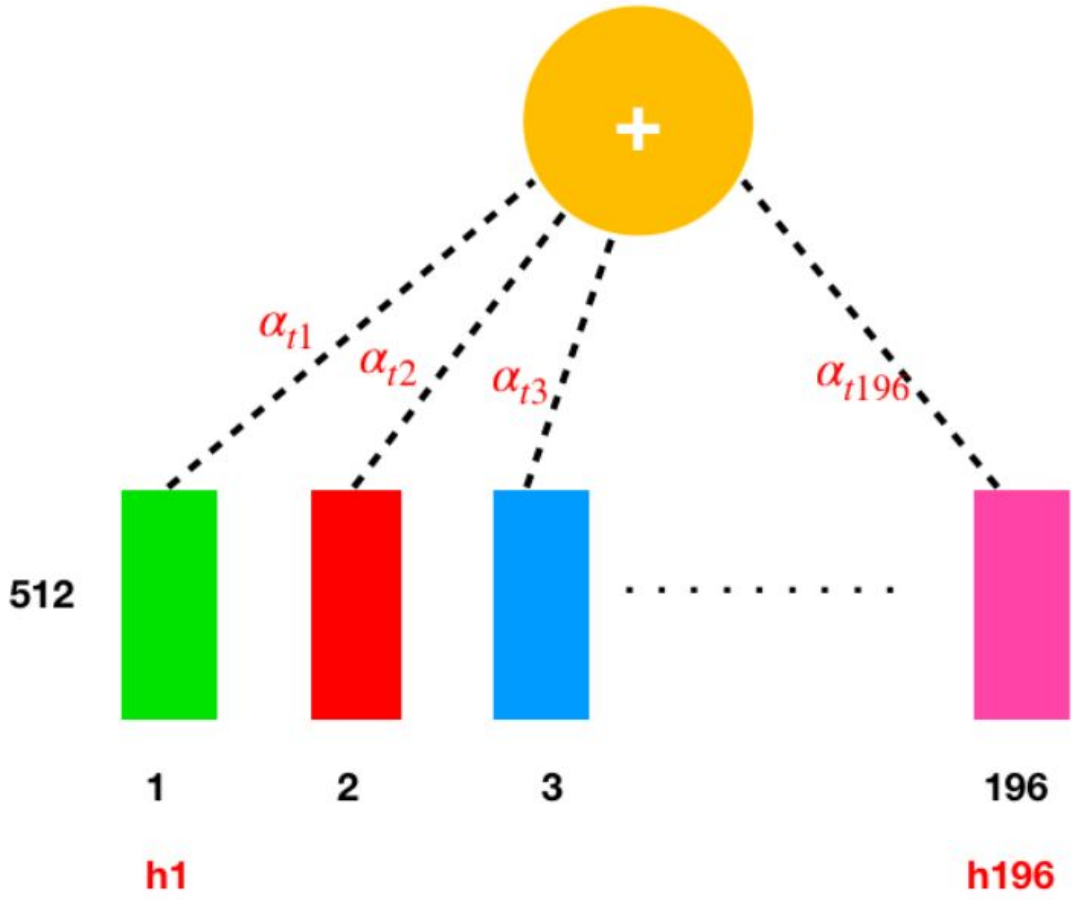
VGG-16

The output of the 5th convolution layer of VGGNet is a 14*14*512 size feature map.

This 5th convolution layer has 14*14 pixel locations which correspond to a certain portion in the image, which means we have 196 such pixel locations.

And finally, we can treat these 196 locations(each having 512-dimensional representation).

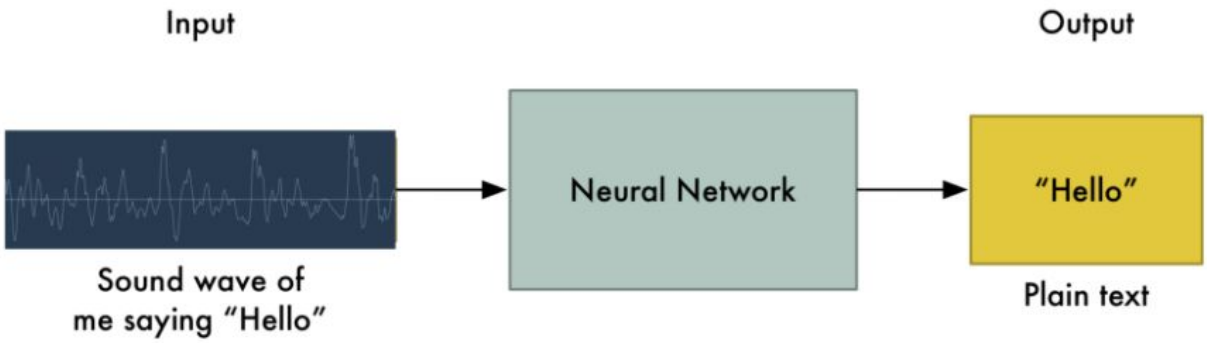
The model will then learn attention over these locations(which in turn corresponds to actual locations in the images).



As shown in the above figure 5th convolution block is represented by 196 locations which can be passed in different time steps.

2) Speech Recognition:

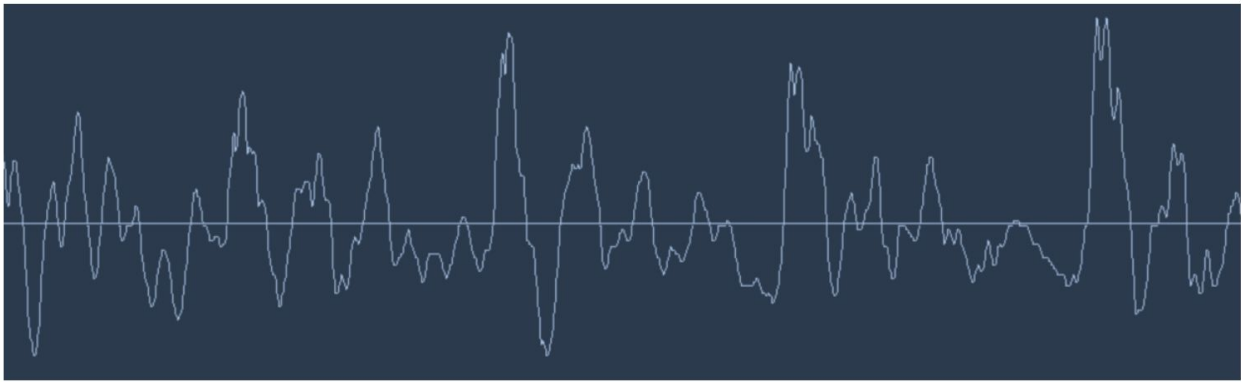
Speech recognition has around for decades and becoming mainstream nowadays. Amazon's The Echo Dot has been so popular this holiday season that they can't seem to keep them in stock! The reason is that deep learning finally made speech recognition accurate enough to be useful outside of carefully controlled environments.



This what happens behind the scenes when we use an speech recognition system or a device.

1) Turning Sounds into Bits

The first step in speech recognition is obvious — we need to feed sound waves into a computer.

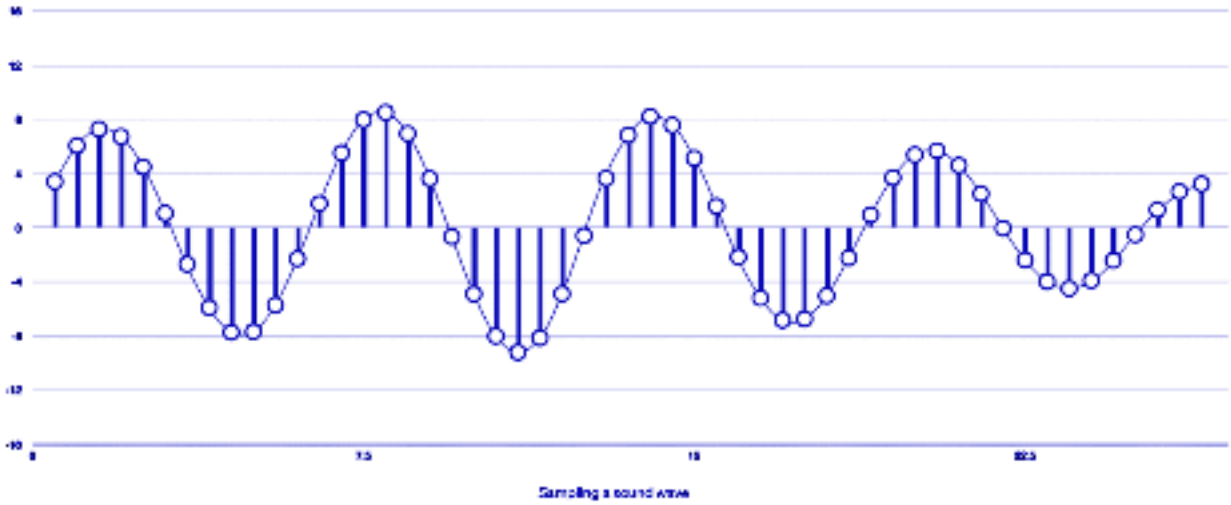


But sound waves look like this so how can we make it understandable to computers.

Sound waves are one-dimensional. At every moment in time, they have a single value based on the height of the wave.

Let's zoom in on one tiny part of the sound wave and take a look:

To turn this sound wave into numbers, we just record the height of the wave at equally-spaced points:

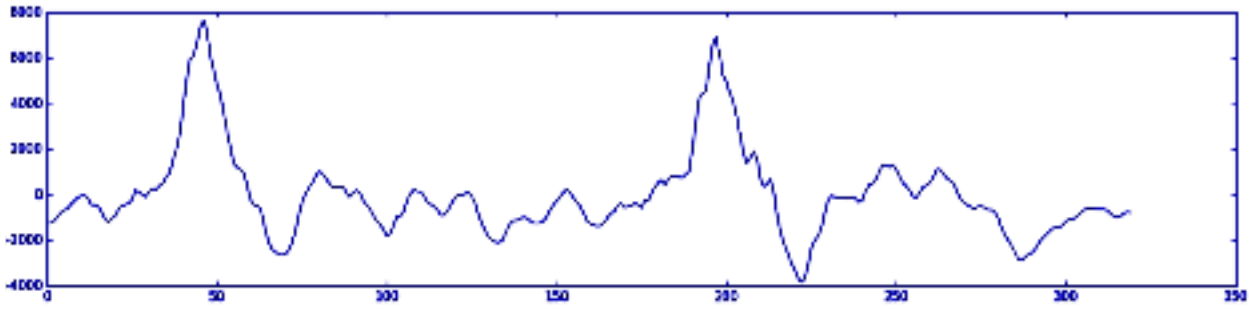


This is called sampling.

Now we need to use the Nyquist theorem to turn it into an array

```
[-1274, -1252, -1160, -986, -792, -692, -614, -429, -286, -134, -57, -41, -169, -4
, -397, -212, 193, 114, -17, -110, 128, 261, 198, 390, 461, 772, 948, 1451, 1974,
 4820, 4353, 3611, 2740, 2004, 1349, 1178, 1085, 901, 301, -262, -499, -488, -707,
1648, -970, -364, 13, 260, 494, 788, 1011, 938, 717, 507, 323, 324, 325, 350, 103,
-1815, -1725, -1341, -971, -959, -723, -261, 51, 210, 142, 152, -92, -345, -439, -
398, -896, -1262, -1610, -1862, -2021, -2077, -2105, -2023, -1697, -1360, -1150, -
03, -707, -468, -300, -116, 92, 224, 72, -150, -336, -541, -820, -1178, -1289, -13
7, -531, -376, -458, -581, -254, -277, 50, 331, 531, 641, 416, 697, 810, 812, 759,
6117, 5244, 4951, 4462, 4124, 3435, 2671, 1847, 1370, 1591, 1900, 1586, 713, 341,
59, -3723, -3134, -2380, -2032, -1831, -1457, -804, -241, -51, -113, -136, -122, -
 1267, 1197, 1291, 1110, 793, 514, 370, 174, -90, -139, 104, 334, 407, 524, 771, 1
-552, -512, -575, -669, -672, -763, -1022, -1435, -1791, -1999, -2242, -2563, -285
 -1495, -1460, -1446, -1345, -1177, -1088, -1072, -1003, -856, -719, -621, -585, -
 -788]
```

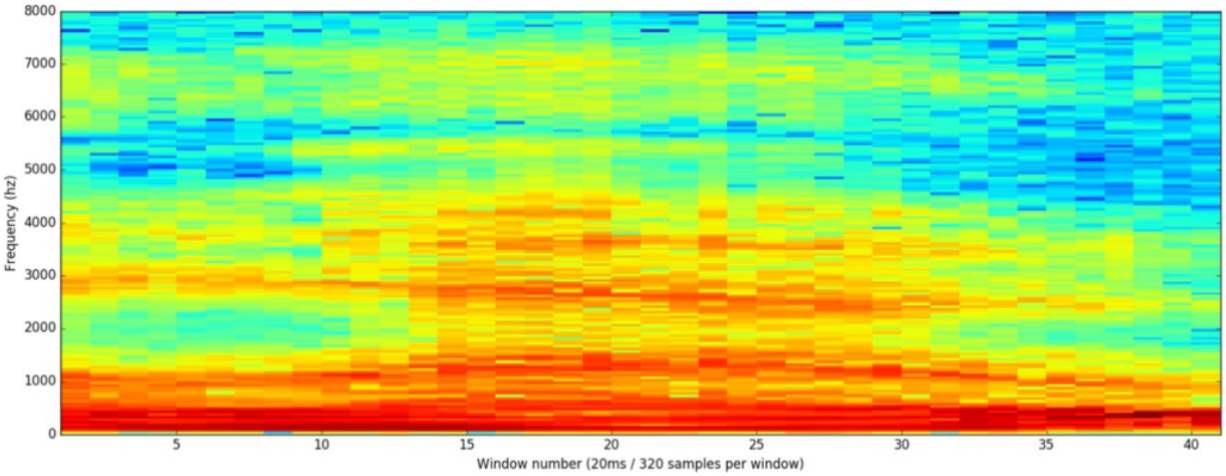
Plotting those numbers as a simple line graph gives us a rough approximation of the original sound wave for those 20 milliseconds:



We do this using a mathematic operation called a Fourier transform.

```
[110.97481594791122, 166.61537247955155, 180.43561044211469, 175.09309469913353, 180.016869109591
, 159.26023828556598, 163.24469810981628, 149.15527353931867, 154.34196586290136, 151.46179061113
8, 146.28632297509679, 164.37233032929228, 158.1282656446088, 147.23266451005145, 133.26597973863
1, 111.80244759457571, 92.590676871856431, 105.75863927434719, 95.673146446282971, 90.39174812806
758, 90.252031938154076, 89.361567351948437, 90.917307309643206, 90.746777849123049, 86.726552726
6669, 105.80328302591124, 109.53029281234707, 116.46408227060996, 129.20890691592615, 130.4346036
29752, 123.93018478493934, 121.19289035588113, 119.03159255422509, 114.23027889344033, 119.171734
000341, 94.376766266598295, 97.850709698634489, 113.37126364077845, 110.24526597732718, 113.72249
81947157, 111.57319012901624, 115.54483708595507, 116.99850750130265, 114.40659619324526, 79.8695
781117835, 82.214144782667248, 67.246072805959614, 66.578937262360313, 74.100307226086798, 64.861
02909362839, 55.693923524361097, 50.776364877715011, 41.196111220671298, 51.062413666348945, 58.4
124915202, 35.413635503802389, 22.705615809958832, 16.458048045346381, 44.910670465379937, 59.282
67526244051, 91.806226496828543, 94.570526932206619, 99.250924315589074, 97.899164767741183, 75.1
8614354976241, 99.366456533638413, 102.18717608176904, 102.06596663023235, 101.78493139911082, 10
10596541338749, 100.75737831526195, 91.742897073196886, 88.307278943069093, 90.936627732905492, 7
.018963766250437, 49.133789791276826, 53.507751009532953, 48.586423555688746, -4.4730776113028883
 56.967128095484341, 49.383247263177985]
```

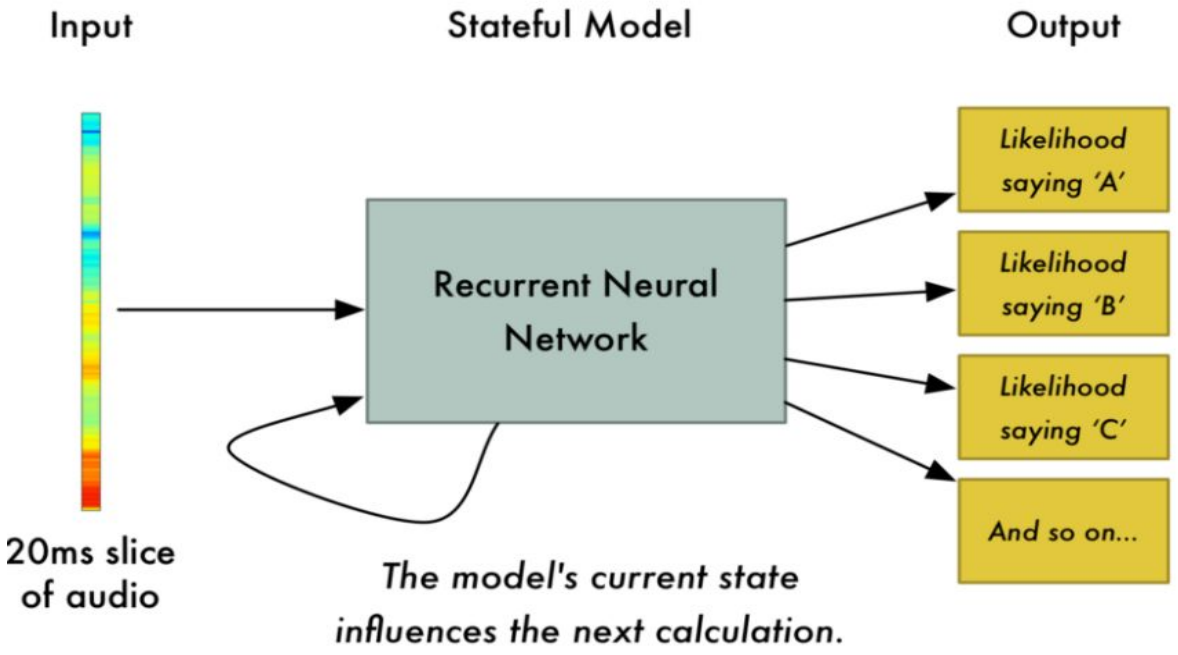
But this is a lot easier to see when you draw this as a chart:



The full spectrogram of the "hello" sound clip

Recognizing Characters from Short Sounds

Now that we have our audio in a format that's easy to process, we will feed it into a deep neural network. The input to the neural network will be 20-millisecond audio chunks. For each little audio slice, it will try to figure out the letter that corresponds to the sound currently being spoken.



Explanation of the above model

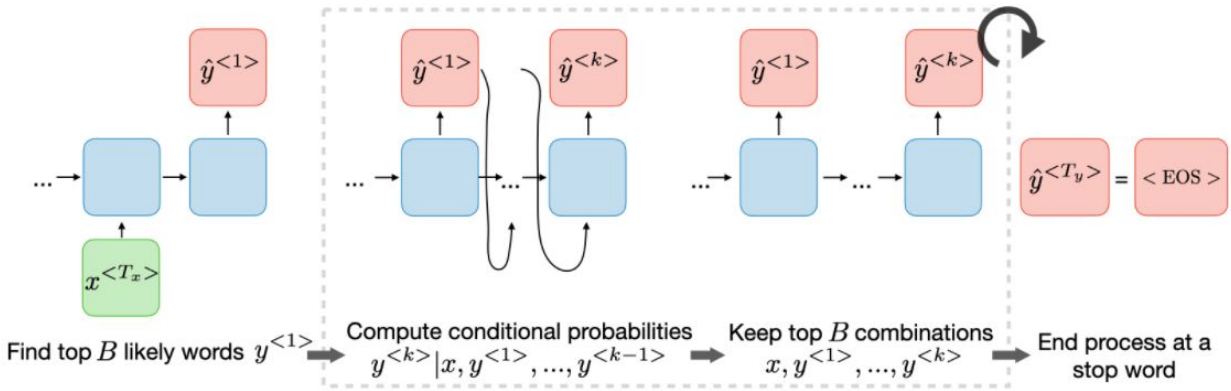
There is an audio file of 20 ms given to the recurrent neural network. RNN has many types such as one to one, one to many, many to one, and many to many but to use speech recognition we need to follow many to many rule where Tx != Ty.

The sentence has given input as y and x.

Step 1: Find top BB likely words y^{< 1 >}y

Step 2: Compute conditional probabilities y^{< k >}|x,y^{< 1 >},...,y^{< k-1>}y <k>|x,y <1>,...,y <k−1>

Step 3: Keep top BB combinations x,y^{< 1>},...,y^{< k >}x,y <1>,...,y <k>



Our neural net is predicting that one likely thing I said was “HHHEE_LL_LLLOOO”. But it also thinks that I might have said “HHHUU_LL_LLLOOO” or even “AAAUU_LL_LLLOOO”.

We need to clean up this output to remove repeating characters and blanks spaces.

That leaves us with three possible transcriptions — “Hello”, “Hullo” and “Aullo”.

That is how we can build our speech recognition model.