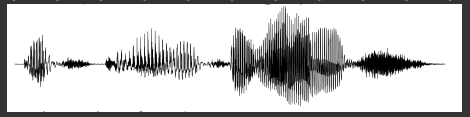
Automatic Speech Recognition

Well, the task at hand is straightforward:

Take a waveform looking like this



& map it to words being uttered → It is raining!

This process is termed as "Speech-to-Text"

But if all this was, then we would not be here (still 9 pages to go)

The raw waveform as such does not tell us anything useful. Put differently, we need features that we can input to a model so that it can spit out what was being said.

But we have Amplitude-vs-time information, so why do we need something else and can't we use waveform as an input to a Deep Learning architecture??

→ The 2 questions are separate and should not be confused. Lets address the 2nd one to begin with

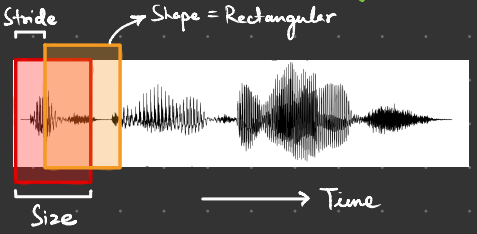
A raw waveform depicts sound on a basic level, which is nothing but vibrations propagating thru a medium. But to make a machine/model understand, we need to convert it from this physical notion to digital signal.

Now a sound disturbance carries with itself → frequency, amplitude, energy, etc. These are the properties of interest that we need to extract, which gets feed into a model.

The waveforms are just one-dimensional. What we need is 'Energy' information across various 'Frequency Bands' at a given 'Time'. Therefore mel spectrograms are what we use as an input to a speech recognition model.

Converting sound into Mel Spectrogram

- First step is to digitize the signal → **Sampling** & **Quantization**
  - Sampling → # Samples per second
  - Quantization → Storing Amplitude as 8 bit / 16 bit integers
- Next we do **Windowing** → the part of signal that is responsible for a particular fundamental phoneme will be extracted through windowing.



Frame

→ The value post windowing ⇒ $w(t) = f(t) \cdot s(t)$

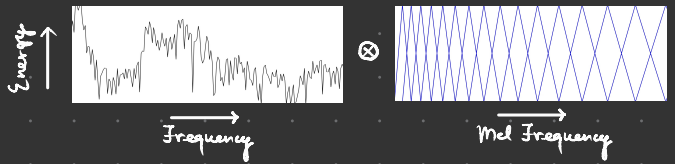
($s(t)$: signal, $f(t)$: window function)

- To extract energy from the different frequency bands we need **Discrete Fourier Transform**

  The magnitude & phase of a k^{th} Frequency component is given by → $E(k) = \sum_{t=0}^{N} w(t) \exp\left(-i \frac{2\pi}{N} kt\right)$

  $e^{i\theta} = \cos\theta + i\sin\theta$ → Euler's Formula

  N → Total number of Frequency Bands
- Finally to aid human recognition, we need to increase the weight of lower frequencies. So we transform frequency to **Mel scale** that lets us collect energies on a 'log scale'



Output of DFT

Mel Scale → $mel(k) = 1127 \ln(1 + f/700)$

Working with Mel Spectrograms

The huge data contains — Utterances
— Phoneme state labels

or → Mel Spectral vectors

Each utterance contains in its first dimension # Frames while # Frequency bands in the second dimension. The individual frame is of size 'x' ms and any adjacent frames will be 'y' ms apart.

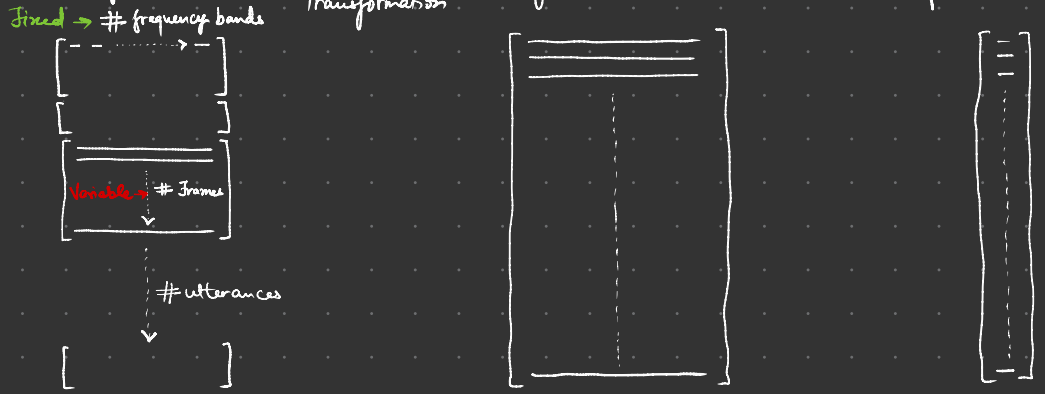
'x' ms → Frame size
'y' ms → Frame stride

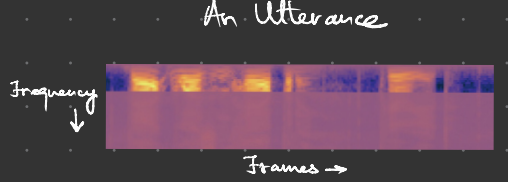
Against each utterance there is a phoneme states vector, which contains a phoneme state for every frame in that utterance. Entries in the array correspond to Energy values (log scale) in different frequency bands that vary across time.

The job is to build a model that learns for every frame a phoneme state out of the 138 classes.

- First naive model will be that takes each frame independently & learns the phoneme label against it. We'll test both MLP & ConvID against this structure of data.

Original Data —Simple Transformation→ Augmented Data —Learn→ Output Phoneme states





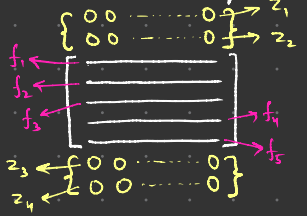
Now given we are working with Temporal Data, taking each frame on its own will only take us to a best ~25% on the accuracy scale. This learning is worse than a model giving random output (~50% accuracy).

The flaw in this plan is that the model only sees at the current frame to learn the current phoneme class.
What if our model got access to energy values in the frequency bands of previous & next frames with respect to the current frame and then learn the phoneme label {given that we need to work with MLP & Conv only} !!

So we concatenate the frames from previous & next time steps to the current frame. One thing to take care of is to not leak this context between utterances. Therefore, we need to do this step of augmentation at an utterance by utterance basis.

Lets say we choose context size = 2, then the first 2 & last 2 frames of each utterance won't have enough features post data augmentation.
To solve this issue, we pad each utterance with an array, of constant = 0, of shape (context size, # frequencies) on each side & then perform the transformation.

Lets take an example utterance



Content Transformation →

Input to model

$$\begin{bmatrix} z_1 \oplus z_2 \oplus f_1 \oplus f_2 \oplus f_3 \\ z_2 \oplus f_1 \oplus f_2 \oplus f_3 \oplus f_4 \\ f_1 \oplus f_2 \oplus f_3 \oplus f_4 \oplus f_5 \\ f_2 \oplus f_3 \oplus f_4 \oplus f_5 \oplus z_3 \\ f_3 \oplus f_4 \oplus f_5 \oplus z_3 \oplus z_4 \end{bmatrix}$$

Learn →

Output Phoneme states

$$\begin{bmatrix} - \\ - \\ - \\ - \\ - \end{bmatrix}$$

We can apply MLP architecture on this type of data

↓

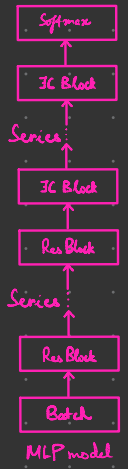
Number of Rows remain same
but the size of each row increases to → # Frequencies * (2 * context size + 1)

Another issue that we face is if the total number of frames of all utterances combined is already large then creating new features by increasing context size will increase the size of data beyond a normal system can handle. Therefore we will use TensorFlow's generator to create batches on the fly which will be called only when we do model.fit, so the whole data won't be loaded into the RAM while training. We'll also modify the iterator to augment the data based on the context size as a hyper-parameter.

Dense Residual Architecture

Fully Connected Block

Residual Block



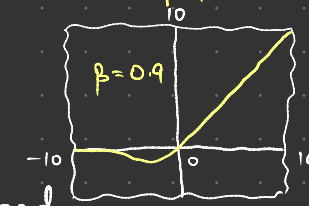
MLP model

- Fully Connected Block is made of 'Dense', 'Activation', 'Dropout' & 'BatchNorm' Layers
- Residual Block offers a 'skip connection' by joining the output of last FC Block & input to pass as the argument of next Res Block post applying a non-linear activation
- Collection of the 'Res Blocks' & then 'FC Blocks' followed by 'Softmax Layer' makes a deep MLP

can be 'concatenation' or 'summation'

SiLU is short for 'Sigmoid-weighted Linear Unit' $\quad silu(x, \beta) = x \cdot sigmoid(\beta x) = \frac{x}{1 + \exp(-\beta x)}$

↳ like a smoothed version of ReLU



→ shape = (batch_size, [2 × context_size + 1] × # frequencies)

What is Dropout ??

→ Consider an image of a human face & car. The task is to train the model to detect a face.

Imagine 3 units of hidden layer & parameters randomly initialized

Each unit learns a feature about the input image But more importantly lets say the 3rd unit learns about the mouth in a face

But what if there was a test image with the mouth not visible

The task Dropout does is it will mask the 3rd unit and force the model to think that it has not learnt about a mouth. So in the next iteration, the model makes some other unit learn about mouth or a new feature to detect face, like nose or ear. The model is then made to learn new features or call it regularization.

What is Batch Normalization ??

To set the record straight BN is not a regularization method. It is in fact an adaptive reparametrization technique to improve the process of optimisation in very deep nets. Lets understand this through a toy example

$= \hat{y} = w_L h_{L-1} = w_L(w_{L-1} h_{L-2}) = (w_L w_{L-1} \dots w_2 w_1)\, x$, consider there is an Identity activation

After one step of Backpropagation → $\hat{y} = (w_L - \alpha g_L)(w_{L-1} - \alpha g_{L-1}) \dots (w_2 - \alpha g_2)(w_1 - \alpha g_1)\, x$ → each weight follows an update

The prediction is of the form → $\hat{y} \sim (\alpha^L g^L w^{L-1} + \alpha^2 g^2 w^{L-2} + \dots + \alpha^L g^L w^0)\, x + \underbrace{w^L x}_{\text{Before BP}}$ $\quad w_i' = w_i - \alpha g_i$,

Assume the gradient to be 1 → fixed by the cost function where g_i = gradient

Now In an optimization process, we want the difference to be small ⇒ Small $\Delta\hat{y}$

$\Delta\hat{y} \sim \alpha w^{L-1} + \alpha^2 w^{L-2} + \dots + \alpha^L w^0$ ⟶ The weights of each layer when multiplied can either be very small or very large

Therefore the choice of learning rate becomes difficult to get stable updates in a deep network

Lets make few changes to the network

$\vec{\beta} \equiv (\mu, \sigma, \lambda, \beta)$

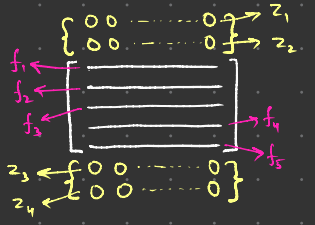
μ, σ : Fixed for every hidden layer

λ, β : Learnable

$$BN = \begin{cases} h_i' = \dfrac{h_i - \mu}{\sigma} \\ h_i = \lambda h_i' + \beta \end{cases}$$

What BatchNorm does is that it normalizes the hidden layer at first. In doing, we are making units standardized. This helps reduce the problem of simultaneous updates across layers of the deep network. But in doing the network becomes vanilla. To restore the balance, we let the layers learn its own mean & standard deviation

Instead of flattening post transformation



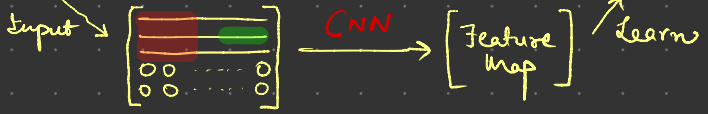
Content Transformation

Input to model

- vertical stacks $(z_1\ z_2\ f_1\ f_2\ f_3)$
- vertical stacks $(z_2\ f_1\ f_2\ f_3\ f_4)$
- vertical stacks $(f_1\ f_2\ f_3\ f_4\ f_5)$
- vertical stacks $(f_2\ f_3\ f_4\ f_5\ z_3)$
- vertical stacks $(f_3\ f_4\ f_5\ z_3\ z_4)$

Output Phoneme states

Here we can consider each stack as collection of temporal vectors and apply Conv1D or see it as an image and see that how good Conv2D performs



What the custom CNN model looks like ??

The architecture will be built using 'Subclassing' & 'Functional' approaches

- There will be an elementary 'Convolution block', comprising of a 'Conv' layer, a 'Batch Normalization' layer & a 'SeLU' layer.
- Using this a recurrent part, we will build a 'Residual block' containing series of {Convolution blocks & a 'Skip Connection'} followed by a 'MaxPooling' layer.
- Lastly, we combine series of Residual blocks & a 'GlobalAveragePooling' layer followed by the output 'dense' layer.