Audio Processing With Deep Learning

By Abhinav(1801002),
Vikas Kumar(1801194)

Supervised By Dr. Bijit kumar Das,
Dr Sudip Biswas

Assisted By Deep Barua

Department of Electronics and communication Technologies
Indian Institute of Information Technology, Guwahati

Problem Statement

Cornell Birdcall Identification

Objective: Challenge in this project is to identify which birds are calling in long recordings, given training data generated in meaningfully different contexts.

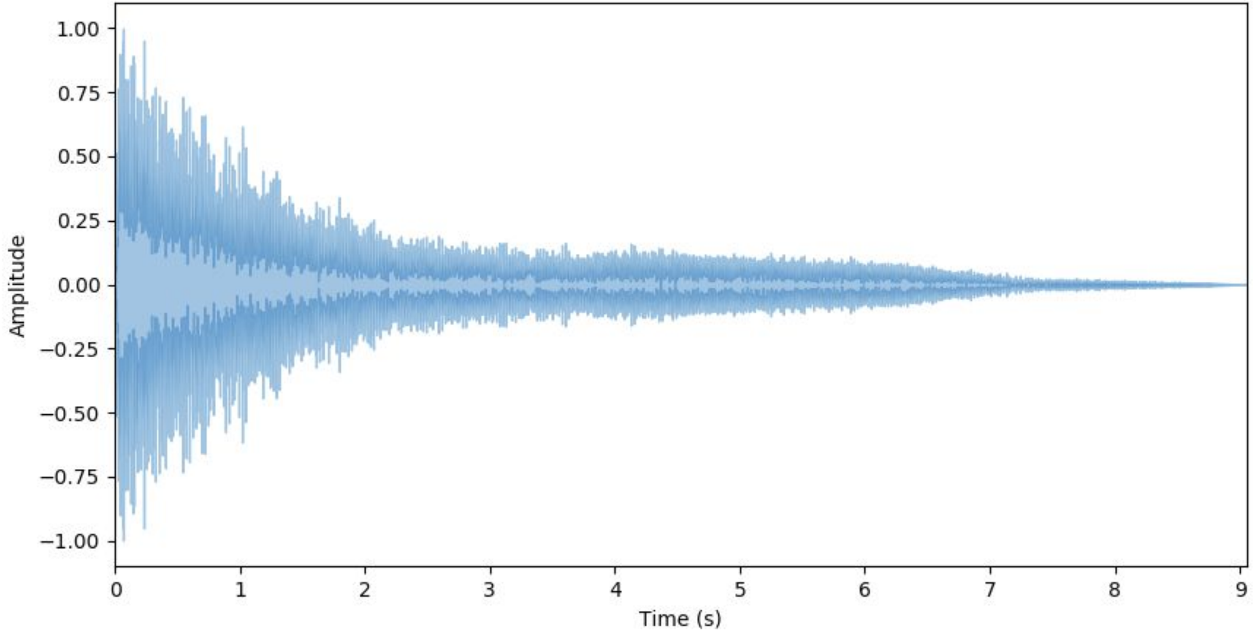
Problem Link: https://www.kaggle.com/c/birdsong-recognition/data

Code Link: https://github.com/abhinav8292/Deep_learning

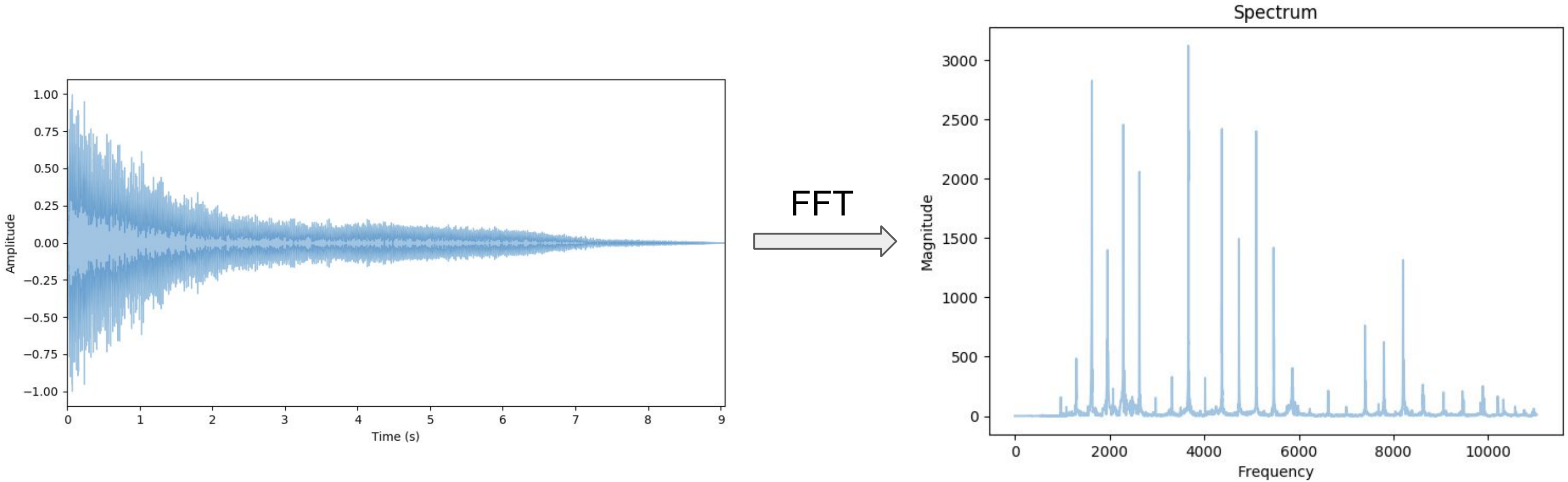
Techstack: Python, Librosa, Numpy, Tensorflow, Matplotlib, Jupyter, VsCode

Understanding audio data for deep learning

A real-world sound wave (piano key)



Fourier transform

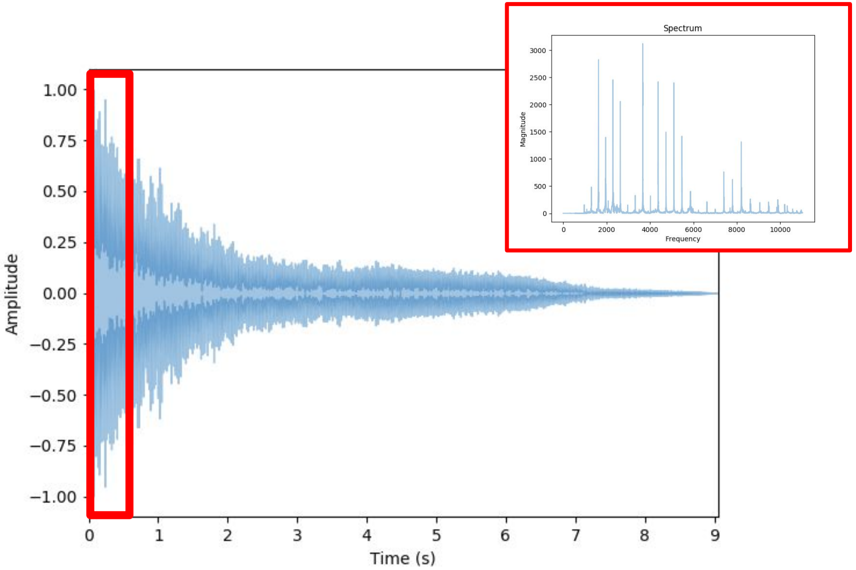


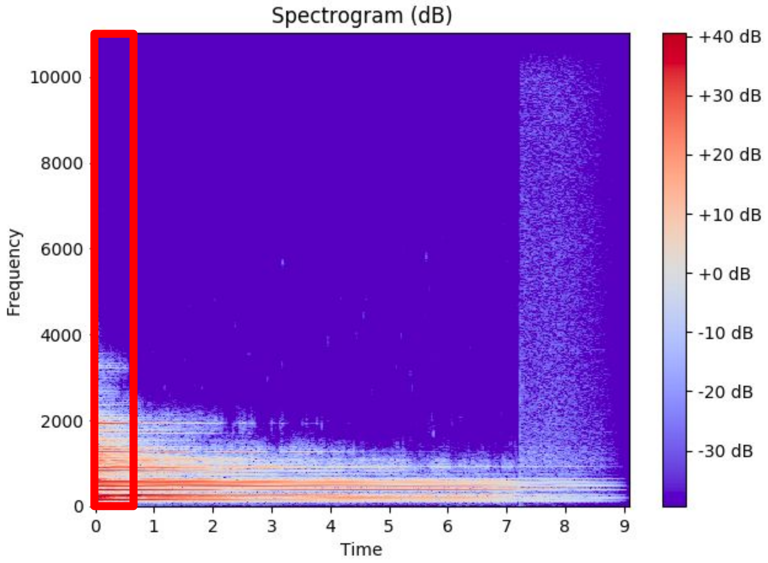
- From *time domain* to *frequency domain*
- No time information

Short Time Fourier Transform (STFT)

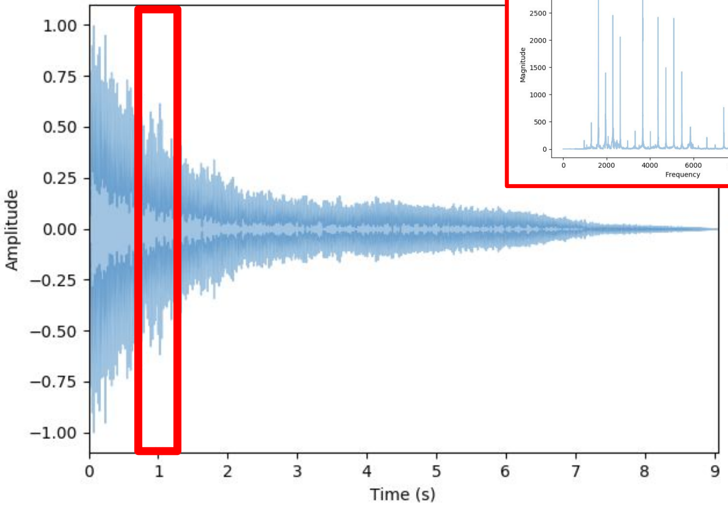
- Computes several FFT at different intervals
- Preserves time information
- Fixed frame size (e.g., 2048 samples)
- Gives a *spectrogram* (time + frequency + magnitude)

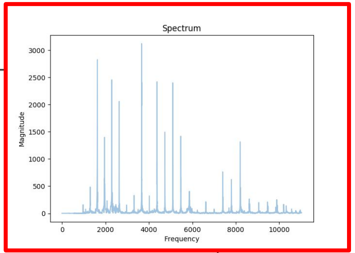
Short Time Fourier Transform (STFT)

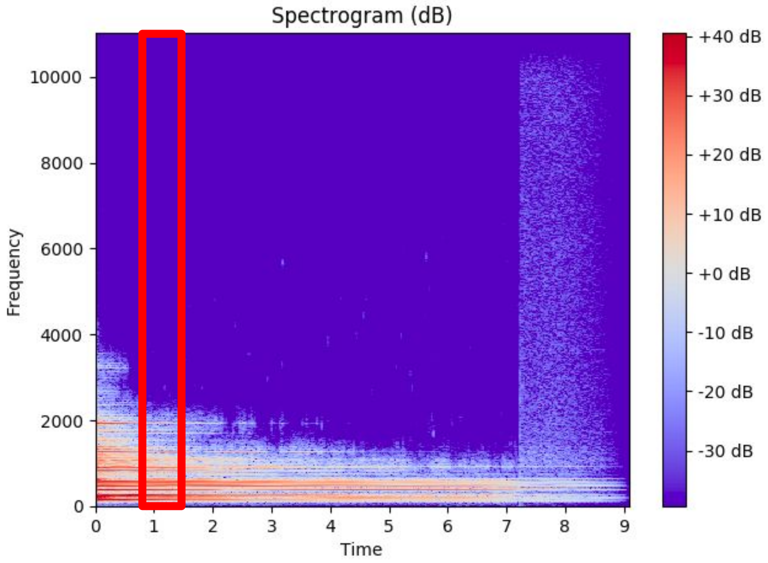




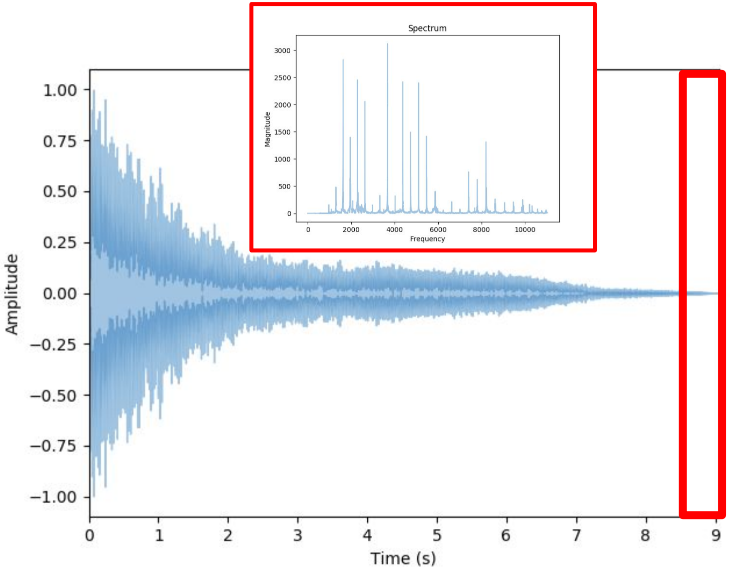
Short Time Fourier Transform (STFT)

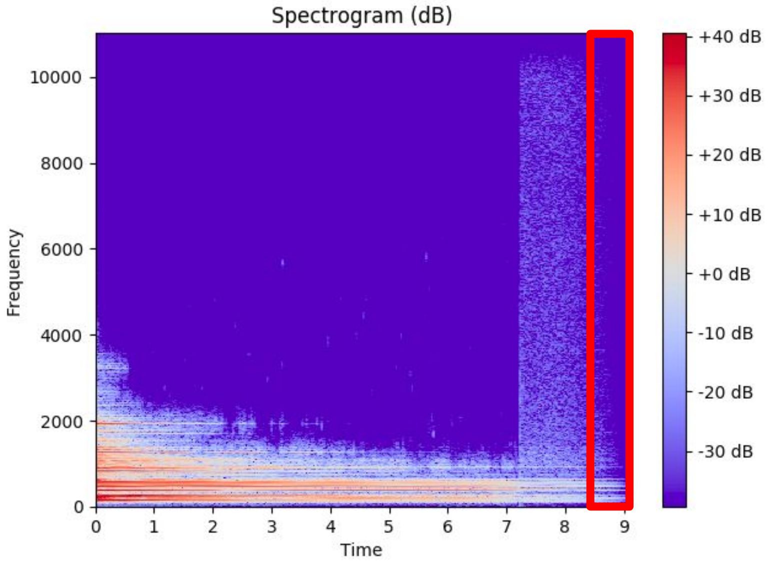




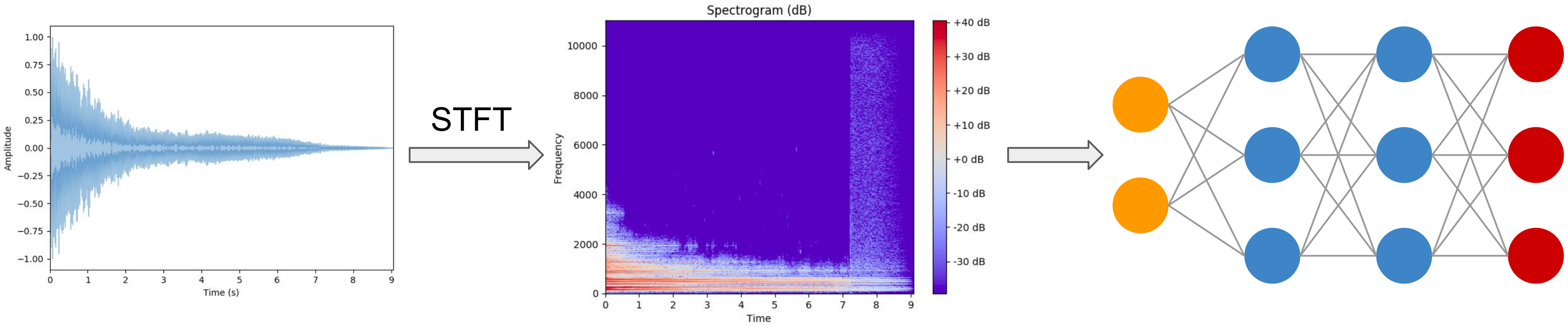


Short Time Fourier Transform (STFT)





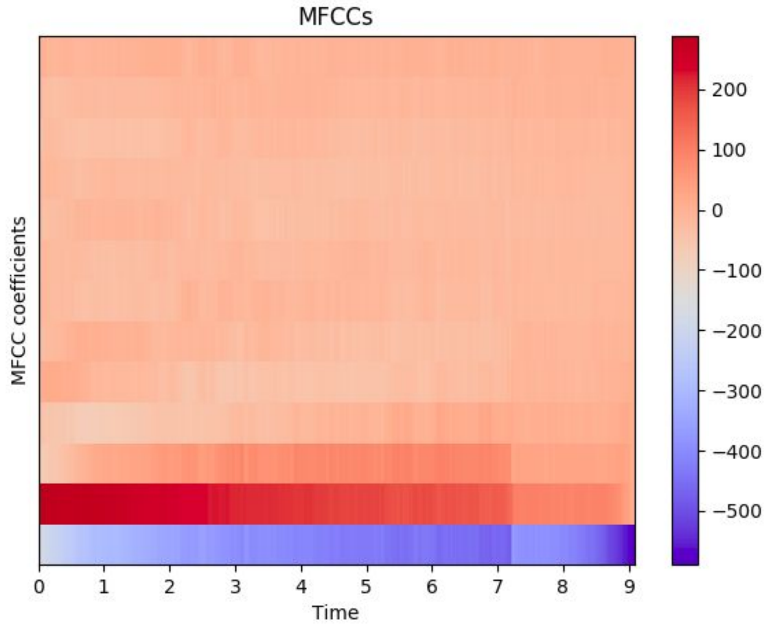
DL pre-proprocessing pipeline for audio data



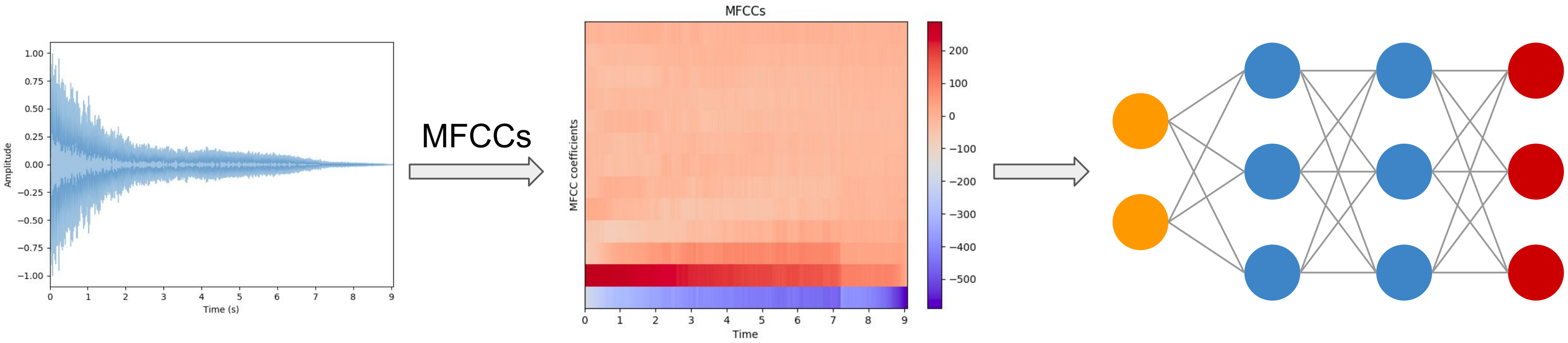
Mel Frequency Cepstral Coefficients (MFCCs)

- Capture timbral/textural aspects of sound
- Frequency domain feature
- Approximate human auditory system
- 13 to 40 coefficients
- Calculated at each frame

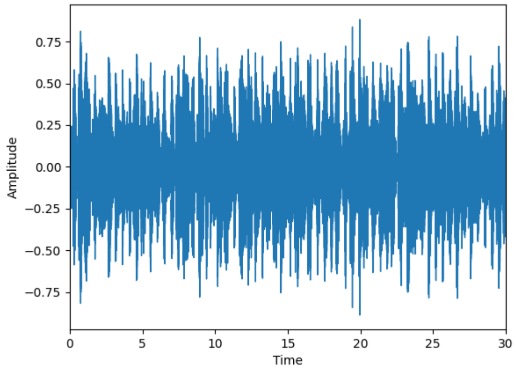
Mel Frequency Cepstral Coefficients (MFCCs)



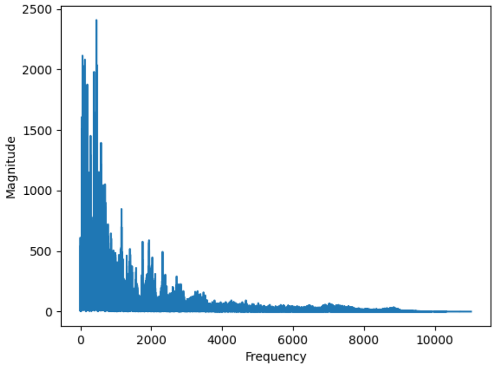
DL pre-proprocessing pipeline for audio data



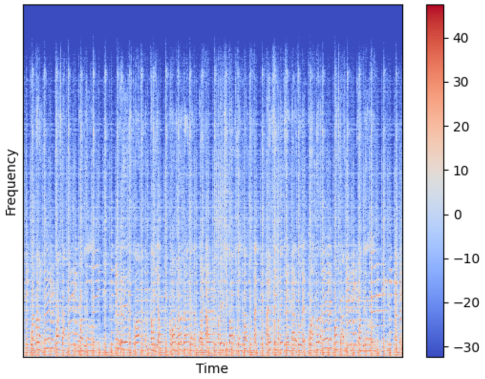
Birdsong Audio Features



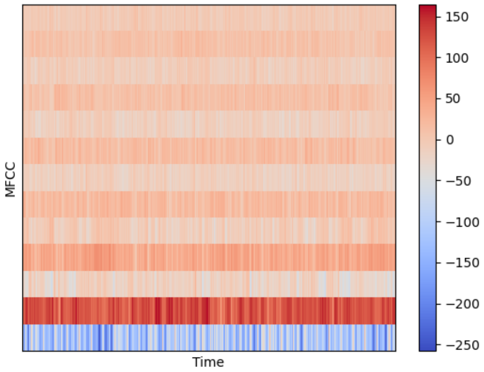
Audio Waveform



FFT

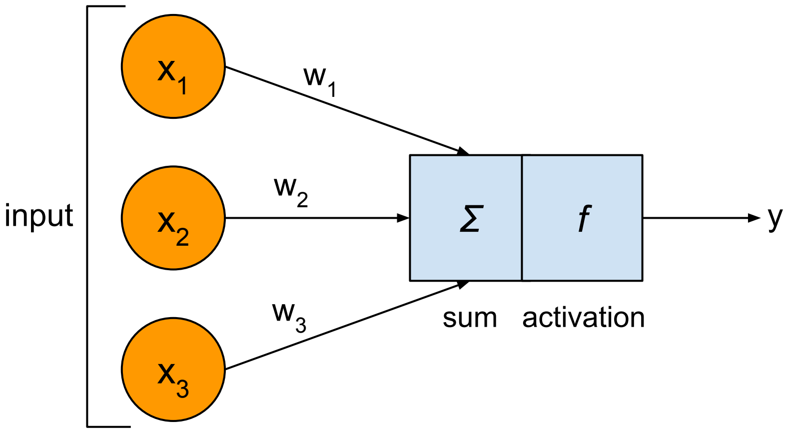


Spectrogram



MFCC

The artificial neuron



$$h = \sum_i x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$y = f(h) = f(x_1 w_1 + x_2 w_2 + x_3 w_3)$$

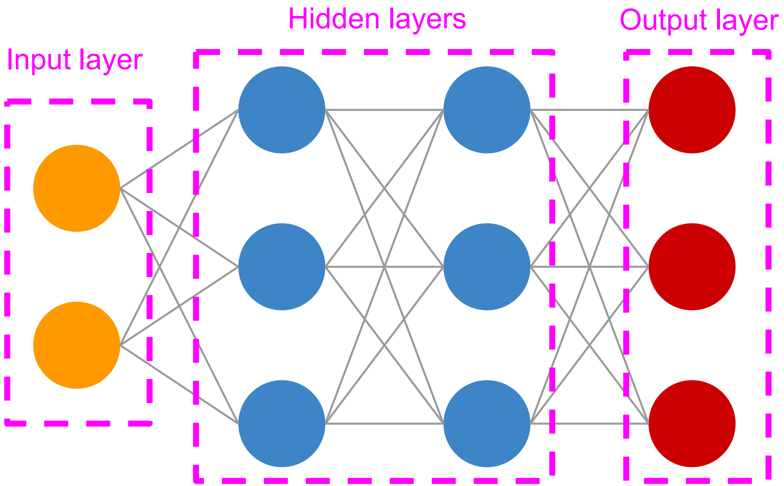
$$y = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_2 + x_3 w_3)}}$$

Why is a neural network needed?

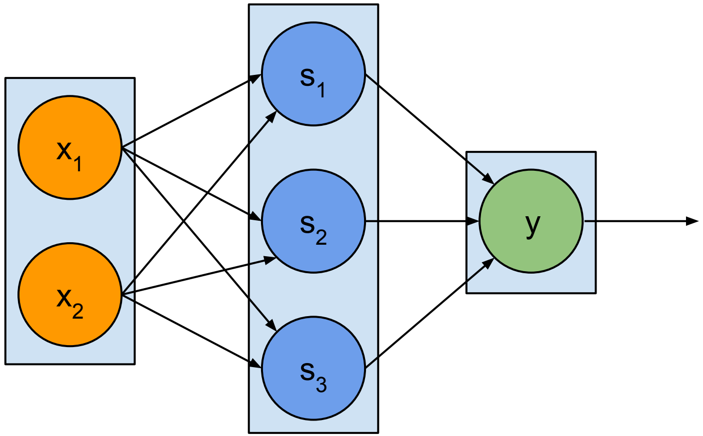
- A single neuron works for linear problems
- Real-world problems are complex
- ANNs can reproduce highly non-linear functions

The components of an artificial neural network (ANN)

- Neurons
- Input, hidden, output layers
- Weighted connections
- Activation function



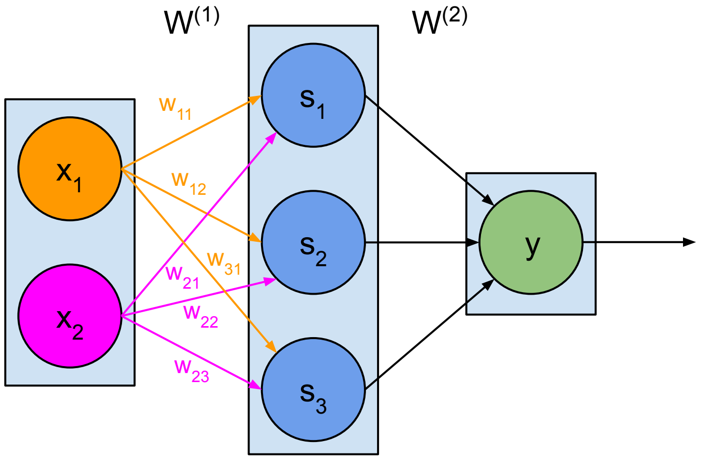
The multilayer perceptron (MLP)



Computation in MLP

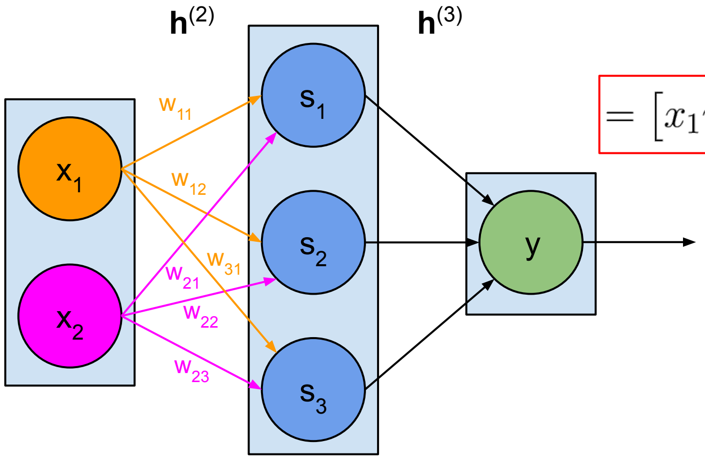
- Weights
- Net inputs (sum of weighted inputs)
- Activations (output of neurons to next layer)

Weights



$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

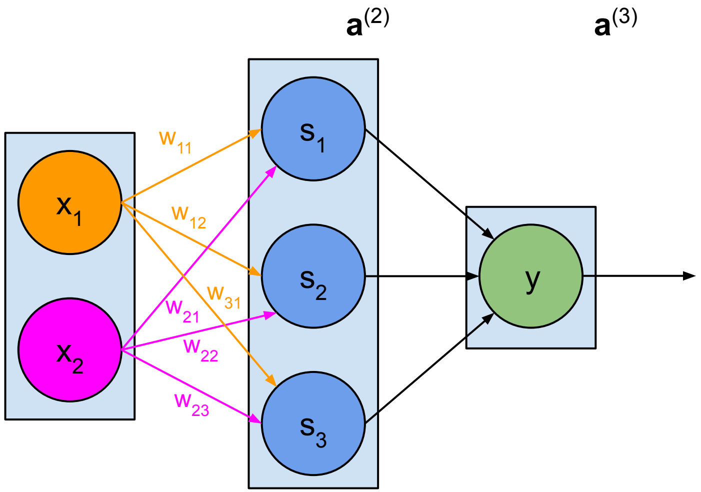
Net input



$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

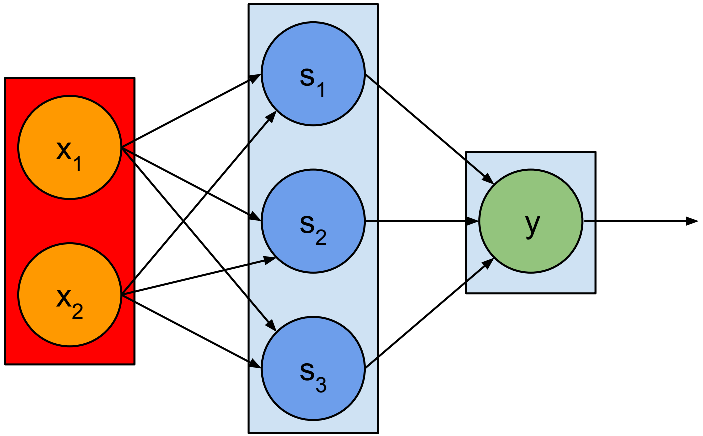
$$\boxed{= \begin{bmatrix} x_1 w_{11} + x_2 w_{21} & x_1 w_{12} + x_2 w_{22} & x_1 w_{13} + x_2 w_{23} \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}}$$

Activation



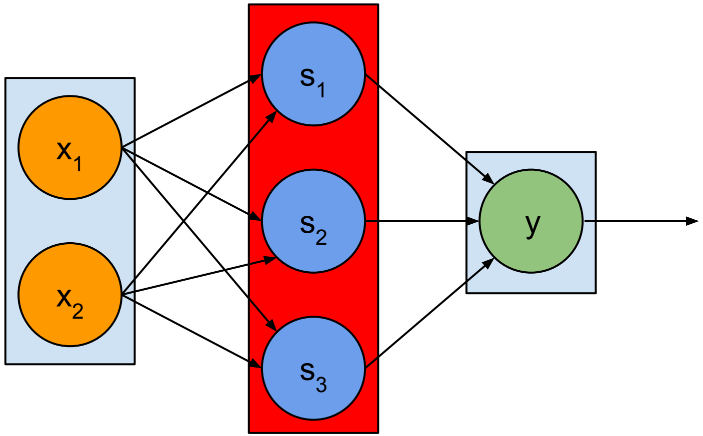
$$\mathbf{a}^{(2)} = f(\mathbf{h}^{(2)})$$

Computation in MLP (1st layer)



$\mathbf{x}$

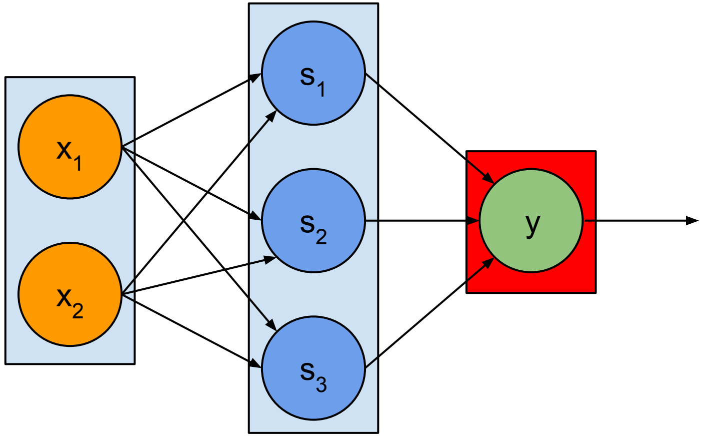
Computation in MLP (2nd layer)



$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$

$$\mathbf{a}^{(2)} = f(\mathbf{h}^{(2)})$$

Computation in MLP (3d layer)



$$\mathbf{h}^{(3)} = \mathbf{a}^{(\mathbf{2})} W^{(2)}$$

$$y = f(\mathbf{h}^{(3)})$$

Results using MLP

Train Accuracy: 32.27%
Test Accuracy: 29.24%

```
Epoch 30/50
219/219 [==============================] - 5s 21ms/step - loss: 3.0097 - accuracy: 0.2564 - val_loss: 3.0020 - val_accuracy: 0.2423
Epoch 31/50
219/219 [==============================] - 5s 21ms/step - loss: 2.9821 - accuracy: 0.2604 - val_loss: 2.9528 - val_accuracy: 0.2543
Epoch 32/50
219/219 [==============================] - 5s 21ms/step - loss: 2.9545 - accuracy: 0.2665 - val_loss: 2.9373 - val_accuracy: 0.2500
Epoch 33/50
219/219 [==============================] - 5s 21ms/step - loss: 2.9208 - accuracy: 0.2691 - val_loss: 2.8952 - val_accuracy: 0.2607
Epoch 34/50
219/219 [==============================] - 5s 21ms/step - loss: 2.8836 - accuracy: 0.2741 - val_loss: 2.8738 - val_accuracy: 0.2664
Epoch 35/50
219/219 [==============================] - 5s 22ms/step - loss: 2.8497 - accuracy: 0.2835 - val_loss: 2.8267 - val_accuracy: 0.2710
Epoch 36/50
219/219 [==============================] - 5s 21ms/step - loss: 2.8143 - accuracy: 0.2810 - val_loss: 2.8008 - val_accuracy: 0.2680
Epoch 37/50
219/219 [==============================] - 5s 21ms/step - loss: 2.7743 - accuracy: 0.2847 - val_loss: 2.7811 - val_accuracy: 0.2657
Epoch 38/50
219/219 [==============================] - 5s 21ms/step - loss: 2.7386 - accuracy: 0.2910 - val_loss: 2.7171 - val_accuracy: 0.2787
Epoch 39/50
219/219 [==============================] - 5s 21ms/step - loss: 2.7143 - accuracy: 0.2927 - val_loss: 2.6848 - val_accuracy: 0.2810
Epoch 40/50
219/219 [==============================] - 5s 22ms/step - loss: 2.6727 - accuracy: 0.2931 - val_loss: 2.6710 - val_accuracy: 0.2804
Epoch 41/50
219/219 [==============================] - 5s 22ms/step - loss: 2.6408 - accuracy: 0.2990 - val_loss: 2.6394 - val_accuracy: 0.2804
Epoch 42/50
219/219 [==============================] - 5s 21ms/step - loss: 2.6056 - accuracy: 0.3063 - val_loss: 2.6051 - val_accuracy: 0.2891
Epoch 43/50
219/219 [==============================] - 5s 21ms/step - loss: 2.5809 - accuracy: 0.3034 - val_loss: 2.5807 - val_accuracy: 0.2881
Epoch 44/50
219/219 [==============================] - 5s 21ms/step - loss: 2.5459 - accuracy: 0.3059 - val_loss: 2.5477 - val_accuracy: 0.2951
Epoch 45/50
219/219 [==============================] - 5s 22ms/step - loss: 2.5165 - accuracy: 0.3107 - val_loss: 2.5258 - val_accuracy: 0.2887
Epoch 46/50
219/219 [==============================] - 5s 22ms/step - loss: 2.4941 - accuracy: 0.3136 - val_loss: 2.4958 - val_accuracy: 0.2931
Epoch 47/50
219/219 [==============================] - 5s 21ms/step - loss: 2.4538 - accuracy: 0.3139 - val_loss: 2.4737 - val_accuracy: 0.2921
Epoch 48/50
219/219 [==============================] - 5s 22ms/step - loss: 2.4252 - accuracy: 0.3200 - val_loss: 2.4484 - val_accuracy: 0.2967
Epoch 49/50
219/219 [==============================] - 5s 21ms/step - loss: 2.4004 - accuracy: 0.3159 - val_loss: 2.4226 - val_accuracy: 0.2994
Epoch 50/50
219/219 [==============================] - 5s 22ms/step - loss: 2.3664 - accuracy: 0.3227 - val_loss: 2.4033 - val_accuracy: 0.2924
```

CNNs

- Mainly used for processing images
- Perform better than multilayer perceptron
- Less parameters than dense layers

Intuition

- Image data is structured
  - Edges, shapes
  - Translation invariance
  - Scale invariance
- CNN emulates human vision system
- Components of a CNN learn to extract different features

CNN components

- Convolution
- Pooling

Convolution

- *Kernel* = grid of weights
- Kernel is “applied” to the image
- Traditionally used in image processing

1	2	-1
0	1	2
-2	1	0

Convolution

5	2	3	1	2	4
2	4	1	0	3	1
5	1	0	2	8	3
0	2	1	5	2	4
2	7	0	0	2	1
1	3	2	8	7	0

Convolution: Zero padding

Image

0	0	0	0	0	0	0	0
0	5	2	3	1	2	4	0
0	2	4	1	0	3	1	0
0	5	1	0	2	8	3	0
0	0	2	1	5	2	4	0
0	2	7	0	0	2	1	0
0	1	3	2	8	7	0	0
0	0	0	0	0	0	0	0

Kernel

1	0	0
2	1	0
1	0	-1

Output

-1					
	18	10	-3	5	
	12	?	?	?	
	?	?	?	?	
	?	?	?	?	

Convolution

Image

0	0	0	0	0	0	0	0
0	5	2	3	1	2	4	0
0	2	4	1	0	3	1	0
0	5	1	0	2	8	3	0
0	0	2	1	5	2	4	0
0	2	7	0	0	2	1	0
0	1	3	2	8	7	0	0
0	0	0	0	0	0	0	0

Kernel

1	0	0
2	1	0
1	0	-1

Output

-1	?	?	?	?	?
?	18	10	-3	5	?
?	12	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?

Kernels

- Feature detectors
- Kernels are learned

Oblique line detector

1	0	0
0	1	0
0	0	1

Vertical line detector

0	1	0
0	1	0
0	1	0

Architectural decisions for convolution

- Grid size
- Stride
- Depth
- Number of kernels

Grid size

- # of pixels for height/width
- Odd numbers

3 by 3

1	2	9
1	6	5
2	2	3

Grid size

- # of pixels for height/width
- Odd numbers

5 by 5

1	2	9	8	7
1	6	5	0	0
2	2	3	1	0
1	1	-3	0	-1
1	-2	2	2	3

Stride

- Step size used for sliding kernel on image
- Indicated in pixels

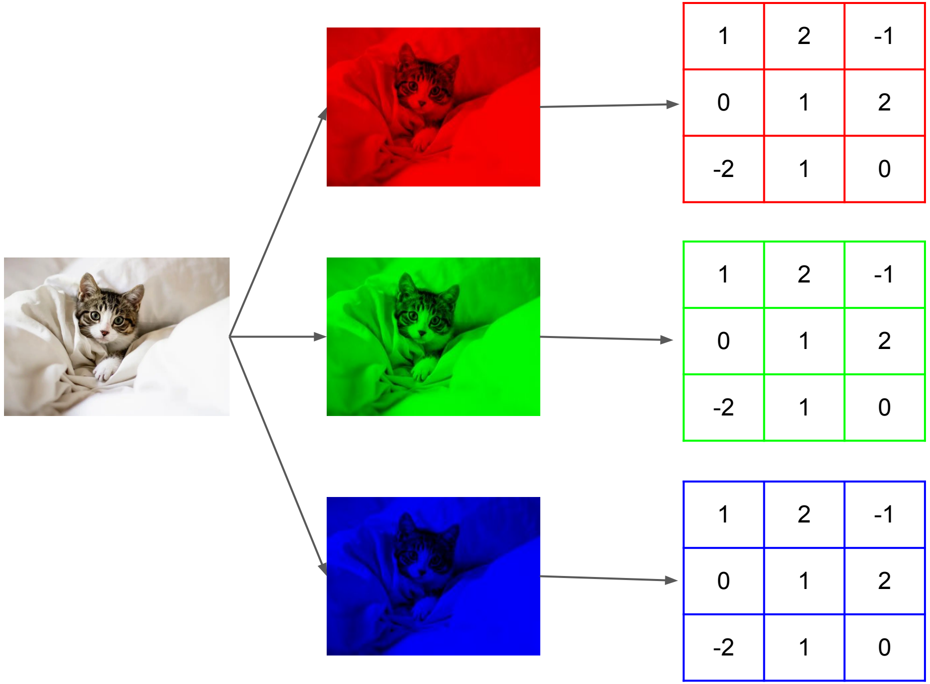
5	2	3	1	2	4
2	4	1	0	3	1
5	1	0	2	8	3
0	2	1	5	2	4
2	7	0	0	2	1
1	3	2	8	7	0

Stride

- Step size used for sliding kernel on image
- Indicated in pixels

5	2	3	1	2	4
2	4	1	0	3	1
5	1	0	2	8	3
0	2	1	5	2	4
2	7	0	0	2	1
1	3	2	8	7	0

Depth



Kernel = 3 x 3 x 3

weights = 27

of kernels

- A conv layer has multiple kernels
- Each kernel outputs a single 2D array
- Output from a layer has as many 2d arrays as # kernels

Pooling

- Downsample the image
- Overlaying grid on image
- Max/average pooling
- No parameters

Pooling settings

- Grid size
- Stride
- Type (e.g., max, average)

Max pooling (2x2, stride 2)

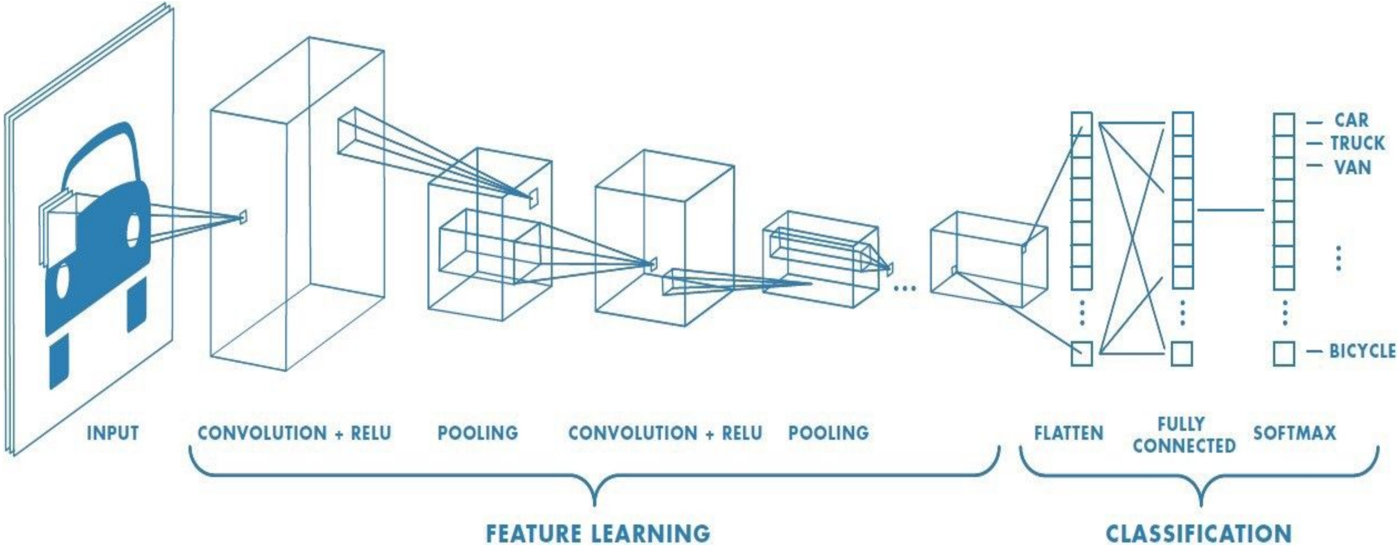
Input

-1	2	0	2
3	18	10	-3
2	12	5	2
1	3	7	4

Output

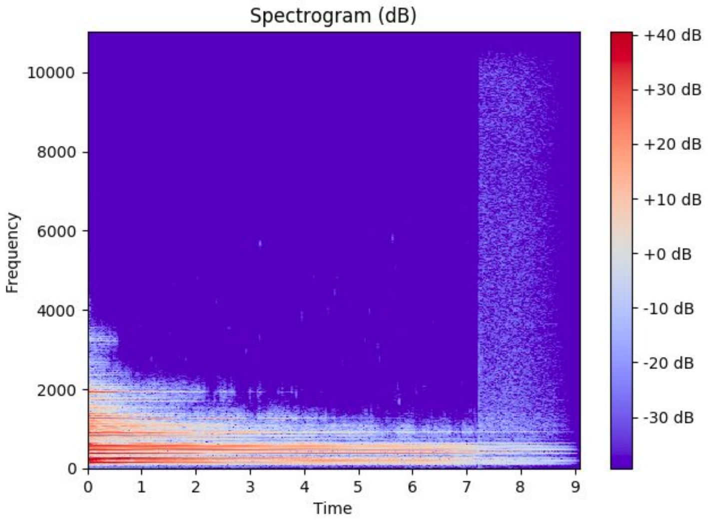
10	18
12	7

CNN architecture



How does convolution/pooling apply to audio?

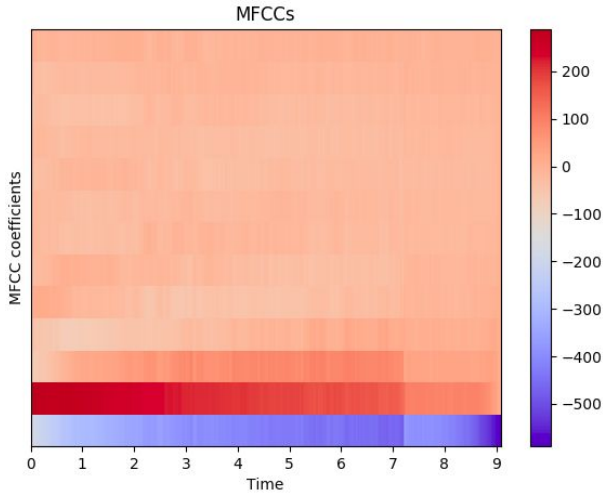
- Spectrogram/MFCC = image
- Time, frequency = x, y
- Amplitude = pixel value



Preparing MFCCs for a CNN

- 13 MFCCs
- Hop length = 512 samples
- # samples in audio file = 51200

Data shape = 100 x 13 x 1



Results Using CNN

Train Accuracy: 74.29%
Test Accuracy: 71.21%%

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Epoch 10/30
188/188 [==============================] - 6s 35ms/step - loss: 1.2037 - accuracy: 0.5635 - val_loss: 1.1246 - val_accuracy: 0.6021
Epoch 11/30
188/188 [==============================] - 6s 34ms/step - loss: 1.1689 - accuracy: 0.5790 - val_loss: 1.0918 - val_accuracy: 0.6075
Epoch 12/30
188/188 [==============================] - 6s 34ms/step - loss: 1.1256 - accuracy: 0.5981 - val_loss: 1.0733 - val_accuracy: 0.6222
Epoch 13/30
188/188 [==============================] - 6s 33ms/step - loss: 1.1000 - accuracy: 0.6006 - val_loss: 1.0493 - val_accuracy: 0.6235
Epoch 14/30
188/188 [==============================] - 6s 34ms/step - loss: 1.0530 - accuracy: 0.6216 - val_loss: 1.0325 - val_accuracy: 0.6375
Epoch 15/30
188/188 [==============================] - 6s 34ms/step - loss: 1.0505 - accuracy: 0.6266 - val_loss: 1.0265 - val_accuracy: 0.6328
Epoch 16/30
188/188 [==============================] - 6s 34ms/step - loss: 1.0009 - accuracy: 0.6385 - val_loss: 1.0051 - val_accuracy: 0.6449
Epoch 17/30
188/188 [==============================] - 6s 34ms/step - loss: 0.9702 - accuracy: 0.6515 - val_loss: 0.9883 - val_accuracy: 0.6529
Epoch 18/30
188/188 [==============================] - 7s 37ms/step - loss: 0.9548 - accuracy: 0.6688 - val_loss: 0.9621 - val_accuracy: 0.6582
Epoch 19/30
188/188 [==============================] - 6s 33ms/step - loss: 0.9304 - accuracy: 0.6738 - val_loss: 0.9608 - val_accuracy: 0.6602
Epoch 20/30
188/188 [==============================] - 6s 34ms/step - loss: 0.9057 - accuracy: 0.6757 - val_loss: 0.9404 - val_accuracy: 0.6702
Epoch 21/30
188/188 [==============================] - 6s 33ms/step - loss: 0.8945 - accuracy: 0.6874 - val_loss: 0.9596 - val_accuracy: 0.6595
Epoch 22/30
188/188 [==============================] - 6s 33ms/step - loss: 0.8668 - accuracy: 0.6954 - val_loss: 0.9220 - val_accuracy: 0.6689
Epoch 23/30
188/188 [==============================] - 6s 34ms/step - loss: 0.8635 - accuracy: 0.6980 - val_loss: 0.8998 - val_accuracy: 0.6816
Epoch 24/30
188/188 [==============================] - 6s 34ms/step - loss: 0.8404 - accuracy: 0.6990 - val_loss: 0.9145 - val_accuracy: 0.6789
Epoch 25/30
188/188 [==============================] - 6s 34ms/step - loss: 0.8233 - accuracy: 0.7096 - val_loss: 0.9015 - val_accuracy: 0.6876
Epoch 26/30
188/188 [==============================] - 6s 33ms/step - loss: 0.7988 - accuracy: 0.7134 - val_loss: 0.8712 - val_accuracy: 0.6943
Epoch 27/30
188/188 [==============================] - 6s 33ms/step - loss: 0.7670 - accuracy: 0.7276 - val_loss: 0.8711 - val_accuracy: 0.6949
Epoch 28/30
188/188 [==============================] - 6s 33ms/step - loss: 0.7701 - accuracy: 0.7311 - val_loss: 0.8691 - val_accuracy: 0.6943
Epoch 29/30
188/188 [==============================] - 6s 33ms/step - loss: 0.7580 - accuracy: 0.7328 - val_loss: 0.8772 - val_accuracy: 0.6876
Epoch 30/30
188/188 [==============================] - 6s 33ms/step - loss: 0.7398 - accuracy: 0.7429 - val_loss: 0.8612 - val_accuracy: 0.6916
79/79 - 1s - loss: 0.8288 - accuracy: 0.7121
Accuracy on test set is: 0.7120544910430908
Expected index: 4, Predicted index, [4]
```

Observation

Got 29.24% using MLP
Got 71.21% using CNN
CNN gave around 42% more accuracy than MLP

Refrences

Problem link: https://www.kaggle.com/c/birdsong-recognition/overview

Learn Data Science: https://www.youtube.com/watch?v=YJFnBPhTi0Y&t=2825s

Analytics Vidya: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

viso.ai: https://viso.ai/deep-learning/deep-neural-network-three-popular-types/