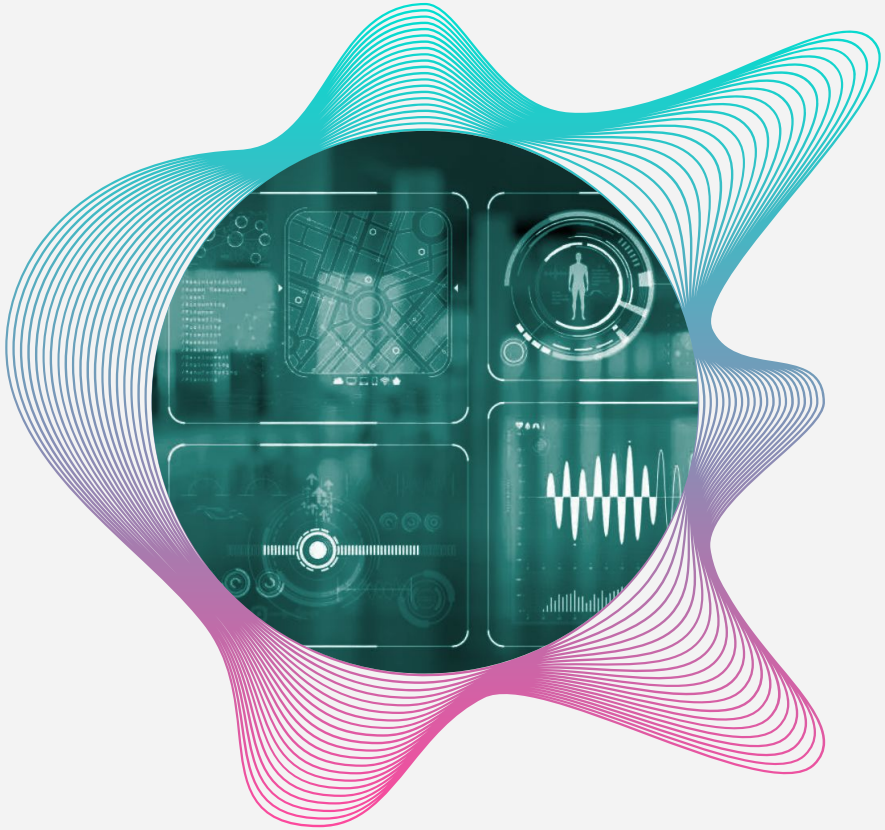


# Speech Emotion Recognition

Catherine Sanso

# Classifying audio data by emotion type

## 01 Introduction

CREMA-D Dataset

## 02 Modeling

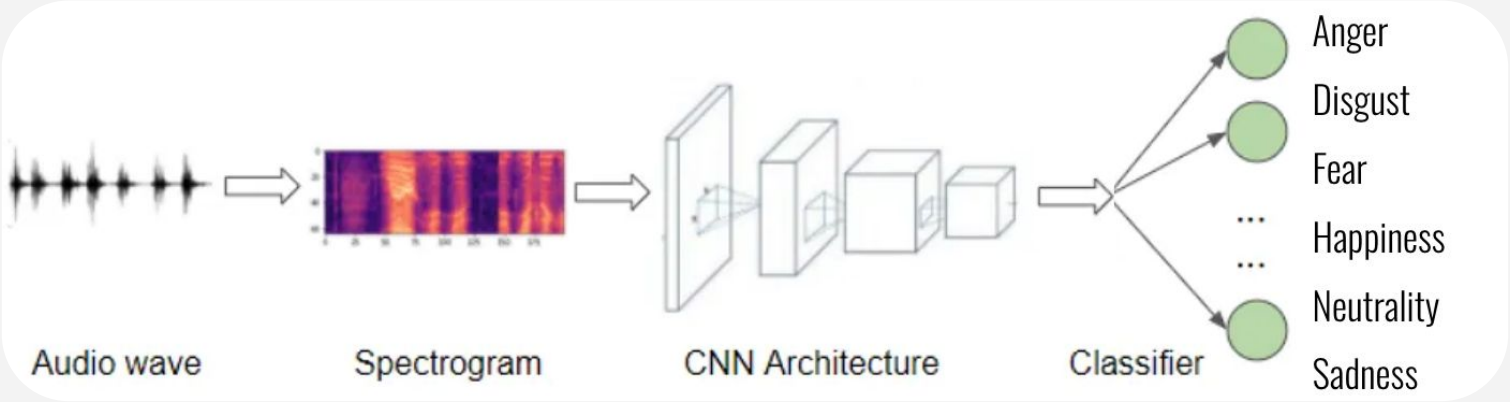
Waveplots, Spectograms, & CNNs

## 03 Analysis

Model Performance

## 04 Discussion

Interpretation & Next Steps



# Speech Emotion Recognition: Uses

- Human-Computer Interaction (HCI)
- Consumer Sentiment
- Fraud Detection
- Financial market analysis

# Emotion Types & CREMA-D

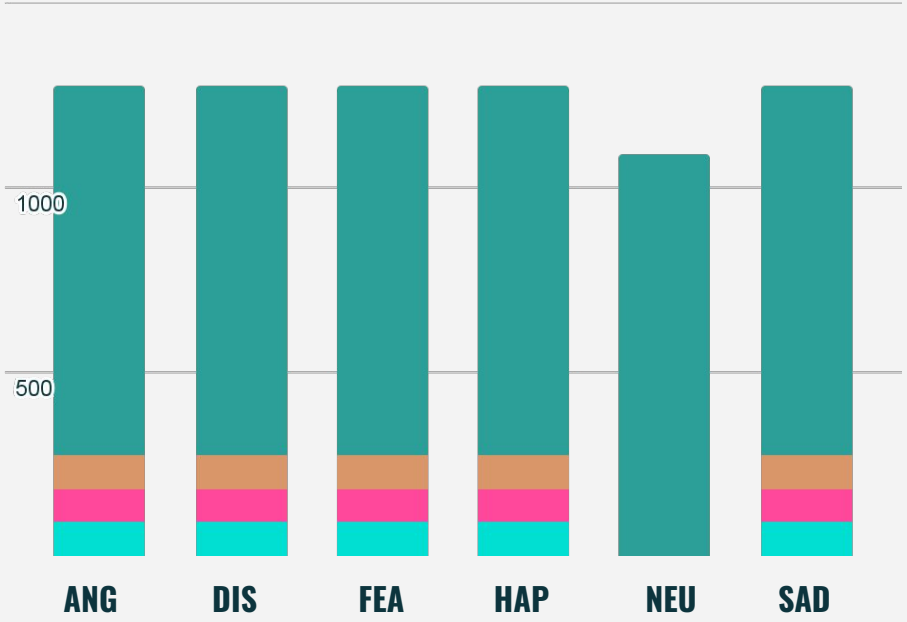
Anger  Happiness

Disgust  Neutrality

Fear  Sadness

Intensity Levels

# Distribution of Emotions by Intensity Level




n = 7,442

## NEU

Neutral intensity only

**1,087**

▼ 14%

## ALL OTHERS

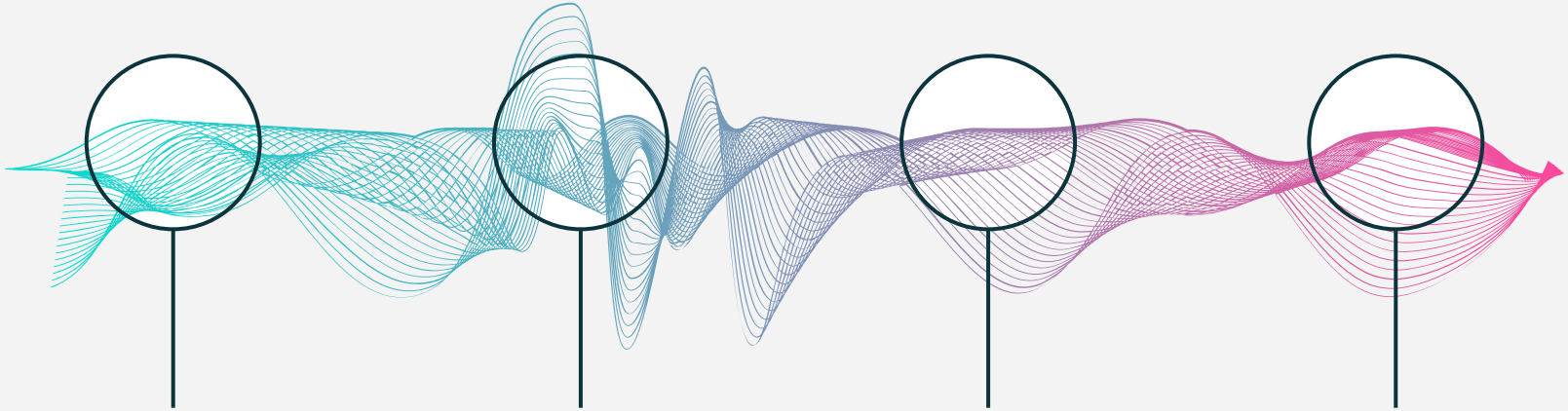
No quantity change between all other categories

**1,271**

▶ 0%

# An audio picture is worth a thousand words

# Data Processing & Augmentation



.wav file

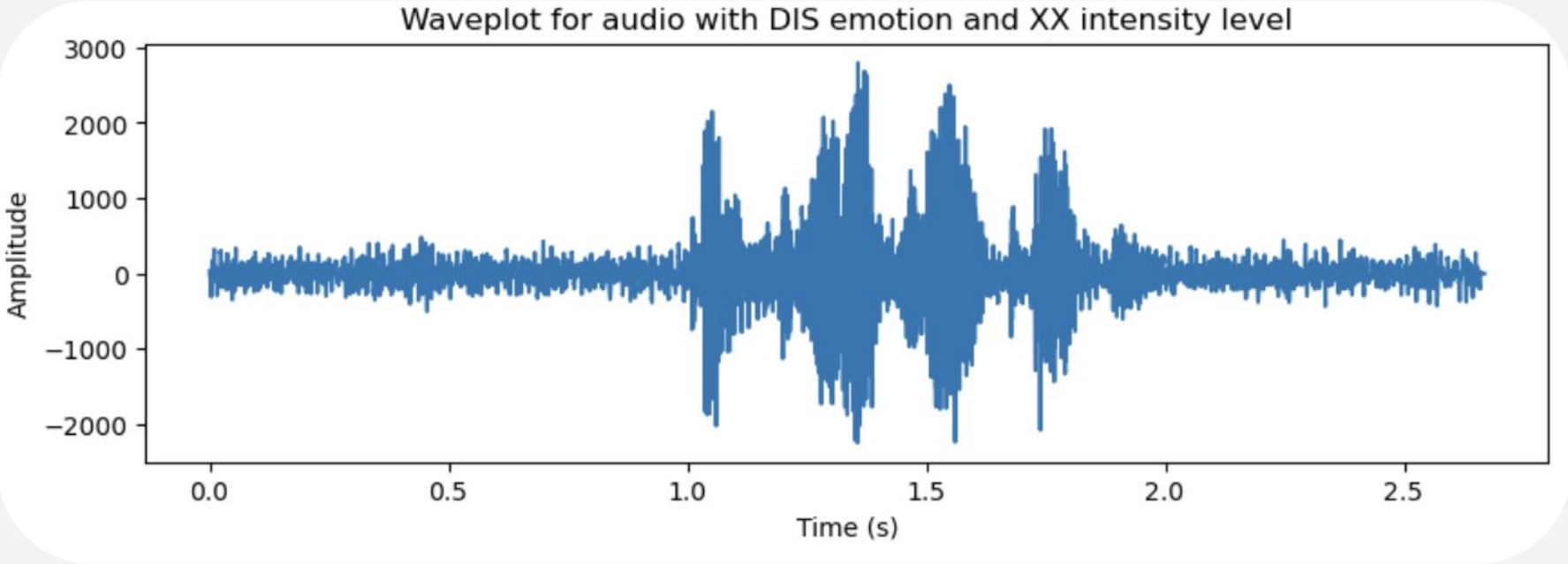
Waveplot

Spectogram

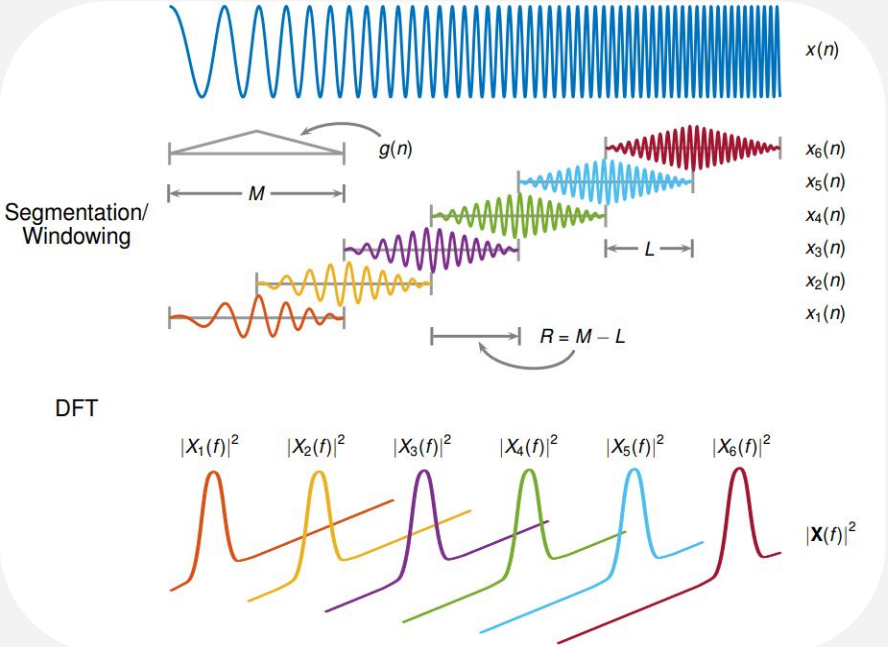
## Data Augmentation

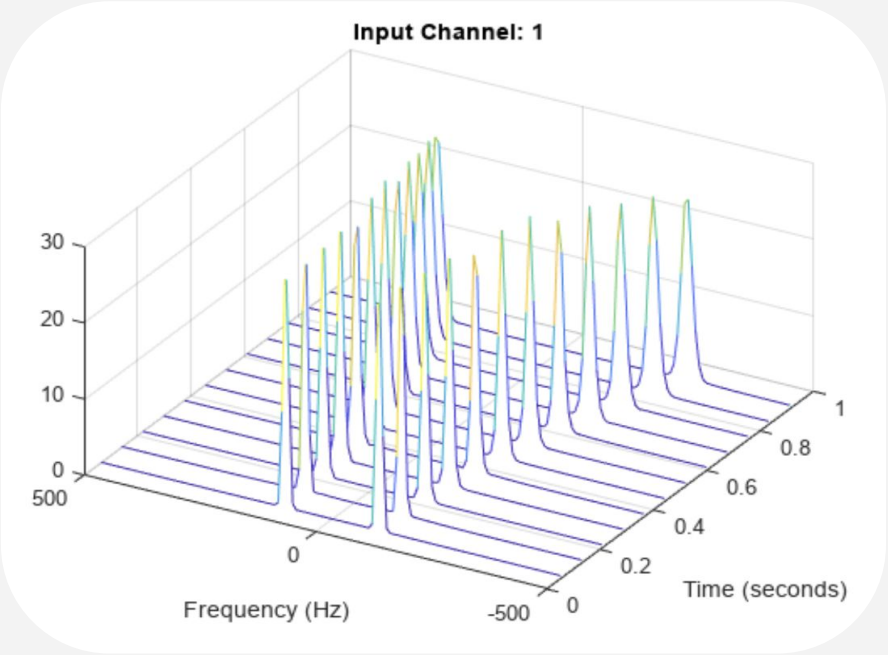
- Noise
- Preemphasis
- Pitch-Shifting
- Speed-Shifting

# Waveplot

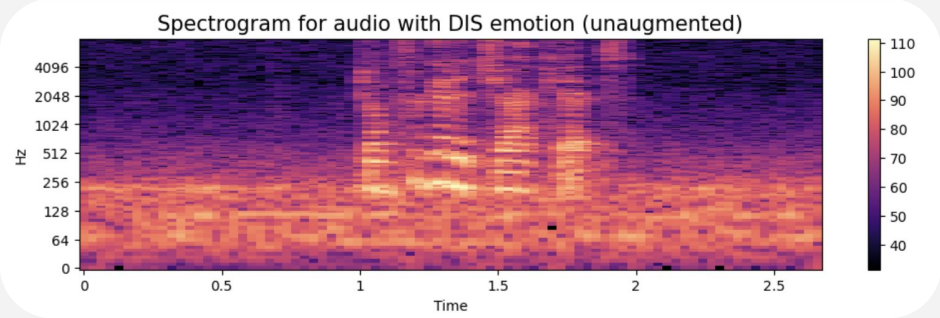

Waveplot for audio with DIS emotion and XX intensity level
Amplitude
3000
2000
1000
0
−1000
−2000
0.0
0.5
1.0
1.5
2.0
2.5
Time (s)

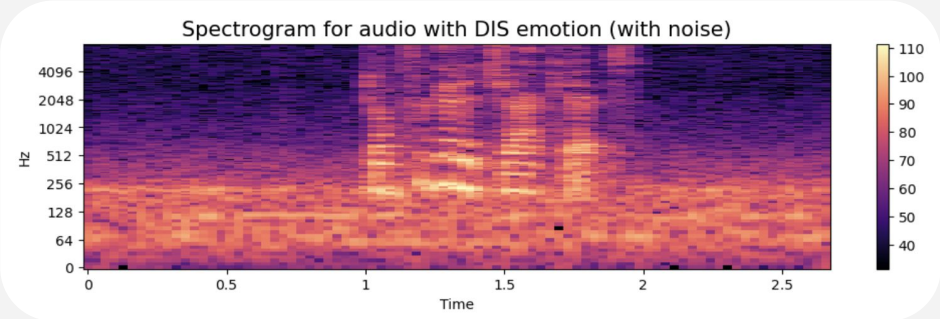
# Short Term Fourier Transform

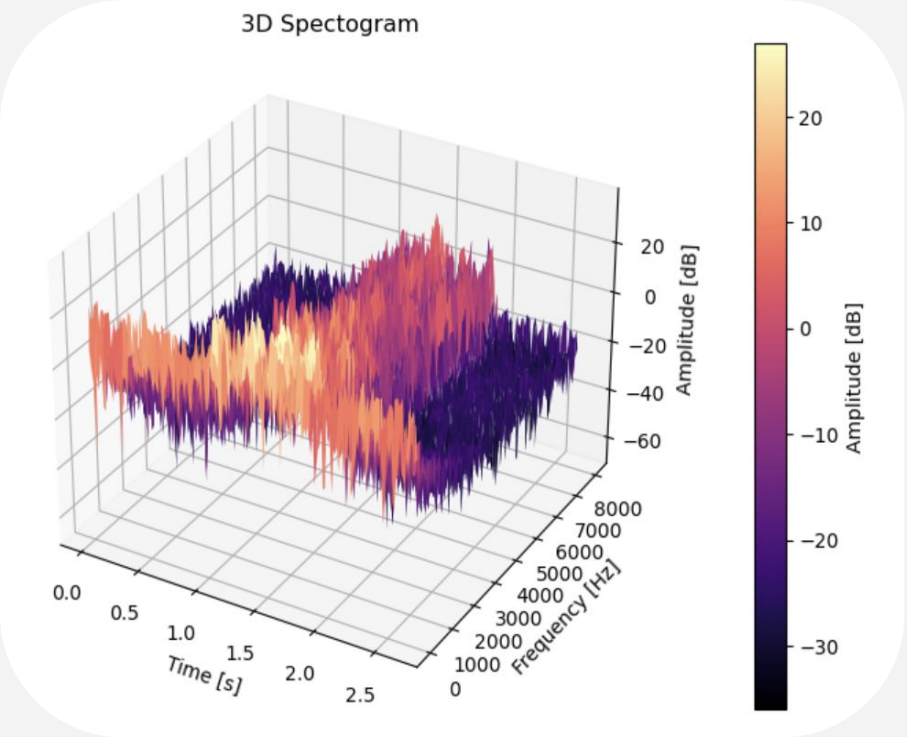


# Spectograms



```python
def plot_spectogram(filename, display_title = False, display_plot = False):
  # Locate the row with the specified filename
  selected_row = df[df['filename'] == filename].iloc[0]

  # Extract information from the selected row
  data = selected_row['audio_data'].astype(np.float32)  # Convert data to float32
  sr = selected_row['sample_rate']
  emotion = selected_row['emotion']

  augmentation_types = ["none", 'noise', 'preemphasis', "pitch", 'speed']

  for augmentation in augmentation_types:
      if augmentation == 'none':
          augmented_data = data
          augmentation_label = 'unaugmented'
      elif augmentation == 'noise':
          noise_level = 0.01  # Adjust noise level as desired
          noise = np.random.normal(0, scale=noise_level, size=len(data))
          augmented_data = data + noise
          augmentation_label = 'with noise'
      elif augmentation == 'preemphasis':
          augmented_data = librosa.effects.preemphasis(data)
          augmentation_label = 'preemphasized'
      elif augmentation == 'pitch':
          n_steps = 2  # Adjust pitch shift amount as desired
          augmented_data = librosa.effects.pitch_shift(data, sr=sr, n_steps=n_steps)
          augmentation_label = 'pitch-shifted'
      elif augmentation == 'speed':
          rate = 1.2  # Adjust speed change rate as desired
          augmented_data = librosa.effects.time_stretch(data, rate=rate)
          augmentation_label = 'speed-changed'

      # Calculate the short-term Fourier transform for the data
      X = librosa.stft(augmented_data)
      Xdb = librosa.amplitude_to_db(abs(X))

      # Create and display the spectrogram
      plt.figure(figsize=(12, 3))
      librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
      plt.colorbar()
```

librosa

# Sample Code



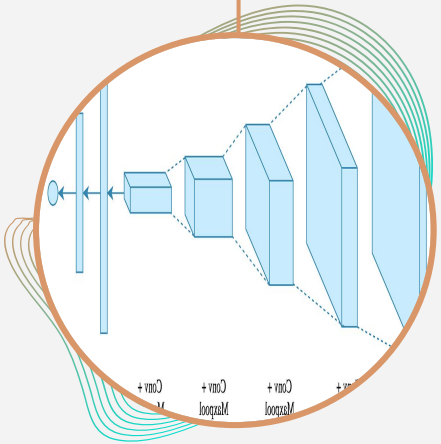
```python
# Visualizing the 3D Spectogram

# Function to plot the 3D spectrogram
def plot_3d_spectrogram(audio_data, sample_rate):
    f, t, Sxx = signal.spectrogram(audio_data, fs=sample_rate)

    fig = plt.figure(figsize=(10, 6))
    ax = fig.add_subplot(111, projection='3d')

    T, F = np.meshgrid(t, f)

    surf = ax.plot_surface(T, F, 10 * np.log10(Sxx), cmap='magma')
    ax.set_xlabel('Time [s]')
    ax.set_ylabel('Frequency [Hz]')
    ax.set_zlabel('Amplitude [dB]')

    # Add a color bar (vertical scale) to the right
    cbar = fig.colorbar(surf, ax=ax, pad=0.1, aspect=20)
    cbar.set_label('Amplitude [dB]')

    plt.title('3D Spectrogram')
    plt.show()

# Load the audio data and sample rate from the specified .wav file
sample_rate, audio_data = wavfile.read(sample_wav_file)

# Plot the 3D spectrogram
plot_3d_spectrogram(audio_data, sample_rate)
```

# CNN Model Creation

## 01

## Self-built CNN

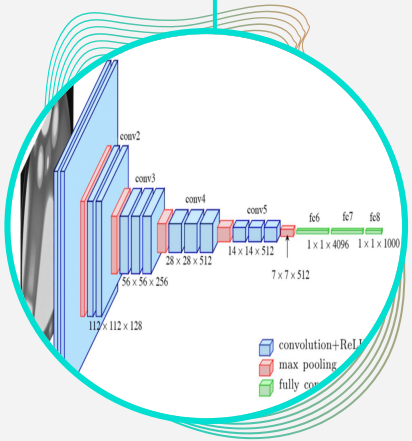
Discretion on convolution layers and maxpool operations



## 02

## VGG16 with Transfer Learning

Specified architecture trained on ImageNet data

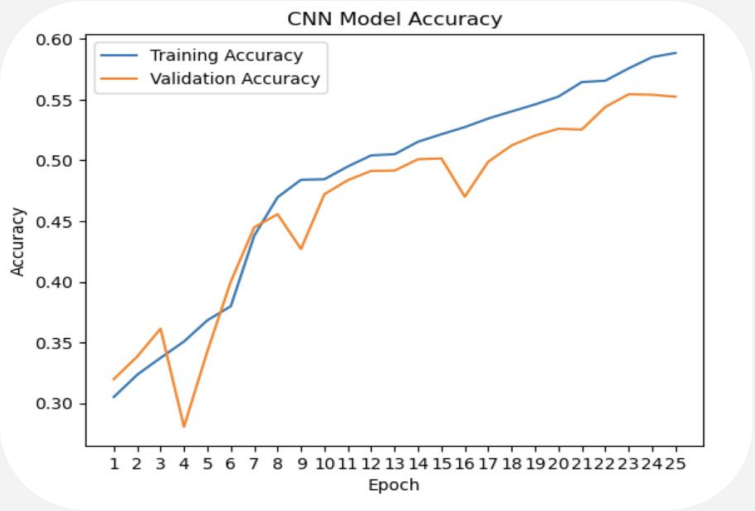


# CNN Performance Metric: Accuracy

## Self-built CNN

Training Set | Validation Set

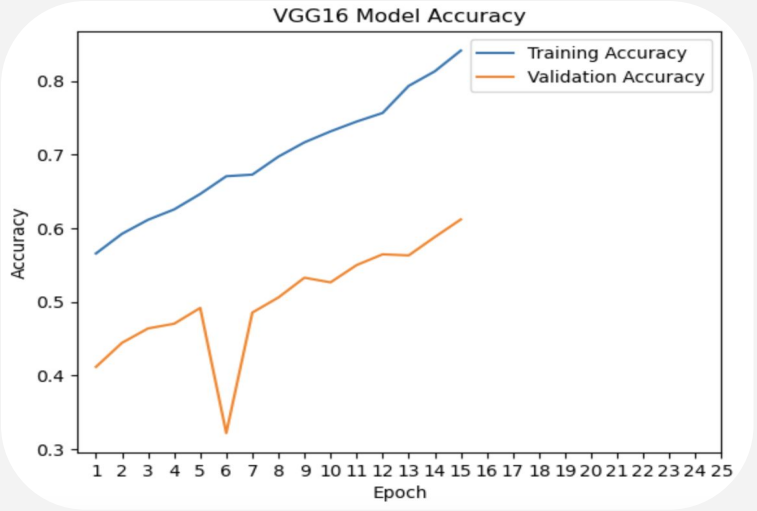
59% 55%




## VGG16

Training Set | Validation Set

99% 93%



# Recommendations for Further Study

## Additional Datasets

RAVDESS, SAVEE, TESS

## Data Augmentation

Add further perturbations

## Audio-specific CNNs

YAMNet, VGGish, UrbanSound8K, ESC-10

## Ensemble Learning

CNN + K-Means Clustering

# Questions?

Thank you for attending!