Speech Emotion Recognition

Name	**ID**
Ranime Ahmed Elsayed Shehata	21010531
Habiba Tarek Ramadan Ali	21010445
Karene Antoine Nassif Guirguis	21010969

Problem Statement:

The goal of this project is to develop a system that can classify emotions from human speech using the CREMA-D dataset. By analyzing audio signals, we aim to classify them into one of six emotions: Angry, Disgust, Fear, Happy, Neutral, and Sad.

Step 1: Download Dataset:

We began by downloading the CREMA-D dataset, which consists of 7442 audio clips recorded by 91 actors (48 male, 43 female). Each filename contains metadata that includes the emotion label, actor ID, and sentence ID. This structured naming enabled us to directly extract the emotion labels from the filenames.

Step 2: Generating Feature Space:

For each audio file, we extracted key features to capture various aspects of the speech signal:

- **Zero Crossing Rate (ZCR)**: Calculated as the number of times the audio waveform crosses the zero amplitude axis. It helps distinguish between voiced and unvoiced sounds.
- **Energy**: Computed as the sum of squares of the signal amplitude. This feature helps capture emotional intensity (e.g., angry or excited speech typically has higher energy).
- **Mel Spectrogram**: Obtained by transforming the audio signal into the frequency domain using the Short-Time Fourier Transform (STFT), then mapping it onto the mel scale. This results in a time-frequency representation that is more aligned with how humans perceive sound.

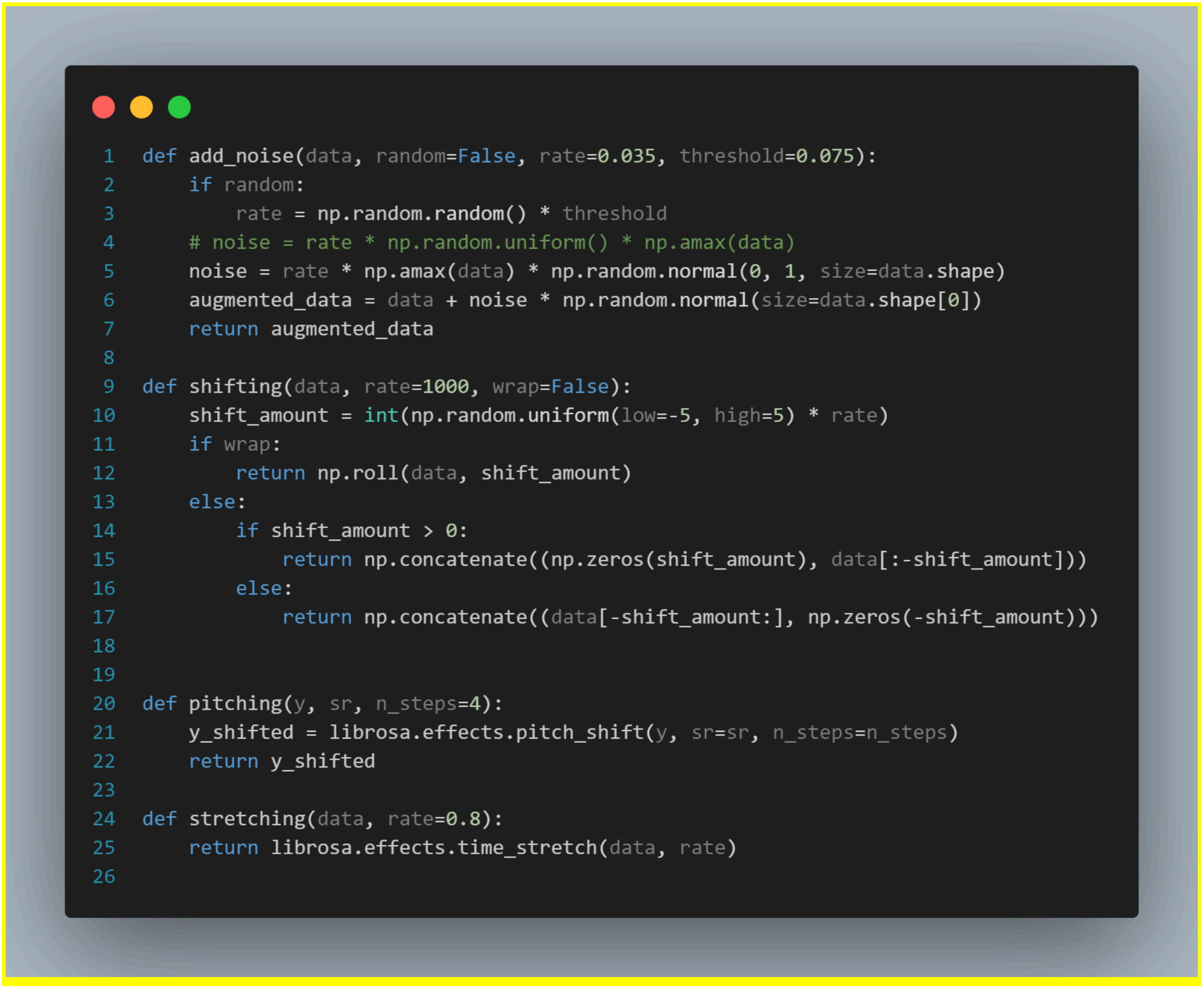
To enhance generalization and reduce overfitting, we applied the following **data augmentation** techniques to every audio sample:

- **Noise Addition**: Injected low-level Gaussian noise.
- **Time Shifting**: Shifted the audio waveform forward or backward.
- **Pitch Shifting**: Raised or lowered the pitch using a pitch scaling factor.
- **Time Stretching**: Stretched or compressed the audio duration without altering pitch.

Each original sample was augmented once, effectively doubling the dataset size.

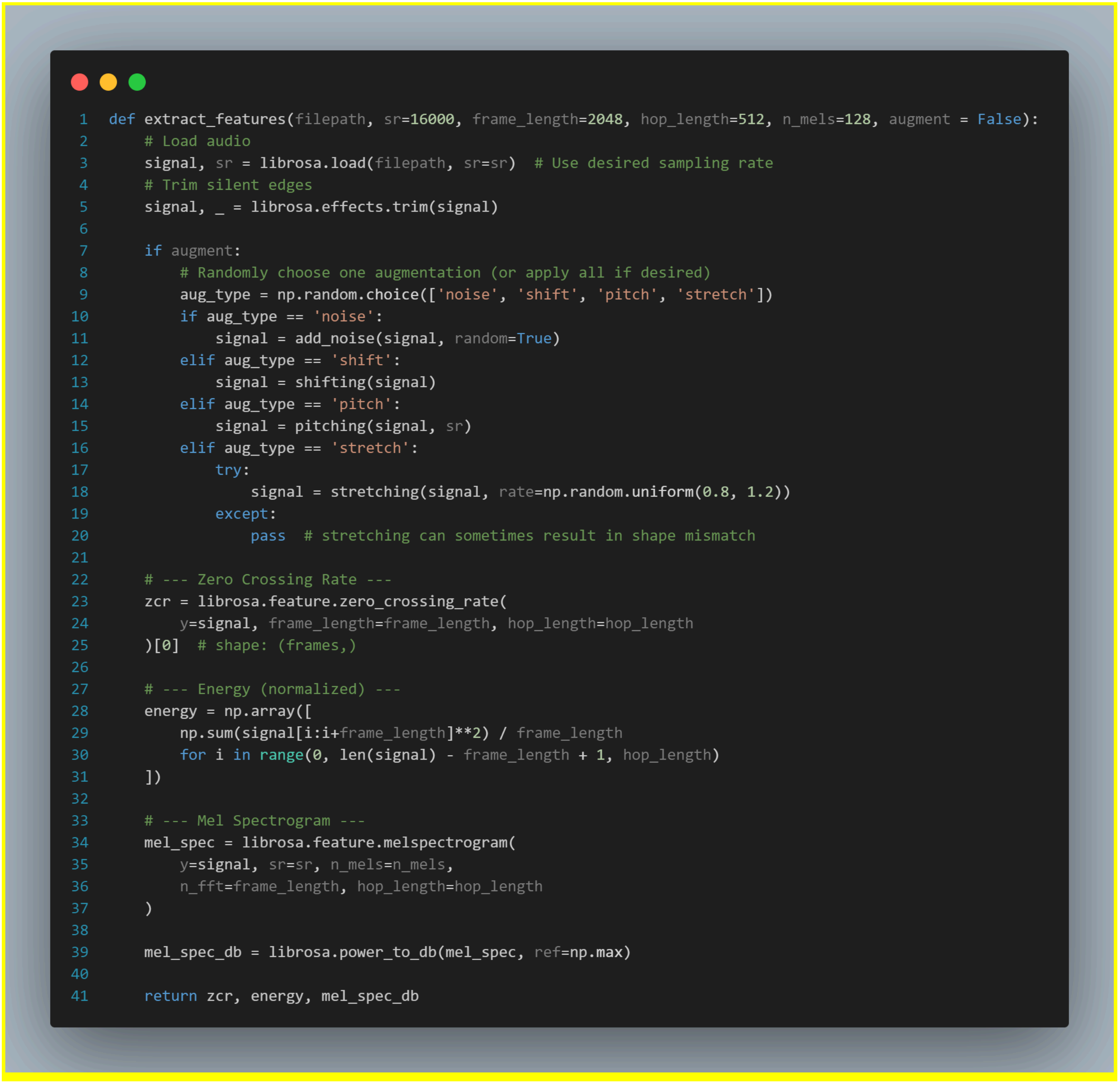
We applied data augmentation only for 1D Model. However, we tried for 2D model, but we faced some RAM limit problem and couldn’t proceed further.

Data Augmentation:



```python
def add_noise(data, random=False, rate=0.035, threshold=0.075):
    if random:
        rate = np.random.random() * threshold
    # noise = rate * np.random.uniform() * np.amax(data)
    noise = rate * np.amax(data) * np.random.normal(0, 1, size=data.shape)
    augmented_data = data + noise * np.random.normal(size=data.shape[0])
    return augmented_data

def shifting(data, rate=1000, wrap=False):
    shift_amount = int(np.random.uniform(low=-5, high=5) * rate)
    if wrap:
        return np.roll(data, shift_amount)
    else:
        if shift_amount > 0:
            return np.concatenate((np.zeros(shift_amount), data[:-shift_amount]))
        else:
            return np.concatenate((data[-shift_amount:], np.zeros(-shift_amount)))


def pitching(y, sr, n_steps=4):
    y_shifted = librosa.effects.pitch_shift(y, sr=sr, n_steps=n_steps)
    return y_shifted

def stretching(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate)

```

Feature Extraction:

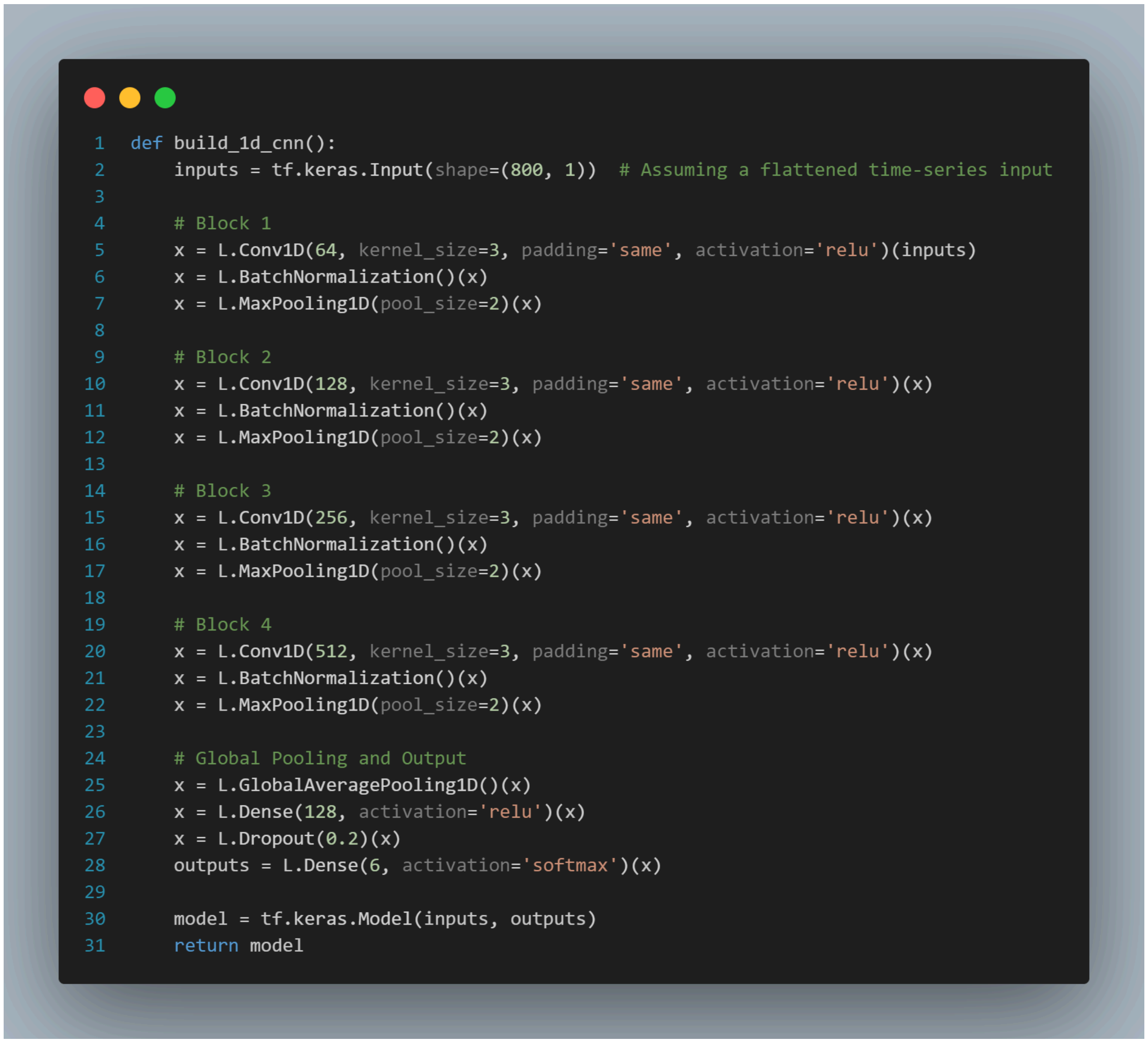


```python
def extract_features(filepath, sr=16000, frame_length=2048, hop_length=512, n_mels=128, augment = False):
    # Load audio
    signal, sr = librosa.load(filepath, sr=sr)  # Use desired sampling rate
    # Trim silent edges
    signal, _ = librosa.effects.trim(signal)

    if augment:
        # Randomly choose one augmentation (or apply all if desired)
        aug_type = np.random.choice(['noise', 'shift', 'pitch', 'stretch'])
        if aug_type == 'noise':
            signal = add_noise(signal, random=True)
        elif aug_type == 'shift':
            signal = shifting(signal)
        elif aug_type == 'pitch':
            signal = pitching(signal, sr)
        elif aug_type == 'stretch':
            try:
                signal = stretching(signal, rate=np.random.uniform(0.8, 1.2))
            except:
                pass  # stretching can sometimes result in shape mismatch

    # --- Zero Crossing Rate ---
    zcr = librosa.feature.zero_crossing_rate(
        y=signal, frame_length=frame_length, hop_length=hop_length
    )[0]  # shape: (frames,)

    # --- Energy (normalized) ---
    energy = np.array([
        np.sum(signal[i:i+frame_length]**2) / frame_length
        for i in range(0, len(signal) - frame_length + 1, hop_length)
    ])

    # --- Mel Spectrogram ---
    mel_spec = librosa.feature.melspectrogram(
        y=signal, sr=sr, n_mels=n_mels,
        n_fft=frame_length, hop_length=hop_length
    )

    mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

    return zcr, energy, mel_spec_db
```

Step 3: 1D CNN Model Architecture:

We built a 1D Convolutional Neural Network using the extracted ZCR and Energy features:

1. **Preprocessing**:
    - Padded features to ensure uniform shape.
    - Standardized each feature to have zero mean and unit variance.
2. **Model Architecture**:
    - Stacked Conv1D layers with ReLU activation.
    - Applied MaxPooling to reduce dimensionality.
    - Added Dense layers for classification.
3. **Training**:
    - Used EarlyStopping and ReduceLROnPlateau callbacks for optimized training

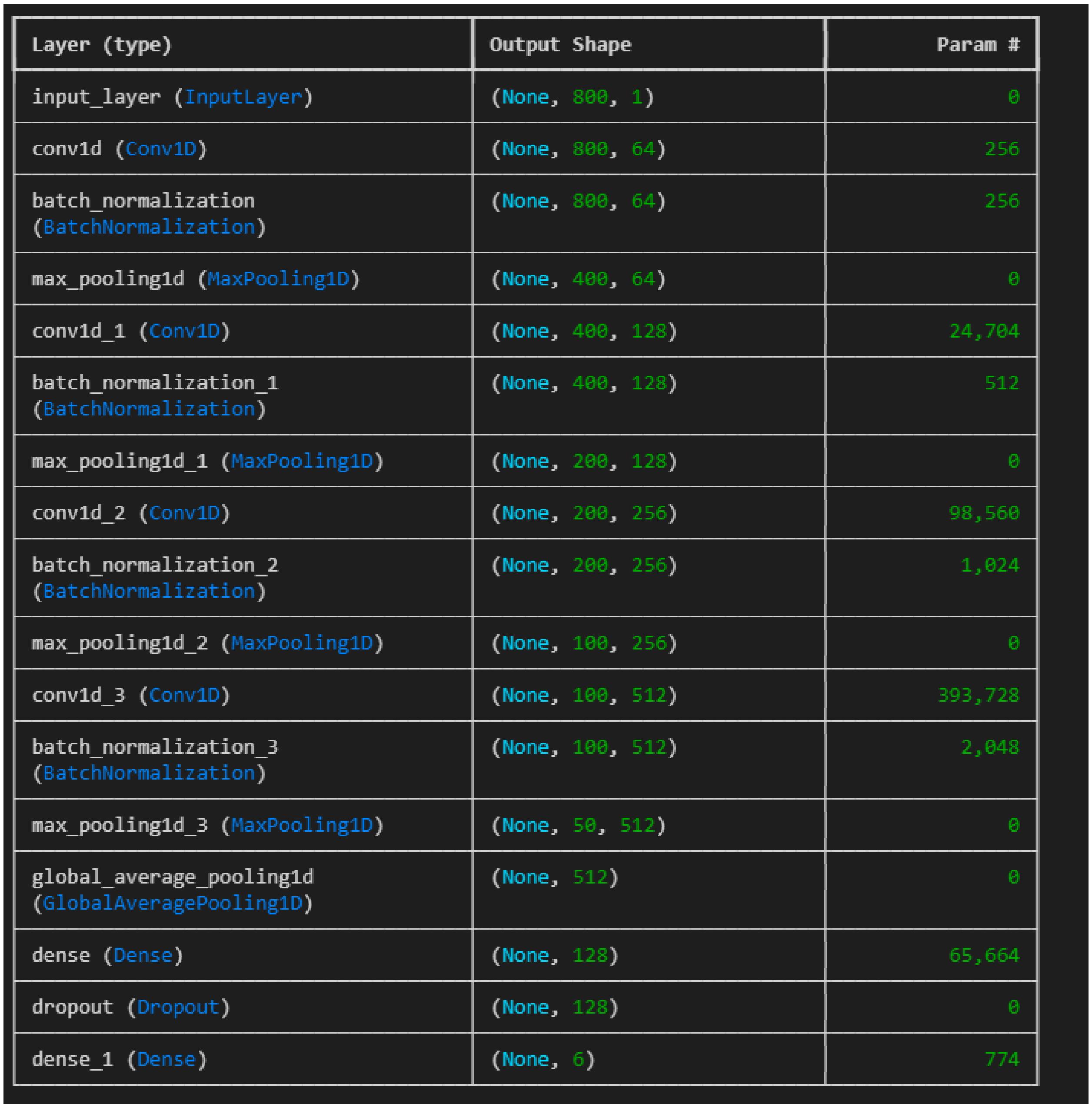


```python
def build_1d_cnn():
    inputs = tf.keras.Input(shape=(800, 1))  # Assuming a flattened time-series input

    # Block 1
    x = L.Conv1D(64, kernel_size=3, padding='same', activation='relu')(inputs)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling1D(pool_size=2)(x)

    # Block 2
    x = L.Conv1D(128, kernel_size=3, padding='same', activation='relu')(x)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling1D(pool_size=2)(x)

    # Block 3
    x = L.Conv1D(256, kernel_size=3, padding='same', activation='relu')(x)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling1D(pool_size=2)(x)

    # Block 4
    x = L.Conv1D(512, kernel_size=3, padding='same', activation='relu')(x)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling1D(pool_size=2)(x)

    # Global Pooling and Output
    x = L.GlobalAveragePooling1D()(x)
    x = L.Dense(128, activation='relu')(x)
    x = L.Dropout(0.2)(x)
    outputs = L.Dense(6, activation='softmax')(x)

    model = tf.keras.Model(inputs, outputs)
    return model
```

- **Layers:** The model consists of 4 convolutional blocks, each with Conv1D, BatchNormalization, and MaxPooling layers, followed by global average pooling and dense layers.
- **Purpose:** The model is designed to learn patterns in 1D sequential data, such as time-series or audio features.
- **GlobalAveragePooling1D:** This layer performs a global average pooling operation across the time dimension, which reduces the data to a single value per feature map, preserving global information while reducing the number of parameters.

- **Dense Layer:** A fully connected layer with 128 units and ReLU activation, which introduces non-linearity and helps the model learn complex representations.

- **Dropout:** A regularization technique where 20% of the units are randomly set to zero during training to prevent overfitting.
- **Output:** The model outputs probabilities for each of the 6 emotion classes. The final output layer has 6 units, each representing one emotion class (Angry, Disgust, Fear, Happy, Neutral, Sad). The `softmax` activation function ensures that the outputs sum to 1 and can be interpreted as probabilities.
- Model Summary:



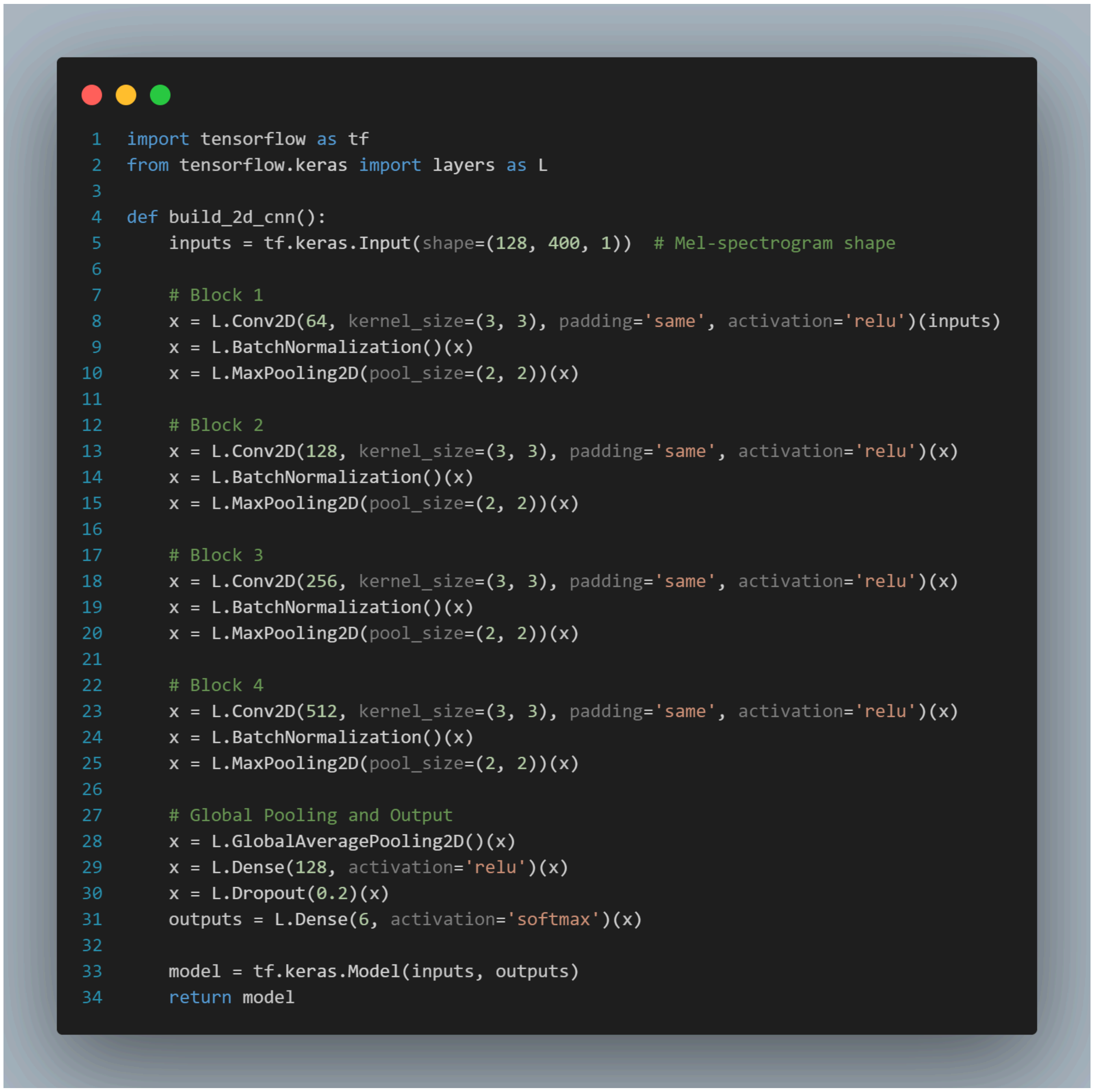
```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                 ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)             │ (None, 800, 1)               │               0 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ conv1d (Conv1D)                      │ (None, 800, 64)              │             256 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ batch_normalization                  │ (None, 800, 64)              │             256 │
│ (BatchNormalization)                 │                              │                 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ max_pooling1d (MaxPooling1D)         │ (None, 400, 64)              │               0 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ conv1d_1 (Conv1D)                    │ (None, 400, 128)             │          24,704 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ batch_normalization_1                │ (None, 400, 128)             │             512 │
│ (BatchNormalization)                 │                              │                 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ max_pooling1d_1 (MaxPooling1D)       │ (None, 200, 128)             │               0 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ conv1d_2 (Conv1D)                    │ (None, 200, 256)             │          98,560 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ batch_normalization_2                │ (None, 200, 256)             │           1,024 │
│ (BatchNormalization)                 │                              │                 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ max_pooling1d_2 (MaxPooling1D)       │ (None, 100, 256)             │               0 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ conv1d_3 (Conv1D)                    │ (None, 100, 512)             │         393,728 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ batch_normalization_3                │ (None, 100, 512)             │           2,048 │
│ (BatchNormalization)                 │                              │                 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ max_pooling1d_3 (MaxPooling1D)       │ (None, 50, 512)              │               0 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ global_average_pooling1d             │ (None, 512)                  │               0 │
│ (GlobalAveragePooling1D)             │                              │                 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 128)                  │          65,664 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 128)                  │               0 │
├──────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 6)                    │             774 │
└──────────────────────────────────────┴──────────────────────────────┴─────────────────┘
```

-
- `Total params: 587,526`

Step 4: 2D CNN Model Architecture:

We trained a 2D CNN using mel spectrogram features:

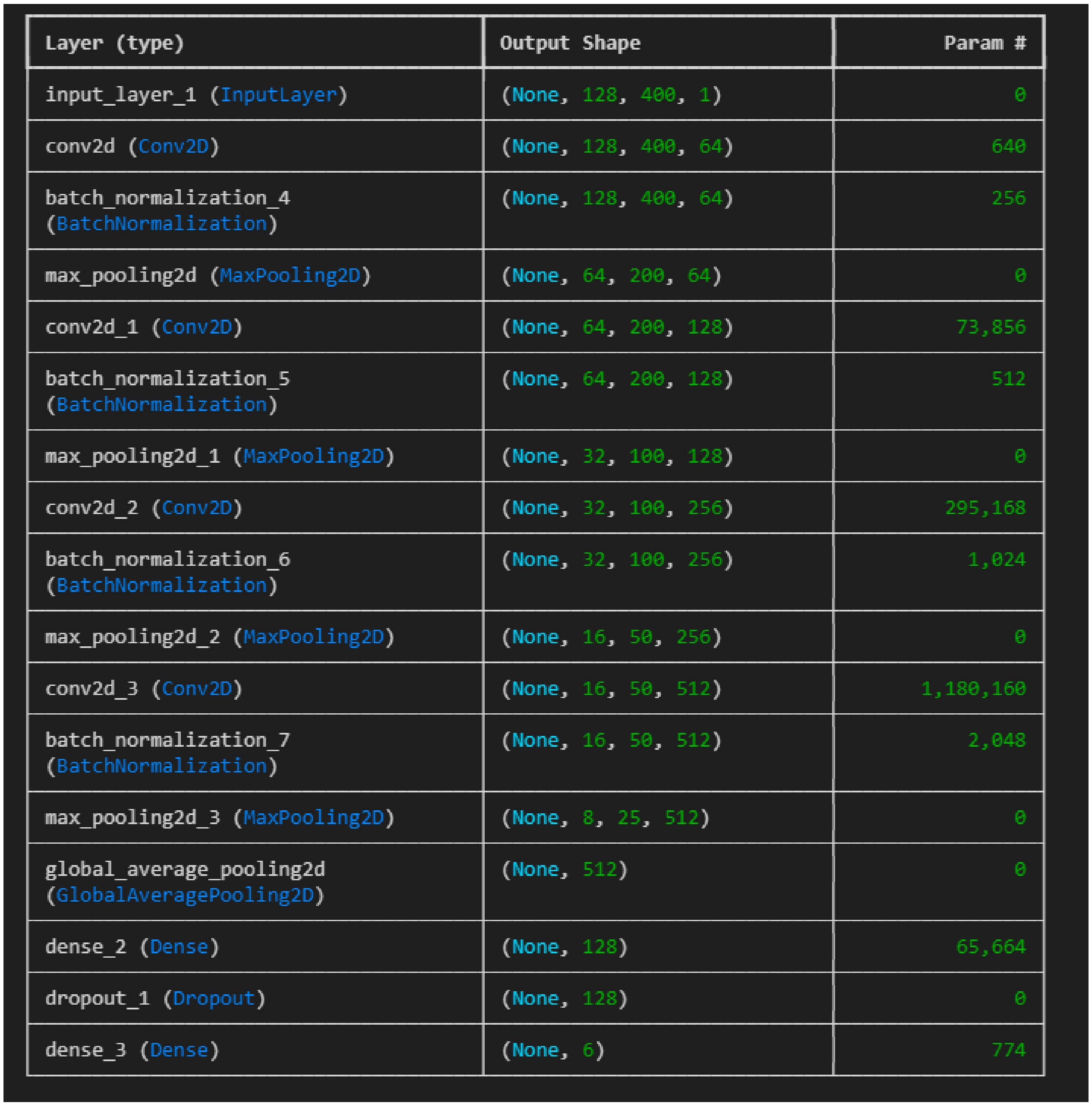
1. **Preprocessing**:
    - Converted mel spectrograms into 2D images.
    - Normalized pixel values.
2. **Model Architecture**:
    - Used Conv2D layers followed by MaxPooling2D.
    - Added BatchNormalization and Dropout to prevent overfitting.
3. **Training**:
    - Similar callbacks as 1D CNN: EarlyStopping and learning rate scheduler.



```python
import tensorflow as tf
from tensorflow.keras import layers as L

def build_2d_cnn():
    inputs = tf.keras.Input(shape=(128, 400, 1))  # Mel-spectrogram shape

    # Block 1
    x = L.Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu')(inputs)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling2D(pool_size=(2, 2))(x)

    # Block 2
    x = L.Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu')(x)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling2D(pool_size=(2, 2))(x)

    # Block 3
    x = L.Conv2D(256, kernel_size=(3, 3), padding='same', activation='relu')(x)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling2D(pool_size=(2, 2))(x)

    # Block 4
    x = L.Conv2D(512, kernel_size=(3, 3), padding='same', activation='relu')(x)
    x = L.BatchNormalization()(x)
    x = L.MaxPooling2D(pool_size=(2, 2))(x)

    # Global Pooling and Output
    x = L.GlobalAveragePooling2D()(x)
    x = L.Dense(128, activation='relu')(x)
    x = L.Dropout(0.2)(x)
    outputs = L.Dense(6, activation='softmax')(x)

    model = tf.keras.Model(inputs, outputs)
    return model
```

- **Layers:** The model has 4 convolutional blocks, each with Conv2D, BatchNormalization, and MaxPooling2D layers. After that, it uses global average pooling and dense layers.
- **Purpose:** The model is designed to learn spatial and temporal patterns from 2D data, such as spectrograms, to classify audio signals into different emotion categories.
  GlobalAveragePooling2D: This layer performs global average pooling across the entire feature map. It reduces the 2D features to a 1D vector, preserving global information while reducing the number of parameters.

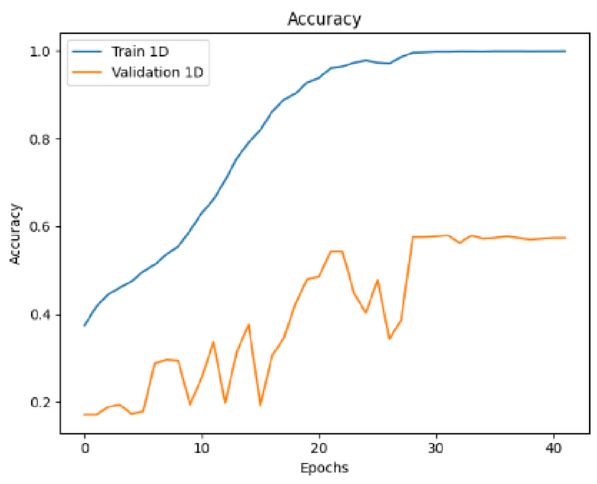
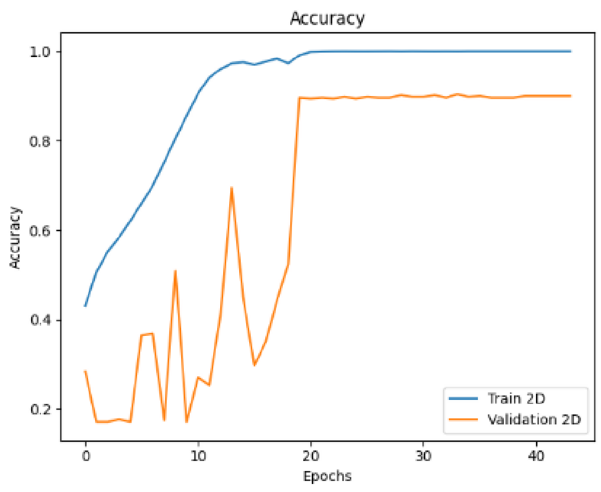
- **Dense Layer:** A fully connected layer with 128 units and ReLU activation. This layer introduces non-linearity and helps the model learn complex relationships between features.
- **Dropout:** A regularization technique that randomly drops 20% of the neurons during training to prevent overfitting.
- **Output:** The model outputs probabilities for each of the 6 emotion classes. A dense layer with 6 units, each corresponding to one emotion class (Angry, Disgust, Fear, Happy, Neutral, Sad). The `softmax` activation ensures the output values are probabilities that sum to 1.
- Model Summary:



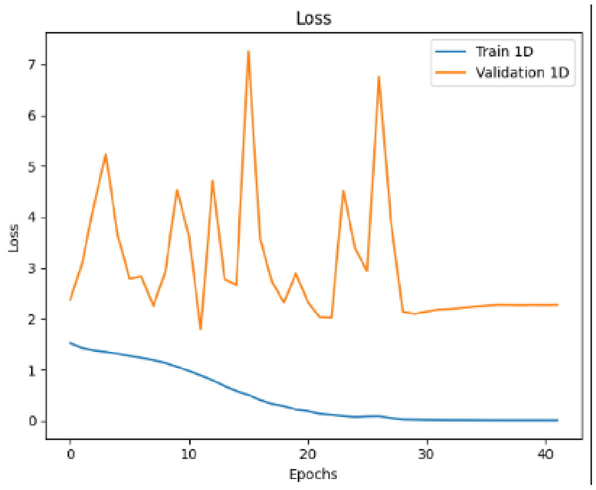
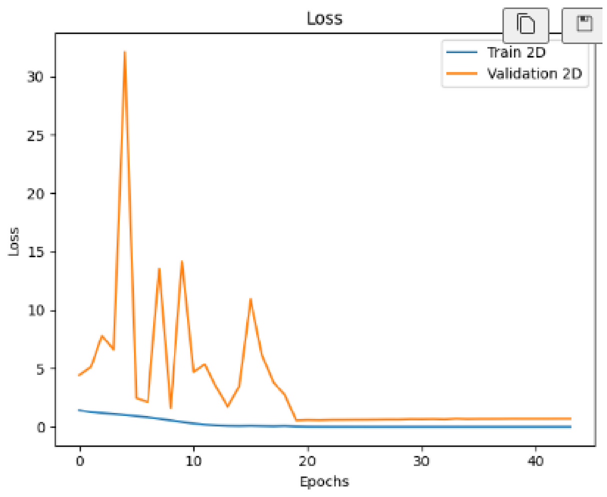
```
┌───────────────────────────────────────┬─────────────────────────────┬───────────────┐
│ Layer (type)                          │ Output Shape                │       Param # │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ input_layer_1 (InputLayer)            │ (None, 128, 400, 1)         │             0 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ conv2d (Conv2D)                       │ (None, 128, 400, 64)        │           640 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ batch_normalization_4                 │ (None, 128, 400, 64)        │           256 │
│ (BatchNormalization)                  │                             │               │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)          │ (None, 64, 200, 64)         │             0 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)                     │ (None, 64, 200, 128)        │        73,856 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ batch_normalization_5                 │ (None, 64, 200, 128)        │           512 │
│ (BatchNormalization)                  │                             │               │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)        │ (None, 32, 100, 128)        │             0 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)                     │ (None, 32, 100, 256)        │       295,168 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ batch_normalization_6                 │ (None, 32, 100, 256)        │         1,024 │
│ (BatchNormalization)                  │                             │               │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)        │ (None, 16, 50, 256)         │             0 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)                     │ (None, 16, 50, 512)         │     1,180,160 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ batch_normalization_7                 │ (None, 16, 50, 512)         │         2,048 │
│ (BatchNormalization)                  │                             │               │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)        │ (None, 8, 25, 512)          │             0 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ global_average_pooling2d              │ (None, 512)                 │             0 │
│ (GlobalAveragePooling2D)              │                             │               │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ dense_2 (Dense)                       │ (None, 128)                 │        65,664 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ dropout_1 (Dropout)                   │ (None, 128)                 │             0 │
├───────────────────────────────────────┼─────────────────────────────┼───────────────┤
│ dense_3 (Dense)                       │ (None, 6)                   │           774 │
└───────────────────────────────────────┴─────────────────────────────┴───────────────┘
```

- **`Total params: 1,620,102`**

Comparison Training & Validation Accuracies:

1D CNN Model	2D CNN Model
	

Comparison Training & Validation Losses:

1D CNN Model	2D CNN Model
	

Step 5: Evaluation:

- 1D CNN Model

Most confusing classes for 1D model: **HAP** and **ANG**

Both Emotions Have High Energy

- Happy and Angry speech both tend to have:
    - Loud volume
    - High intensity
- Energy alone cannot distinguish between positive excitement (happy) and negative excitement (angry).

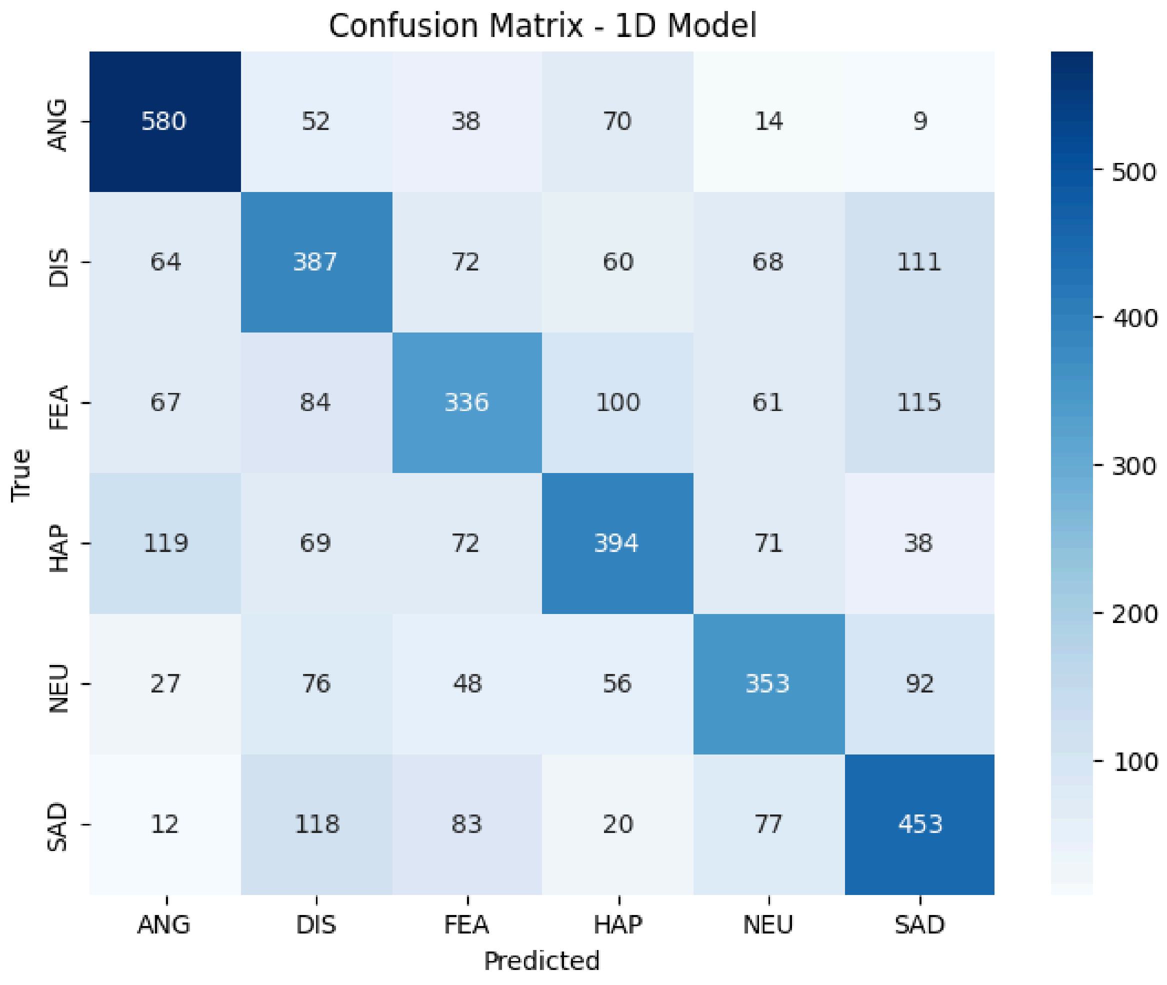
So energy just tells you *how strong* the emotion is, not *what* it is.

Both Can Have High ZCR

- ZCR reflects how often the signal changes sign — linked to:
    - Voicing characteristics
    - Noisiness
- Angry speech can be more harsh, so it may have slightly higher ZCR due to more noise.
- Happy speech is often voiced and rapid, so it can also have high ZCR.
  In practice, ZCR alone is too coarse to reliably separate these emotions.

Test Accuracy: 56.05%

F1 Score: 0.5575


Confusion Matrix - 1D Model
True
Predicted
ANG
DIS
FEA
HAP
NEU
SAD
580 52 38 70 14 9
64 387 72 60 68 111
67 84 336 100 61 115
119 69 72 394 71 38
27 76 48 56 353 92
12 118 83 20 77 453
500
400
300
200
100

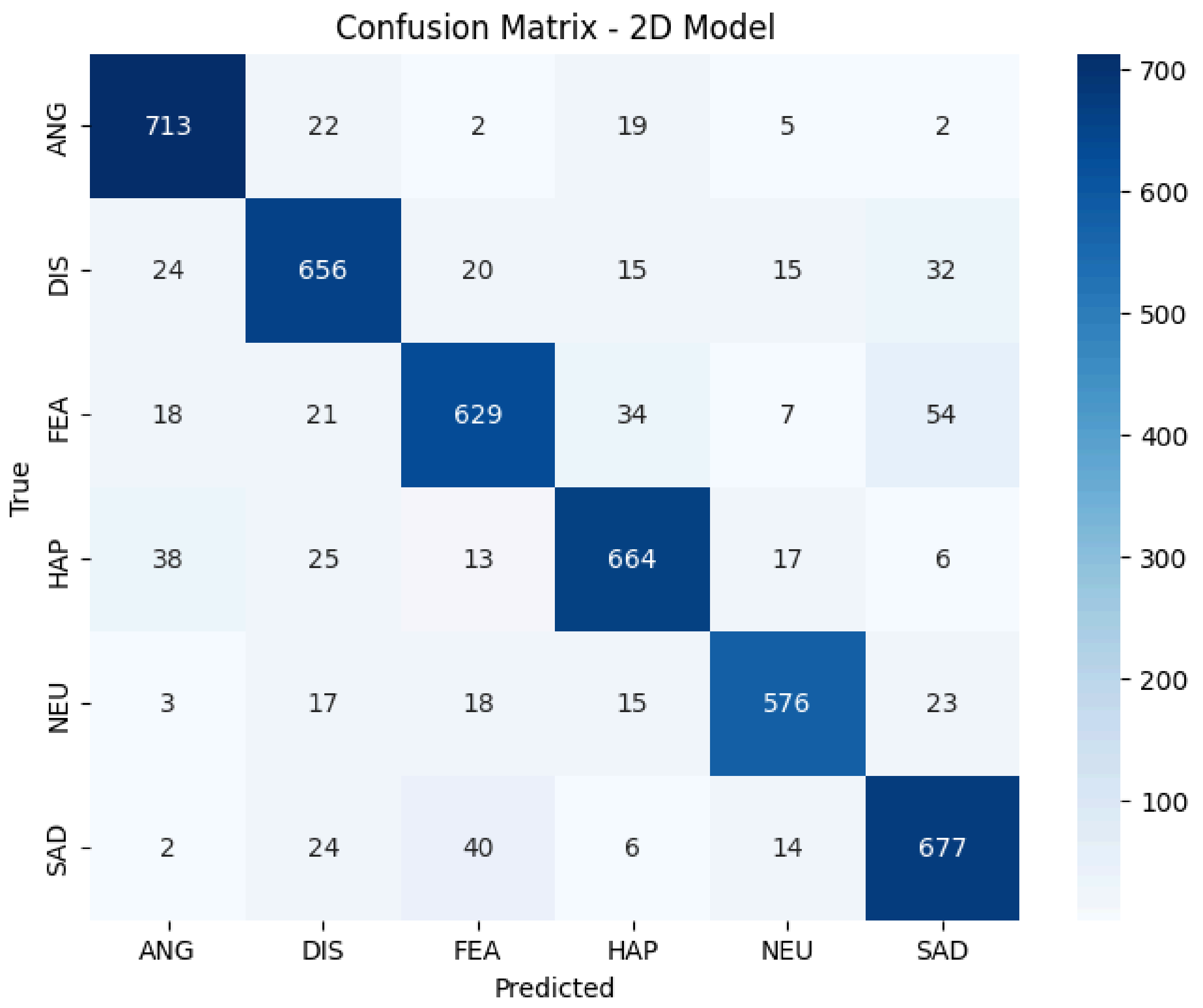
- <u>2D CNN Model</u>

Most confusing classes for 2D model: **FEA** and **SAD**

1. Both Are Low-Arousal Emotions

- Spectrograms primarily reflect energy distribution across frequency and time.
- Fear and sadness both show:
    - Low energy overall → spectrogram appears darker.
    - Low to mid frequencies → little high-frequency activity.
    - Few transients or sudden bursts → smooth, flat appearance.

Test Accuracy: 87.66%
F1 Score: 0.8764


Confusion Matrix - 2D Model
True
ANG
DIS
FEA
HAP
NEU
SAD
713 22 2 19 5 2
24 656 20 15 15 32
18 21 629 34 7 54
38 25 13 664 17 6
3 17 18 15 576 23
2 24 40 6 14 677
ANG
DIS
FEA
HAP
NEU
SAD
Predicted
700
600
500
400
300
200
100

Comparison between the 2 Models

Metrics	**1D CNN**	**2D CNN**
Test Accuracy	0.5605	0.8766
F1 Score	0.5575	0.8764