CSE 676: Deep Learning, Spring 2024

Final Project

Team Members: Epuru Sai Muralidhar, Revanth Siva Sai Ram Balineni, Shashanka Reddy Kapu

Title: Urban Sound Classification Using Deep Learning

Dataset Download Link: https://urbansounddataset.weebly.com/urbansound8k.html

Preprocessing and Normalization

Data Selection: A subset of the Urban Sound Classification dataset, specifically folds 1 to 4, is used.

```
[ ] df = pd.read_csv("/content/drive/MyDrive/UrbanSound/UrbanSound8K.csv")
    #using only a subset of the data
    df = df[df["fold"].isin([1, 2, 3, 4])]
    df.head(5)
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
14	100652-3-0-0.wav	100652	0.0	4.0	1	2	3	dog_bark
15	100652-3-0-1.wav	100652	0.5	4.5	1	2	3	dog_bark
16	100652-3-0-2.wav	100652	1.0	5.0	1	2	3	dog_bark
17	100652-3-0-3.wav	100652	1.5	5.5	1	2	3	dog_bark
64	101415-3-0-2.wav	101415	1.0	5.0	1	1	3	dog_bark

Spectrogram Generation: For each audio file, a mel-spectrogram is generated using ‘librosa.feature.melspectrogram’. This process converts the audio signals into a visual representation that captures the frequency and time information, which is more suitable for processing by CNNs.

```python
def create_spectrogram(file_path):
    y, sr = librosa.load(file_path)
    spec = librosa.feature.melspectrogram(y=y, sr=sr)
    spec_conv = librosa.amplitude_to_db(spec, ref=np.max)
    return spec_conv, data_paths[i][1]

print(f"{len(data_paths)} total spectograms")

spectrograms = []  # List to hold spectrogram and label tuples

for i, (file_path, label) in enumerate(data_paths):
    spec, label = create_spectrogram(file_path)
    spectrograms.append((spec, label))

    if i % 100 == 0 and i != 0:
        print(f"{i} spectrograms created!")

final_df = pd.DataFrame(spectrograms, columns=["spec", "label"])
```

```
3676 total spectograms
100 spectrograms created!
200 spectrograms created!
300 spectrograms created!
/usr/local/lib/python3.10/dist-packages/librosa/core/spectrum.py:257: UserWarning: n_fft=2048 is too large for input signal of length=1323
  warnings.warn(
400 spectrograms created!
500 spectrograms created!
600 spectrograms created!
700 spectrograms created!
800 spectrograms created!
900 spectrograms created!
1000 spectrograms created!
1100 spectrograms created!
1200 spectrograms created!
1300 spectrograms created!
1400 spectrograms created!
1500 spectrograms created!
1600 spectrograms created!
1700 spectrograms created!
1800 spectrograms created!
1900 spectrograms created!
2000 spectrograms created!
2100 spectrograms created!
```

Amplitude Conversion: The amplitude of the spectrogram is converted to decibels (dbs) with 'librosa.amplitude_to_db', which helps in normalizing the wide range of values in the spectrogram, making it easier for the neural network to process.

Label Extraction: The corresponding class ID for each audio file is extracted from the dataset metadata.

Normalization of Input Data: Before feeding into the model, the spectrogram data is normalized by dividing by 255.0, bringing the pixel values into the range [0, 1]. This normalization step is crucial for helping the model to converge faster during training.

```
[ ] from sklearn.model_selection import train_test_split
    from tensorflow.keras.utils import to_categorical

    # Train-Validation-Test Split
    X_train, X_temp, y_train, y_temp = train_test_split(X_new, y_new, test_size=0.15, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.15, random_state=42)

    # Reshape inputs
    input_shape = (128, 173, 1)
    X_train = np.expand_dims(X_train, axis=-1)
    X_val = np.expand_dims(X_val, axis=-1)
    X_test = np.expand_dims(X_test, axis=-1)

    # Convert labels to one-hot encoded vectors
    num_classes = len(np.unique(y_train))
    y_train = to_categorical(y_train, num_classes=num_classes)
    y_val = to_categorical(y_val, num_classes=num_classes)
    y_test = to_categorical(y_test, num_classes=num_classes)

    # Normalize inputs
    X_train = X_train / 255.0
    X_val = X_val / 255.0
    X_test = X_test / 255.0
```

Model Architecture

The model used is a Convolutional Neural Network (CNN), which is the taken choice for this task. This includes:

Input Layer: Accepts the preprocessed spectrogram data.

Convolutional Layers: Three convolutional layers with ReLU activation to extract features from the spectrogram.

Max Pooling Layers: Applied after the first and second convolutional layers to reduce dimensionality and computation.

Flattening Layer: Converts the 2D feature maps into a 1D vector for the dense layers.

Dense Layers with Dropout: Two dense layers with dropout in between to prevent overfitting and to make decisions based on the features extracted by the convolutional layers.

Output Layer: A dense layer with softmax activation that outputs a probability distribution over the 10 classes of urban sounds.

```python
# Define input layer
inputs = tf.keras.Input(shape=(128, 173, 1))

# Convolutional and max pooling layers
x = tf.keras.layers.Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding="same", activation="relu")(inputs)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)

x = tf.keras.layers.Conv2D(128, kernel_size=(3, 3), strides=(1, 1), padding="same", activation="relu")(x)
x = tf.keras.layers.MaxPooling2D((2, 2))(x)

x = tf.keras.layers.Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding="same", activation="relu")(x)

# Flatten layer
x = tf.keras.layers.Flatten()(x)

# Dense layers with dropout
x = tf.keras.layers.Dense(64, activation="relu")(x)
x = tf.keras.layers.Dropout(0.5)(x)

x = tf.keras.layers.Dense(32, activation="relu")(x)
x = tf.keras.layers.Dropout(0.5)(x)

# Output layer
outputs = tf.keras.layers.Dense(10, activation="softmax")(x)

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

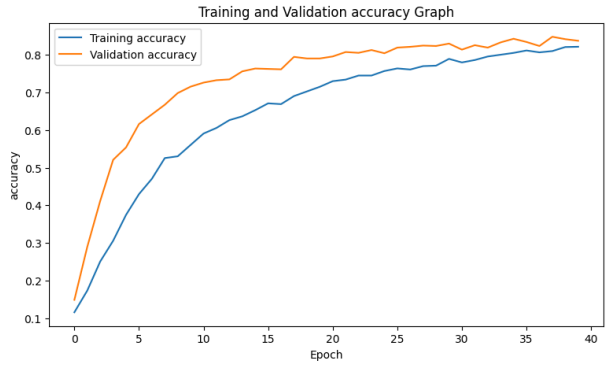
Training and Evaluation

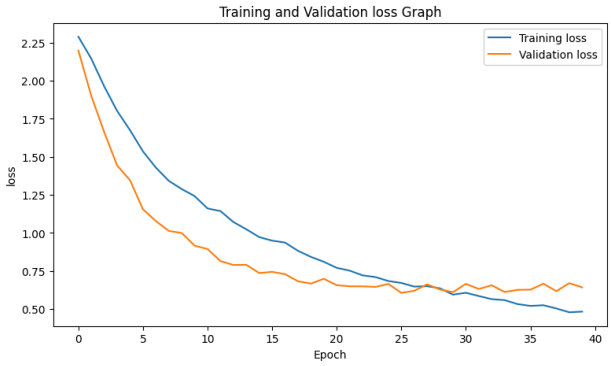
Compilation: The model is compiled with the Adam optimizer and categorical crossentropy as the loss function.

Training: The model is trained for 40 epochs with a batch size of 128. Validation data is used to monitor the performance on unseen data.

Results Visualization: Training and validation accuracy and loss are plotted over epochs to visualize the learning progress and to identify overfitting or underfitting.

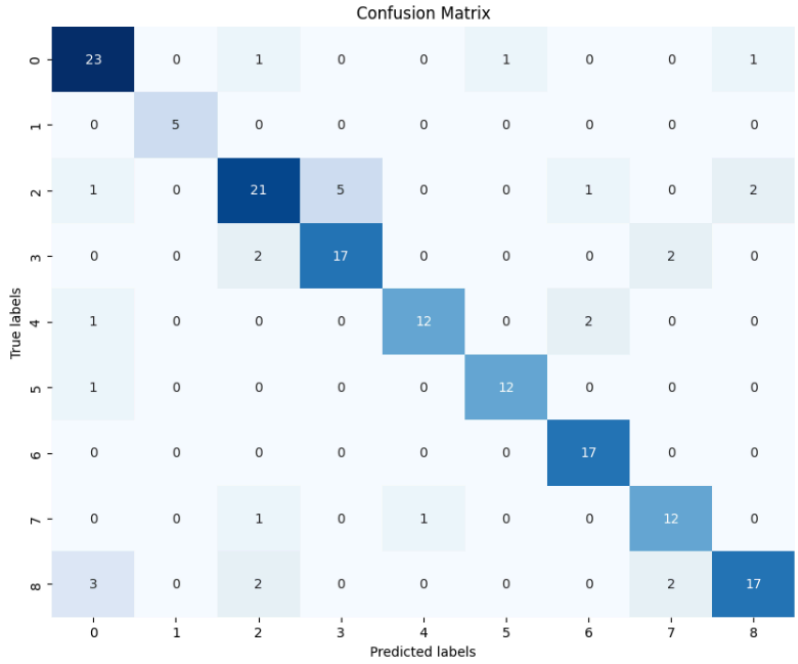
```
21/21 [==============================] - 1s 69ms/step - loss: 1.0607 - accuracy: 0.5938 - val_loss: 0.8557 - val_accuracy: 0.7259
Epoch 24/40
21/21 [==============================] - 1s 69ms/step - loss: 0.9660 - accuracy: 0.6346 - val_loss: 0.8376 - val_accuracy: 0.7487
Epoch 25/40
21/21 [==============================] - 2s 73ms/step - loss: 0.9671 - accuracy: 0.6368 - val_loss: 0.8489 - val_accuracy: 0.7208
Epoch 26/40
21/21 [==============================] - 2s 73ms/step - loss: 0.9258 - accuracy: 0.6483 - val_loss: 0.8000 - val_accuracy: 0.7056
Epoch 27/40
21/21 [==============================] - 1s 69ms/step - loss: 0.9144 - accuracy: 0.6448 - val_loss: 0.8338 - val_accuracy: 0.7360
Epoch 28/40
21/21 [==============================] - 2s 73ms/step - loss: 0.8979 - accuracy: 0.6646 - val_loss: 0.8709 - val_accuracy: 0.7157
Epoch 29/40
21/21 [==============================] - 2s 74ms/step - loss: 0.8948 - accuracy: 0.6730 - val_loss: 0.8129 - val_accuracy: 0.7386
Epoch 30/40
21/21 [==============================] - 2s 73ms/step - loss: 0.8573 - accuracy: 0.6692 - val_loss: 0.8148 - val_accuracy: 0.7589
Epoch 31/40
21/21 [==============================] - 2s 78ms/step - loss: 0.8488 - accuracy: 0.6703 - val_loss: 0.7859 - val_accuracy: 0.7538
Epoch 32/40
21/21 [==============================] - 1s 69ms/step - loss: 0.8141 - accuracy: 0.6841 - val_loss: 0.7669 - val_accuracy: 0.7538
Epoch 33/40
21/21 [==============================] - 1s 70ms/step - loss: 0.8101 - accuracy: 0.6913 - val_loss: 0.7920 - val_accuracy: 0.7614
Epoch 34/40
21/21 [==============================] - 1s 69ms/step - loss: 0.7744 - accuracy: 0.6920 - val_loss: 0.7729 - val_accuracy: 0.7437
Epoch 35/40
21/21 [==============================] - 1s 70ms/step - loss: 0.7606 - accuracy: 0.7077 - val_loss: 0.7938 - val_accuracy: 0.7563
Epoch 36/40
21/21 [==============================] - 2s 73ms/step - loss: 0.7336 - accuracy: 0.7175 - val_loss: 0.7838 - val_accuracy: 0.7538
Epoch 37/40
21/21 [==============================] - 1s 70ms/step - loss: 0.7276 - accuracy: 0.7236 - val_loss: 0.8240 - val_accuracy: 0.7538
Epoch 38/40
21/21 [==============================] - 2s 75ms/step - loss: 0.7487 - accuracy: 0.7130 - val_loss: 0.7557 - val_accuracy: 0.7589
Epoch 39/40
21/21 [==============================] - 2s 78ms/step - loss: 0.7197 - accuracy: 0.7141 - val_loss: 0.7465 - val_accuracy: 0.7716
Epoch 40/40
21/21 [==============================] - 2s 72ms/step - loss: 0.6568 - accuracy: 0.7484 - val_loss: 0.7609 - val_accuracy: 0.7665
```





The training and validation loss graph shows a desirable downward trend, indicating learning progress.

Confusion matrix:



Testing accuracy and test loss:

```
6/6 [==============================] - 0s 13ms/step - loss: 0.7592 - accuracy: 0.8242
Testing Accuracy: 82.42%
Test Loss:  0.7592489719390869
```

Metrics:

```
6/6 [==============================] - 0s 19ms/step
Precision: 0.8489
Recall: 0.8536
F1 Score: 0.8484
```

This model demonstrates steady progress in training, achieving a maximum training accuracy of around 74% and validation accuracy around 77%. However, the noticeable gap between training and validation accuracy indicates some degree of overfitting, as the model performs better on the training data than on unseen validation data. The test results show an accuracy of 82.42%, which is consistent with the validation trends. The precision, recall, and F1 score all hover around 0.85, indicating a balanced performance in identifying positive instances and minimizing false classifications.

Hyperparameter optimized model:

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 128, 173, 1)]     0

 conv2d_3 (Conv2D)           (None, 128, 173, 128)     1280

 batch_normalization (Batch  (None, 128, 173, 128)     512
 Normalization)

 max_pooling2d_2 (MaxPoolin  (None, 64, 86, 128)       0
 g2D)

 conv2d_4 (Conv2D)           (None, 64, 86, 256)       295168

 batch_normalization_1 (Bat  (None, 64, 86, 256)       1024
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 32, 43, 256)       0
 g2D)

 conv2d_5 (Conv2D)           (None, 32, 43, 128)       295040

 flatten_1 (Flatten)         (None, 176128)            0

 dense_3 (Dense)             (None, 128)               22544512

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 32)                2080

 dense_6 (Dense)             (None, 10)                330

=================================================================
Total params: 23148202 (88.30 MB)
Trainable params: 23147434 (88.30 MB)
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
```

Model Architecture

Input Specification: The model takes grayscale images with dimensions of 128x173 pixels. This matches the shape defined in the input layer.

Convolutional Layers:

The first convolutional layer has 128 filters, followed by a batch normalization layer.

The second convolutional layer increases the number of filters to 256, also followed by a batch normalization layer.

The third convolutional layer reduces the number of filters back to 128 for refinement, followed by a batch normalization layer.

Pooling Layers: Two max pooling layers are utilized to reduce spatial dimensions by half after the first and second convolutional layers, which helps in reducing the number of parameters and computation in the network, and also helps in controlling overfitting.

Flattening and Dense Layers:

A flattening layer is used to convert the 3D feature maps to 1D feature vectors.

Dense layers with decreasing units (128, 64, 32) increase the network's decision-making capability.

Dropout layers are introduced after each of the first two dense layers with a dropout rate of 0.3 to prevent overfitting.

Output Layer:

The final layer is a dense layer with 10 units, representing 10 classes, with softmax activation to output a probability distribution over these classes.

Optimization and Loss:

The model uses the Adam optimizer, which is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications.

It employs categorical cross-entropy as the loss function, suitable for multi-class classification problems.

Model Summary:

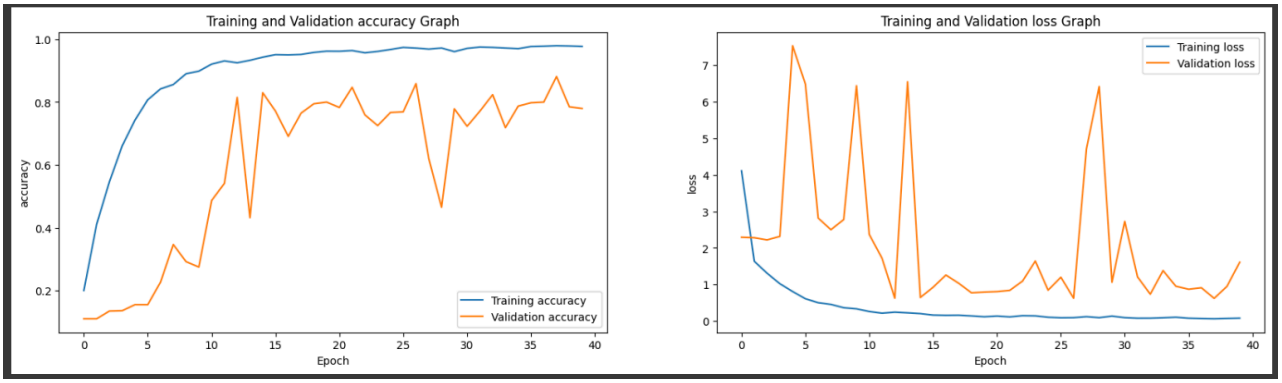
Total parameters: 23,148,202

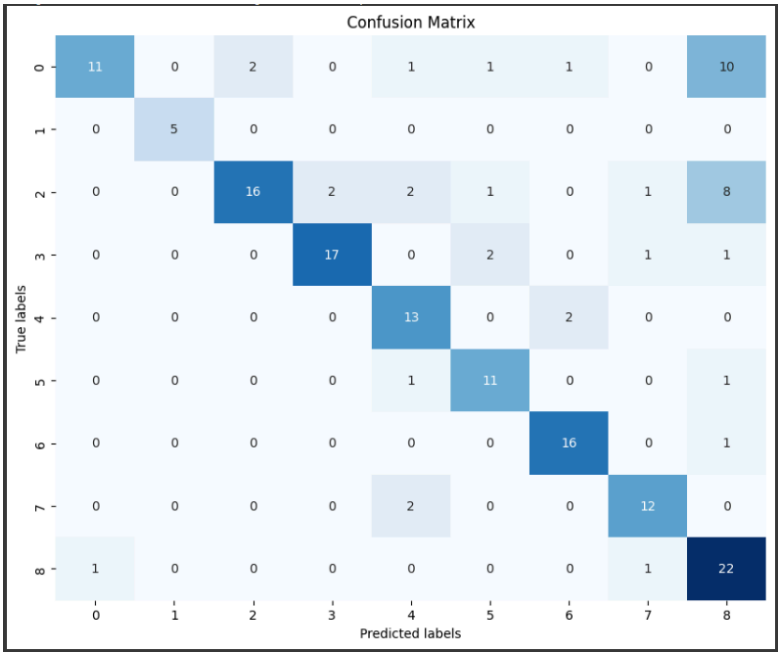
Trainable parameters: 23,147,434

Non-trainable parameters: 768

Result Visualization:

```
49/49 [==============================] - 21s 421ms/step - loss: 0.1077 - accuracy: 0.9674 - val_loss: 0.8468 - val_accuracy: 0.7668
Epoch 26/40
49/49 [==============================] - 20s 409ms/step - loss: 0.0938 - accuracy: 0.9741 - val_loss: 1.2002 - val_accuracy: 0.7690
Epoch 27/40
49/49 [==============================] - 20s 411ms/step - loss: 0.0968 - accuracy: 0.9719 - val_loss: 0.6271 - val_accuracy: 0.8588
Epoch 28/40
49/49 [==============================] - 20s 409ms/step - loss: 0.1236 - accuracy: 0.9687 - val_loss: 4.7046 - val_accuracy: 0.6203
Epoch 29/40
49/49 [==============================] - 20s 410ms/step - loss: 0.0964 - accuracy: 0.9721 - val_loss: 6.4082 - val_accuracy: 0.4652
Epoch 30/40
49/49 [==============================] - 20s 409ms/step - loss: 0.1384 - accuracy: 0.9607 - val_loss: 1.0646 - val_accuracy: 0.7786
Epoch 31/40
49/49 [==============================] - 20s 410ms/step - loss: 0.0971 - accuracy: 0.9709 - val_loss: 2.7241 - val_accuracy: 0.7230
Epoch 32/40
49/49 [==============================] - 20s 407ms/step - loss: 0.0810 - accuracy: 0.9753 - val_loss: 1.2098 - val_accuracy: 0.7711
Epoch 33/40
49/49 [==============================] - 21s 420ms/step - loss: 0.0820 - accuracy: 0.9741 - val_loss: 0.7371 - val_accuracy: 0.8235
Epoch 34/40
49/49 [==============================] - 20s 409ms/step - loss: 0.0937 - accuracy: 0.9721 - val_loss: 1.3813 - val_accuracy: 0.7187
Epoch 35/40
49/49 [==============================] - 20s 410ms/step - loss: 0.1083 - accuracy: 0.9700 - val_loss: 0.9545 - val_accuracy: 0.7872
Epoch 36/40
49/49 [==============================] - 20s 408ms/step - loss: 0.0808 - accuracy: 0.9770 - val_loss: 0.8722 - val_accuracy: 0.7979
Epoch 37/40
49/49 [==============================] - 20s 409ms/step - loss: 0.0716 - accuracy: 0.9780 - val_loss: 0.9132 - val_accuracy: 0.8000
Epoch 38/40
49/49 [==============================] - 21s 419ms/step - loss: 0.0656 - accuracy: 0.9794 - val_loss: 0.6233 - val_accuracy: 0.8813
Epoch 39/40
49/49 [==============================] - 20s 408ms/step - loss: 0.0740 - accuracy: 0.9786 - val_loss: 0.9502 - val_accuracy: 0.7850
Epoch 40/40
49/49 [==============================] - 20s 407ms/step - loss: 0.0822 - accuracy: 0.9774 - val_loss: 1.6105 - val_accuracy: 0.7797
```



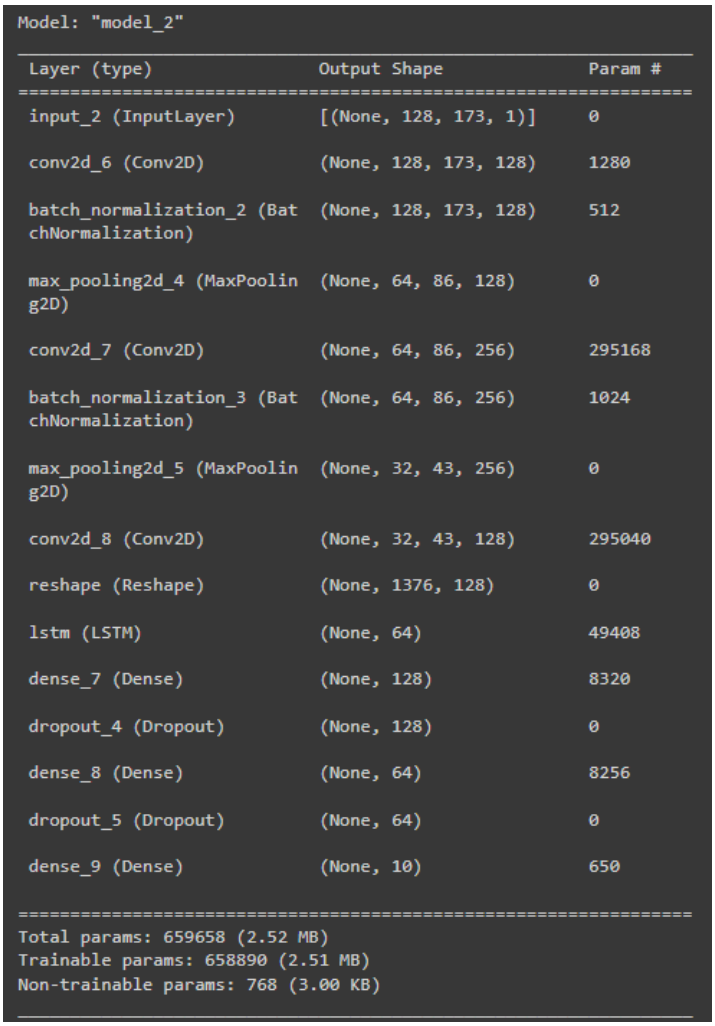



```
6/6 [==============================] - 0s 25ms/step - loss: 1.5369 - accuracy: 0.7455
Testing Accuracy: 74.55%
Test Loss:  1.536901593208313
```

```
6/6 [==============================] - 0s 24ms/step
Precision: 0.8080
Recall: 0.7993
F1 Score: 0.7799
```

The training results show high accuracy (around 97.8%) but there is a noticeable disparity with the validation and test results, where accuracies drop to around 80% and 74.55% respectively, and the validation loss increases significantly across the epochs. This suggests that the model might be overfitting the training data, where it learns specific details and noise that do not generalize well to unseen data. The increasing validation loss across epochs, particularly a sharp increase in the final epoch, further supports this.

Model: Enhanced CNN-LSTM Model Architecture



```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 128, 173, 1)]     0

 conv2d_6 (Conv2D)           (None, 128, 173, 128)     1280

 batch_normalization_2 (Bat  (None, 128, 173, 128)     512
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 64, 86, 128)       0
 g2D)

 conv2d_7 (Conv2D)           (None, 64, 86, 256)       295168

 batch_normalization_3 (Bat  (None, 64, 86, 256)       1024
 chNormalization)

 max_pooling2d_5 (MaxPoolin  (None, 32, 43, 256)       0
 g2D)

 conv2d_8 (Conv2D)           (None, 32, 43, 128)       295040

 reshape (Reshape)           (None, 1376, 128)         0

 lstm (LSTM)                 (None, 64)                49408

 dense_7 (Dense)             (None, 128)               8320

 dropout_4 (Dropout)         (None, 128)               0

 dense_8 (Dense)             (None, 64)                8256

 dropout_5 (Dropout)         (None, 64)                0

 dense_9 (Dense)             (None, 10)                650

=================================================================
Total params: 659658 (2.52 MB)
Trainable params: 658890 (2.51 MB)
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
```

Model Architecture:

Input Layer: Accepts single-channel grayscale images of size 128x173 pixels. This is for image data that might also have temporal characteristics to be captured in subsequent layers.

Convolutional Layers:

First Convolutional Layer: Uses 128 filters for initial feature extraction, paired with batch normalization to enhance model stability and learning efficiency.

Second Convolutional Layer: Increases filter count to 256 for deeper feature extraction, also followed by batch normalization.

Third Convolutional Layer: the number of filters back to 128, aimed at refining features, followed by batch normalization.

Pooling Layers: Two max pooling layers follow the first and second convolutional layers, reducing feature dimensionality and computational complexity, and helping prevent overfitting.

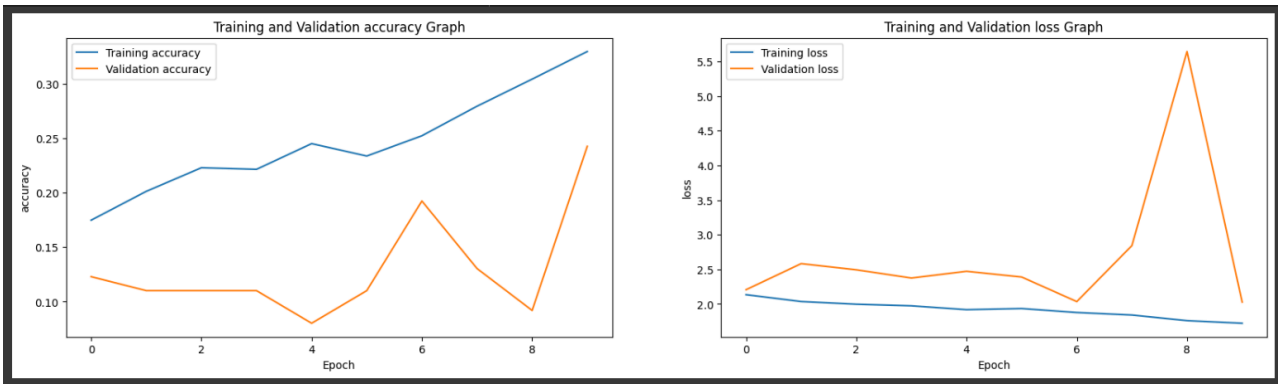
Reshape Layer: Transforms the output of the convolutional layers to suit the input requirements of the LSTM layer, for the transition from spatial to temporal data processing.

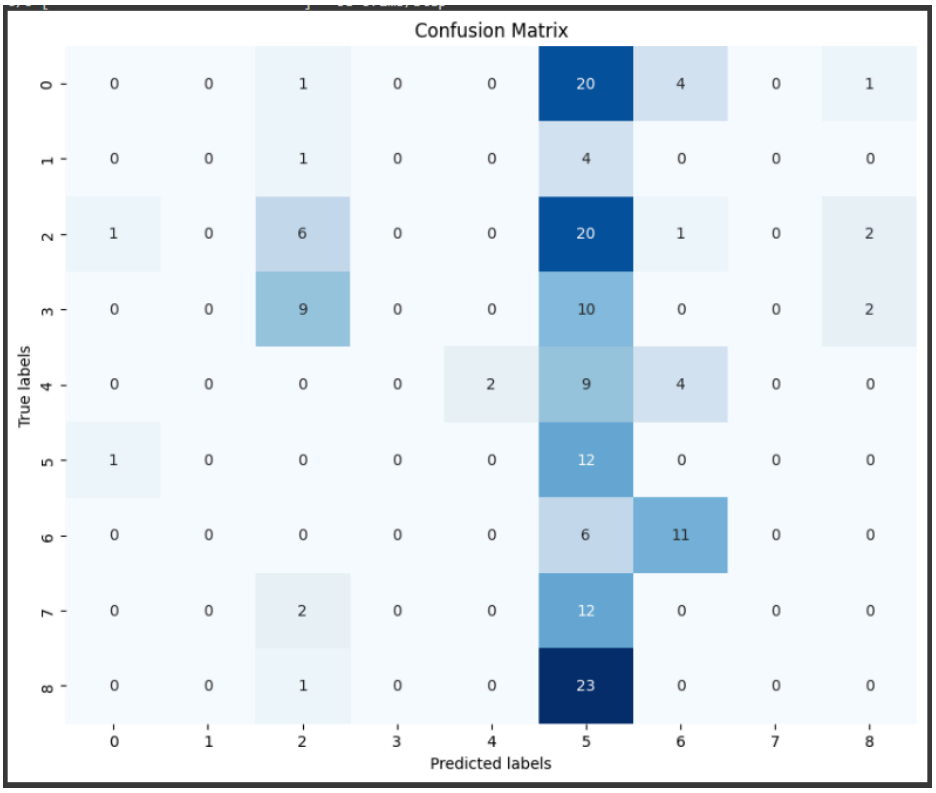
LSTM Layer: A single LSTM layer with 64 units captures temporal dynamics. This is particularly useful for sequences where temporal relationships within the data are critical for accurate predictions.

Dense and Dropout Layers:
First Dense Layer: Consists of 128 units, serves as a fully connected layer that processes features from the LSTM.
Second Dense Layer: Further processes the features through 64 units.
Dropout Layers: Placed after each dense layer with a dropout rate designed to prevent overfitting by randomly omitting a subset of features during training.
Output Layer: The final dense layer with 10 units uses softmax activation to output a probability distribution over 10 classes, suitable for multi-class classification.
Model Parameters:
Total Parameters: 659,658
Trainable Parameters: 658,890
Non-trainable Parameters: 768

Result Visualization:

```
Epoch 1/10
49/49 [==============================] - 255s 5s/step - loss: 2.1344 - accuracy: 0.1749 - val_loss: 2.2059 - val_accuracy: 0.1230
Epoch 2/10
49/49 [==============================] - 249s 5s/step - loss: 2.0355 - accuracy: 0.2014 - val_loss: 2.5822 - val_accuracy: 0.1102
Epoch 3/10
49/49 [==============================] - 245s 5s/step - loss: 1.9968 - accuracy: 0.2231 - val_loss: 2.4933 - val_accuracy: 0.1102
Epoch 4/10
49/49 [==============================] - 247s 5s/step - loss: 1.9738 - accuracy: 0.2216 - val_loss: 2.3747 - val_accuracy: 0.1102
Epoch 5/10
49/49 [==============================] - 255s 5s/step - loss: 1.9183 - accuracy: 0.2452 - val_loss: 2.4717 - val_accuracy: 0.0802
Epoch 6/10
49/49 [==============================] - 247s 5s/step - loss: 1.9346 - accuracy: 0.2338 - val_loss: 2.3892 - val_accuracy: 0.1102
Epoch 7/10
49/49 [==============================] - 250s 5s/step - loss: 1.8770 - accuracy: 0.2524 - val_loss: 2.0348 - val_accuracy: 0.1925
Epoch 8/10
49/49 [==============================] - 250s 5s/step - loss: 1.8409 - accuracy: 0.2796 - val_loss: 2.8425 - val_accuracy: 0.1305
Epoch 9/10
49/49 [==============================] - 249s 5s/step - loss: 1.7598 - accuracy: 0.3045 - val_loss: 5.6425 - val_accuracy: 0.0920
Epoch 10/10
49/49 [==============================] - 245s 5s/step - loss: 1.7223 - accuracy: 0.3299 - val_loss: 2.0273 - val_accuracy: 0.2428
```






```
6/6 [==============================] - 2s 370ms/step - loss: 2.1967 - accuracy: 0.1879
Testing Accuracy: 18.79%
Test Loss:  2.196742296218872
```

```
6/6 [==============================] - 3s 560ms/step
Precision: 0.2170
Recall: 0.2115
F1 Score: 0.1395
```

The models training and validation performance indicates some issues, primarily pointing to little high overfitting and potential underfitting. The training accuracy improves modestly over epochs, but remains low. Conversely, validation accuracy is even lower, fluctuating significantly and showing poor generalization to unseen data. The substantial and erratic jumps in validation loss, including a peak at over 5.6, suggest the model is struggling to learn the underlying patterns of the data effectively.

Additionally, metrics (accuracy, precision, recall, F1 score) reinforce the model's inability to make reliable predictions on new data. This could be due to several factors including insufficient training data, inappropriate model complexity, or inadequate training time.

Transformer Model Architecture:

```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_5 (InputLayer)        [(None, 128, 173, 1)]     0         []

 conv2d_15 (Conv2D)          (None, 128, 173, 128)     1280      ['input_5[0][0]']

 max_pooling2d_11 (MaxPooli  (None, 64, 86, 128)       0         ['conv2d_15[0][0]']
 ng2D)

 conv2d_16 (Conv2D)          (None, 64, 86, 256)       295168    ['max_pooling2d_11[0][0]']

 max_pooling2d_12 (MaxPooli  (None, 32, 43, 256)       0         ['conv2d_16[0][0]']
 ng2D)

 conv2d_17 (Conv2D)          (None, 32, 43, 128)       295040    ['max_pooling2d_12[0][0]']

 max_pooling2d_13 (MaxPooli  (None, 16, 21, 128)       0         ['conv2d_17[0][0]']
 ng2D)

 reshape_2 (Reshape)         (None, 336, 128)          0         ['max_pooling2d_13[0][0]']

 lambda_1 (Lambda)           (None, 336, 128)          0         ['reshape_2[0][0]']

 multi_head_attention_1 (Mu  (None, 336, 128)          263808    ['lambda_1[0][0]',
 ltiHeadAttention)                                               'lambda_1[0][0]']

 global_average_pooling1d_1  (None, 128)               0         ['multi_head_attention_1[0][0]
  (GlobalAveragePooling1D)                                       ']

 dropout_9 (Dropout)         (None, 128)               0         ['global_average_pooling1d_1[0
                                                                 ][0]']

 dense_16 (Dense)            (None, 64)                8256      ['dropout_9[0][0]']

 dense_17 (Dense)            (None, 10)                650       ['dense_16[0][0]']

==================================================================================================
Total params: 864202 (3.30 MB)
Trainable params: 864202 (3.30 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

Architecture:
Input Layer: Designed to handle single-channel grayscale images with dimensions 128x173 pixels, suitable for capturing spatial features without color information.
CNN Architecture:
First Convolutional Layer (conv2d_15): Utilizes 128 filters with a kernel size of 3x3 and ReLU activation, aimed at initial feature extraction.
Max Pooling Layer (max_pooling2d_11): Reduces the spatial dimensions of the feature maps by half, decreasing the computational load and helping to mitigate overfitting.
Second Convolutional Layer (conv2d_16): Increases the number of filters to 256, capturing more complex features with the same 3x3 kernel and ReLU activation.
Max Pooling Layer (max_pooling2d_12): Further reduces feature map dimensions, preserving the most essential features.

Third Convolutional Layer (conv2d_17): Maintains 256 filters for continued feature refinement.
Max Pooling Layer (max_pooling2d_13): Final reduction in spatial dimensions, preparing the data for transformation into sequences.

Data Reshaping and Positional Encoding:

Reshape Layer (reshape_2): Transforms the 2D feature maps into a sequence of vectors suitable for processing by the Transformer block.
Lambda Layer (lambda_1): Applies positional encoding to the sequence to embed temporal information, crucial for maintaining the order of data points in the Transformer.

Transformer Block:

Multi-head Attention (multi_head_attention_1): Employs a multi-head attention mechanism to focus on different parts of the input sequence simultaneously, enhancing the model's ability to capture dependencies across the data.
Pooling and Output Processing:
Global Average Pooling (global_average_pooling1d_1): Condenses the entire output of the Transformer into a single vector, summarizing the extracted features.
Dropout Layer (dropout_9): Applied with an aim to prevent overfitting by randomly dropping units during training.
Dense Layers: Sequential fully connected layers (dense_16 with 64 units and dense_17 with 10 units) process the pooled features into final predictions.
Output Layer (dense_17): Uses softmax activation to produce a probability distribution across 10 classes.

Optimization and Loss:

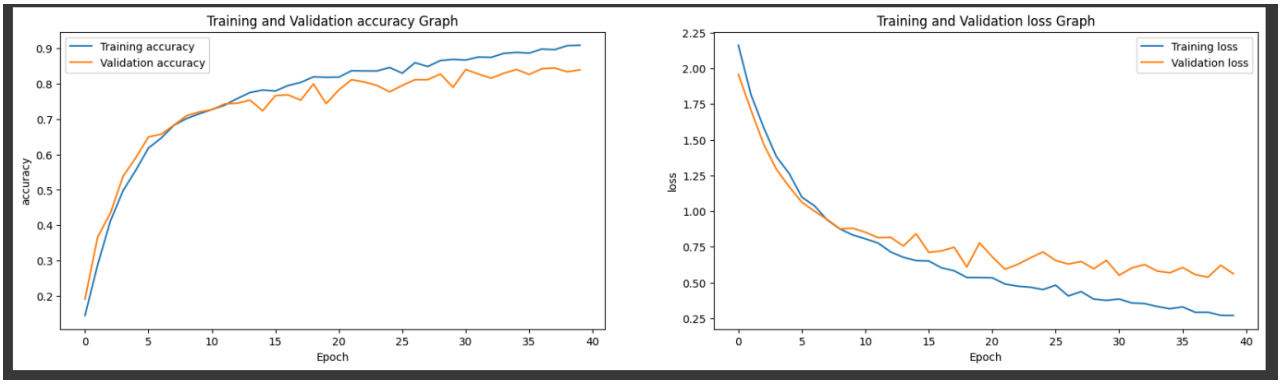
The model utilizes the Adam optimizer for efficient stochastic optimization and categorical cross-entropy loss for handling multi-class classification tasks.

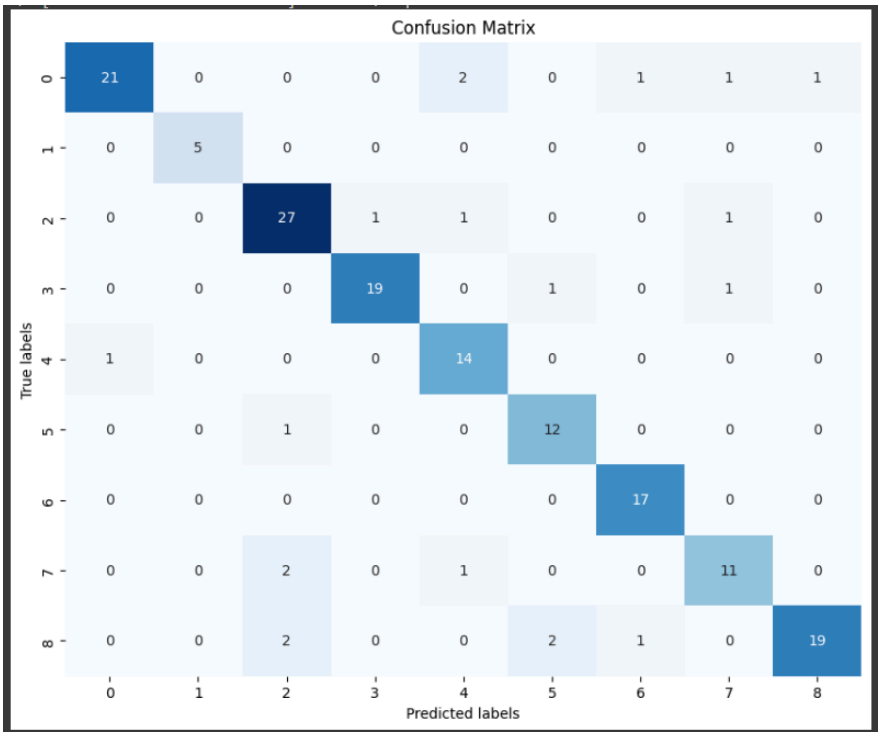
Model Parameters:

Total Parameters: 864,202
Trainable Parameters: 864,202

Result Visualization:

```
49/49 [==============================] - 19s 392ms/step - loss: 0.3839 - accuracy: 0.8648 - val_loss: 0.5967 - val_accuracy: 0.8267
Epoch 30/40
49/49 [==============================] - 20s 401ms/step - loss: 0.3747 - accuracy: 0.8680 - val_loss: 0.6551 - val_accuracy: 0.7893
Epoch 31/40
49/49 [==============================] - 20s 401ms/step - loss: 0.3843 - accuracy: 0.8662 - val_loss: 0.5512 - val_accuracy: 0.8396
Epoch 32/40
49/49 [==============================] - 19s 391ms/step - loss: 0.3566 - accuracy: 0.8747 - val_loss: 0.6020 - val_accuracy: 0.8267
Epoch 33/40
49/49 [==============================] - 19s 393ms/step - loss: 0.3523 - accuracy: 0.8738 - val_loss: 0.6258 - val_accuracy: 0.8150
Epoch 34/40
49/49 [==============================] - 19s 393ms/step - loss: 0.3318 - accuracy: 0.8850 - val_loss: 0.5801 - val_accuracy: 0.8289
Epoch 35/40
49/49 [==============================] - 20s 402ms/step - loss: 0.3158 - accuracy: 0.8877 - val_loss: 0.5681 - val_accuracy: 0.8396
Epoch 36/40
49/49 [==============================] - 19s 392ms/step - loss: 0.3290 - accuracy: 0.8857 - val_loss: 0.6051 - val_accuracy: 0.8257
Epoch 37/40
49/49 [==============================] - 20s 402ms/step - loss: 0.2914 - accuracy: 0.8972 - val_loss: 0.5553 - val_accuracy: 0.8417
Epoch 38/40
49/49 [==============================] - 20s 401ms/step - loss: 0.2916 - accuracy: 0.8955 - val_loss: 0.5374 - val_accuracy: 0.8439
Epoch 39/40
49/49 [==============================] - 20s 401ms/step - loss: 0.2702 - accuracy: 0.9069 - val_loss: 0.6204 - val_accuracy: 0.8332
Epoch 40/40
49/49 [==============================] - 19s 391ms/step - loss: 0.2690 - accuracy: 0.9078 - val_loss: 0.5613 - val_accuracy: 0.8385
```






```
6/6 [==============================] - 0s 28ms/step - loss: 0.5093 - accuracy: 0.8788
Testing Accuracy: 87.88%
Test Loss:  0.5093083381652832
```

```
6/6 [==============================] - 0s 26ms/step
Precision: 0.8841
Recall: 0.8940
F1 Score: 0.8858
```

This model demonstrates robust performance with consistent improvement across training epochs, reflected in the convergence of training and validation loss and accuracy. The final testing results show a high accuracy of 87.88%, with precision, recall, and F1 score all above 0.88, indicating a strong balance between correctly identifying positive cases and minimizing false positives and negatives. The convergence of training and validation metrics suggests good

generalization to unseen data, making this a well-fitted model for its specific classification task. This balance of performance metrics and effective learning dynamics makes it the best model among those tested.

Key Observations:

-> We used a key technique to transform **‘audio signals into a visual representation’** that captures both **frequency and time** information. By using **‘librosa.feature.melspectrogram’**, audio data is converted into **spectrograms**, which display variations in sound intensity across different frequencies over time. These visual representations allow convolutional neural networks (CNNs) to effectively identify patterns similar to how they process image data. Additionally, converting the amplitude of these spectrograms to decibels with **‘librosa.amplitude_to_db’** normalizes the range, enhancing the model’s ability to process and learn from these features efficiently, thereby improving accuracy.

-> In the project, generating **spectrograms was a significant preprocessing step** that took **about an hour** to complete. This process involved converting audio signals into mel-spectrograms using librosa.feature.melspectrogram.

-> Initially, to optimize these models, we trained on a subset of the data, specifically folds 1 to 4 of the dataset. This strategy allowed for iterative adjustments and refinements in the model architectures, leading to the final, more effective models .

Comparisons:

Model	Total Params	Training Accuracy	Validation Accuracy	Testing Accuracy	Precision	Recall	F1 Score	Best Model Reasoning
Basic CNN Model	5786858	82.16%	83.74%	82.42%	0.8489	0.8536	0.8484	Good General Balance
Hyperparamet r- CNN Model	23,148,202	97.74%	77.97%	74.55.%	0.8080	0.7993	0.7799	Significant Overfitting
LSTM Model	659,658	32.99%	24.28%	18.79%	0.2170	0.2115	0.1395	Not good performance
Transformer model	864,202	90.7%	83.85%	87.88%	0.8841	0.8940	0.8858	Very High level accurate model

The **‘Transformer Model’** stands out as the best among the ones tested, as:

Accuracy: It has the highest testing accuracy of 87.88%, which is closely aligned with its training and validation accuracies, indicating excellent model generalization.

Balance of Metrics: Precision, recall, and F1 score are all above 0.88, suggesting that the model is not only accurate overall but also effective in classifying each class with minimal bias and error.

Parameter Efficiency: Despite having a comparable number of parameters to the Enhanced CNN Model, the Transformer Model shows better performance and stability, making it the most effective in utilizing its architectural complexity.

In this, model's success is due to the use of multi-head attention mechanisms, which enhance its ability to capture dependencies in the data effectively, a critical factor in complex sound classification tasks.

Bonus Claims:

a) Real-world deep learning application:

The dataset used in the project is essential for developing systems that automate the identification of urban sounds from real-world environments.

This dataset typically includes diverse sound categories such as car horns, construction noise, human voices, and ambient street sounds. Utilizing deep learning, **specifically the Transformer model which achieved high accuracy**, enables applications like monitoring urban noise pollution, enhancing public safety by detecting anomalous sounds, and improving urban planning.

By accurately classifying these sounds, such systems can contribute to smarter, more responsive urban environments, illustrating the practical application of machine learning in managing and interpreting complex urban data.

b) If you have come up with a new model or setup, that is almost the same as the current models or even better, your work is eligible for this bonus. In your report, you must highlight the novelty.

The novelty of the Transformer model developed for this dataset lies in its integration of Convolutional Neural Networks (CNNs) with **Transformer architecture,** uniquely tailored to handle the intricacies of sound data represented as spectrograms. This hybrid model harnesses the spatial feature extraction capabilities of CNNs and the sequence modeling prowess of Transformers within a unified framework, a distinctive approach particularly suited to analyzing time-frequency representations of sounds.

Unique Model Setup:

Convolutional Layers: The model starts with three convolutional layers with filter sizes tailored to incrementally capture complex features from the spectrograms, followed by max pooling to reduce dimensionality and focus on salient features.

Positional Encoding: Post convolution and flattening, the model incorporates positional encoding directly into the flattened feature vectors. This step is crucial for adding temporal context to the features, enabling the Transformer to effectively interpret sequences as time-series data.

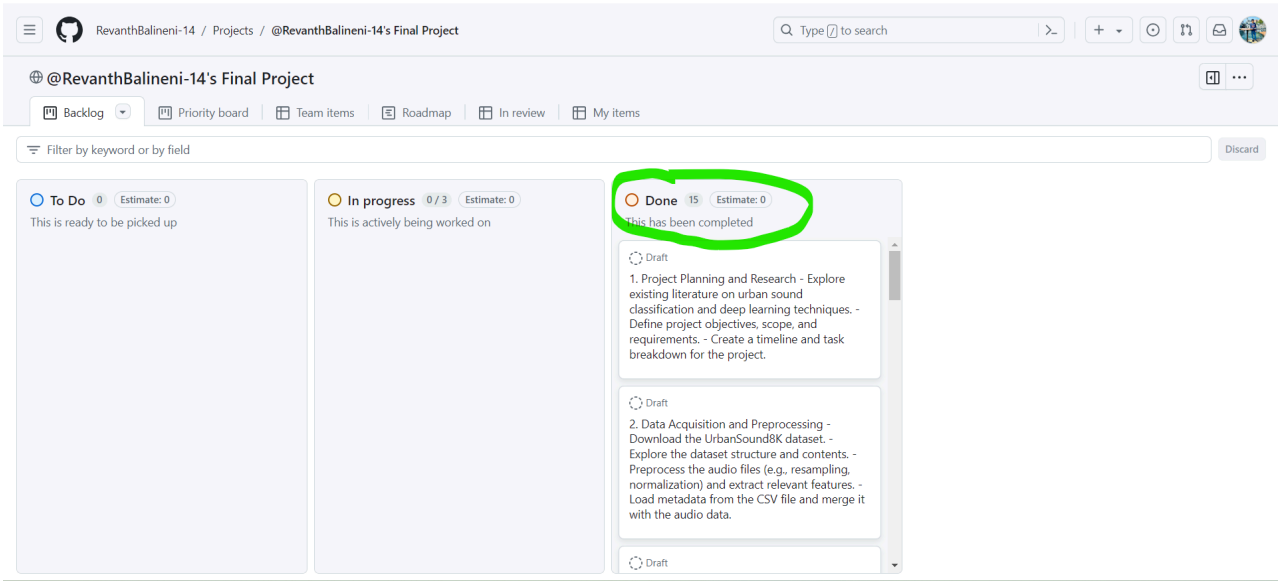
Transformer Block: The core of the model's novelty is the Transformer block that employs multi-head attention mechanisms. This allows the model to attend to different parts of the input sequence simultaneously, enhancing its ability to discern intricate dependencies across various parts of the audio signal.

Performance and Accuracy:

The model achieved a **testing** accuracy of **87.88%**, significantly outperforming traditional models. This high level of accuracy demonstrates the model's effectiveness in generalizing well from training to unseen data, owing to its advanced capability to capture both spatial and temporal dependencies in urban sound data.

This innovative setup not only achieves high accuracy but also sets a new precedent in sound classification, providing a robust solution for analyzing complex urban environments. This model can be seen as a significant enhancement in the domain of audio analysis, especially in real-world applications where understanding the context and dynamics of urban sounds is crucial.

Project Management tool: (Github Project)



Contributions:

Epuru Sai Muralidhar	50537867	33.33%
Revanth Balineni	50540873	33.33%
Shashank Reddy Kapu	50533630	33.33%

References:

https://librosa.org/doc/latest/index.html
https://www.tensorflow.org/api_docs/python/tf/keras

https://towardsdatascience.com/the-lstm-reference-card-6163ca98ae87

https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5

https://huggingface.co/docs/transformers/en/tasks/audio_classification