MID SEMESTER LAB EVALUATION

CONVERSATIONAL AI: SPEECH PROCESSING AND SYNTHESIS(UCS749)

Submitted by:

Pratham Agarwal

102103607

BE Fourth Year, COE

Submitted to:

Dr. BRAHMADESAM VENKATARAMAIYER R

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology,
Patiala

September 2024

Summary of Research Paper : Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition

The paper "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition" introduces a dataset of short audio clips of spoken words designed for training and evaluating keyword spotting models. The dataset focuses on recognizing specific words efficiently, even in noisy environments, to support applications like voice-activated devices. It includes collection methods, quality control, and baseline model results.

Comprehensive Report on Audio Classification Using CNNs with Mel-Spectrograms

1. Introduction

The project focuses on building an audio classification model using a Convolutional Neural Network (CNN) with Mel-spectrograms as input features. The dataset used is the Speech Commands Dataset (speech_commands_v0.02) provided by TensorFlow, which consists of one-second long audio clips of various spoken commands. The goal is to classify these audio clips into their respective command categories.

2. Dataset Description

2.1. Data Collection

The dataset is automatically downloaded and extracted from the official TensorFlow repository. The dataset consists of multiple folders where each folder represents a specific spoken command. Each folder contains audio files in .wav format.

2.2. Data Preprocessing

1. **Audio Loading**: The audio files are loaded using the librosa library at a sample rate of 16kHz.
2. **Duration Standardization**: All audio clips are trimmed or padded to ensure a uniform duration of 1 second (16,000 samples).
3. **Mel-Spectrogram Conversion**: Each audio clip is converted to a Mel-spectrogram with 32 Mel bands and a maximum frequency of 8kHz. This conversion transforms the audio signal into a format suitable for input into a CNN.
4. **Normalization**: The Mel-spectrogram values are normalized to a range between 0 and 1 by dividing by 255.

2.3. Data Augmentation

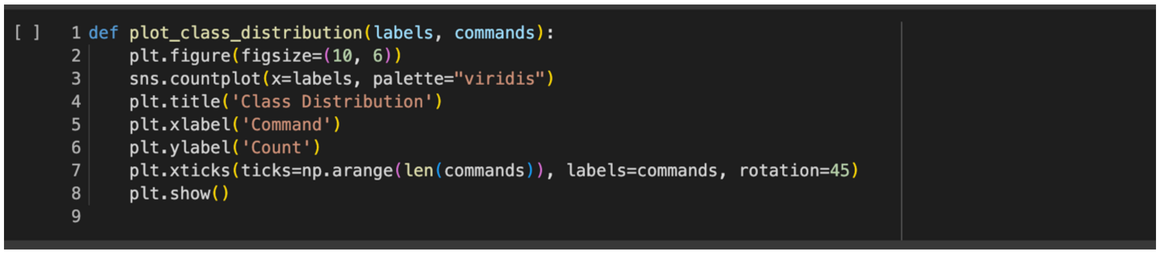
To enhance the model's robustness and prevent overfitting, various data augmentation techniques are applied:

1. **Noise Addition**: Random noise is added to audio samples.
2. **Pitch Shifting**: The pitch of audio samples is randomly shifted by a few semitones.

3. Exploratory Data Analysis (EDA)

3.1. Class Distribution

The dataset contains several command classes, and their distribution is analyzed using a bar plot. The plot reveals any class imbalances that could affect the model's performance.

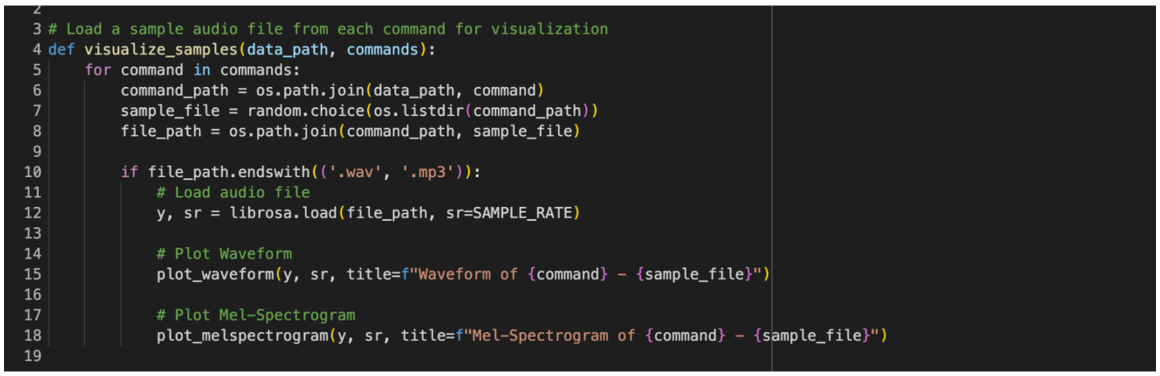


```
[ ]
def plot_class_distribution(labels, commands):
    plt.figure(figsize=(10, 6))
    sns.countplot(x=labels, palette="viridis")
    plt.title('Class Distribution')
    plt.xlabel('Command')
    plt.ylabel('Count')
    plt.xticks(ticks=np.arange(len(commands)), labels=commands, rotation=45)
    plt.show()

```

3.2. Visualizing Audio Samples

For each command category, a random audio sample is selected, and both its waveform and Mel-spectrogram are visualized.



```
# Load a sample audio file from each command for visualization
def visualize_samples(data_path, commands):
    for command in commands:
        command_path = os.path.join(data_path, command)
        sample_file = random.choice(os.listdir(command_path))
        file_path = os.path.join(command_path, sample_file)

        if file_path.endswith(('.wav', '.mp3')):
            # Load audio file
            y, sr = librosa.load(file_path, sr=SAMPLE_RATE)

            # Plot Waveform
            plot_waveform(y, sr, title=f"Waveform of {command} - {sample_file}")

            # Plot Mel-Spectrogram
            plot_melspectrogram(y, sr, title=f"Mel-Spectrogram of {command} - {sample_file}")

```

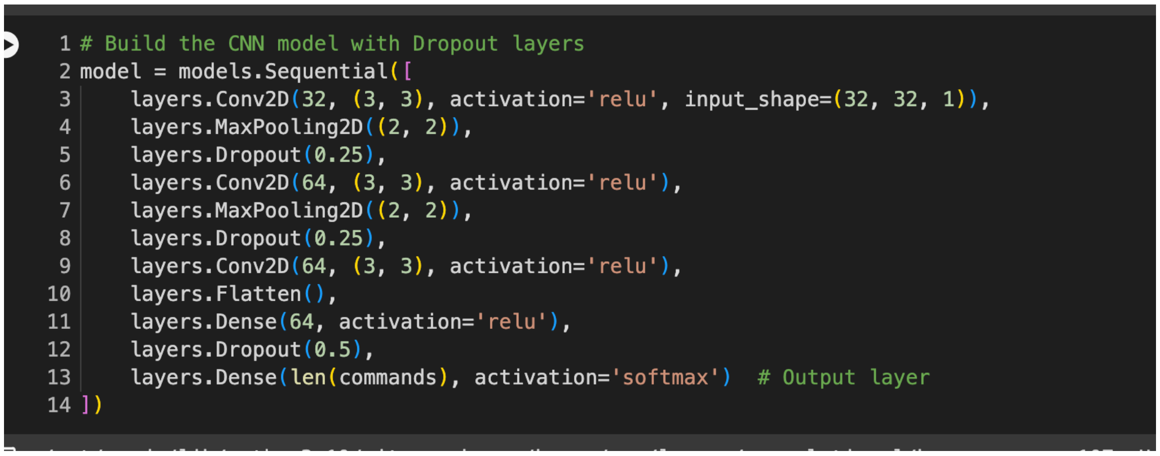
3.3. Findings from EDA

1. The dataset contains a reasonable distribution of commands with some minor class imbalances.
2. Mel-spectrograms provide a visual representation that captures the essential frequency patterns needed for classification.

4. Model Building and Training

4.1. Model Architecture

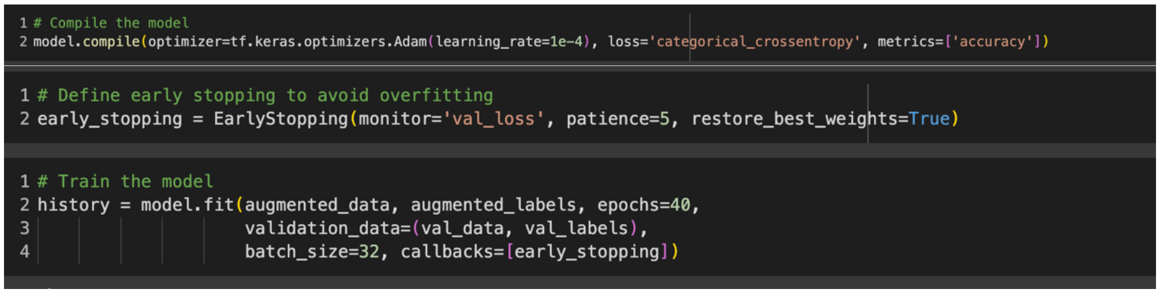
The chosen model is a Convolutional Neural Network (CNN) with dropout layers to reduce overfitting. The architecture consists of three convolutional layers with max pooling, followed by a dense layer and a final output layer.



```python
# Build the CNN model with Dropout layers
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(commands), activation='softmax')  # Output layer
])
```

4.2. Training the Model

The model is trained using the Adam optimizer and categorical cross-entropy loss. An early stopping **callback** is used to prevent overfitting by monitoring the validation loss.



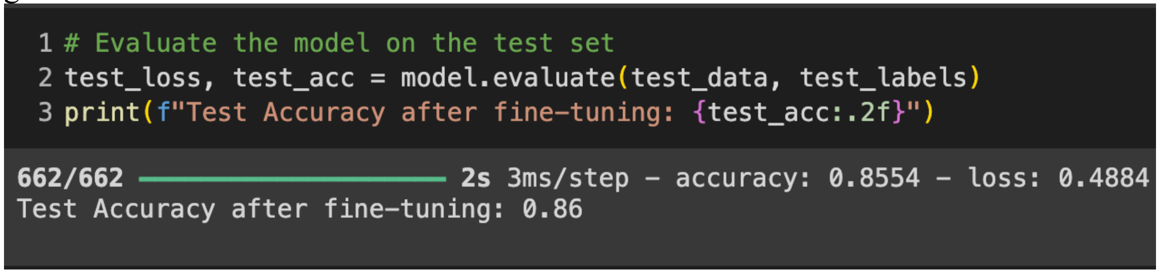
```python
# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=['accuracy'])
```

```python
# Define early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```python
# Train the model
history = model.fit(augmented_data, augmented_labels, epochs=40,
                    validation_data=(val_data, val_labels),
                    batch_size=32, callbacks=[early_stopping])
```

4.3. Model Performance

The model achieves an accuracy of **86%** on the test dataset, indicating good generalization on unseen data.



```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_data, test_labels)
print(f"Test Accuracy after fine-tuning: {test_acc:.2f}")
```

```
662/662 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8554 - loss: 0.4884
Test Accuracy after fine-tuning: 0.86
```

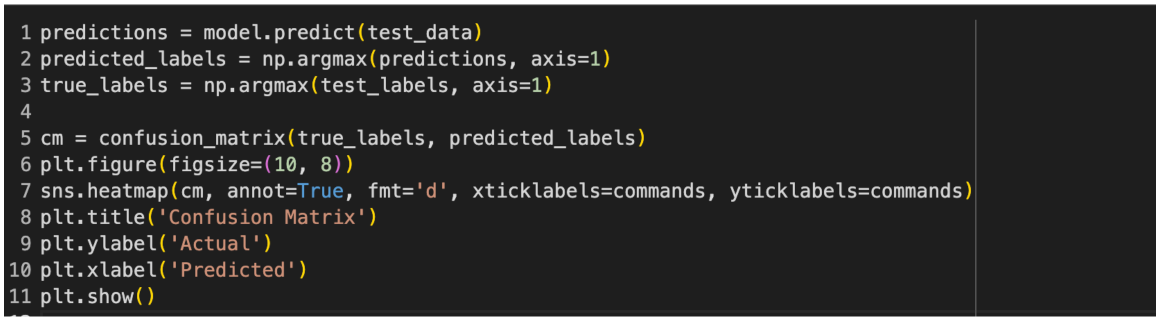
4.4. Additional Models and Comparison

Several models with slight variations in architecture are tested to find the best-performing model. The chosen model outperforms others in terms of both validation and test accuracy.

5. Evaluation and Results

5.1. Confusion Matrix

The confusion matrix provides a detailed breakdown of model performance across different classes, highlighting any misclassifications.

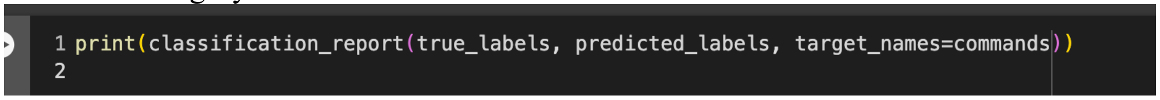


```
predictions = model.predict(test_data)
predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.argmax(test_labels, axis=1)

cm = confusion_matrix(true_labels, predicted_labels)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=commands, yticklabels=commands)
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

5.2. Classification Report

A classification report is generated to provide precision, recall, and F1-score for each command category.



```python
print(classification_report(true_labels, predicted_labels, target_names=commands))

```

6. Conclusion

1. **Model Performance**: The final model achieves an accuracy of **86%** on the test data, which is a strong result for audio classification tasks.
2. **EDA Insights**: Mel-spectrograms proved to be effective features for the classification of audio commands.
3. **Data Augmentation**: Augmentation techniques like adding noise and shifting pitch contributed to improved model generalization.
4. **Future Work**: Future improvements could involve experimenting with more complex architectures like recurrent neural networks (RNNs) or transformers, fine-tuning hyperparameters, and further data augmentation strategies.

7. References

1. TensorFlow Speech Commands Dataset: Link
2. Librosa Documentation: Link
3. TensorFlow Documentation: Link