LAB EVALUATION - KUSH GUPTA - 4CO26 - 103103734

Summary of the research paper:

The "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition" dataset, published by Pete Warden, provides a large corpus of one-second long utterances of short words which essentially act as keywords. The dataset is designed for building and evaluating keyword-spotting systems. It includes more than 65,000 utterances from thousands of different speakers, focusing on recognizing basic commands like "yes," "no," "stop," and others.

CNN Based multi-class classification of the voice samples:

Code:

```
#import required libraries
import os
import librosa
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import layers, models

print('Modules imported')

#load and pre-process the data
def load_audio_files_from_folder(folder_path,
sample_rate=16000):
    audio_files = []
    labels = []
    class_names = sorted(os.listdir(folder_path))  # Assuming
folder names are the class labels
```

```
    for class_name in class_names:
        class_folder = os.path.join(folder_path, class_name)
        if os.path.isdir(class_folder):
            for file_name in os.listdir(class_folder):
                if file_name.endswith('.wav'):
                    file_path = os.path.join(class_folder, file_name)
                    audio, _ = librosa.load(file_path, sr=sample_rate)

                    # Ensure the audio is 1 second long
                    if len(audio) > sample_rate:
                        audio = audio[:sample_rate]
                    elif len(audio) < sample_rate:
                        padding = sample_rate - len(audio)
                        audio = np.pad(audio, (0, padding), 'constant')

                    # Normalize the audio
                    audio = audio / np.max(np.abs(audio))

                    audio_files.append(audio)
                    labels.append(class_name)

    return np.array(audio_files), np.array(labels)
```

```python
train_dir = 'speech_commands_v0.02'
audios, labels = load_audio_files_from_folder(train_dir)
print("Audio data shape:", audios.shape)
print("Labels shape:", labels.shape)
print('Data loaded')

label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)

print("Class names:", label_encoder.classes_)

print("Labels encoded")

train_audios, val_audios, train_labels, val_labels = train_test_split(
    audios, labels_encoded, test_size=0.2, random_state=42, stratify=labels_encoded
)

print(f"Training set size: {train_audios.shape}")
print(f"Validation set size: {val_audios.shape}")


train_dataset = tf.data.Dataset.from_tensor_slices((train_audios, train_labels))
val_dataset = tf.data.Dataset.from_tensor_slices((val_audios, val_labels))
```

```python
BATCH_SIZE = 64
train_dataset = train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)

print("Converted to tensorflow dataset")


model = models.Sequential([
    layers.Input(shape=(16000,)),
    layers.Reshape((16000, 1)),

    # First convolutional layer
    layers.Conv1D(filters=32, kernel_size=3, activation='relu', padding='same'),
    layers.MaxPooling1D(pool_size=4),

    # Second convolutional layer
    layers.Conv1D(filters=64, kernel_size=3, activation='relu', padding='same'),
    layers.MaxPooling1D(pool_size=4),

    # Third convolutional layer
    layers.Conv1D(filters=128, kernel_size=3, activation='relu', padding='same'),
    layers.MaxPooling1D(pool_size=4),

    # Flattening
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),

    # Output layer
    layers.Dense(len(label_encoder.classes_), activation='softmax')
])


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


model.summary()

print("Model constructed")


EPOCHS = 10

history = model.fit(
    train_dataset,
    epochs=EPOCHS,
    validation_data=val_dataset
)
print("Model trained")
```

```
model.save('keyword_recognition_cnn_2.h5')
print("Model saved successfully!")
```

Data collection of my own voice samples:

Code:

```
import sounddevice as sd
import wavio
import os
```

```
Function to record audio
def record_audio(filename, duration=2, fs=16000):
    print(f"Recording {filename} for {duration} seconds...")
    recording = sd.rec(int(duration * fs), samplerate=fs, channels=1)
    sd.wait()  # Wait until the recording is finished
    wavio.write(filename, recording, fs, sampwidth=2)
    print(f"Saved {filename}")
```

```python
# Function to create directories and record for each label
def create_dataset(labels, num_samples_per_label=10, duration=2, dataset_dir="my_custom_dataset"):
    if not os.path.exists(dataset_dir):
        os.makedirs(dataset_dir)

    for label in labels:
        label_dir = os.path.join(dataset_dir, label)
        if not os.path.exists(label_dir):
            os.makedirs(label_dir)

        for i in range(num_samples_per_label):
            filename = os.path.join(label_dir, f"{label}_sample_{i + 1}.wav")
            record_audio(filename, duration=duration)
            print(f"Recorded and saved {filename}")


# List of labels/words you want to record
labels = ['_background_noise_','backward','bed','bird','cat','dog','down','eight','five','follow', 'forward', 'four' ,'go', 'happy', 'house', 'learn', 'left','marvin' ,'nine' ,'no', 'off', 'on' ,'one', 'right', 'seven' ,'sheila', 'six','stop', 'three', 'tree', 'two', 'up', 'visual', 'wow', 'yes', 'zero']
create_dataset(labels, num_samples_per_label=30)  # Record 5 samples per word
```

Fine Tuning the model on my own data:

Code:

```
import os
import numpy as np
import librosa
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
def load_audio_files_from_folder(folder_path, sample_rate=16000):
    audio_files = []
    labels = []
    class_names = sorted(os.listdir(folder_path))

    for class_name in class_names:
        class_folder = os.path.join(folder_path, class_name)
        if os.path.isdir(class_folder):
            for file_name in os.listdir(class_folder):
                if file_name.endswith('.wav'):
                    file_path = os.path.join(class_folder, file_name)
                    audio, _ = librosa.load(file_path, sr=sample_rate)

                    # Ensure audio is 1 second long
                    if len(audio) > sample_rate:
                        audio = audio[:sample_rate]
                    elif len(audio) < sample_rate:
                        padding = sample_rate - len(audio)
                        audio = np.pad(audio, (0, padding), 'constant')

                    # Normalize audio
                    audio = audio / np.max(np.abs(audio))

                    audio_files.append(audio)
                    labels.append(class_name)

    return np.array(audio_files), np.array(labels)


dataset_dir = 'my_custom_dataset'
audios, labels = load_audio_files_from_folder(dataset_dir)

print("Audio data shape:", audios.shape)
print("Labels shape:", labels.shape)
print('Pre processing done')

from sklearn.preprocessing import LabelEncoder


label_encoder = LabelEncoder()
```

```
labels_encoded = label_encoder.fit_transform(labels)


print("Class names:", label_encoder.classes_)
print('Labels encoded')

from sklearn.model_selection import train_test_split

train_audios, val_audios, train_labels, val_labels = train_test_split(
    audios, labels_encoded, test_size=0.2, random_state=42, stratify=labels_encoded
)

print(f"Training set size: {train_audios.shape}")
print(f"Validation set size: {val_audios.shape}")
print('splitted data')

import tensorflow as tf
from tensorflow.keras import layers, models


pretrained_model = tf.keras.models.load_model('keyword_recognition_cnn.h5')


for layer in pretrained_model.layers[:-2]:
    layer.trainable = False
```

```
model = models.Sequential(pretrained_model.layers[:-2])
```

```
model.add(layers.Flatten(name='new_flatten'))
model.add(layers.Dense(128, activation='relu', name='new_dense_1'))
model.add(layers.Dropout(0.3, name='new_dropout'))
model.add(layers.Dense(len(label_encoder.classes_), activation='softmax', name='new_output'))
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.summary()
```

```
print('model trained')
```

```
train_dataset = tf.data.Dataset.from_tensor_slices((train_audios, train_labels))
val_dataset = tf.data.Dataset.from_tensor_slices((val_audios, val_labels))
```

```
BATCH_SIZE = 32
train_dataset = train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)


EPOCHS = 100

history = model.fit(
    train_dataset,
    epochs=EPOCHS,
    validation_data=val_dataset
)
print('Model trained')


model.save('finetuned_model.h5')
print("Model fine-tuned and saved successfully!")

val_loss, val_acc = model.evaluate(val_dataset)
print(f"Validation Loss: {val_loss}")
print(f"Validation Accuracy: {val_acc}")
```

```
# Get predictions from the model
y_true = []
y_pred = []

for x, y in val_dataset:
    y_true.extend(y.numpy())  # Get true labels
    y_pred.extend(np.argmax(model.predict(x), axis=1))  # Get predicted labels

y_true = np.array(y_true)
y_pred = np.array(y_pred)

# Calculate metrics

# Accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy}")

# Precision, Recall, and F1-score
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
# Classification Report
print("\nClassification Report:")
print(classification_report(y_true, y_pred,
target_names=label_encoder.classes_))

# Confusion Matrix
print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_true, y_pred)
print(conf_matrix)
print('Metrics calculated')
```