

Summary:

The paper discusses speech command recognition using machine learning, based on a dataset of one-second audio clips containing 30 short words. It highlights deep learning techniques for classifying commands, making it suitable for voice-activated systems. The work focuses on improving accuracy, robustness, and real-time performance in various environments.

```
# Download the dataset
!wget http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz

# Create a new directory for extraction
!mkdir speech_commands_dataset

# Extract the dataset into the new directory
!tar -xvzf speech_commands_v0.02.tar.gz -C speech_commands_dataset

[111] Streaming output truncated to the last 5000 lines.
... ./on/fc3ba625_nohash_0.wav
./on/d9d952d2_nohash_0.wav
./on/72242187_nohash_4.wav
./on/b8f5b164_nohash_0.wav
./on/95aa8c90_nohash_1.wav
./on/525eaa67_nohash_0.wav
./on/42e31669_nohash_0.wav
./on/94d3780f_nohash_4.wav
./on/bf8d5617_nohash_0.wav
./on/e2b7d6d6_nohash_2.wav
./on/89ed36ab_nohash_0.wav
./on/387d6d03_nohash_1.wav
./on/3d4c0853_nohash_0.wav
./on/7cbf6454_nohash_0.wav
./on/c98b8bda_nohash_2.wav
./on/9a7c1493_nohash_5.wav
./on/3589bc72_nohash_0.wav
./on/3ae5c84f_nohash_0.wav
./on/adcc216c5_nohash_2.wav
./on/c58f55b8_nohash_15.wav
./on/324218d4_nohash_2.wav
./on/c120e80e_nohash_2.wav
./on/69886e40_nohash_0.wav
./on/fce85676_nohash_1.wav
./on/8dc216c5_nohash_2.wav
./on/c58f55b8_nohash_15.wav
./on/324218d4_nohash_2.wav
./on/c120e80e_nohash_2.wav
./on/69886e40_nohash_0.wav
./on/dce85676_nohash_1.wav
...
./follow/3d794813_nohash_1.wav
./follow/2927c001_nohash_2.wav
./follow/2acae72_nohash_2.wav
./follow/ceef6090_nohash_3.wav
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings.

import os

# Define the correct path to the extracted dataset
dataset_path = 'speech_commands_dataset' # Adjusted for the new extraction directory

# List available commands (subdirectories within the dataset folder)
commands = os.listdir(dataset_path)
print(f'Available commands: {commands}')

[133] Available commands: ['no', 'right', 'left', 'zero', 'seven', 'forward', 'six', '_background_noise_', 'two', 'wow', 'happy', 'four', 'one', 'down', 'sheila', 'learn', 'go', 'bed', 'validation_1']
...

commands = [d for d in os.listdir(dataset_path) if os.path.isdir(os.path.join(dataset_path, d))]

[143]

samples_per_command = {command: len(os.listdir(os.path.join(dataset_path, command))) for command in commands}

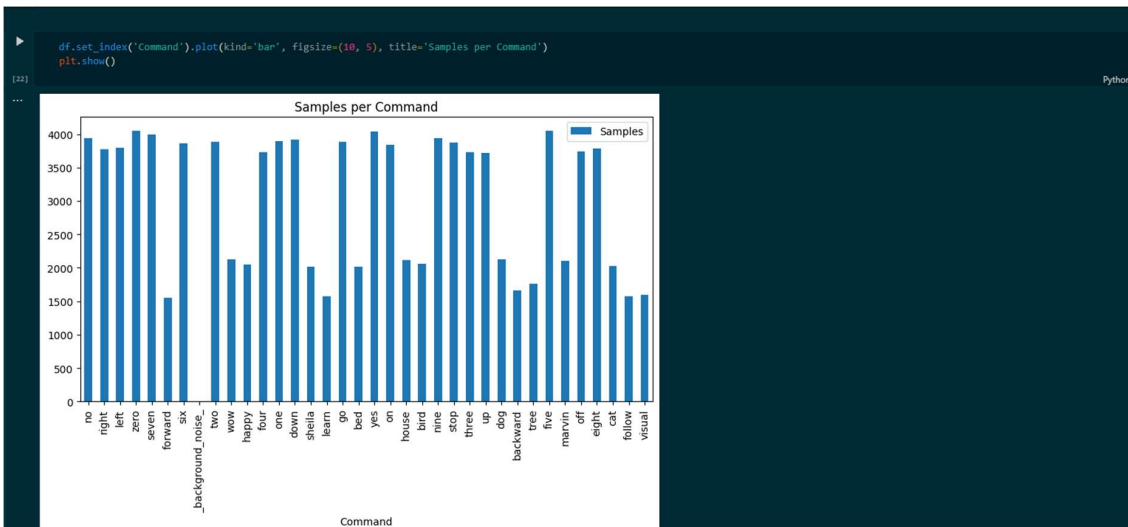
[137]

print(f'Available commands: {commands}')
print(f'Samples per command: {samples_per_command}')

[143] Available commands: ['no', 'right', 'left', 'zero', 'seven', 'forward', 'six', '_background_noise_', 'two', 'wow', 'happy', 'four', 'one', 'down', 'sheila', 'learn', 'go', 'bed', 'yes', 'on', '']
Samples per command: {'no': 3941, 'right': 3778, 'left': 3881, 'zero': 4052, 'seven': 3998, 'forward': 1557, 'six': 3860, '_background_noise_': 7, 'two': 3880, 'wow': 2123, 'happy': 2054, 'four': }

import os
import pandas as pd
import matplotlib.pyplot as plt
df = pd.DataFrame(list(samples_per_command.items()), columns=['Command', 'Samples'])

[141]
```



```
import os
import numpy as np
import pandas as pd
import librosa
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Define constants
DATA_DIR = './speech_commands_dataset'
SAMPLE_RATE = 16000
DURATION = 1.0
N_MFCC = 13
N_FFT = 2048
HOP_LENGTH = 512

def load_data(data_dir):
    data = []
    labels = []
    categories = [d for d in os.listdir(data_dir) if os.path.isdir(os.path.join(data_dir, d))]
    for category in categories:
        path = os.path.join(data_dir, category)
        for file in os.listdir(path):
            if file.endswith('.wav'):
                file_path = os.path.join(path, file)
                audio, _ = librosa.load(file_path, sr=SAMPLE_RATE, duration=DURATION)
                if len(audio) == SAMPLE_RATE:
                    mfccs = librosa.feature.mfcc(y=audio, sr=SAMPLE_RATE, n_mfcc=N_MFCC, n_fft=N_FFT, hop_length=HOP_LENGTH)
                    data.append(mfccs)
                    labels.append(category)
    return np.array(data), np.array(labels)
```

```
# Load and preprocess data
X, y = load_data(DATA_DIR)
X = np.expand_dims(X, axis=-1) # Add channel dimension for CNN

# Adjust X shape if necessary
if len(X.shape) == 3:
    # If X is already in the shape (samples, time_steps, features)
    pass
elif len(X.shape) == 4:
    # If X is in the shape (samples, features, time_steps, 1)
    X = np.squeeze(X, axis=-1)
    X = X.transpose(0, 2, 1)
else:
    raise ValueError(f"Unexpected shape of X: {X.shape}")

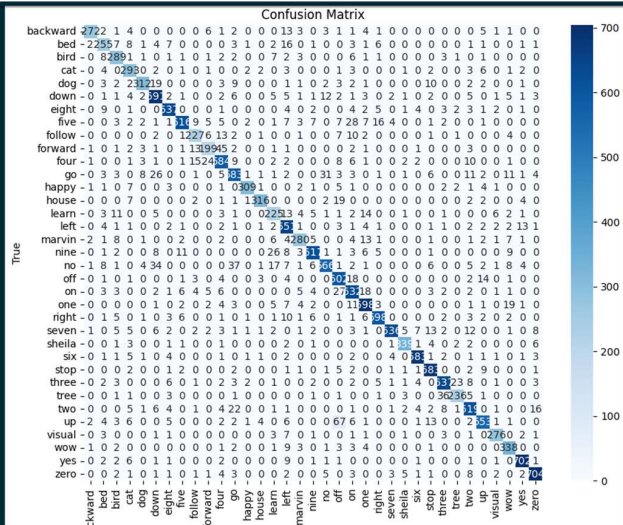
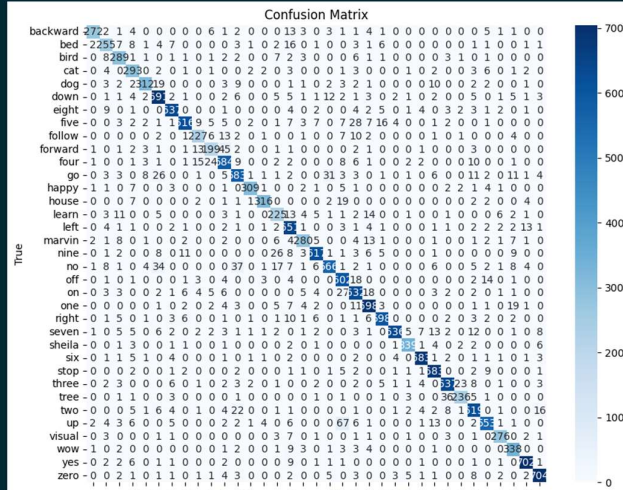
print(f"Adjusted X shape: {X.shape}")

# Encode labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
```

Adjusted X shape: (95400, 32, 13)

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings.



```

import os
import numpy as np
import librosa

Tabnine: Edit | Test | Explain | Document | Ask
def load_custom_data(data_dir, sample_rate=16000, duration=1.0, n_mfcc=13, n_fft=2048, hop_length=512):
    data = []
    filenames = []
    for command in os.listdir(data_dir):
        command_dir = os.path.join(data_dir, command)
        if os.path.isdir(command_dir):
            for file in os.listdir(command_dir):
                if file.endswith('.wav'):
                    file_path = os.path.join(command_dir, file)
                    audio, _ = librosa.load(file_path, sr=sample_rate, duration=duration)
                    if len(audio) == sample_rate:
                        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=n_mfcc, n_fft=n_fft, hop_length=hop_length)
                        data.append(mfccs)
                        filenames.append(file)
    return np.array(data), np.array(filenames)

# Load your custom dataset
CUSTOM_DATA_DIR = '/content/extracted_files/VoiceCommandDataset_User110'
X_custom, filenames_custom = load_custom_data(CUSTOM_DATA_DIR)

# Preprocess the data
X_custom = np.expand_dims(X_custom, axis=-1)
if len(X_custom.shape) == 4:
    X_custom = np.squeeze(X_custom, axis=-1)
    X_custom = X_custom.transpose(0, 2, 1)

print("Custom dataset shape:", X_custom.shape)

```

... Custom dataset shape: (934, 32, 13)

```

import os
import numpy as np
import pandas as pd
import librosa
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Define constants
DATA_DIR = '/content/extracted_files/VoiceCommandDataset_User110'
SAMPLE_RATE = 16000
DURATION = 1.0
N_MFCC = 13
N_FFT = 2048
HOP_LENGTH = 512

Tabnine: Edit | Test | Explain | Document | Ask
def load_data(data_dir):
    data = []
    labels = []
    categories = [d for d in os.listdir(data_dir) if os.path.isdir(os.path.join(data_dir, d))]
    for category in categories:
        path = os.path.join(data_dir, category)
        for file in os.listdir(path):
            if file.endswith('.wav'):
                file_path = os.path.join(path, file)
                audio, _ = librosa.load(file_path, sr=SAMPLE_RATE, duration=DURATION)
                if len(audio) == SAMPLE_RATE:
                    mfccs = librosa.feature.mfcc(y=audio, sr=SAMPLE_RATE, n_mfcc=N_MFCC, n_fft=N_FFT, hop_length=HOP_LENGTH)
                    data.append(mfccs)
                    labels.append(category)

```

[3]

```

# Adjust X shape if necessary
if len(X.shape) == 3:
    # If X is already in the shape (samples, time_steps, features)
    pass
elif len(X.shape) == 4:
    # If X is in the shape (samples, features, time_steps, 1)
    X = np.squeeze(X, axis=-1)
    X = X.transpose(0, 2, 1)
else:
    raise ValueError(f"Unexpected shape of X: {X.shape}")

print("Adjusted X shape:", X.shape)

# Encode labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

```

[4]

... Adjusted X shape: (934, 32, 13)

Python

```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X.shape[1], X.shape[2])),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(len(np.unique(y)), activation='softmax')
])

# Print model summary
model.summary()

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

# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2, batch_size=32)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 30, 32)	1,280
max_pooling1d (MaxPooling1D)	(None, 15, 32)	0
conv1d_1 (Conv1D)	(None, 13, 64)	6,208
max_pooling1d_1 (MaxPooling1D)	(None, 6, 64)	0
max_pooling1d_1 (MaxPooling1D)	(None, 6, 64)	0
conv1d_2 (Conv1D)	(None, 4, 64)	12,352
global_average_pooling1d (GlobalAveragePooling1D)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dense_1 (Dense)	(None, 25)	1,625

Total params: 25,625 (100.10 KB)

Trainable params: 25,625 (100.10 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10
19/19 — 3s 21ms/step - accuracy: 0.0354 - loss: 25.3599 - val_accuracy: 0.0133 - val_loss: 4.6275

Epoch 2/10
19/19 — 0s 8ms/step - accuracy: 0.1340 - loss: 3.3097 - val_accuracy: 0.2867 - val_loss: 2.5115

Epoch 3/10
19/19 — 0s 8ms/step - accuracy: 0.4311 - loss: 2.1764 - val_accuracy: 0.5067 - val_loss: 1.8559

Epoch 4/10
19/19 — 0s 15ms/step - accuracy: 0.6526 - loss: 1.5065 - val_accuracy: 0.5867 - val_loss: 1.2995

Epoch 5/10
19/19 — 0s 13ms/step - accuracy: 0.7545 - loss: 0.9469 - val_accuracy: 0.8200 - val_loss: 0.7571

Epoch 6/10
19/19 — 0s 14ms/step - accuracy: 0.9148 - loss: 0.5306 - val_accuracy: 1.0000 - val_loss: 0.3373

Epoch 7/10
19/19 — 1s 17ms/step - accuracy: 0.9979 - loss: 0.2242 - val_accuracy: 1.0000 - val_loss: 0.1464

Epoch 8/10
19/19 — 1s 16ms/step - accuracy: 1.0000 - loss: 0.0922 - val_accuracy: 1.0000 - val_loss: 0.0683

Layer (type)	Output Shape	Param #
max_pooling1d_1 (MaxPooling1D)	(None, 6, 64)	0
conv1d_2 (Conv1D)	(None, 4, 64)	12,352
global_average_pooling1d (GlobalAveragePooling1D)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dense_1 (Dense)	(None, 25)	1,625

Total params: 25,625 (100.10 KB)

Trainable params: 25,625 (100.10 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10
19/19 — 3s 21ms/step - accuracy: 0.0354 - loss: 25.3599 - val_accuracy: 0.0133 - val_loss: 4.6275

Epoch 2/10
19/19 — 0s 8ms/step - accuracy: 0.1340 - loss: 3.3097 - val_accuracy: 0.2867 - val_loss: 2.5115

Epoch 3/10
19/19 — 0s 8ms/step - accuracy: 0.4311 - loss: 2.1764 - val_accuracy: 0.5067 - val_loss: 1.8559

Epoch 4/10
19/19 — 0s 15ms/step - accuracy: 0.6526 - loss: 1.5065 - val_accuracy: 0.5867 - val_loss: 1.2995

Epoch 5/10
19/19 — 0s 13ms/step - accuracy: 0.7545 - loss: 0.9469 - val_accuracy: 0.8200 - val_loss: 0.7571

Epoch 6/10
19/19 — 0s 14ms/step - accuracy: 0.9148 - loss: 0.5306 - val_accuracy: 1.0000 - val_loss: 0.3373

Epoch 7/10
19/19 — 1s 17ms/step - accuracy: 0.9979 - loss: 0.2242 - val_accuracy: 1.0000 - val_loss: 0.1464

Epoch 8/10
19/19 — 1s 16ms/step - accuracy: 1.0000 - loss: 0.0922 - val_accuracy: 1.0000 - val_loss: 0.0683

```

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')

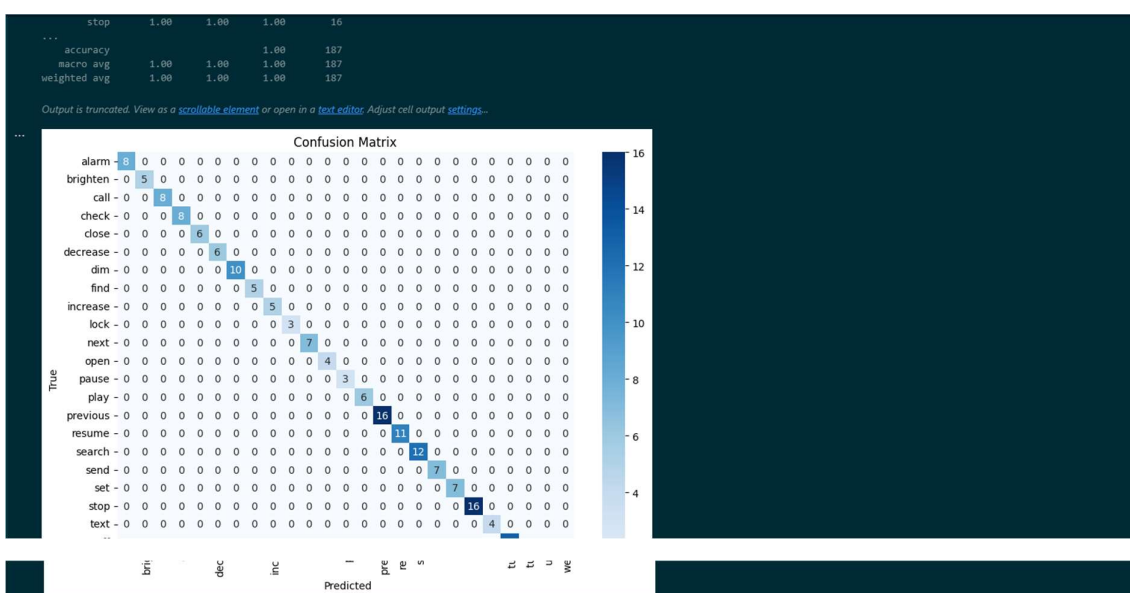
# Generate classification report
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_classes, target_names=le.classes_))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

```

6/6 0s 5ms/step - accuracy: 1.0000 - loss: 0.0222
Test accuracy: 1.0
6/6 0s 32ms/step

	precision	recall	f1-score	support
alarm	1.00	1.00	1.00	8
brighten	1.00	1.00	1.00	5
call	1.00	1.00	1.00	8
check	1.00	1.00	1.00	8
close	1.00	1.00	1.00	6
decrease	1.00	1.00	1.00	6
dim	1.00	1.00	1.00	10
find	1.00	1.00	1.00	5



```

loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test accuracy: {accuracy}')

```

Test accuracy: 0.89

```

import tensorflow as tf
import numpy as np
import os
import hashlib

# Define the save path
save_path = './saved_model_and_data'
os.makedirs(save_path, exist_ok=True)

# Save model
model_path = os.path.join(save_path, 'speech_command_model.keras')
model.save(model_path)
print(f'Model saved to {model_path}')

# Save dataset
dataset_path = os.path.join(save_path, 'speech_command_dataset.npz')
np.savez(dataset_path, X=X, y=y)
print(f'Dataset saved to {dataset_path}')

# Function to compute MD5 checksum
def compute_md5(file_path):
    with open(file_path, 'rb') as file:
        md5_hash = hashlib.md5()
        for chunk in iter(lambda: file.read(4096), b''):
            md5_hash.update(chunk)
    return md5_hash.hexdigest()

```

p

```
def main():  
    # Compute and save checksums  
    checksums = {}  
    checksums['speech_command_model.keras'] = compute_md5(model_path)  
    checksums['speech_command_dataset.npz'] = compute_md5(dataset_path)  
  
    checksum_path = os.path.join(save_path, 'checksums.txt')  
    with open(checksum_path, 'w') as f:  
        for file_name, checksum in checksums.items():  
            f.write(f"{file_name}: {checksum}\n")  
  
    print(f"Checksums saved to {checksum_path}")  
[18]  
... Model saved to ./saved_model_and_data/speech_command_model.keras  
Dataset saved to ./saved_model_and_data/speech_command_dataset.npz  
Checksums saved to ./saved_model_and_data/checksums.txt  
Python
```

Submitted By:

Jalaj Singh

4CO12

102103330