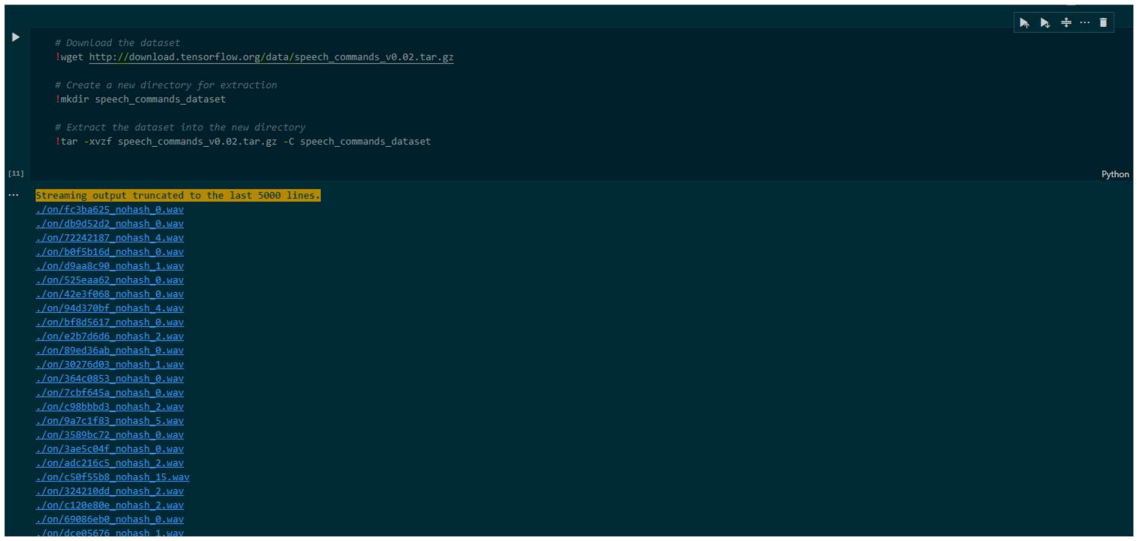
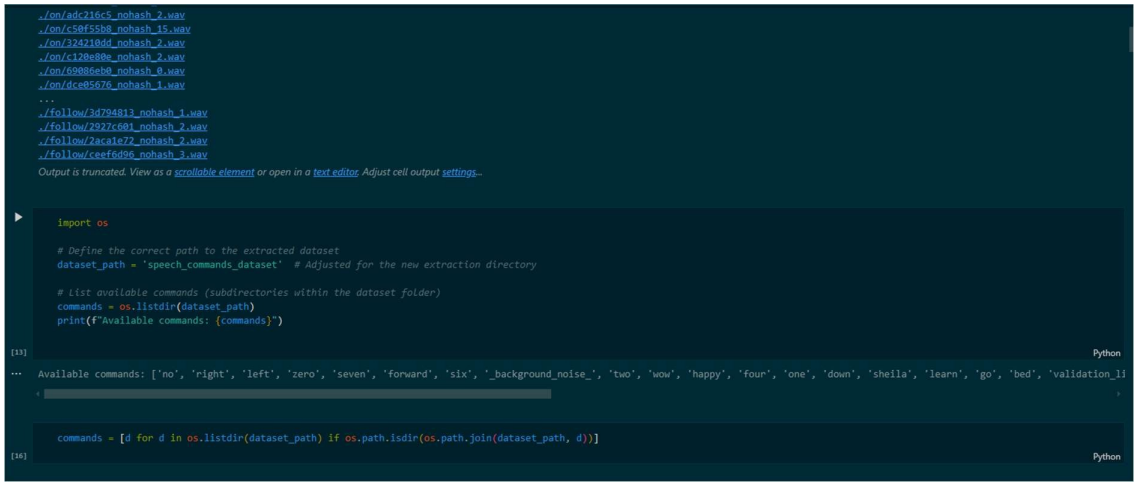
Summary:

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.

Drive_Link:

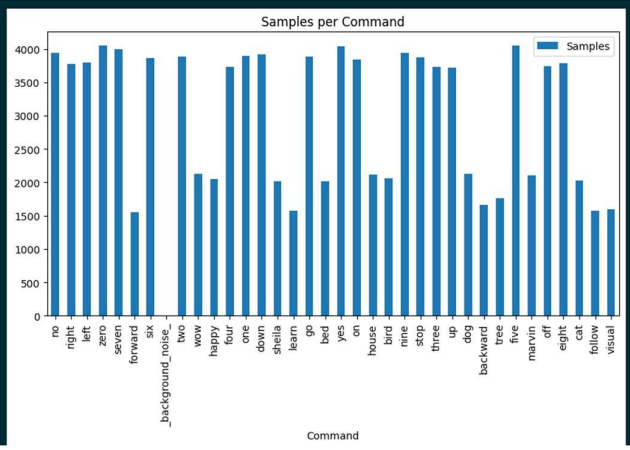
https://drive.google.com/drive/folders/1Uzv8tGJL9rNpt7RQauZ9r3W-Wtyttofl





```python
df.set_index('Command').plot(kind='bar', figsize=(10, 5), title='Samples per Command')
plt.show()
```

[22] Python




```python
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

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

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

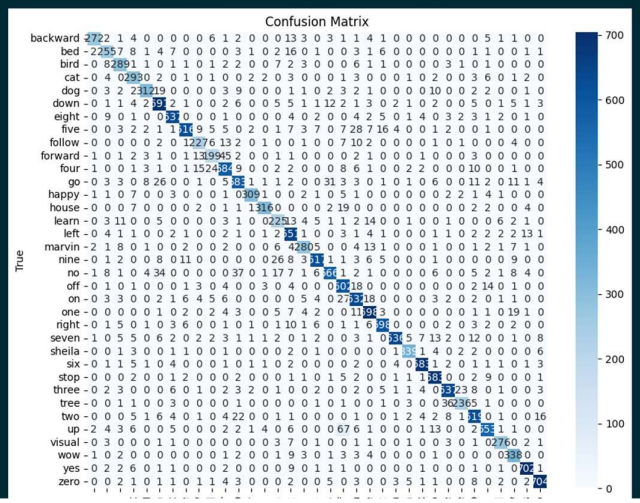
[97] Python

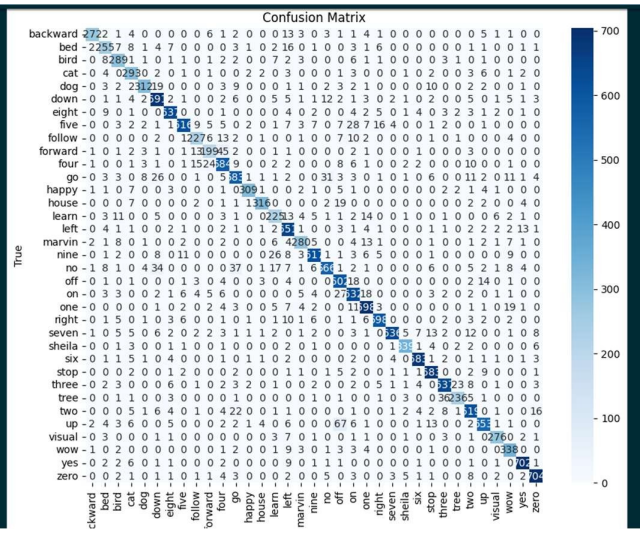
```python
# Adjust X shape if necessary
if len(X.shape) == 3:
    # If X is already in the shape (samples, time_steps, features)
    pass
elif len(X.shape) == 4:
    # If X is in the shape (samples, features, time_steps, 1)
    X = np.squeeze(X, axis=-1)
    X = X.transpose(0, 2, 1)
else:
    raise ValueError(f"Unexpected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

[98] Python

```
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 zipfile
import os

# Define the path to your ZIP file
zip_file_path = 'VoiceCommandDataset_User110.zip'  # Replace with your ZIP file name

# Define the directory to extract to
extract_to_dir = '/content/extracted_files'  # You can choose any directory

# Create the directory if it does not exist
if not os.path.exists(extract_to_dir):
    os.makedirs(extract_to_dir)

# Extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to_dir)

print(f'Files extracted to {extract_to_dir}')
```

[1] Python

```
Files extracted to /content/extracted_files
```

```python
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)
```

```python
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]

```python
# 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] Python

```
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.Conv1D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(64, 3, activation='relu'),
    tf.keras.layers.GlobalAveragePooling1D(),
    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)
```

[5] Python

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
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: 0.0693
```


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)

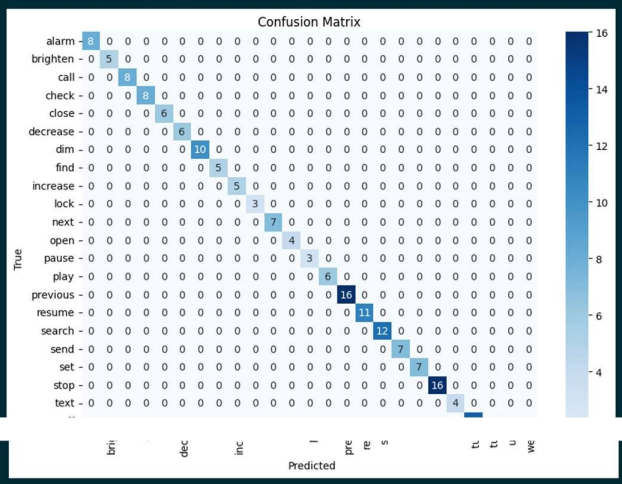
```
Epoch 1/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.0354 - loss: 25.3599 - val_accuracy: 0.0133 - val_loss: 4.6275
Epoch 2/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1340 - loss: 3.3097 - val_accuracy: 0.2867 - val_loss: 2.5115
Epoch 3/10
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - 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: 0.0693
```

```python
# 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] Python

```
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
        stop       1.00      1.00      1.00        16
...
    accuracy                           1.00       187
   macro avg       1.00      1.00      1.00       187
weighted avg       1.00      1.00      1.00       187
```

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



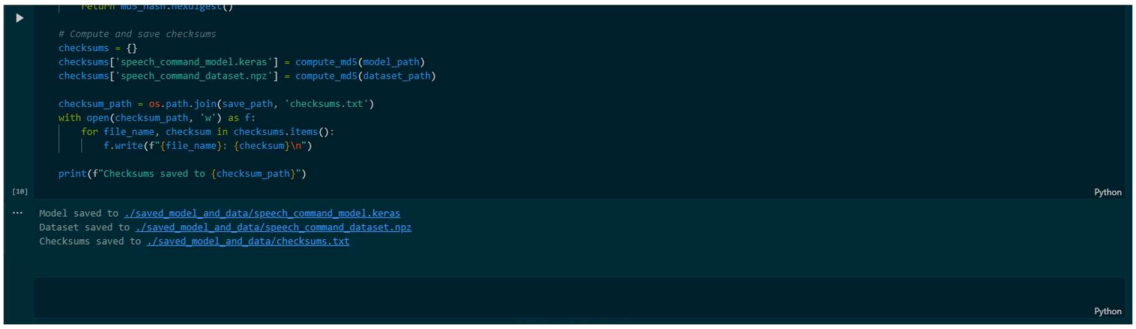

```python
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test accuracy: {accuracy}")
```

[7] Python

```
Test accuracy: 0.89
```

```python
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
Tabnine: Edit | Test | Explain | Document | Ask
def compute_md5(file_path):
    with open(file_path, 'rb') as file:
        md5_hash = hashlib.md5()
```

[10]



Submitted By:
Sukhmandeep
4CS10
102116056