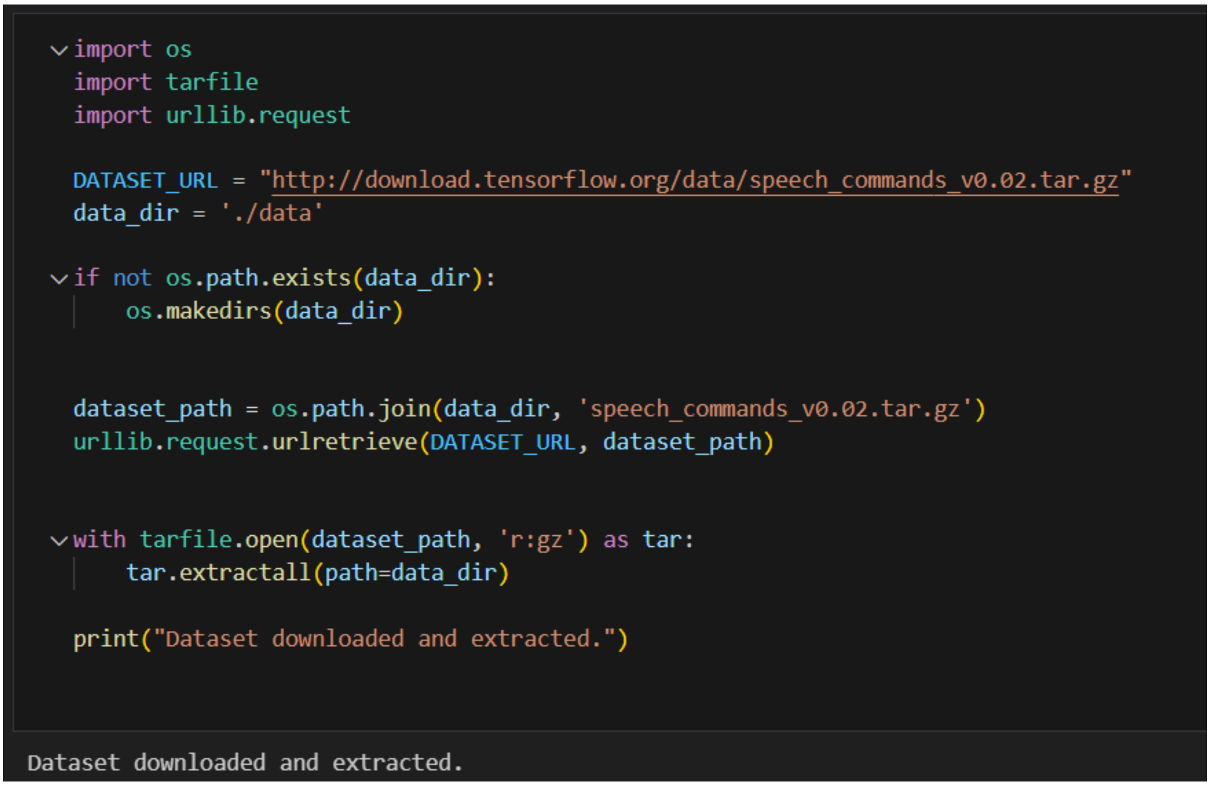
Research Paper Summary (https://arxiv.org/pdf/1804.03209)

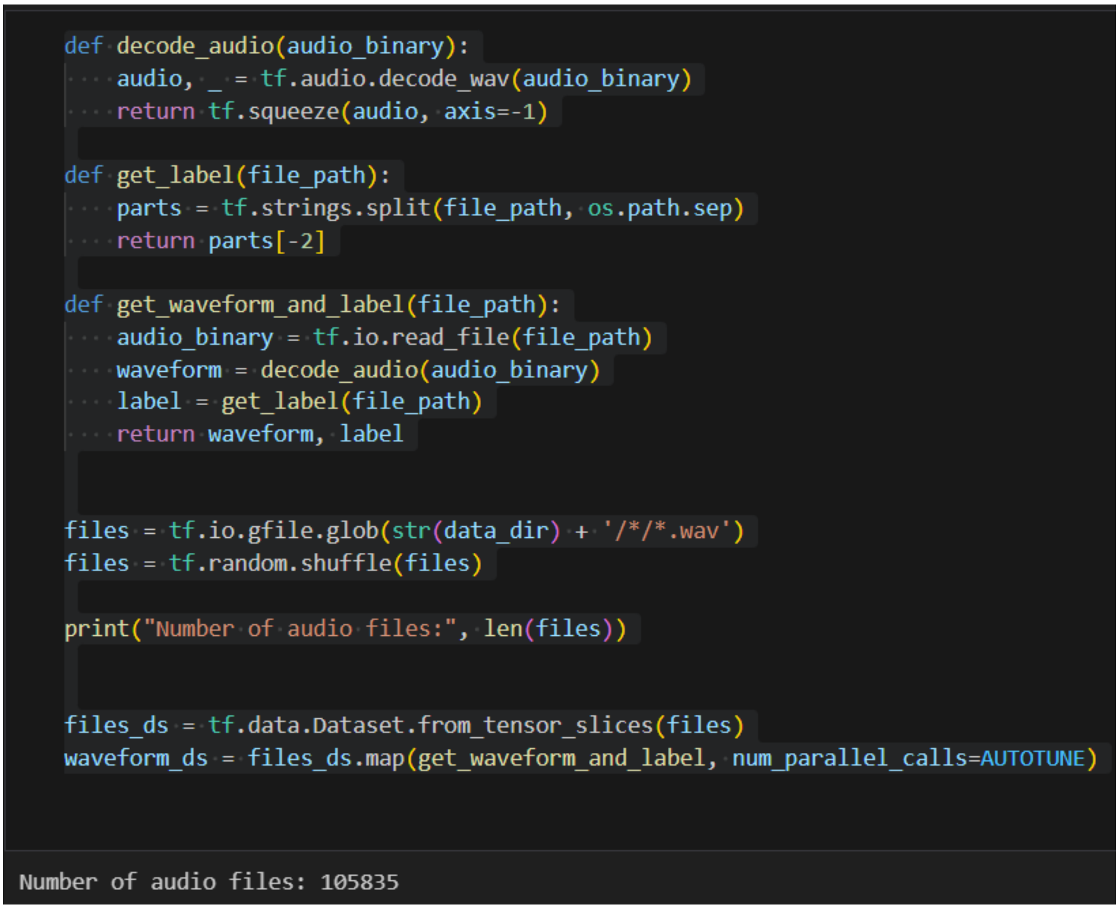
The Speech Commands dataset contains 105,829 utterances from 2,618 speakers, with audio files in WAVE format. It's divided into training, validation, and testing sets, ensuring consistent evaluation and avoiding overfitting. Evaluation metrics include matched percentage, wrong percentage, and false positives. The dataset enables comparisons across models and has been widely used in research on speech recognition, noise tolerance, and adversarial attacks.

Dataset Analysis



```python
import os
import tarfile
import urllib.request

DATASET_URL = "http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz"
data_dir = './data'

if not os.path.exists(data_dir):
    os.makedirs(data_dir)


dataset_path = os.path.join(data_dir, 'speech_commands_v0.02.tar.gz')
urllib.request.urlretrieve(DATASET_URL, dataset_path)


with tarfile.open(dataset_path, 'r:gz') as tar:
    tar.extractall(path=data_dir)

print("Dataset downloaded and extracted.")
```

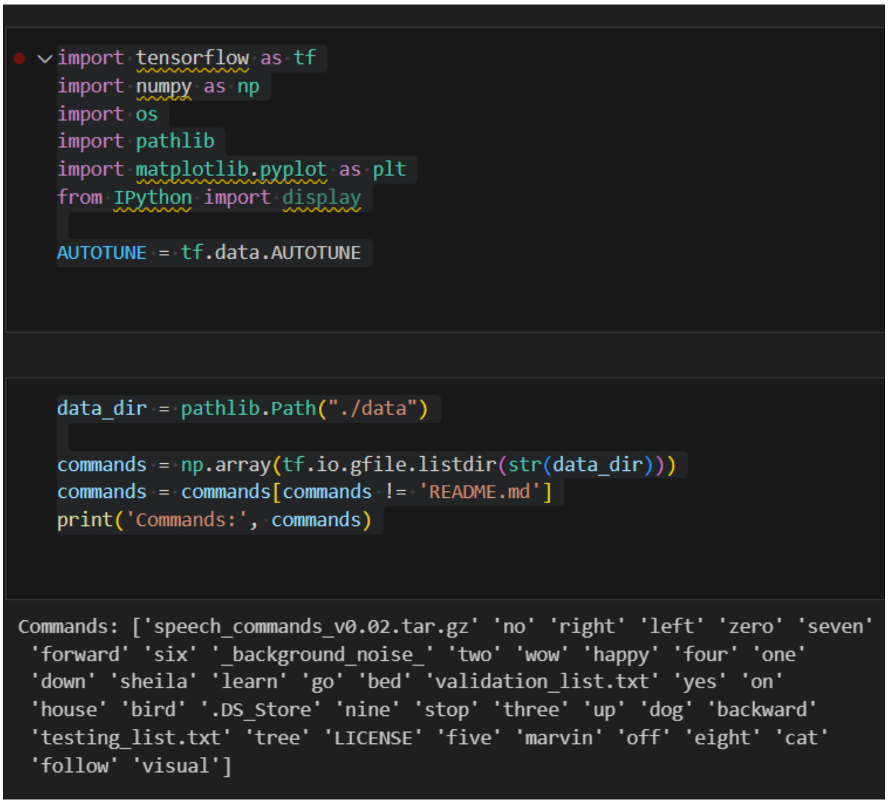
```
Dataset downloaded and extracted.
```



```python
def decode_audio(audio_binary):
    audio, _ = tf.audio.decode_wav(audio_binary)
    return tf.squeeze(audio, axis=-1)

def get_label(file_path):
    parts = tf.strings.split(file_path, os.path.sep)
    return parts[-2]

def get_waveform_and_label(file_path):
    audio_binary = tf.io.read_file(file_path)
    waveform = decode_audio(audio_binary)
    label = get_label(file_path)
    return waveform, label


files = tf.io.gfile.glob(str(data_dir) + '/*/*.wav')
files = tf.random.shuffle(files)

print("Number of audio files:", len(files))


files_ds = tf.data.Dataset.from_tensor_slices(files)
waveform_ds = files_ds.map(get_waveform_and_label, num_parallel_calls=AUTOTUNE)
```

```
Number of audio files: 105835
```

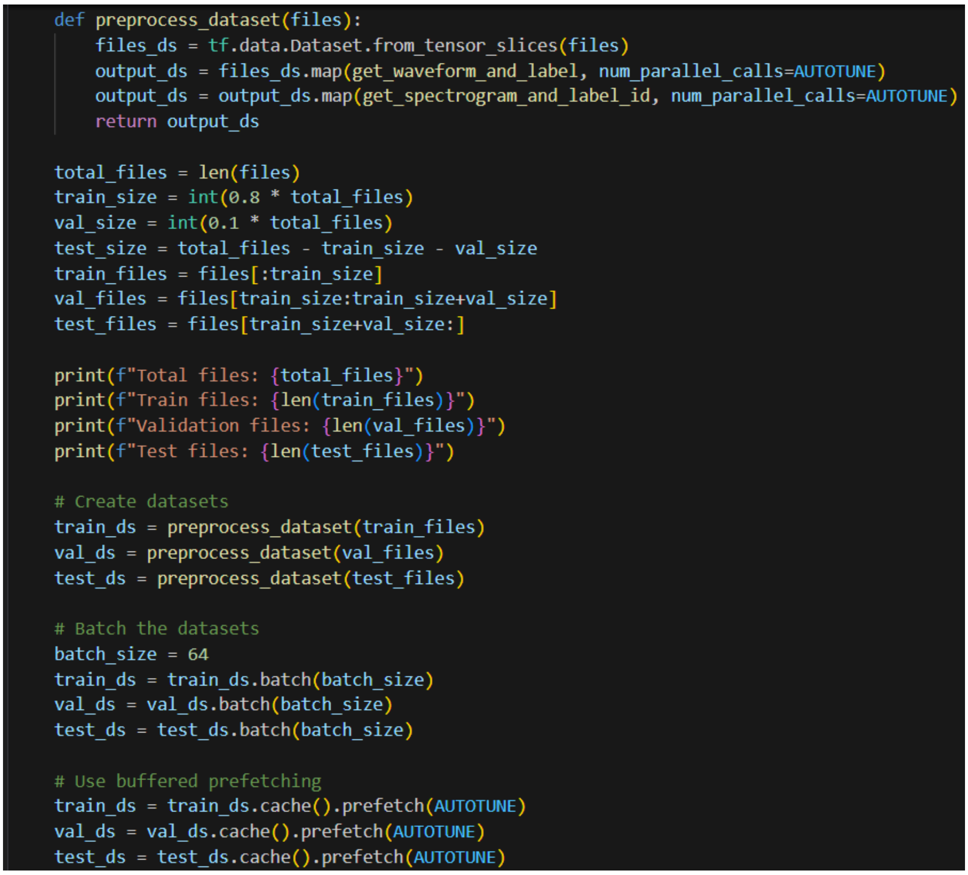
Classifier Training:



```
import tensorflow as tf
import numpy as np
import os
import pathlib
import matplotlib.pyplot as plt
from IPython import display

AUTOTUNE = tf.data.AUTOTUNE
```

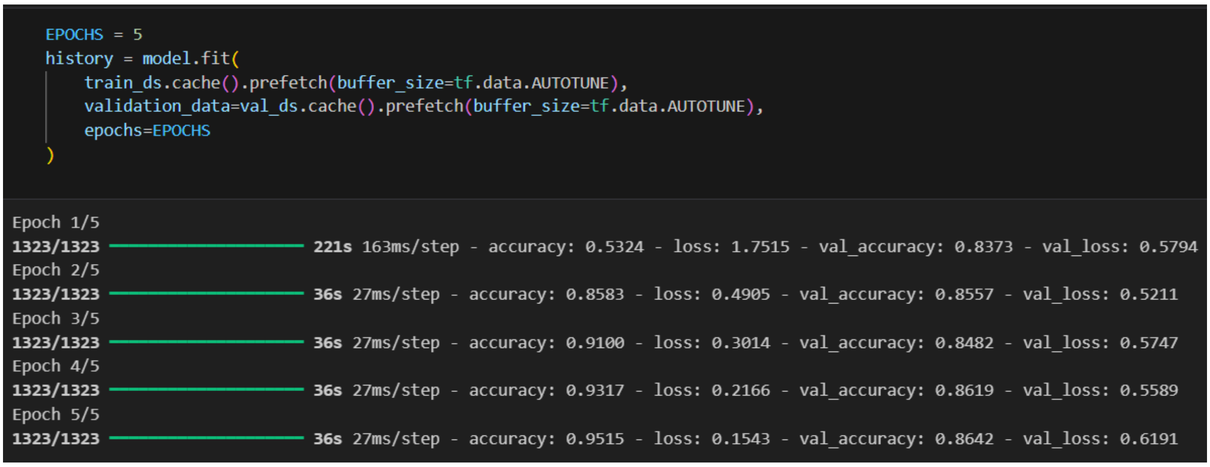
```
data_dir = pathlib.Path("./data")

commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[commands != 'README.md']
print('Commands:', commands)
```

```
Commands: ['speech_commands_v0.02.tar.gz' 'no' 'right' 'left' 'zero' 'seven'
 'forward' 'six' '_background_noise_' 'two' 'wow' 'happy' 'four' 'one'
 'down' 'sheila' 'learn' 'go' 'bed' 'validation_list.txt' 'yes' 'on'
 'house' 'bird' '.DS_Store' 'nine' 'stop' 'three' 'up' 'dog' 'backward'
 'testing_list.txt' 'tree' 'LICENSE' 'five' 'marvin' 'off' 'eight' 'cat'
 'follow' 'visual']
```



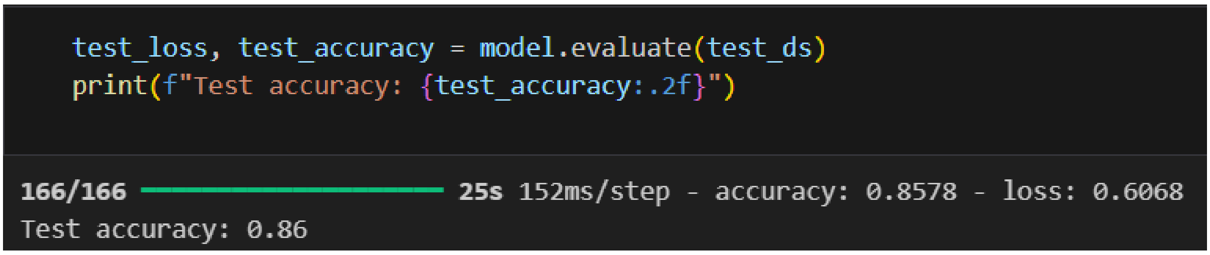
```
def preprocess_dataset(files):
    files_ds = tf.data.Dataset.from_tensor_slices(files)
    output_ds = files_ds.map(get_waveform_and_label, num_parallel_calls=AUTOTUNE)
    output_ds = output_ds.map(get_spectrogram_and_label_id, num_parallel_calls=AUTOTUNE)
    return output_ds

total_files = len(files)
train_size = int(0.8 * total_files)
val_size = int(0.1 * total_files)
test_size = total_files - train_size - val_size
train_files = files[:train_size]
val_files = files[train_size:train_size+val_size]
test_files = files[train_size+val_size:]

print(f"Total files: {total_files}")
print(f"Train files: {len(train_files)}")
print(f"Validation files: {len(val_files)}")
print(f"Test files: {len(test_files)}")

# Create datasets
train_ds = preprocess_dataset(train_files)
val_ds = preprocess_dataset(val_files)
test_ds = preprocess_dataset(test_files)

# Batch the datasets
batch_size = 64
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)
test_ds = test_ds.batch(batch_size)

# Use buffered prefetching
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)
test_ds = test_ds.cache().prefetch(AUTOTUNE)
```

Performance Results using standard benchmarks.



```
EPOCHS = 5
history = model.fit(
    train_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE),
    validation_data=val_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE),
    epochs=EPOCHS
)
```

```
Epoch 1/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 221s 163ms/step - accuracy: 0.5324 - loss: 1.7515 - val_accuracy: 0.8373 - val_loss: 0.5794
Epoch 2/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 36s 27ms/step - accuracy: 0.8583 - loss: 0.4905 - val_accuracy: 0.8557 - val_loss: 0.5211
Epoch 3/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 36s 27ms/step - accuracy: 0.9100 - loss: 0.3014 - val_accuracy: 0.8482 - val_loss: 0.5747
Epoch 4/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 36s 27ms/step - accuracy: 0.9317 - loss: 0.2166 - val_accuracy: 0.8619 - val_loss: 0.5589
Epoch 5/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 36s 27ms/step - accuracy: 0.9515 - loss: 0.1543 - val_accuracy: 0.8642 - val_loss: 0.6191
```



```
test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test accuracy: {test_accuracy:.2f}")
```

```
166/166 ━━━━━━━━━━━━━━━━━━━━ 25s 152ms/step - accuracy: 0.8578 - loss: 0.6068
Test accuracy: 0.86
```