Digits Speech Recognition on Google Speech Commands

This notebook demonstrates how to predict spoken digits from audio files using a neural network model. The process involves the following steps:

- Preprocessing the audio files from dataset to extract MFCC features.
- Loading a trained model.
- Using the model to predict the digit for a given audio file.

Let's get started!

Imports and Setup

We begin by importing the necessary libraries and setting up the environment for audio processing and model prediction.

In [ ]:
```
import pathlib
import seaborn
from IPython import display
from sklearn.preprocessing import LabelEncoder


import numpy as np
import os
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.utils import to_categorical
import librosa
import librosa.display
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
!kaggle datasets download -d neehakurelli/google-speech-commands --path /content/drive
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (s
erver 1.6.14 / client 1.6.12)
Dataset URL: https://www.kaggle.com/datasets/neehakurelli/google-speech-commands
License(s): unknown
Downloading google-speech-commands.zip to /content/drive/MyDrive/SpeechCommands
100% 1.38G/1.38G [00:24<00:00, 50.4MB/s]
100% 1.38G/1.38G [00:24<00:00, 60.0MB/s]
```

In [ ]:

```
import os
import zipfile

# Define the path to the zip file and the extraction directory
zip_path = '/content/drive/MyDrive/SpeechCommands/google-speech-commands.zip'
extract_dir = '/content/drive/MyDrive/SpeechCommands'

# Check if the zip file exists
if not os.path.exists(zip_path):
    print(f"Zip file not found at {zip_path}")
else:
    try:
        # Attempt to open and extract the zip file
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            zip_ref.extractall(extract_dir)
        print(f"Extraction completed successfully to {extract_dir}")
    except zipfile.BadZipFile:
        print(f"The file at {zip_path} is not a zip file or it is corrupted.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

```
Extraction completed successfully to /content/drive/MyDrive/SpeechCommands
```

In [ ]:

```
extract_dir = '/content/drive/MyDrive/SpeechCommands'
# List the extracted files and folders
extracted_files = os.listdir(extract_dir)
print("Extracted files and folders:")
print(len(extracted_files))
```

```
Extracted files and folders:
36
```

Data Preprocessing

This section covers the preprocessing of audio files to extract MFCC features. These features are essential for the model to make accurate predictions.

Mel-Frequency Cepstral Coefficients (MFCC)

MFCC is a representation of the short-term power spectrum of a sound. It's widely used in speech and audio processing because it effectively captures the timbral aspects of the sound. The process involves:

1. Taking the Fourier transform of a windowed signal.

2. Mapping the powers of the spectrum to the mel scale.
3. Taking the logarithm of the powers at each mel frequency.
4. Taking the discrete cosine transform of the resulting coefficients.

Loading and Preprocessing the Dataset

In this section, we load and preprocess the dataset. The dataset contains audio recordings of spoken digits ("zero" to "nine"). The preprocessing steps include:

1. Loading each audio file.
2. Extracting MFCC features from the audio.
3. Ensuring consistent shape of the MFCC features.

In [ ]:

```
DATASET_PATH = '/content/drive/My Drive/SpeechCommands'
DIGITS = ['zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nir

def load_data(path, digits):
    x_data, y_data = [], []
    for i, digit in enumerate(digits):
        digit_path = os.path.join(path, digit)
        for filename in os.listdir(digit_path):
            if filename.endswith('.wav'):
                file_path = os.path.join(digit_path, filename)
                y, sr = librosa.load(file_path, sr=16000)
                mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
                if mfcc.shape[1] == 32:  # Ensure consistent shape
                    x_data.append(mfcc)
                    y_data.append(i)
    return np.array(x_data), np.array(y_data)

x_data, y_data = load_data(DATASET_PATH, DIGITS)
x_data = x_data[..., np.newaxis]  # Add channel dimension
y_data = to_categorical(y_data, num_classes=len(DIGITS))

# Split data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2, rar
```

Model Architecture

The model used for digit recognition is a Convolutional Neural Network (CNN). Here's a detailed breakdown of each layer:

- **Conv2D (32, (3, 3), activation='relu', input_shape=(40, 32, 1)):** The first convolutional layer with 32 filters, a kernel size of (3, 3), ReLU activation, and an input shape matching the MFCC feature dimensions.

- **MaxPooling2D ((2, 2)):** A max pooling layer with a pool size of (2, 2) to reduce the spatial dimensions.
- **Dropout (0.25):** A dropout layer with a dropout rate of 0.25 to prevent overfitting.
- **Conv2D (64, (3, 3), activation='relu'):** The second convolutional layer with 64 filters and a kernel size of (3, 3), again with ReLU activation.
- **MaxPooling2D ((2, 2)):** Another max pooling layer with a pool size of (2, 2).
- **Dropout (0.25):** Another dropout layer with a dropout rate of 0.25 to further prevent overfitting.
- **Flatten ():** A flatten layer to convert the 2D matrix into a 1D vector for the fully connected layers.
- **Dense (128, activation='relu'):** A dense (fully connected) layer with 128 units and ReLU activation.
- **Dropout (0.5):** A dropout layer with a higher dropout rate of 0.5 for more aggressive regularization.
- **Dense (len(DIGITS), activation='softmax'):** The output layer with a number of units equal to the number of digit classes and softmax activation to output class probabilities.

The model is compiled with the Adam optimizer, categorical crossentropy loss function, and accuracy as the evaluation metric.

In [ ]:
```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(40, 32, 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(DIGITS), activation='softmax')
])

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 38, 30, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2  (None, 19, 15, 32)        0         
 D)                                                              
                                                                 
 dropout (Dropout)           (None, 19, 15, 32)        0         
                                                                 
 conv2d_1 (Conv2D)           (None, 17, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 8, 6, 64)          0         
 g2D)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 8, 6, 64)          0         
                                                                 
 flatten (Flatten)           (None, 3072)              0         
                                                                 
 dense (Dense)               (None, 128)               393344    
                                                                 
 dropout_2 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 413450 (1.58 MB)
Trainable params: 413450 (1.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

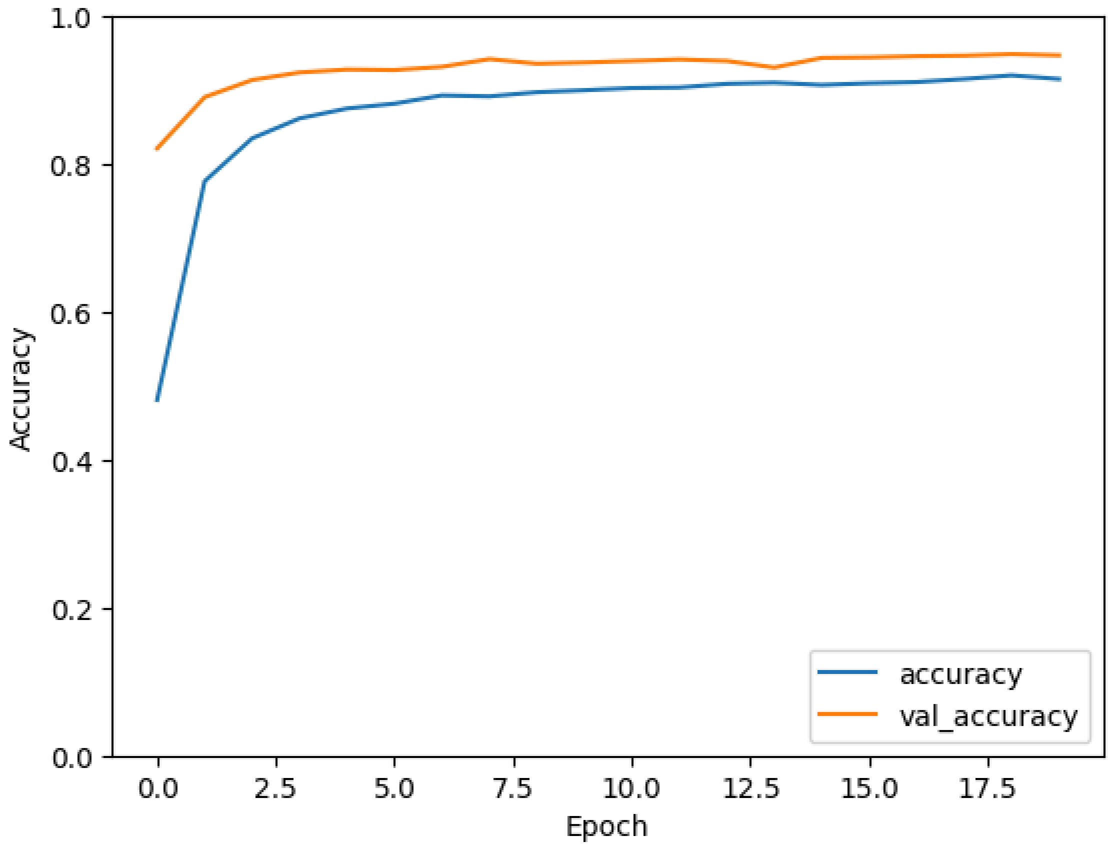
In [ ]:
```
history = model.fit(x_train, y_train, epochs=20, batch_size=32, validation_data=(x_tes
```

```
Epoch 1/20
540/540 [==============================] - 32s 57ms/step - loss: 2.5464 - accuracy:
0.1029 - val_loss: 2.3032 - val_accuracy: 0.0888
Epoch 2/20
540/540 [==============================] - 30s 56ms/step - loss: 2.3029 - accuracy:
0.1043 - val_loss: 2.3035 - val_accuracy: 0.0895
Epoch 3/20
540/540 [==============================] - 29s 54ms/step - loss: 2.2943 - accuracy:
0.1169 - val_loss: 2.2710 - val_accuracy: 0.1194
Epoch 4/20
540/540 [==============================] - 29s 53ms/step - loss: 2.1783 - accuracy:
0.1846 - val_loss: 1.9355 - val_accuracy: 0.3123
Epoch 5/20
540/540 [==============================] - 31s 58ms/step - loss: 1.5876 - accuracy:
0.4349 - val_loss: 0.8964 - val_accuracy: 0.7090
Epoch 6/20
540/540 [==============================] - 33s 62ms/step - loss: 1.0386 - accuracy:
0.6430 - val_loss: 0.5257 - val_accuracy: 0.8535
Epoch 7/20
540/540 [==============================] - 30s 55ms/step - loss: 0.7993 - accuracy:
0.7375 - val_loss: 0.4084 - val_accuracy: 0.8831
Epoch 8/20
540/540 [==============================] - 30s 56ms/step - loss: 0.6837 - accuracy:
0.7779 - val_loss: 0.3540 - val_accuracy: 0.9005
Epoch 9/20
540/540 [==============================] - 31s 57ms/step - loss: 0.6266 - accuracy:
0.7956 - val_loss: 0.3282 - val_accuracy: 0.9022
Epoch 10/20
540/540 [==============================] - 31s 57ms/step - loss: 0.5698 - accuracy:
0.8157 - val_loss: 0.3112 - val_accuracy: 0.9068
Epoch 11/20
540/540 [==============================] - 31s 58ms/step - loss: 0.5188 - accuracy:
0.8318 - val_loss: 0.2810 - val_accuracy: 0.9170
Epoch 12/20
540/540 [==============================] - 31s 57ms/step - loss: 0.5105 - accuracy:
0.8364 - val_loss: 0.2813 - val_accuracy: 0.9172
Epoch 13/20
540/540 [==============================] - 31s 58ms/step - loss: 0.4777 - accuracy:
0.8474 - val_loss: 0.2662 - val_accuracy: 0.9195
Epoch 14/20
540/540 [==============================] - 33s 62ms/step - loss: 0.4541 - accuracy:
0.8525 - val_loss: 0.2500 - val_accuracy: 0.9251
Epoch 15/20
540/540 [==============================] - 31s 57ms/step - loss: 0.4623 - accuracy:
0.8526 - val_loss: 0.2617 - val_accuracy: 0.9200
Epoch 16/20
540/540 [==============================] - 31s 57ms/step - loss: 0.4236 - accuracy:
0.8635 - val_loss: 0.2556 - val_accuracy: 0.9214
Epoch 17/20
540/540 [==============================] - 31s 58ms/step - loss: 0.4085 - accuracy:
0.8688 - val_loss: 0.2367 - val_accuracy: 0.9267
Epoch 18/20
540/540 [==============================] - 31s 58ms/step - loss: 0.4014 - accuracy:
0.8714 - val_loss: 0.2412 - val_accuracy: 0.9267
Epoch 19/20
540/540 [==============================] - 30s 55ms/step - loss: 0.3907 - accuracy:
0.8743 - val_loss: 0.2374 - val_accuracy: 0.9297
Epoch 20/20
540/540 [==============================] - 29s 54ms/step - loss: 0.3934 - accuracy:
0.8747 - val_loss: 0.2427 - val_accuracy: 0.9291
```

In [ ]:
```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {accuracy*100:.2f}%")

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
135/135 [==============================] - 2s 13ms/step - loss: 0.2213 - accuracy: 0.
9362
Test accuracy: 93.62%
```



In [ ]:
```
model.save('/content/drive/My Drive/digit_speech_recognition_TRAILmodel.h5')
```

In [ ]:
```
!pip install wandb
```

```
Requirement already satisfied: wandb in /usr/local/lib/python3.10/dist-packages (0.1
7.0)
Requirement already satisfied: click!=8.0.0,>=7.1 in /usr/local/lib/python3.10/dist-p
ackages (from wandb) (8.1.7)
Requirement already satisfied: docker-pycreds>=0.4.0 in /usr/local/lib/python3.10/dis
t-packages (from wandb) (0.4.0)
Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in /usr/local/lib/python3.1
0/dist-packages (from wandb) (3.1.43)
Requirement already satisfied: platformdirs in /usr/local/lib/python3.10/dist-package
s (from wandb) (4.2.1)
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.19.0 in /usr/local/lib/python
3.10/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/dist-packag
es (from wandb) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (fro
m wandb) (6.0.1)
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.10/dist-p
ackages (from wandb) (2.31.0)
Requirement already satisfied: sentry-sdk>=1.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from wandb) (2.2.0)
Requirement already satisfied: setproctitle in /usr/local/lib/python3.10/dist-package
s (from wandb) (1.3.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages
(from wandb) (67.7.2)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages
(from docker-pycreds>=0.4.0->wandb) (1.16.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.10/dist-pack
ages (from gitpython!=3.1.29,>=1.0.0->wandb) (4.0.11)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
dist-packages (from requests<3,>=2.0.0->wandb) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
s (from requests<3,>=2.0.0->wandb) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
ackages (from requests<3,>=2.0.0->wandb) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
ackages (from requests<3,>=2.0.0->wandb) (2024.2.2)
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.10/dist-pack
ages (from gitdb<5,>=4.0.1->gitpython!=3.1.29,>=1.0.0->wandb) (5.0.1)
```

Using Weights & Biases (wandb) for Experiment Tracking

Weights & Biases (wandb) is a tool for tracking and visualizing machine learning experiments. It provides an easy way to log various metrics, visualize model performance, and compare different runs. Here’s a detailed explanation of how wandb is used in this notebook:

- **Tracking Metrics:** wandb allows you to track important metrics like training and validation accuracy, loss, and more in real-time. This helps you monitor the progress of your model training and detect any issues early on.
- **Logging Hyperparameters:** You can log hyperparameters used in your experiments, such as learning rate, batch size, and number of epochs. This makes it easy to keep track of different configurations and their effects on model performance.

- **Visualizing Performance:** wandb provides interactive plots and charts that help you visualize the performance of your model over time. This includes plots for accuracy, loss, and other custom metrics.
- **Comparing Runs:** With wandb, you can compare different training runs side-by-side. This is particularly useful for hyperparameter tuning and understanding the impact of different changes to your model or training process.
- **Collaboration:** wandb facilitates collaboration by allowing you to share your experiments and results with your team. You can easily access and review the results from different runs, making it easier to work together on improving the model.

In this notebook, we use `wandb.keras.WandbCallback()` as a callback in the `model2.fit()` function. This callback integrates wandb with Keras, automatically logging metrics and parameters for each epoch during the training process.

In [ ]:
```python
import wandb

wandb.init(project='digit-speech-recognition')
```

```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wand
b.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
 ··········
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

Tracking run with wandb version 0.17.0

Run data is saved locally in `/content/wandb/run-20240519_100205-qbn7fd1w`

Syncing run **swift-galaxy-1** to Weights & Biases (docs)

View project at https://wandb.ai/alexandria-university_foe/digit-speech-recognition

View run at https://wandb.ai/alexandria-university_foe/digit-speech-recognition/runs/qbn7fd1w

Out[ ]: Display W&B run

Changes in Model Architecture

Compared to the previous model, `model2` incorporates several significant changes to potentially improve performance:

- **Increased Filters:** The number of filters in each convolutional layer has been increased:
  - The first `Conv2D` layer now has 64 filters instead of 32.
  - The second `Conv2D` layer now has 128 filters instead of 64.
  - A new third `Conv2D` layer has been added with 256 filters.
- **Additional Convolutional Layer:** An additional convolutional layer with 256 filters has been added to increase the model's capacity to capture more complex features from the input data.

- **Increased Units in Dense Layer:** The number of units in the fully connected `Dense` layer has been increased from 128 to 512, allowing the model to learn more complex representations before the final classification layer.

These changes are intended to enhance the model's ability to learn from the data by providing more parameters and depth, which can be beneficial for capturing intricate patterns in the audio features.

In [ ]:
```
model2 = Sequential([
    Conv2D(64, (3, 3), activation='relu', input_shape=(40, 32, 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(len(DIGITS), activation='softmax')
])
```

In [ ]:
```
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 38, 30, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2  (None, 19, 15, 32)        0         
 D)                                                              
                                                                 
 dropout (Dropout)           (None, 19, 15, 32)        0         
                                                                 
 conv2d_1 (Conv2D)           (None, 17, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 8, 6, 64)          0         
 g2D)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 8, 6, 64)          0         
                                                                 
 flatten (Flatten)           (None, 3072)              0         
                                                                 
 dense (Dense)               (None, 128)               393344    
                                                                 
 dropout_2 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 413450 (1.58 MB)
Trainable params: 413450 (1.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:

```python
history = model2.fit(
    x_train, y_train,
    epochs=20,
    batch_size=32,
    validation_data=(x_test, y_test),
    callbacks=[wandb.keras.WandbCallback()]
)
```

```
Epoch 1/20
539/540 [============================>.] - ETA: 0s - loss: 1.5699 - accuracy: 0.4809
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarnin
g: You are saving your model as an HDF5 file via `model.save()`. This file format is
considered legacy. We recommend using instead the native Keras format, e.g. `model.sa
ve('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/file
s/model-best)... Done. 0.1s
540/540 [==============================] - 117s 214ms/step - loss: 1.5699 - accuracy:
0.4810 - val_loss: 0.5848 - val_accuracy: 0.8212
Epoch 2/20
539/540 [============================>.] - ETA: 0s - loss: 0.6819 - accuracy: 0.7767
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarnin
g: You are saving your model as an HDF5 file via `model.save()`. This file format is
considered legacy. We recommend using instead the native Keras format, e.g. `model.sa
ve('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/file
s/model-best)... Done. 0.1s
```

```
540/540 [==============================] - 102s 188ms/step - loss: 0.6819 - accuracy: 0.7767 - val_loss: 0.3380 - val_accuracy: 0.8906
Epoch 3/20
539/540 [============================>.] - ETA: 0s - loss: 0.5152 - accuracy: 0.8348
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/files/model-best)... Done. 0.1s
540/540 [==============================] - 98s 182ms/step - loss: 0.5151 - accuracy: 0.8348 - val_loss: 0.2700 - val_accuracy: 0.9137
Epoch 4/20
539/540 [============================>.] - ETA: 0s - loss: 0.4425 - accuracy: 0.8618
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/files/model-best)... Done. 0.1s
540/540 [==============================] - 100s 185ms/step - loss: 0.4425 - accuracy: 0.8618 - val_loss: 0.2448 - val_accuracy: 0.9240
Epoch 5/20
539/540 [============================>.] - ETA: 0s - loss: 0.4018 - accuracy: 0.8752
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/files/model-best)... Done. 0.1s
540/540 [==============================] - 96s 178ms/step - loss: 0.4018 - accuracy: 0.8752 - val_loss: 0.2400 - val_accuracy: 0.9279
Epoch 6/20
540/540 [==============================] - 94s 174ms/step - loss: 0.3765 - accuracy: 0.8817 - val_loss: 0.2415 - val_accuracy: 0.9272
Epoch 7/20
539/540 [============================>.] - ETA: 0s - loss: 0.3378 - accuracy: 0.8930
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/files/model-best)... Done. 0.1s
540/540 [==============================] - 95s 176ms/step - loss: 0.3380 - accuracy: 0.8929 - val_loss: 0.2224 - val_accuracy: 0.9316
Epoch 8/20
539/540 [============================>.] - ETA: 0s - loss: 0.3462 - accuracy: 0.8916
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/files/model-best)... Done. 0.1s
```

```
540/540 [==============================] - 95s 176ms/step - loss: 0.3462 - accuracy:
0.8916 - val_loss: 0.1895 - val_accuracy: 0.9420
Epoch 9/20
540/540 [==============================] - 95s 175ms/step - loss: 0.3438 - accuracy:
0.8972 - val_loss: 0.2137 - val_accuracy: 0.9358
Epoch 10/20
540/540 [==============================] - 92s 171ms/step - loss: 0.3301 - accuracy:
0.8999 - val_loss: 0.2142 - val_accuracy: 0.9374
Epoch 11/20
540/540 [==============================] - 101s 187ms/step - loss: 0.3160 - accuracy:
0.9028 - val_loss: 0.1973 - val_accuracy: 0.9395
Epoch 12/20
540/540 [==============================] - 99s 183ms/step - loss: 0.3226 - accuracy:
0.9036 - val_loss: 0.1983 - val_accuracy: 0.9416
Epoch 13/20
540/540 [==============================] - 94s 174ms/step - loss: 0.3096 - accuracy:
0.9086 - val_loss: 0.1985 - val_accuracy: 0.9395
Epoch 14/20
540/540 [==============================] - 95s 177ms/step - loss: 0.3083 - accuracy:
0.9102 - val_loss: 0.2212 - val_accuracy: 0.9307
Epoch 15/20
540/540 [==============================] - 94s 174ms/step - loss: 0.3146 - accuracy:
0.9070 - val_loss: 0.1907 - val_accuracy: 0.9437
Epoch 16/20
540/540 [==============================] - 96s 178ms/step - loss: 0.2982 - accuracy:
0.9094 - val_loss: 0.1954 - val_accuracy: 0.9444
Epoch 17/20
539/540 [============================>.] - ETA: 0s - loss: 0.3020 - accuracy: 0.9110
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarnin
g: You are saving your model as an HDF5 file via `model.save()`. This file format is
considered legacy. We recommend using instead the native Keras format, e.g. `model.sa
ve('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/file
s/model-best)... Done. 0.1s
540/540 [==============================] - 95s 176ms/step - loss: 0.3021 - accuracy:
0.9110 - val_loss: 0.1821 - val_accuracy: 0.9460
Epoch 18/20
539/540 [============================>.] - ETA: 0s - loss: 0.2862 - accuracy: 0.9151
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarnin
g: You are saving your model as an HDF5 file via `model.save()`. This file format is
considered legacy. We recommend using instead the native Keras format, e.g. `model.sa
ve('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/file
s/model-best)... Done. 0.1s
540/540 [==============================] - 95s 176ms/step - loss: 0.2862 - accuracy:
0.9151 - val_loss: 0.1745 - val_accuracy: 0.9467
Epoch 19/20
539/540 [============================>.] - ETA: 0s - loss: 0.2710 - accuracy: 0.9200
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarnin
g: You are saving your model as an HDF5 file via `model.save()`. This file format is
considered legacy. We recommend using instead the native Keras format, e.g. `model.sa
ve('my_model.keras')`.
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240519_100205-qbn7fd1w/file
s/model-best)... Done. 0.1s
```

```
540/540 [==============================] - 99s 184ms/step - loss: 0.2710 - accuracy: 0.9201 - val_loss: 0.1697 - val_accuracy: 0.9488
Epoch 20/20
540/540 [==============================] - 99s 182ms/step - loss: 0.2938 - accuracy: 0.9151 - val_loss: 0.2013 - val_accuracy: 0.9471
```

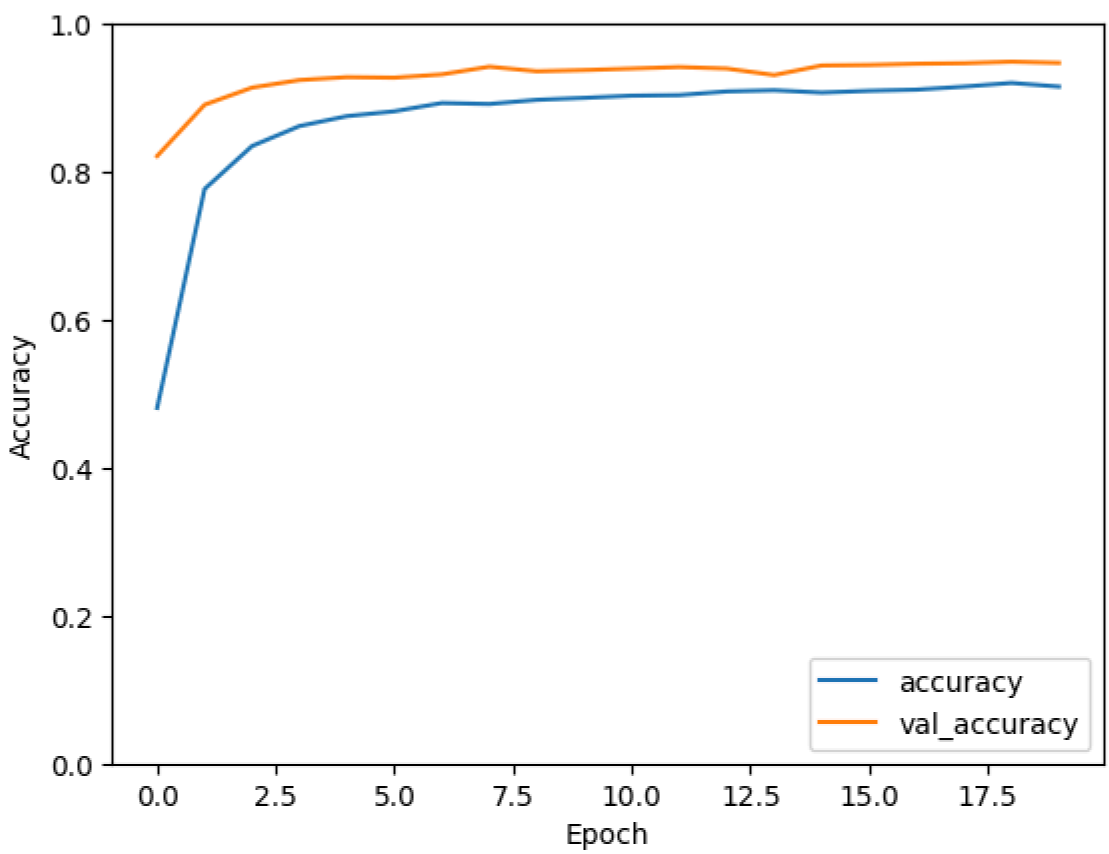
In [ ]:

```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {accuracy*100:.2f}%")
```

```
135/135 [==============================] - 2s 12ms/step - loss: 0.2213 - accuracy: 0.9362
Test accuracy: 93.62%
```

In [ ]:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



In [ ]:

```
model.save('/content/drive/MyDrive/SavedSpeechModel/digit_speech_recognition_Finalmode
```

Audio Preprocessing and Digit Prediction

This section loads the previous pre-trained digit recognition model and defines functions to preprocess audio files and predict the corresponding digit:

- **Preprocess Audio:** Loads an audio file, extracts MFCC features, resizes them to a consistent shape, and prepares them for model input.
- **Predict Digit:** Uses the preprocessed audio to predict the digit with the trained model.

In [ ]:
```
import numpy as np
import tensorflow as tf
import librosa
import librosa.display

# Load the trained model
model = tf.keras.models.load_model('/content/drive/MyDrive/Lab6_number0 to 9 using CNN

# Function to preprocess audio
def preprocess_audio(audio_path, sample_rate=16000, expected_shape=(40, 32, 1)):
    y, sr = librosa.load(audio_path, sr=sample_rate)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)

    # Check the current shape
    print(f"Original MFCC shape: {mfcc.shape}")

    # Resize the MFCC to the expected shape
    if mfcc.shape[1] < expected_shape[1]:
        pad_width = expected_shape[1] - mfcc.shape[1]
        mfcc = np.pad(mfcc, ((0, 0), (0, pad_width)), mode='constant')
    else:
        mfcc = mfcc[:, :expected_shape[1]]

    # Check the new shape
    print(f"Resized MFCC shape: {mfcc.shape}")

    mfcc = mfcc[np.newaxis, ..., np.newaxis]  # Add batch and channel dimension
    return mfcc

# Function to predict digit
def predict_digit(audio_path):
    preprocessed_audio = preprocess_audio(audio_path)
    prediction = model.predict(preprocessed_audio)
    predicted_digit = np.argmax(prediction)
    return predicted_digit
```

Predictions

Finally, we predict the digits for a few test audio files and print the results.

In [ ]:
```
# Predict the digit for the uploaded audio file
audio_file = 'test_audio9.wav'  # Path to the uploaded audio file
predicted_digit = predict_digit(audio_file)
print("Predicted Digit:", predicted_digit)
```

```
Original MFCC shape: (40, 75)
Resized MFCC shape: (40, 32)
1/1 [==============================] - 0s 99ms/step
Predicted Digit: 9
```

In [ ]:

```
# Predict the digit for the uploaded audio file
audio_file = 'test_audio1.wav'  # Path to the uploaded audio file
predicted_digit = predict_digit(audio_file)
print("Predicted Digit:", predicted_digit)
```

```
Original MFCC shape: (40, 57)
Resized MFCC shape: (40, 32)
1/1 [==============================] - 0s 71ms/step
Predicted Digit: 1
```

In [ ]:

```
# Predict the digit for the uploaded audio file
audio_file = 'test_audio7.wav'  # Path to the uploaded audio file
predicted_digit = predict_digit(audio_file)
print("Predicted Digit:", predicted_digit)
```

```
Original MFCC shape: (40, 59)
Resized MFCC shape: (40, 32)
1/1 [==============================] - 0s 150ms/step
Predicted Digit: 7
```