APPENDIX

1. CONVOLUTIONAL NEURAL NETWORK

IMPORT MODULE

```
1  import os
2  import pathlib
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import seaborn as sns
6  import tensorflow as tf
7  from tensorflow.keras import layers
8  from tensorflow.keras import models
9  from tensorflow import keras
10 from IPython import display
```

SET SEED VALUE

```
1  seed = 42
2  tf.random.set_seed(seed)
3  np.random.seed(seed)
```

IMPORT MINI SPEECH COMMANDS DATASET

```
1  DATASET_PATH = 'data/mini_speech_commands'
2  data_dir = pathlib.Path(DATASET_PATH)
3  if not data_dir.exists():
4  tf.keras.utils.get_file(
5  'mini_speech_commands.zip',
6  origin="http://storage.googleapis.com/download.tensorflow.org/data/
   mini_speech_commands.zip",
7  extract=True
8  cache_dir='.', cache_subdir='data')
```

THE DATASET AUDIO CLIPS ARE STORED IN EIGHT FOLDERS

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

EXTRACTED THE AUDIO CLIPS INTO A LIST CALLED FILENAMES

```
1  filenames = tf.io.gfile.glob(str(data_dir) + '/*/*')
2  filenames = tf.random.shuffle(filenames)
3  num_samples = len(filenames)
4  print('Number of total examples:', num_samples)
5  print('Number of examples per label:',
6  len(tf.io.gfile.listdir(str(data_dir/commands[0]))))
7  print('Example file tensor:', filenames[0])
```

SPLIT FILENAMES INTO TRAINING, VALIDATION, AND TEST SETS

```
1  train_files = filenames[:6400]
```

```
2  val_files = filenames[6400: 6400 + 800]
3  test files = filenames[-800:]
4  print('Training set size', len(train_files)
5  print('Validation set size', len(val_files)
6  print('Test set size', len(test_files)
```

READ THE AUDIO FILES AND THEIR LABELS

```
1  test_file=tf.io.read_file(DATASET_PATH+'/down0a9f9af7_nohash_0.wav')
2  test_audio,_ = tf.audio.decode_wav(contents=test_file)
3  test_audio.shape
```

FUNCTION THAT PREPROCESSES AUDIO FILES INTO AUDIO TENSOR

```
1  def decode_audio(audio_binary):
2  audio,_ = tf.audio.decode_wav(contents=audio_binary)
3  return tf.sueeze(audio, axis=-1)
```

FUNCTION THAT CREATES LABELS USING THE PARENT DIRECTORIES

```
1  def get_label(file_path):
2  parts = tf.strings.split(
3  input=file_path,
4  sep=os.path.sep)
5  return parts[-2]
```

DEFINE ANOTHER HELPER FUNCTION

```
1  def get_waveform_and_label(file_path):
2  label = get_label(file_path)
3  audio_binary = tf.io.read_file(file_path)
4  waveform = decode_audio(audio_binary)
5  return waveform, label
```

BUILD THE TRAINING SET TO EXTRACT THE AUDIO-LABEL PAIRS

```
1  AUTOTUNE = tf.data.AUTOTUNE
2  files_ds = tf.data.Dataset.from_tensor_slices(train_files)
3  waveform_ds = files_ds.map(
4  map_func=get_waveform_and_label,
5  num_parallel_calls=AUTOTUNE)
```

PLOT A FEW AUDIO WAVEFORMS

```
1  rows = 3
2  cols = 3
3  n = rows * cols
4  fig, axes = plt.subplots(rows, cols, figsize=(10, 12))
5  for i, (audio, label) in enumerate(waveform_ds.take(n)):
6  r = i // cols
7  c = i % cols
8  ax = axes[r][c]
9  ax.plot(audio.numpy())
10 ax.set_yticks(np.arange(-1.2, 1.2, 0.2))
11 label = label.numpy().decode('utf-8')
12 ax.set_title(label)
13 plt.show()
```

CONVERT WAVEFORM TO SPECTROGRAM

```
input_len = 16000
waveform = waveform[:input_len]
zero_padding = tf.zeros(
[16000] – tf.shape(waveform),
dtype=tf.float32)
waveform = tf.cast(waveform, dtype=tf.float32)
equal_length = tf.concat([waveform, zero_padding], 0)
spectrogram = tf.signal.stft(
equal_length, frame_length=255, frame_step=128)
spectrogram = tf.abs(spectrogram)
spectrogram = spectrogram[…, tf.newaxis]
return spectrogram
```

EXPLORING THE DATA

```
for waveform, label in waveform_ds.take(1):
label = label.numpy().decode('utf-8')
spectrogram = get_spectrogram(waveform)
print('Label:', label)
print('Waveform shape:', waveform.shape)
print('Spectrogram shape:', spectrogram.shape)
print('Audio playback')
display.display(display.Audio(waveform, rate=16000))
```

DEFINE A FUNCTION FOR DISPLAYING A SEPCTROGRAM

```
def plot_spectrogram(spectrogram, ax):
if len(spectrogram.shape) > 2:
assert len(spectrogram.shape) == 3
spectrogram = np.squeeze(spectrogram, axis=-1)
log_spec = np.log(spectrogram.T + np.finfo(float).eps)
height = log_spec.shape[0]
width = log_spec.shape[1]
X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
Y = range(height)
ax.pcolormesh(X, Y, log_spec)
```

PLOT THE EXAMPLE'S WAVEFORM

```
fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])
plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.show()
```

FUNCTION THAT TRANSFORM THE WAVEFORM INTO SPECTROGRAMS

```
def get_spectrogram_and_label_id(audio, label):
spectrogram = get_spectrogram(audio)
label_id = tf.argmax(label == commands)
return spectrogram, label_id
```

MAP get_spectrogram_and_label_id ACROSS THE DATASET'S ELEMENTS

```
spectrogram_ds = waveform_ds.map(
map_func=get_spectrogram_and_label_id,
num_parallel_calls=AUTOTUNE)
```

EXAMINE THE SPECTROGRAM FOR DIFFERENT EXAMPLE DATASET

```
rows = 3
cols = 3
n = rows * cols
fig, axes = plt.subplots(rows, cols, figsize=(10, 10))
for i, (spectrogram, label_id) in enumerate(spectrogram_ds.take(n)):
r = i // cols
c = i % cols
ax = axes[r][c]
plot_spectrogram(spectrogram.numpy(), ax)
ax.set_title(commands[label_id.numpy()])
ax.axis('off')
plt.show()
```

BUILD AND TRAIN THE MODEL

```
def preprocess_dataset(files):
files_ds = tf.data.Dataset.from_tensor_slices(files)
output_ds = files_ds.map(
map_func=get_waveform_and_label,
num_parallel_calls=AUTOTUNE)
output_ds = output_ds.map(
map_func=get_spectrogram_and_label_id,
num_parallel_calls=AUTOTUNE)
return output_ds
train ds = spectrogram_ds
val_ds = preprocess_dataset(val_files)
test_ds preprocess_dataset(test_files)
```

BATCH THE TRAINING AND VALIDATION SETS FOR MODEL TRAINING

```
batch_size = 64
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)
```

ADD OPERATIONS TO REDUCE READ LATENCY

```
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)
```

IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORK

```
for spectrogram, _ in spectrogram_ds.take(1):
input_shape = spectrogram.shape
print('Input shape:', input_shape)
num_labels = len(commands)
norm_layer = layers.Normalization()
```

```
norm_layer.adapt(data=spectrogram_ds.map(map_func=lambda spec, label: spec))
model = model.Sequential([
layers.Input(shape=input_shape),
layers.Resizing(32, 32),
norm_layer,
layers.Conv2D(16, 3, activation='relu'),
layers.Conv2D(32, 3, activation='relu'),
layers.MaxPooling2D(),
layers.Dropout(0.25),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dropout(0.5),
layers.Dense(num_labels),
])
model.summary()
```

CONFIGURE THE KERAS MODEL WITH ADAM OPTIMIZER

```
model.compile(
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'],
)
```

TRAIN THE MODEL OVER 10 EPOCHS

```
EPOCHS = 10
history = model.fit(
train_ds,
validation_data=val_ds,
epochs=EPOCHS,
callbacks=tf.keras.callbacks.EarlyStopping(verbose=0, patience=10),
)
```

PLOT THE TRAINING AND VALIDATION LOSS CURVES

```
metrics = history.history
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```

RUN THE MODEL ON THE TEST SET

```
test_audio = []
test_labels = []
for audio, label in test_ds:
test_audio.append(audio.numpy())
test_labels.append(label.numpy())
test_audio = np.array(test_audio)
test_labels = np.array(test_labels)
y_pred = np.argmax(model.predict(test_audio), axis=1)
y_true = test_labels
test_acc = sum(y_pred == y_true) / len(y_true)
print(f'Test set accuracy: {test_acc:.0%}')
```

DISPLAY A CONFUSION MATRIX

```
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx,
xticklabels=commands,
yticklabels=commands,
annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

RUN INFERENCE ON AN AUDIO FILE

```
sample_file = data_dir/'no/01bb6a2a_nohash_0.wav'
sample_ds = preprocess_dataset([str(sample_file)])
for spectrogram, label in sample_ds.batch(1):
prediction = model(spectrogram)
plt.bar(commands, tf.nn.softmax(prediction[0]))
plt.title(f'Predictions for "{commands[label[0]]}"')
plt.show()
```

2. DEEP NEURAL NETWORK

IMPORT MODULE

```
import os
import pathlib
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow import keras
from IPython import display
```

SET SEED VALUE

```
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

IMPORT MINI SPEECH COMMANDS DATASET

```
DATASET_PATH = 'data/mini_speech_commands'
data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
tf.keras.utils.get_file(
'mini_speech_commands.zip',
origin="http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip",
extract=True
cache_dir='.', cache_subdir='data')
```

THE DATASET AUDIO CLIPS ARE STORED IN EIGHT FOLDERS

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

EXTRACTED THE AUDIO CLIPS INTO A LIST CALLED FILENAMES

```
filenames = tf.io.gfile.glob(str(data_dir) + '/*/*')
filenames = tf.random.shuffle(filenames)
num_samples = len(filenames)
print('Number of total examples:', num_samples)
print('Number of examples per label:',
len(tf.io.gfile.listdir(str(data_dir/commands[0]))))
print('Example file tensor:', filenames[0])
```

SPLIT FILENAMES INTO TRAINING, VALIDATION, AND TEST SETS

```
train_files = filenames[:6400]
val_files = filenames[6400: 6400 + 800]
test files = filenames[-800:]
print('Training set size', len(train_files)
print('Validation set size', len(val_files)
```

```
6  print('Test set size', len(test_files)
```

READ THE AUDIO FILES AND THEIR LABELS

```
1  test_file=tf.io.read_file(DATASET_PATH+'/down0a9f9af7_nohash_0.wav')
2  test_audio,_ = tf.audio.decode_wav(contents=test_file)
3  test_audio.shape
```

FUNCTION THAT PREPROCESSES AUDIO FILES INTO AUDIO TENSOR

```
1  def decode_audio(audio_binary):
2  audio,_ = tf.audio.decode_wav(contents=audio_binary)
3  return tf.sueeze(audio, axis=-1)
```

FUNCTION THAT CREATES LABELS USING THE PARENT DIRECTORIES

```
1  def get_label(file_path):
2  parts = tf.strings.split(
3  input=file_path,
4  sep=os.path.sep)
5  return parts[-2]
```

DEFINE ANOTHER HELPER FUNCTION

```
1  def get_waveform_and_label(file_path):
2  label = get_label(file_path)
3  audio_binary = tf.io.read_file(file_path)
4  waveform = decode_audio(audio_binary)
5  return waveform, label
```

BUILD THE TRAINING SET TO EXTRACT THE AUDIO-LABEL PAIRS

```
1  AUTOTUNE = tf.data.AUTOTUNE
2  files_ds = tf.data.Dataset.from_tensor_slices(train_files)
3  waveform_ds = files_ds.map(
4  map_func=get_waveform_and_label,
5  num_parallel_calls=AUTOTUNE)
```

PLOT A FEW AUDIO WAVEFORMS

```
1  rows = 3
2  cols = 3
3  n = rows * cols
4  fig, axes = plt.subplots(rows, cols, figsize=(10, 12))
5  for i, (audio, label) in enumerate(waveform_ds.take(n)):
6  r = i // cols
7  c = i % cols
8  ax = axes[r][c]
9  ax.plot(audio.numpy())
10 ax.set_yticks(np.arange(-1.2, 1.2, 0.2))
11 label = label.numpy().decode('utf-8')
12 ax.set_title(label)
13 plt.show()
```

CONVERT WAVEFORM TO SPECTROGRAM

```
input_len = 16000
waveform = waveform[:input_len]
zero_padding = tf.zeros(
[16000] – tf.shape(waveform),
dtype=tf.float32)
waveform = tf.cast(waveform, dtype=tf.float32)
equal_length = tf.concat([waveform, zero_padding], 0)
spectrogram = tf.signal.stft(
equal_length, frame_length=255, frame_step=128)
spectrogram = tf.abs(spectrogram)
spectrogram = spectrogram[…, tf.newaxis]
return spectrogram
```

EXPLORING THE DATA

```
for waveform, label in waveform_ds.take(1):
label = label.numpy().decode('utf-8')
spectrogram = get_spectrogram(waveform)
print('Label:', label)
print('Waveform shape:', waveform.shape)
print('Spectrogram shape:', spectrogram.shape)
print('Audio playback')
display.display(display.Audio(waveform, rate=16000))
```

DEFINE A FUNCTION FOR DISPLAYING A SEPCTROGRAM

```
def plot_spectrogram(spectrogram, ax):
if len(spectrogram.shape) > 2:
assert len(spectrogram.shape) == 3
spectrogram = np.squeeze(spectrogram, axis=-1)
log_spec = np.log(spectrogram.T + np.finfo(float).eps)
height = log_spec.shape[0]
width = log_spec.shape[1]
X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
Y = range(height)
ax.pcolormesh(X, Y, log_spec)
```

PLOT THE EXAMPLE'S WAVEFORM

```
fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])
plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.show()
```

FUNCTION THAT TRANSFORM THE WAVEFORM INTO SPECTROGRAMS

```
def get_spectrogram_and_label_id(audio, label):
spectrogram = get_spectrogram(audio)
label_id = tf.argmax(label == commands)
return spectrogram, label_id
```

MAP get_spectrogram_and_label_id ACROSS THE DATASET'S ELEMENTS

```
spectrogram_ds = waveform_ds.map(
map_func=get_spectrogram_and_label_id,
num_parallel_calls=AUTOTUNE)
```

EXAMINE THE SPECTROGRAM FOR DIFFERENT EXAMPLE DATASET

```
rows = 3
cols = 3
n = rows * cols
fig, axes = plt.subplots(rows, cols, figsize=(10, 10))
for i, (spectrogram, label_id) in enumerate(spectrogram_ds.take(n)):
r = i // cols
c = i % cols
ax = axes[r][c]
plot_spectrogram(spectrogram.numpy(), ax)
ax.set_title(commands[label_id.numpy()])
ax.axis('off')
plt.show()
```

BUILD AND TRAIN THE MODEL

```
def preprocess_dataset(files):
files_ds = tf.data.Dataset.from_tensor_slices(files)
output_ds = files_ds.map(
map_func=get_waveform_and_label,
num_parallel_calls=AUTOTUNE)
output_ds = output_ds.map(
map_func=get_spectrogram_and_label_id,
num_parallel_calls=AUTOTUNE)
return output_ds
train ds = spectrogram_ds
val_ds = preprocess_dataset(val_files)
test_ds preprocess_dataset(test_files)
```

BATCH THE TRAINING AND VALIDATION SETS FOR MODEL TRAINING

```
batch_size = 64
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)
```

ADD OPERATIONS TO REDUCE READ LATENCY

```
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)
```

IMPLEMENTATION OF DEEP NEURAL NETWORK

```
for spectrogram, _ in spectrogram_ds.take(1):
input_shape = spectrogram.shape
print('Input shape:', input_shape)
num_labels = len(commands)
norm_layer = layers.Normalization()
norm_layer.adapt(data=spectrogram_ds.map(map_func=lambda spec, label:
spec))
```

```
model = model.Sequential([
layers.Input(shape=input_shape),
layers.Resizing(32, 32),
norm_layer,
layers.Flatten(),
layers.Dense(512, activation='relu'),
layers.Dense(256, activation='relu'),
layers.Dense(128, activation='relu'),
layers.Dense(num_labels),
])
model.summary()
```

CONFIGURE THE KERAS MODEL WITH ADAM OPTIMIZER

```
model.compile(
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'],
)
```

TRAIN THE MODEL OVER 10 EPOCHS

```
EPOCHS = 10
history = model.fit(
train_ds,
validation_data=val_ds,
epochs=EPOCHS,
callbacks=tf.keras.callbacks.EarlyStopping(verbose=0, patience=10),
)
```

PLOT THE TRAINING AND VALIDATION LOSS CURVES

```
metrics = history.history
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()
```

RUN THE MODEL ON THE TEST SET

```
test_audio = []
test_labels = []
for audio, label in test_ds:
test_audio.append(audio.numpy())
test_labels.append(label.numpy())
test_audio = np.array(test_audio)
test_labels = np.array(test_labels)
y_pred = np.argmax(model.predict(test_audio), axis=1)
y_true = test_labels
test_acc = sum(y_pred == y_true) / len(y_true)
print(f'Test set accuracy: {test_acc:.0%}')
```

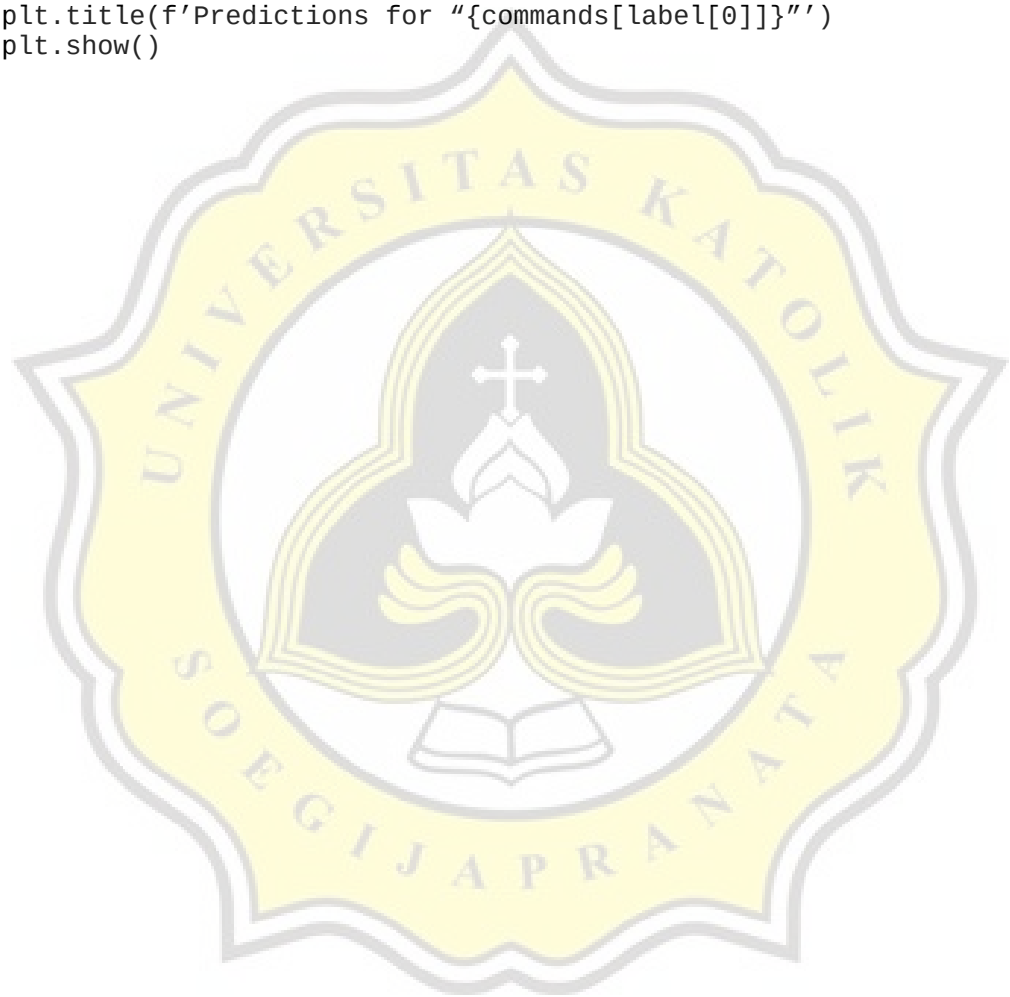
DISPLAY A CONFUSION MATRIX

```
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx,
```

```
4  xticklabels=commands,
5  yticklabels=commands,
6  annot=True, fmt='g')
7  plt.xlabel('Prediction')
8  plt.ylabel('Label')
9  plt.show()
```

RUN INFERENCE ON AN AUDIO FILE

```
1  sample_file = data_dir/'no/01bb6a2a_nohash_0.wav'
2  sample_ds = preprocess_dataset([str(sample_file)])
3  for spectrogram, label in sample_ds.batch(1):
4  prediction = model(spectrogram)
5  plt.bar(commands, tf.nn.softmax(prediction[0]))
6  plt.title(f'Predictions for "{commands[label[0]]}"')
7  plt.show()
```



PLAGIARISM CHECK.ORG

7.39% PLAGIARISM APPROXIMATELY | 0.07% IN QUOTES

Report #14281625

INTRODUCTION Background As modern technology is developing, especially artificial intelligence, many smartphones or other devices are equipped with artificial intelligence. By developing artificial intelligence, devices or the like are able to work effectively, and when viewed from the development of artificial intelligence becomes one of the modern technologies that we must be able to implement in the future. In this project, I tried to develop a virtual assistant in the form of voice commands on the laptop/PC so that the user can give commands to the laptop/PC and the command is done in accordance depends what the user said. To develop speech recognition requires appropriate algorithms and datasets. Deep Neural Network Algorithm is the most suitable for the development of speech recognition system because the algorithm is able to recognize human voice into inputs that later produce output in the form of commands and run in accordance with the command. Little information why in this project I use Deep Neural Network