CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

Chapter 5 explains the implementation and testing of projects development about virtual assistants using Deep Neural Network and Convolutional Neural Network Algorithms. Below is the code of the Deep Neural Network and Convolutional Neural Network algorithms used to obtain results from the project developed.

1. **Convolutional Neural Network**

```
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 IPython import display

# Set the seed value for experiment reproducibility.
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

Figure 5.1: Import Library of Convolutional Neural Network Algorithm

Import necessary modules and dependencies :

1. import os : used to import modules, it is a module in python so that python itself interacts directly with the operating system.
2. import pathlib : transfer most file system functions to a path object.
3. import matplotlib.pyplot as plt : a collection of functions that make some changes to an image, for example, creating an image, creating a plot areas in an image, adding labels in the plot and more.
4. import numpy as np : it is one of the python libraries that functions for numerical computing processes. NumPy has the ability to create N-dimensional array objects.
5. import seaborn as sns : for visualization in this program.

6. import tensorflow as tf : it is one of the official facilities provided by tensorflow to train deep learning models, either using a CPU or GPU.
7. from tensorflow.keras import layers : use keras as a library to import layers.
8. from tensorflow.keras import models : use keras as a library to import models.
9. from IPython import display : public API for display tools in Ipython.

```
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')
```

```
Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip
182083584/182082353 [==============================] - 4s 0us/step
182091776/182082353 [==============================] - 4s 0us/step
```

Figure 5.2: Import Mini Speech Commands Dataset

The original dataset consists of over 105,000 audio files in the WAV (Waveform) audio file format of people saying 35 different words. This data was collected by Google and released under a CC BY license. Download and extract the mini_speech_commands.zip file containing the smaller Speech Commands Datasets with *tf.keras.utils.get_file*

```
def get_spectrogram(waveform):
  # Zero-padding for an audio waveform with less than 16,000 samples.
  input_len = 16000
  waveform = waveform[:input_len]
  zero_padding = tf.zeros(
      [16000] - tf.shape(waveform),
      dtype=tf.float32)
  # Cast the waveform tensors' dtype to float32.
  waveform = tf.cast(waveform, dtype=tf.float32)
  # Concatenate the waveform with `zero_padding`, which ensures all audio
  # clips are of the same length.
  equal_length = tf.concat([waveform, zero_padding], 0)
  # Convert the waveform to a spectrogram via a STFT.
  spectrogram = tf.signal.stft(
      equal_length, frame_length=255, frame_step=128)
  # Obtain the magnitude of the STFT.
  spectrogram = tf.abs(spectrogram)
  # Add a `channels` dimension, so that the spectrogram can be used
  # as image-like input data with convolution layers (which expect
  # shape (`batch_size`, `height`, `width`, `channels`).
  spectrogram = spectrogram[..., tf.newaxis]
  return spectrogram
```

Figure 5.3: Convert Waveforms to Spectrograms

The waveforms in the dataset are represented in the time domain. Next, transform the waveforms from the time-domain signals into the time-frequency-domain signals by cimputing the short-time Fourier transform (STFT) to convert the waveforms to as spectrograms, which show frequency changes over time and can be represented as 2D images. Feed the spectrogram images into your neural network to train the model.

A Fourier transform (tf.signal.fft) converts a signal to its component frequencies, but loses all time information. In comparison, STFT (tf.signal.stft) splits the signal into windows of time and runs a Fourier transform on each window, preserving some time information, and returning a 2D tensor that you can run standard convolutions on.

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

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    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),
])
```

Figure 5.4: Implementation of Convolutinal Neural Network

For the model, using a simple Convolutional Neural Network (CNN), since it has transformed the audio files into spectrogram images. *tf.keras.Sequential* model using the following Keras pre-processing layers:

- *tf.keras.layers.Resizing* : to downsample the input to enable the model to train faster.
- *tf.keras.layers.Normalization* : to normalize each pixel in the image based on its mean and standard deviation.

```
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%}')
```

Figure 5.5: Evaluate The Model Performance

Run the Model on the test set and check the model performance. The main metrics used to evaluate a classification model is accuracy. Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

2. Deep Neural Network

```
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 IPython import display

# Set the seed value for experiment reproducibility.
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

Figure 5.6: Import Library of Deep Neural Network Algorithm

Import necessary modules and dependencies :

1. import os : used to import modules, it is a module in python so that python itself interacts directly with the operating system.
2. import pathlib : transfer most file system functions to a path object.
3. import matplotlib.pyplot as plt : a collection of functions that make some changes to an image, for example, creating an image, creating a plot areas in an image, adding labels in the plot and more.
4. import numpy as np : it is one of the python libraries that functions for numerical computing processes. NumPy has the ability to create N-dimensional array objects.
5. import seaborn as sns : for visualization in this program.
6. import tensorflow as tf : it is one of the official facilities provided by tensorflow to train deep learning models, either using a CPU or GPU.
7. from tensorflow.keras import layers : use keras as a library to import layers.
8. from tensorflow.keras import models : use keras as a library to import models.
9. from IPython import display : public API for display tools in Ipython.

```
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')
```

```
Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip
182083584/182082353 [==============================] - 4s 0us/step
182091776/182082353 [==============================] - 4s 0us/step
```

Figure 5.7: Import Mini Speech Commands Dataset

The original dataset consists of over 105,000 audio files in the WAV (Waveform) audio file format of people saying 35 different words. This data was collected by Google and released under a CC BY license. Download and extract the mini_speech_commands.zip file containing the smaller Speech Commands Datasets with *tf.keras.utils.get_file*

```
def get_spectrogram(waveform):
  # Zero-padding for an audio waveform with less than 16,000 samples.
  input_len = 16000
  waveform = waveform[:input_len]
  zero_padding = tf.zeros(
      [16000] - tf.shape(waveform),
      dtype=tf.float32)
  # Cast the waveform tensors' dtype to float32.
  waveform = tf.cast(waveform, dtype=tf.float32)
  # Concatenate the waveform with `zero_padding`, which ensures all audio
  # clips are of the same length.
  equal_length = tf.concat([waveform, zero_padding], 0)
  # Convert the waveform to a spectrogram via a STFT.
  spectrogram = tf.signal.stft(
      equal_length, frame_length=255, frame_step=128)
  # Obtain the magnitude of the STFT.
  spectrogram = tf.abs(spectrogram)
  # Add a `channels` dimension, so that the spectrogram can be used
  # as image-like input data with convolution layers (which expect
  # shape (`batch_size`, `height`, `width`, `channels`).
  spectrogram = spectrogram[..., tf.newaxis]
  return spectrogram
```

Figure 5.8: Convert Waveforms to Spectrograms

The waveforms in the dataset are represented in the time domain. Next, transform the waveforms from the time-domain signals into the time-frequency-domain signals by cimputing the short-time Fourier transform (STFT) to convert the waveforms to as spectrograms, which show frequency changes over time and can be represented as 2D images.

Feed the spectrogram images into your neural network to train the model. A Fourier transform (tf.signal.fft) converts a signal to its component frequencies, but loses all time information.

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

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    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()
```

Figure 5.9: Implementation of Deep Neural Network

In the implementation of Deep Neural Network, have different architectures in the layers. In experiments conducted, Deep Neural Network using 4 layers with filters on each layer has a different value.

```
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)
```

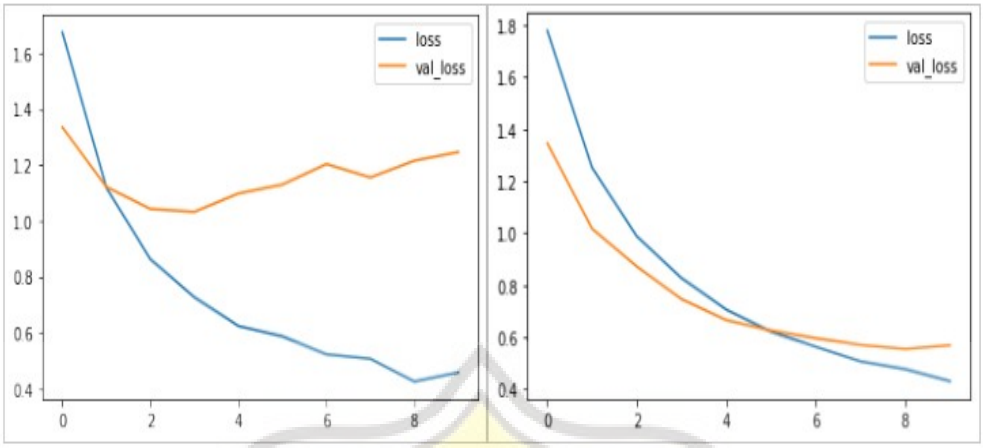
```
y_pred = np.argmax(model.predict(test_audio), axis=1)
y_true = test_labels

test_acc = sum(y_pred == y_true) / len(y_true)
print(f'Test set accuracy: {test_acc:.0%}')
```

Figure 5.10: Evaluate The Model Performance

Run the Model on the test set and check the model performance. The main metrics used to evaluate a classification model is accuracy. Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

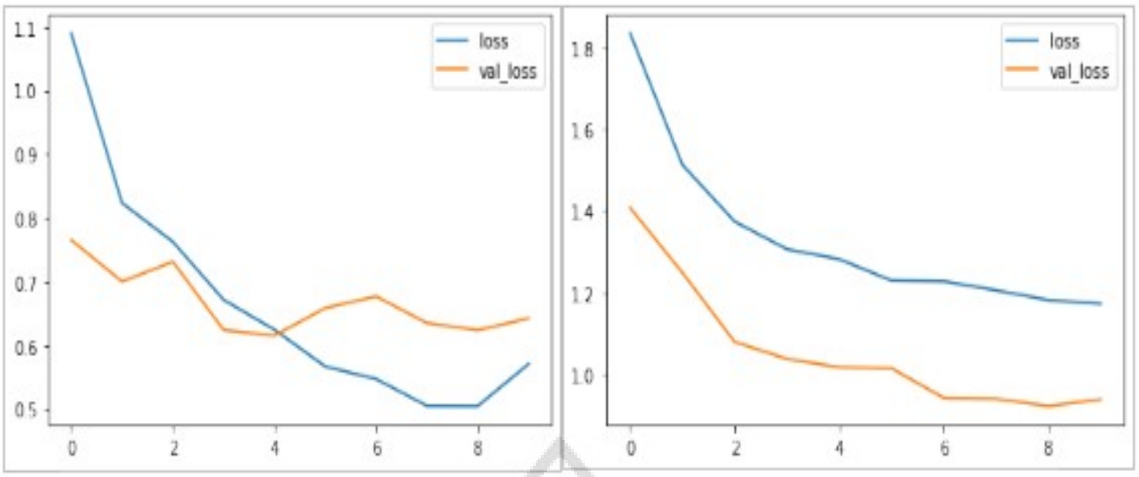
5.2. Results



(a) (b)

Figure 5.11: (a) Deep Neural Network Loss Function Graph and (b) Convolutional Neural Network Loss Function Graph (1)

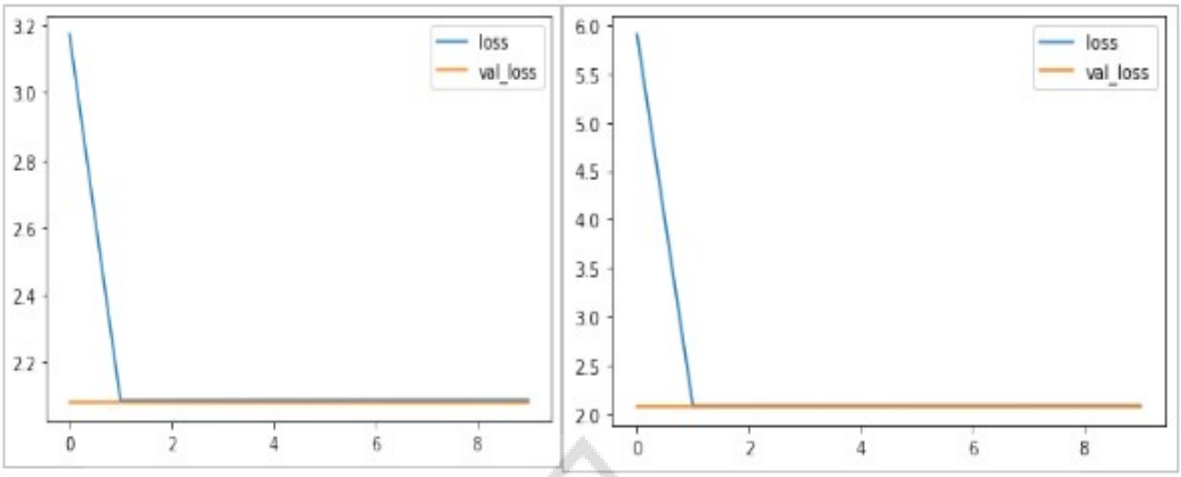
The figure above shows the loss function graph of both algorithms. On the loss function graph using parameters that are learning rate 0.001 both Deep Neural Network and Convolutional Network. In Figure 5.11, the loss function graph of the Deep Neural Network is explained that validation loss on the Deep Neural Network does not decrease even as unstable so that the resulting accuracy is less high. While the loss function graph of the Convolutional Neural Network is explained that validation loss on the Convolutional Neural Network is more stable and the graph decreases so that the accuracy generated can be higher than Deep Neural Network.



(a) (b)

Figure 5.12: (a) Deep Neural Network Loss Function Graph and (b) Convolutional Neural Network Loss Function Graph (2)

The figure above shows the loss function graph of both algorithms. On the loss function graph using parameters that are learning rate 0.01 both Deep Neural Network and Convolutional Network. In Figure 5.12, the loss function graph of the Deep Neural Network again shows a graph that is less stable than before and even the calculation of errors is greater than the previous results. While, the loss function graph from Convolutional Neural Network still shows a graph that decreases although not as well as the previous one but the calculation of errors that Convolutional Neural Network still remains small.



(a) (b)

Figure 5.13: (a) Deep Neural Network Loss Function Graph and (b) Convolutional Neural Network Loss Function Graph (3)

The figure above shows the loss function graph of both algorithms. On the loss function graph using parameters that are learning rate 0.1 both Deep Neural Network and Convolutional Network. In Figure 5.13, the loss function graph of both algorithms shows a very different graph than before. So these two algorithms are the same resulting in a bad graph because the graph does not decrease so that the calculation of the resulting error is very large and makes accuracy very small.

Table 5.1: Table Accuracy of Two Algorithms Results

Algorithm	Model	Parameters			Accuracy
		Learning Rate	Epochs	Batch Size	
Deep Neural Network	Sequential	0.1	10	64	14%
	Sequential	0.01	10	64	56%
	Sequential	**0.001**	**10**	**64**	**67%**
Convolutional Neural Network	Sequential	0.1	10	64	12%
	Sequential	0.01	10	64	81%
	Sequential	**0.001**	**10**	**64**	**82%**

Table 5.1 shows the experimental results of both tested algorithms with the three different parameters. From the experiments that have been done, obtained the best accuracy from the Deep Neural Network Algorithm of 67% and the accuracy of the Convolutional Neural Network Algorithm of 82%. Judging from both accuracy, Convolutional Neural Network Algorithm are better than Deep Neural Network Algorithm in terms of speech recognition implementation. From the experiment table above also shows that the learning rate parameters greatly affect the magnitude of accuracy produced, the smaller the learning rate, the higher the accuracy result, while the greater the learning rate, the smaller the accuracy. The success rate of the experiments conducted is measured by the high accuracy generated, and from the trials conducted by both algorithms resulting in high accuracy.

In the Deep Neural Network Algorithm conducted several experiments with the same input shape (124, 129, 1) with 3 different resizing layers namely (16, 16), (32, 32), (64, 64) and described in the 3 tables below:

Table 5.2: Model Summary of Deep Neural Network (1)

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 16, 16, 1)	0
normalization (Normalization)	(None, 16, 16, 1)	3
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 512)	131584
dense (Dense)	(None, 256)	131328
dense (Dense)	(None, 128)	32896
dense (Dense)	(None, 8)	1032
Total params :	296,843	
Trainable params:	296,840	
Non-trainable params:	3	
Test set accuracy:	66%	

Table 5.3: Model Summary of Deep Neural Network (2)

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dense (Dense)	(None, 256)	131328
dense (Dense)	(None, 128)	32896
dense (Dense)	(None, 8)	1032
Total params :	690,059	
Trainable params:	690,056	
Non-trainable params:	3	
Test set accuracy:	67%	

Table 5.4: Model Summary of Deep Neural Network (3)

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 64, 64, 1)	0
normalization (Normalization)	(None, 64, 64, 1)	3
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 512)	2097664
dense (Dense)	(None, 256)	131328
dense (Dense)	(None, 128)	32896
dense (Dense)	(None, 8)	1032
Total params :	2,262,923	
Trainable params:	2,262,920	
Non-trainable params:	3	
Test set accuracy:	72%	

The 3 tables above are trials conducted on Deep Neural Network Algorithms with 3 different resizing layers. The result of resizing layer used against the resulting accuracy is that the greater the resizing layer, the greater the accuracy of the resulting also while the smaller the resizing layer, the smaller the accuracy of the resulting layer. In the 3 experiments above each using 4 layers with Dense type and 1 layer with Flatten type, and using the same filters and activation function that is relu. From the results of the experiment above obtained different accuracy results from the 3 experiments conducted. The three experiments saw improved accuracy after the resizing layer was changed.

All of the above experiments can be summarized into a conclusion that indeed to implement speech recognition Convolutional Neural Network is more suitable than Deep Neural Network because datasets that were previously in audio form are converted into spectrogram or image data, in this process Convolutional Neural Network utilizes the convolution process by moving a certain-sized convolution kernel (filter) to an image, the computer gets new representative information from the results of multiplying the image with the filter used. While in Deep Neural Network there is no convolution process so the resulting accuracy is lower than the Convolutional Neural Network.