National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Aayush Aggarwal
Student ID: x21232911

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aayush Aggarwal
Student ID:	x21232911
Programme:	MSc in Data Analytics
Year:	2023
Module:	MSc Research Project
Lecturer:	Dr. Catherine Mulwa
Submission Due Date:	14/08/2023
Project Title:	American Sign Language Recognition using Computer Vision and Deep Learning
Word Count:	977
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Aayush Aggarwal
Date:	14th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aayush Aggarwal
Student ID: x21232911

1 Introduction

The configuration manual provides various components that were required to perform the research project. The description of hardware and software configuration used for model implementation and evaluation is described in Section 2 and Section 3. The overall flow of this configuration manual is stated below, along with the snapshots of code artefacts.

- Data collection and loading
- Data splitting and pre-processing
- Model Implementation for sign language recognition
- Model Evaluation

2 Hardware Configuration

The hardware configuration used to implement the research project is described below in Table 1.

Table 1: Hardware Configuration

Machine Name	Apple MacBook Pro M1 chip
Processor	macOS Ventura version 13.4.1, Apple M1 Chip, 8-core CPU, and 8-core GPU, 16-core Neural Engine
RAM	8 GB

3 Software Configuration

Python code has been performed on Google Colab. Different tools and libraries used for implementation of the research project is mentioned below and Fig. 1 depicts the same.

- Python 3.10.12
- Google Colab Pro Plus - A100 GPU accelerator
- Keras and TensorFlow
- OpenCV and few other for pre-processing
- Matplotlib for visualisation

```
[ ] # Load data
    import cv2
    import os

    # Model Training
    import tensorflow as tf
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.optimizers import Adam

[ ] # Data Visualisation
    import matplotlib.pyplot as plt
```

Fig. 1: Python libraries

4 Data Collection and Loading

American sign language dataset was selected for recognising hand gestures which was obtained from Kaggle. Due to large image size, the data is firstly uploaded to google drive and zip file was created. Then, Google drive was mounted with Google Colab. Further, these zipped folders are located and stored in a destination folder at '/content/my_data'. Then, these folders are unzipped to prepare for data splitting and pre-processing. Below Fig. 2 shows the code artefact of the same.

```
[ ] # Mount google drive
    from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[ ] if not os.path.exists("./my_data"):
        os.makedirs("./my_data")

[ ] import zipfile

    #locate the zip folder
    zip_file_path='/content/drive/My Drive/asl_alphabet_train.zip'
    destination_folder = '/content/my_data'

    # Unzip the folder
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(destination_folder)
```

Fig. 2: Data Loading

5 Data Splitting

The author has implemented two different models, namely Convolutional Neural Network (CNN) and Residual Network 50 (ResNet50). The data is splitted into two different types for each of them which is explained further in next sub-sections (5.1 and 5.2).

5.1 CNN model

In CNN model, few libraries such as random, train_test_split, and NumPy are loaded for iteration and splitting of dataset. Below Fig. 3 shows the iteration of each class folder to collect the image paths and their labels. The data is then shuffled and converted to NumPy arrays. Finally, the training dataset is divided into 60:20:20 ratio which contains 60% training images, 20% validation and testing images. This splitting was then verified by printing their shapes which is shown in Fig. 4.

```
import random
from sklearn.model_selection import train_test_split
import numpy as np

# Assuming your dataset path is '/content/drive/MyDrive/dataset/'
dataset_path = '/content/my_data/asl_alphabet_train'

# List all the folders in the dataset path (assuming each folder is a different class)
class_folders = os.listdir(dataset_path)

# Remove the '.DS_Store' folder from the list if present
if '.DS_Store' in class_folders:
    class_folders.remove('.DS_Store')

# Initialize empty lists to store image paths and corresponding labels
image_paths = []
labels = []

# Iterate through each class folder and collect image paths and labels
for class_idx, class_folder in enumerate(class_folders):
    class_path = os.path.join(dataset_path, class_folder)
    image_filenames = os.listdir(class_path)
    image_paths.extend([os.path.join(class_path, img_filename) for img_filename in image_filenames])
    labels.extend([class_idx] * len(image_filenames))

# Shuffle the data
combined = list(zip(image_paths, labels))
random.shuffle(combined)
image_paths[:], labels[:] = zip(*combined)
```

```
# Convert lists to NumPy arrays
image_paths = np.array(image_paths)
labels = np.array(labels)

# Split the dataset into training, validation and testing sets (60-20-20% ratio)
X_train, X_test, y_train, y_test = train_test_split(image_paths, labels, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=27)

# Now, you have X_train and y_train for training, X_val and y_val for validation and X_test and y_test for testing
```

Fig. 3: Splitting of dataset - CNN model

```
[ ] print(X_train.shape,y_train.shape)
    print(X_val.shape,y_val.shape)
    print(X_test.shape, y_test.shape)

    (142767,) (142767,)
    (35692,) (35692,)
    (44615,) (44615,)
```

Fig. 4: Splitted dataset - CNN model

5.2 ResNet50 model

In ResNet50 model, a new base path is created for training and testing dataset. The training dataset consist of 29 folders and hence the same folders are created with same name in testing dataset. Then, the images are moved randomly from training set to testing set in 80:20 ratio. Fig. 5 shows the code snippet for the same.

```
import random
import shutil

# Path to your original dataset
original_dataset_path = '/content/my_data/asl_alphabet_train'

# Path to create the training and testing datasets
base_path = '/content/asl_dataset'
training_path = os.path.join(base_path, 'training')
testing_path = os.path.join(base_path, 'testing')

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

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

# Loop through each folder in the original dataset
for folder_name in os.listdir(original_dataset_path):
    folder_path = os.path.join(original_dataset_path, folder_name)

    if os.path.isdir(folder_path):
        images = os.listdir(folder_path)
        num_images = len(images)
        num_testing_images = int(0.2 * num_images)

        # Create the destination folders in the training and testing directories
        training_folder = os.path.join(training_path, folder_name)
        testing_folder = os.path.join(testing_path, folder_name)
```

```
        os.makedirs(training_folder, exist_ok=True)
        os.makedirs(testing_folder, exist_ok=True)

        # Randomly select images for testing
        testing_images = random.sample(images, num_testing_images)

        # Move images to respective training and testing folders
        for image in images:
            source_path = os.path.join(folder_path, image)
            destination_folder = testing_folder if image in testing_images else training_folder
            destination_path = os.path.join(destination_folder, image)
            shutil.copy(source_path, destination_path)
```

Fig. 5: Splitting of dataset - ResNet50 model

6 Data Pre-Processing

Once the data is loaded, some visualizations were performed to check the distribution of the datasets. A plot for number of samples for each dataset is depicted. In addition, a sample images of ASL dataset were also shown. Both the diagrams are represented in technical report. After this, the images were loaded and pre-processed which is explained separately for both the models in next sub-sections (6.1 and 6.2).

6.1 CNN model

For data pre-processing in Fig. 6, the images are firstly read from the specified path which takes the input path and their labels. They decode the 3 colour channels in the images and then perform resizing and normalisation. Further, autotuning and shuffling of images takes place with a batch size of 32. The same process is then repeated for validation and testing sets.

```
# Function to load and preprocess images
def load_and_preprocess_image(image_path, label):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)  # Adjust channels accordingly (RGB or grayscale)
    image = tf.image.resize(image, (64, 64))  # Resize images to a common size (e.g., 64x64)
    image = image / 255.0  # Normalize pixel values to [0, 1]
    return image, label

# Create TensorFlow datasets
batch_size = 32  # Adjust the batch size as needed
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_dataset = train_dataset.map(load_and_preprocess_image, num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(buffer_size=len(X_train))
train_dataset = train_dataset.batch(batch_size)
train_dataset = train_dataset.prefetch(tf.data.AUTOTUNE)

validation_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val))
validation_dataset = validation_dataset.map(load_and_preprocess_image, num_parallel_calls=tf.data.AUTOTUNE)
validation_dataset = validation_dataset.batch(batch_size)
validation_dataset = validation_dataset.prefetch(tf.data.AUTOTUNE)

test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_dataset = test_dataset.map(load_and_preprocess_image, num_parallel_calls=tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(batch_size)
test_dataset = test_dataset.prefetch(tf.data.AUTOTUNE)
```

Fig. 6: Data Pre-Processing - CNN model

6.2 ResNet50 model

For data pre-processing of second model, an ImageDataGenerator function is applied for data augmentation method. An image height and width are considered as 224, with 3 channels. Later, some transformations like rescaling, rotation, zooming, flipping, etc, were applied on training and testing dataset. Below Fig. 7 shows the same.

```python
data_dir = '/content/asl_dataset'

img_height=224
img_width=224
num_channels=3
```

```python
# Read asl data from directories
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True,
                                   fill_mode='nearest')



train_generator = train_datagen.flow_from_directory(
        data_dir+ '/training',
        target_size=(img_height, img_width),
        batch_size=32,
        class_mode='categorical')
```

```python
validation_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

validation_generator = validation_datagen.flow_from_directory(
        data_dir+ '/testing',
        target_size=(img_height, img_width),
        batch_size=32,
        class_mode='categorical')
```

Fig. 6: Data Pre-Processing - ResNet50 model

7 Model Implementation

The author implemented two Deep learning algorithms, CNN and ResNet50 for sign language recognition. The feature extraction and modelling for both the models are explained below.

7.1 CNN model

CNN model consist of three convolutional and three maxpooling layers with different filter sizes. It is followed by ReLU activation function. These layers perform the feature extraction by adding some filters to them. A flatten and dense layer is also included with a softmax activation function. This model is then compiled using Adam optimizer and the summary is printed. Early stopping callback is provided during training. Below Fig. 7 is the code snapshot for the same.

```python
from tensorflow.keras import layers, models
#from tensorflow.keras.metrics import Precision, Recall

# This callback will stop the training when there is no improvement in the loss for three consecutive epochs.
# patience defines the number of epochs to check for little to no improvement in loss
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3) #Early stopping method

# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(class_folders), activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy', 'SparseTopKCategoricalAccuracy'])

# Display the model summary
model.summary()
```

Fig. 7: CNN model construction

This model is then trained on 5 and 10 epochs with a batch size of 32 samples. The model's performance is checked on testing data with matrices such as accuracy, loss, Top k accuracy, and Binary recall. Fig. 8 shows the same.

```python
# Train the model
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=5, batch_size = 32, callbacks=[callback], verbose=True)

# Evaluate the model on the test dataset
test_loss, test_accuracy, test_sparse_top_k_categorical_accuracy = model.evaluate(test_dataset)
print("Test Accuracy:", test_accuracy)
print('Test Top-k Accuracy', test_sparse_top_k_categorical_accuracy)
```

```
Epoch 1/5
4462/4462 [==============================] - 66s 5ms/step - loss: 0.6787 - accuracy: 0.8008 - sparse_top_k_categorical_accuracy: 0.9256 - val_loss: 0.1757
Epoch 2/5
4462/4462 [==============================] - 54s 5ms/step - loss: 0.1321 - accuracy: 0.9596 - sparse_top_k_categorical_accuracy: 0.9962 - val_loss: 0.1267
Epoch 3/5
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0787 - accuracy: 0.9765 - sparse_top_k_categorical_accuracy: 0.9986 - val_loss: 0.0868
Epoch 4/5
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0609 - accuracy: 0.9821 - sparse_top_k_categorical_accuracy: 0.9992 - val_loss: 0.0869
Epoch 5/5
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0460 - accuracy: 0.9863 - sparse_top_k_categorical_accuracy: 0.9997 - val_loss: 0.0880
1395/1395 [==============================] - 5s 4ms/step - loss: 0.0843 - accuracy: 0.9783 - sparse_top_k_categorical_accuracy: 0.9982
Test Accuracy: 0.9782808423042297
Test Top-k Accuracy 0.9982293248176575
```

```python
# Train the model
history = model.fit(train_dataset, validation_data=validation_dataset, epochs=10, batch_size = 32, callbacks=[callback], verbose=True)

# Evaluate the model on the test dataset
test_loss, test_accuracy, test_sparse_top_k_categorical_accuracy = model.evaluate(test_dataset)
print("Test Accuracy:", test_accuracy)
print('Test Top-k Accuracy', test_sparse_top_k_categorical_accuracy)
```

```
Epoch 1/10
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0398 - accuracy: 0.9885 - sparse_top_k_categorical_accuracy: 0.9997 - val_loss: 0.1009
Epoch 2/10
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0360 - accuracy: 0.9896 - sparse_top_k_categorical_accuracy: 0.9998 - val_loss: 0.0785
Epoch 3/10
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0340 - accuracy: 0.9908 - sparse_top_k_categorical_accuracy: 0.9999 - val_loss: 0.1109
Epoch 4/10
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0292 - accuracy: 0.9920 - sparse_top_k_categorical_accuracy: 0.9999 - val_loss: 0.0775
Epoch 5/10
4462/4462 [==============================] - 54s 5ms/step - loss: 0.0290 - accuracy: 0.9920 - sparse_top_k_categorical_accuracy: 0.9999 - val_loss: 0.0883
```

Fig. 8: Training CNN model

7.2 ResNet50 model

A base model of ResNet50 was created which was added with convolutional layers to form a master model. This combined model is compiled and trained on training dataset with 5 epochs and 32 batch size. The models performance is checked on testing dataset with evaluation matrices such as accuracy, precision, recall, AUC, and loss as seen in Fig. 9.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Input
import cv2
import numpy as np

# Load ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Create CNN model
model = tf.keras.models.Sequential([
    Conv2D(32, (3, 3), activation='relu',padding='valid'),
    MaxPooling2D((1, 1)),
    Conv2D(64, (3, 3), activation='relu',padding='valid'),
    MaxPooling2D((1, 1)),
    Conv2D(256, (3, 3), activation='relu',padding='valid'),
    MaxPooling2D((1, 1)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(29, activation='softmax')
])

# Combine models and create a master model
x = base_model.output
x = model(x)
combined_model = Model(inputs=base_model.input, outputs=x)
```

```
# print(model.summary())
# print(base_model.summary())

# Fit the model
# combined_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
combined_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()

history = combined_model.fit(
    train_generator,
    epochs=5,
    batch_size=32,
    validation_data=validation_generator,  # Use the same generator for validation
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_steps=validation_generator.samples // validation_generator.batch_size,  # Number of validation steps
    verbose=1
)

print(combined_model.summary())
```

Fig. 9: ResNet50 model construction and training

8 Model Evaluation

Both the models are evaluated on few matrices which are mentioned in above section. After testing the model's performance, CNN model outperforms ResNet50. Further, a plot of accuracy and loss was demonstrated using Matplotlib library. Fig. 10 shows the code snippet for CNN visualisation and Fig. 11 depicts the plots.

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Fig. 10: Code snapshot for CNN visualisation

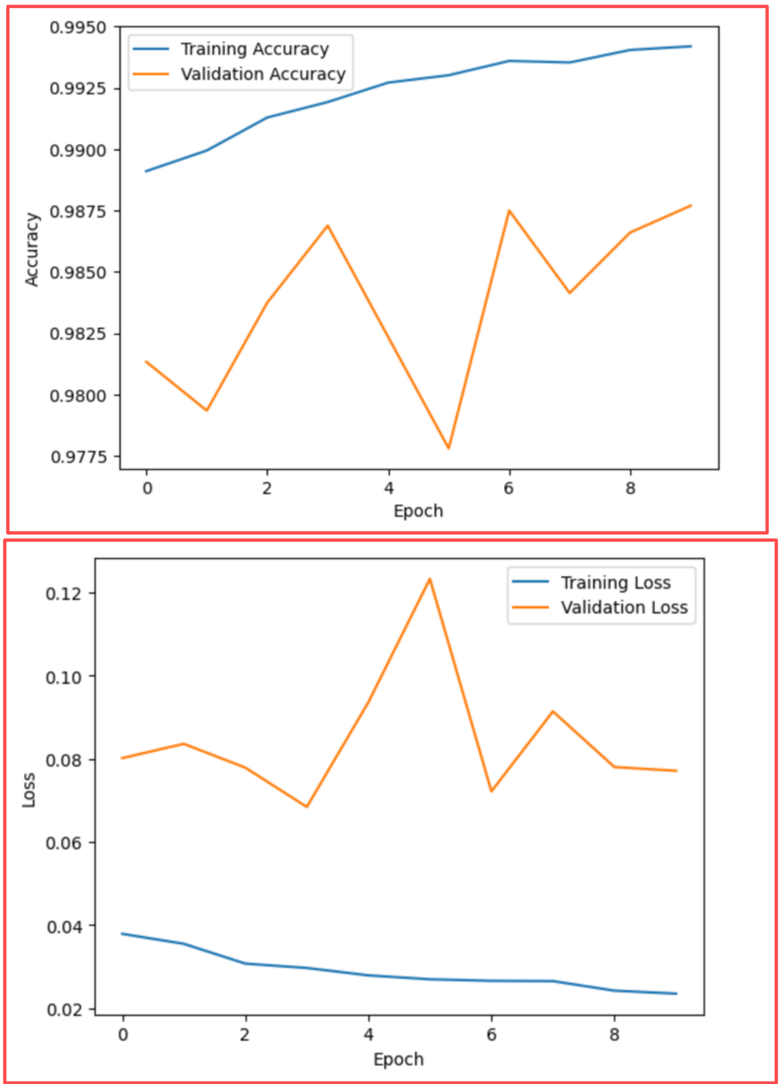

Fig. 11: Plot of accuracy and loss