Configuration Manual

MSc Research Project
Data Analytics

Deepti Deepak Tiwari
Student ID: X21240302

School of Computing
National College of Ireland

Supervisor: Prashanth Nayak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Deepti Deepak Tiwari
Student ID:	X21240302
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prashanth Nayak
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	556
Page Count:	10

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:	Deepti Deepak Tiwari
Date:	14/08/2023

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

Deepti Deepak Tiwari
x21240302

1 Overview

This document provides the step to step manual configuration for ”PolyCystic Ovary Syndrome Detection Using CapsuleNet and Synthetic Data”. This document will help to setup and install the prerequisites required to execute this research in future.

2 Hardware and software Requirement

2.1 Hardware Requirements

The research used the below-listed hardware in order to execute the code used in this research.

- Operating system used was Windows 10 Pro Language version 22H2.
- Processor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
- storage: RAM 8.00 GB
- System type: 64-bit operating system, x64-based processor

2.2 Software Requirements

The below are software used to complete the research proposed

- Google Colab and Jupyter Notebook
- Python 3.7 Scripting Language
- Google Drive for Storage
- Notepad++. Word, Excel, Overleaf

3 Setting Environment

3.1 Google Colab

Firstly to start with the research project Google Colab was set up. In order to set up the Colab we need a Gmail account. The purpose to use Google Colab was as it provides free and high GPU for processing. Select the GPU before execution.

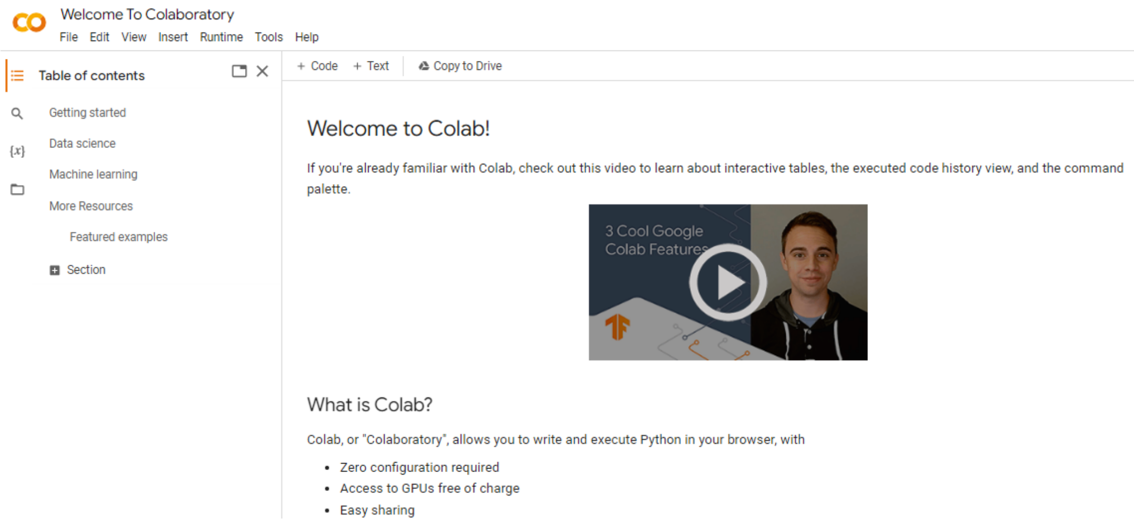

Figure 1: Google Colab

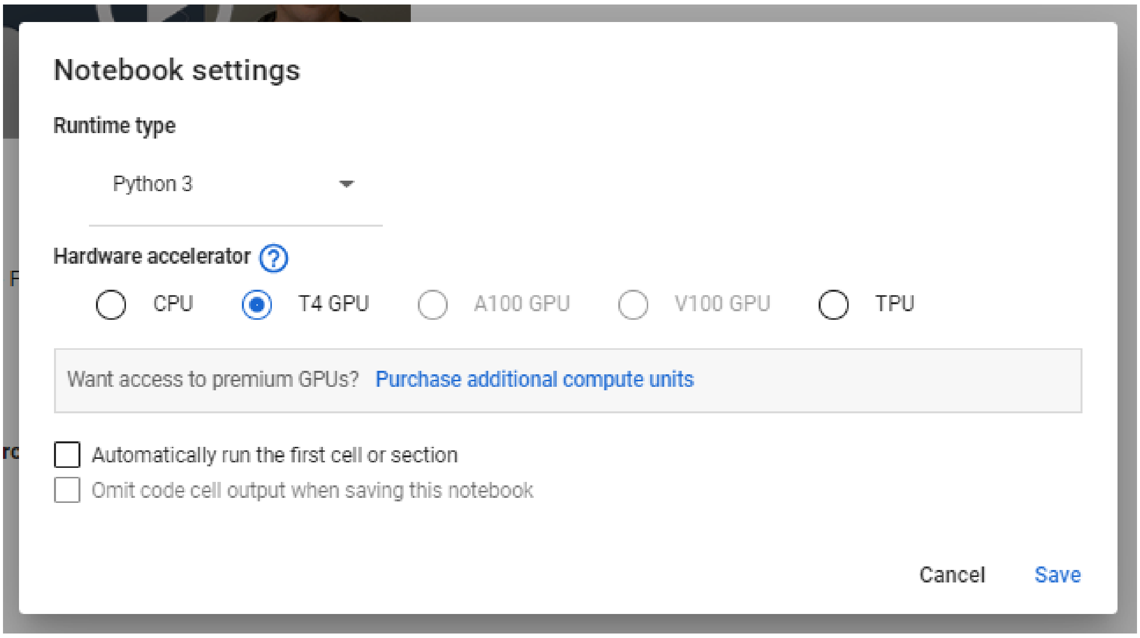

Figure 2: Selecting GPU

4 Data Selection

The data used in this research work was retrieved from the Kaggle data repository. The dataset consists of 3856 image files which were about 132 MB.

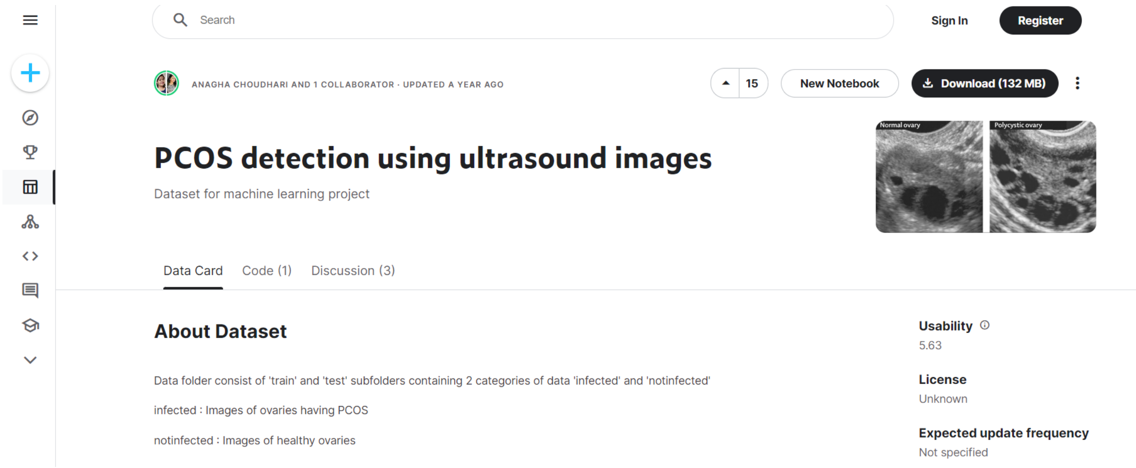

Figure 3: dataset

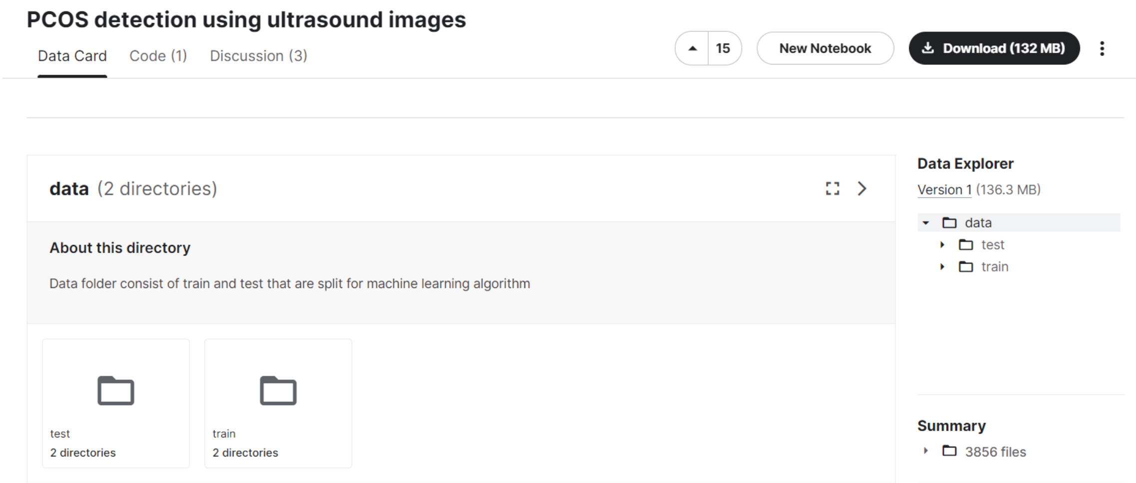

Figure 4: dataset files

5 Data Storing and Model Building 5

5.1 Storing the data into Google Drive

The data fetched from google drive was uploaded to the drive in order to read and use the data easily while model building and compilation.

5.2 Prerequisite installation and library importing

This research used the below-mentioned libraries for model building, evaluation and data cleaning.

- numpy
- tensorflow.keras.utils
- PIL
- matplotlib
- torch
- random
- sklearn
- itertools
- keras
- google.colab
- ImageDataGenerator
- seaborn
- sklearn.metrics

```
from google.colab import drive
import os
import numpy as np
from PIL import Image
from tensorflow.keras.utils import to_categorical
from PIL import UnidentifiedImageError
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
import tensorflow as tf

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

Figure 5: Library Used

5.3 Connecting Google Drive and Colab

Google Colab was linked to Drive in order to retrieve and use the data as shown below 6. This piece of code was executed to mount the Google Colab and Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

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

Figure 6: Mounting Google Drive

5.4 Data loading, Pre-processing and splitting of data

5.4.1 Data Loading and reading

After mounting the drive data was read as shown below 7.

```
[ ] def load_images(directory):
        X = []
        y = []
        classes = os.listdir(directory)
        c_indices = {name_of_class: idx for idx, name_of_class in enumerate(classes)}

        for name_of_class in classes:
            class_directory = os.path.join(directory, name_of_class)
            if os.path.isdir(class_directory):
                for image_name in os.listdir(class_directory):
                    image_path = os.path.join(class_directory, image_name)
                    if os.path.isfile(image_path):
                        try:
                            image = Image.open(image_path).convert('RGB')
                            image = image.resize((128, 128))
                            image = np.array(image) / 255.0
                            X.append(image)
                            y.append(c_indices[name_of_class])
                        except (IOError, OSError, UnidentifiedImageError) as e:
                            print(f"Error opening image: {image_path} - {e}")

        X = np.array(X)
        y = np.array(y)
        return X, y
```

Figure 7: Data Reading

5.4.2 Data Pre-proceesing

After reading the data the data is in image format. It was mandatory to transfer the image in greyscale and should be of the same size before using it with the model. The image processing was done as shown below 8

```
image = Image.open(image_path).convert('RGB')
image = image.resize((128, 128))
image = np.array(image) / 255.0
X.append(image)
```

Figure 8: Pre-Processing

5.4.3 Setting the path for train and test

The path for train data and test data was set as shown below 9

```
# Set the paths to the train and test directories
train_directory ='/content/drive/MyDrive/r1data/train'
test_directory = '/content/drive/MyDrive/r1data/test'
```

Figure 9: Setting path for Train and Test

6 Model building

6.0.1 Capsule Network

The Figure Below depicts a Capsule Network with 3 layers, primary and digital capsule layers with 256 convolution layers, 9x9 filters and routing agreement.

```
def CapsNet(input_shape, num_classes):
    model = tf.keras.Sequential()

    # First layer: Convolutional layer with 9x9 filter and stride of 1
    model.add(layers.Conv2D(256, kernel_size=(9, 9), strides=(1, 1), activation='relu', input_shape=input_shape))

    # Second layer: Primary Capsule Network formed by 9x9 convolutions and stride of 2
    model.add(layers.Conv2D(256, kernel_size=(9, 9), strides=(2, 2), activation='relu'))

    # Third layer: Routing by Agreement process
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dense(256, activation='relu'))

    # Last layer: Fully connected layer with softmax activation
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model
```

Figure 10: Base Model Building

The model was compiled and used along with test data for evaluation purposes. 3

```
# Define and compile the CapsuleNet model
model = CapsNet(input_shape=(128, 128, 3), num_classes=num_classes)
```

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

Figure 11: compile model

6.0.2 Data Augmentation

To generate synthetic data. Traditional Augmentation was implemented as shown below.

```python
# Preprocess the training and testing sets using an image data generator
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

datagen.fit(X_train)
```

Figure 12: Data augmentation

6.0.3 Fine tune the model

Model tuning was performed in this research to enhance the results. The code and design of the model were defined as shown in fig 13

```python
def CapsNetwork(input_shape, number_of_classes):
    tune_base_model = tf.keras.Sequential()

    # First layer: Convolutional layer with 9x9 filter and stride of 1
    tune_base_model.add(layers.Conv2D(256, kernel_size=(9, 9), strides=(1, 1), activation='relu', input_shape=input_shape))

    # Second layer: Primary Capsule Network formed by 9x9 convolutions and stride of 2
    tune_base_model.add(layers.Conv2D(256, kernel_size=(9, 9), strides=(2, 2), activation='relu'))

    # Third layer: Routing by Agreement process
    tune_base_model.add(layers.Flatten())
    tune_base_model.add(layers.Dense(512, activation='relu'))
    tune_base_model.add(layers.Dense(256, activation='relu'))

    # Last layer: Fully connected layer with softmax activation
    tune_base_model.add(layers.Dense(number_of_classes, activation='softmax'))

    return tune_base_model
```

```python
# Load the pretrained model
tune_base_model = CapsNetwork(input_shape=(128, 128, 3), number_of_classes=number_of_classes)
```

```python
# Stage 1: Train only the last layer
for layer in tune_base_model.layers[:-1]:
    layer.trainable = False

tune_base_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the last layer on the rotated data
tune_base_model.fit(X_train_rotated_image, y_train_rotated_image, batch_size=32, epochs=5, validation_data=(Xtest, ytest))

# Stage 2: Train the entire model with a lower learning rate
for layer in tune_base_model.layers:
    layer.trainable = True

tune_base_model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 13: Tune Model

7 Evaluation

This research consists of different experiments to evaluate the model. The model is evaluated on the base of accuracy, precision, recall, sensitivity, and specificity and by comparing train and test accuracy.

```python
#  predictions for the test set
y_prediction = base_model.predict(Xtest)
y_prediction_labels = np.argmax(y_prediction, axis=1)

# Convert one-hot encoded test labels back to original class labels
y_test_labels = np.argmax(ytest, axis=1)

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test_labels, y_prediction_labels)

# Calculate specificity
specificity_base_model = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1])

# Calculate sensitivity (recall)
sensitivity_base_model = recall_score(y_test_labels, y_prediction_labels, average='weighted')

# Calculate F1 score
f1 = f1_score(y_test_labels, y_prediction_labels, average='weighted')

print(f"Specificity: {specificity_base_model:.4f}")
print(f"Sensitivity (Recall): {sensitivity_base_model:.4f}")
print(f"F1 Score: {f1:.4f}")
```

Figure 14: Accuracy of Experiment 1

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
# Print confusion matrix as a heatmap
name_of_class = ['infected','non-infected']
confusion_df = pd.DataFrame(conf_matrix, index=name_of_class, columns=name_of_class)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_df, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

Figure 15: Confusion for Experiment 1

```python
# Plot loss graph
plt.figure(figsize=(8, 6))
plt.plot(base_model_history.history['loss'], label='Training Loss')
plt.plot(base_model_history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()

# Plot accuracy graph
plt.figure(figsize=(8, 6))
plt.plot(base_model_history.history['accuracy'], label='Training Accuracy')
plt.plot(base_model_history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```

Figure 16: Graph for Experiment 1

```python
# Accuracy values for each dataset
accuracy_values = [accuracy_1_to_1, accuracy_1_to_2, accuracy_1_to_3]

# Dataset labels
dataset_labels = ['accuracy 1:1', 'accuracy 1:2', 'accuracy 1:3']

# Create line graph
plt.plot(dataset_labels, accuracy_values, marker='o', linestyle='-', color='b')

# Set axis labels and title
plt.xlabel('Dataset')
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracy on Different Datasets')

# Show gridlines
plt.grid(True)

# Show the line graph
plt.show()
```

Figure 17: Experiment 2 Graph

```python
# Assuming you have already trained the model and have predictions for the test set
y_predictions_synthetic_data1 = tune_base_model.predict(X_test_rotated_image)
y_pred_labels_synthetic_data1 = np.argmax(y_predictions_synthetic_data1, axis=1)

# Convert one-hot encoded test labels back to original class labels
y_test_labels_synthetic1 = np.argmax(y_test_rotated_image, axis=1)

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test_labels_synthetic1, y_pred_labels_synthetic_data1)

# Calculate specificity
specificity = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1])

# Calculate sensitivity (recall)
sensitivity = recall_score(y_test_labels_synthetic1, y_pred_labels_synthetic_data1, average='weighted')

# Calculate F1 score
f1 = f1_score(y_test_labels_synthetic1, y_pred_labels_synthetic_data1, average='weighted')

print(f"Specificity_synthetic_data: {specificity:.4f}")
print(f"Sensitivity (Recall)_synthetic_data: {sensitivity:.4f}")
print(f"F1 Score_synthetic_data: {f1:.4f}")
```

```
9/9 [==============================] - 1s 109ms/step
Specificity_synthetic_data: 1.0000
Sensitivity (Recall)_synthetic_data: 0.9674
F1 Score_synthetic_data: 0.9675
```

Figure 18: Accuracy of Experiment 3

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
# Print confusion matrix as a heatmap
class_names = ['infected','non-infected']
conf_df = pd.DataFrame(conf_matrix, index=class_names, columns=class_names)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_df, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix original data')
plt.show()
```

Figure 19: Confusion for Experiment 3

```python
# Plotting the graphs
import matplotlib.pyplot as plt

# Plot loss graph
plt.figure(figsize=(8, 6))
plt.plot(original_data_tune_model.history['loss'], label='Training Loss')
plt.plot(original_data_tune_model.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()

# Plot accuracy graph
plt.figure(figsize=(8, 6))
plt.plot(original_data_tune_model.history['accuracy'], label='Training Accuracy')
plt.plot(original_data_tune_model.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```

Figure 20: Graph for Experiment 3