National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Aniket Piralkar
Student ID: 20177291

School of Computing
National College of Ireland

Supervisor: Dr. Musfira Jilani, Dr. Paul Stynes, Dr. Pramod Pathak

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Aniket Piralkar
Student ID:	20177291
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Musfira Jilani, Dr. Paul Stynes, Dr. Pramod Pathak
Submission Due Date:	15/08/2012
Project Title:	Configuration Manual
Word Count:	1368
Page Count:	7

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:	
Date:	12th August 2022

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

Aniket Piralkar
20177291

1 Introduction

The aim of this configuration manual is to provide a detailed list of all the tasks that must be completed during the project implementation phase. All the software and hardware specifications are provided so that project can be reproduced in the future. The steps that need to be followed in order to run the code are covered in this manual, along with the coding and deployment processes.

2 System Configurations

2.1 Hardware Configuration

Hardware configurations of the system on which the code was implemented and executed are outlined in this section. Figure 1 shows the configurations of the system.

Figure 1: Hardware Configuration

The figure shows that the system has an Apple silicone M1 chip with 8GB of RAM and seven core GPU.

2.2 Software Configuration

As shown in figure 1, the system is running on macOS Monterey (version 12.5). The following section describes all the software used to implement this research.

2.2.1 Google Colaboratory

Google Colab, which is Google's computing platform, is used in this research project for all the code development [1]. Colab PRO subscription is used in order to increase the availability of RAM and to minimize the code run time. All the required libraries for the project have been imported into Colab and used to develop the model. First, the entire dataset and the metadata for the project are uploaded on Google drive. Then it is accessed in the Colab notebook from the same Google drive account. Figure 2 shows the Google drive mounting code block. Drive library from google.colab needs to be imported.

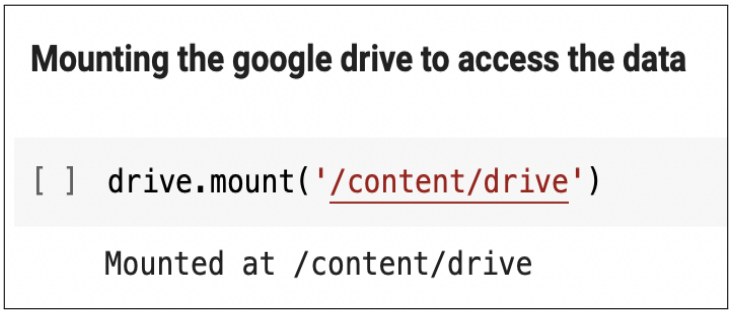

Figure 2: Google drive mount

Before start to run the code, Colab notebook settings should be updated by selecting **'Change runtime type'** from **'Runtime'** option. Figure 3 shows the notebook settings used in this research. GPU is selected in order to run the model faster. High RAM is selected to increase the runtime for the notebook. After running the command shown in figure 2, it will redirect to the authorization page of google. Once authorization is completed for that google account, data from google drive can be accessed into the Colab notebook for further processing.

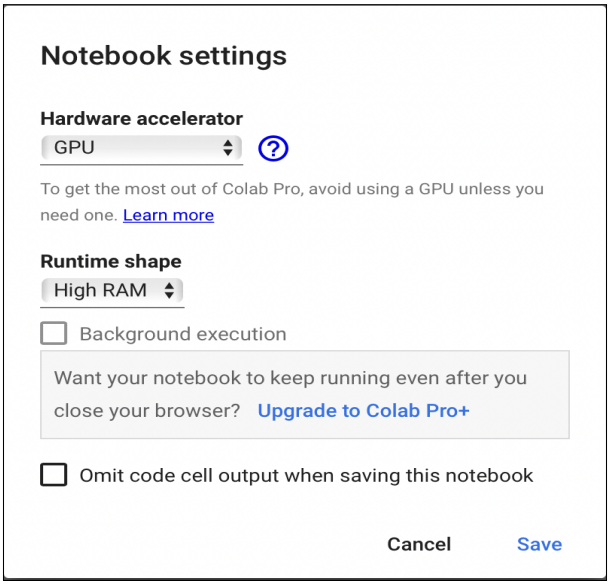

Figure 3: Google drive mount

[1] https://colab.research.google.com

2.2.2 Other software

Safari browser is used to run the Google Colab notebooks. Overleaf (which is an online Latex editor) is used for all the documentation and report writing of this project [2]. Overleaf is very easy to use and prepares the documents in real-time using the online editor. It can be accessed from any browser; no need to install it. Figure 4 shows the overleaf editor opened in the safari browser.

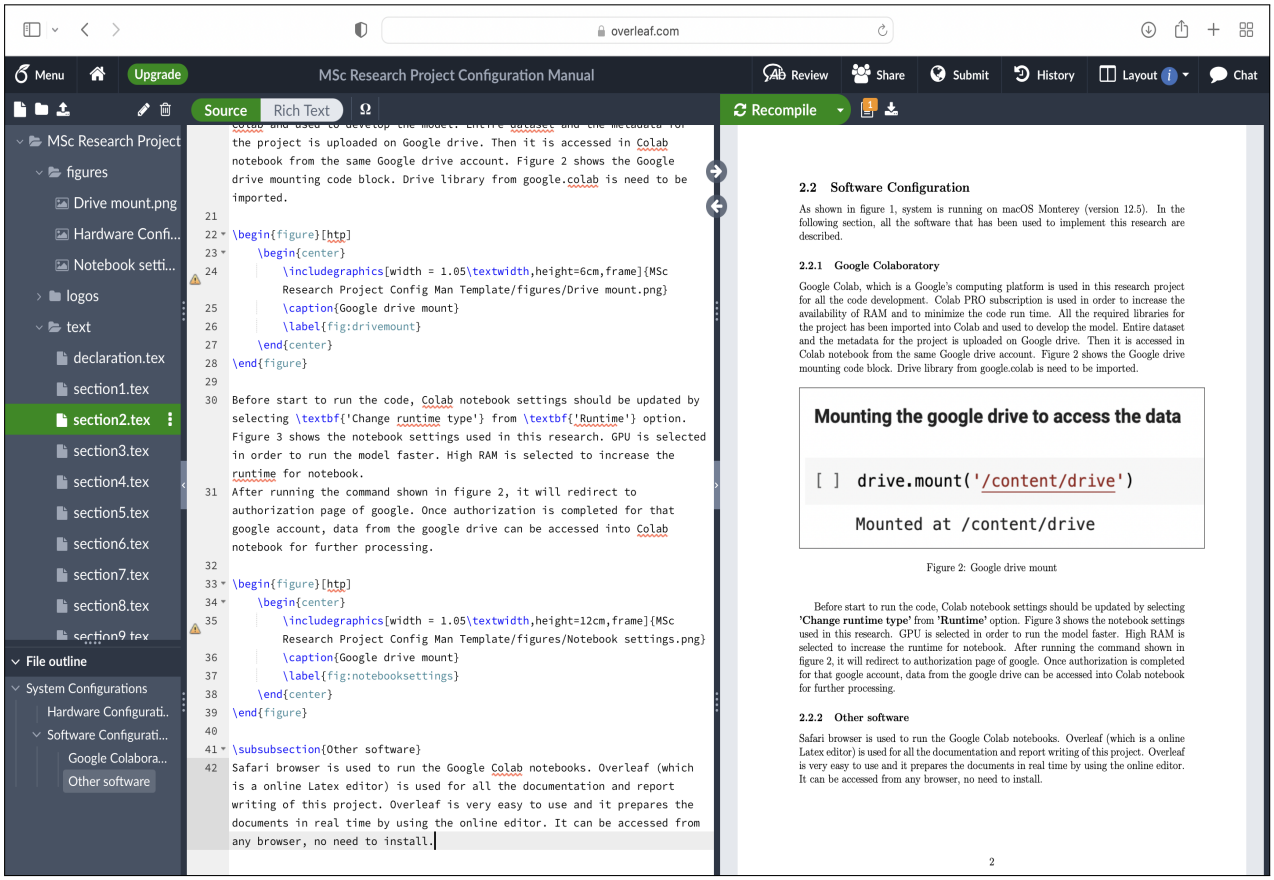

Figure 4: Overleaf editor

3 Data Preparation

Dataset used in this research is the German Traffic Sign Recognition Benchmark (GTSRB), and it is downloaded from the Kaggle website [3]. The dataset has train as well as test data. Train data has around 39K images in 43 different classes of traffic signs, and test data has 12K images. The metadata file is also present in the dataset in CSV format. It includes image dimensions, ROI, class id, and path information. Once the train dataset is read into the Colab notebook, it is converted into a NumPy array and shuffled randomly. The code block for the same is shown in figure 5.

Following that, it is divided into train and validation data with the ratio of 80:20, respectively, in order to train the model. Finally, one hot encoding with the class id label

[2] https://www.overleaf.com
[3] https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

Importing the train data

```
# Reading the train data
image_data = []
image_labels = []

for i in range(total_classes):
    path = train_path + '/' + str(i)
    images = os.listdir(path)

    for img in images:
        try:
            image = cv2.imread(path + '/' + img)
            image_fromarray = Image.fromarray(image, 'RGB')
            resize_image = image_fromarray.resize((img_height, img_width))
            image_data.append(np.array(resize_image))
            image_labels.append(i)
        except:
            print("Error in " + img)

# Converting the train data into numoy array
image_data = np.array(image_data)
image_labels = np.array(image_labels)

# Shuffling the train data
shuffle_indexes = np.arange(image_data.shape[0])
np.random.shuffle(shuffle_indexes)
image_data = image_data[shuffle_indexes]
image_labels = image_labels[shuffle_indexes]
a
print(image_data.shape, image_labels.shape)
```

Figure 5: Importing the train data

is performed on y set of train and validation data. Figure 6 shows the code block for the same.

Splitting the train data into train and val data

```
train_x, val_x, train_y, val_y = train_test_split(image_data, image_labels, test_size=0.2, random_state=42, shuffle=True) # Splitting ratio is 80:20

train_x = train_x/255
val_x = val_x/255

# One hot encoding of class id
train_y = keras.utils.to_categorical(train_y, total_classes)
val_y = keras.utils.to_categorical(val_y, total_classes)

print("Shape of train_x", train_x.shape)
print("Shape of val_x", val_x.shape)
print("Shape of train_y", train_y.shape)
print("Shape of val_y", val_y.shape)
```

Figure 6: Splitting the data into train and val data

4 Model Implementation

Model is implemented in this project by using Residual Network (ResNet) and data augmentation, which is a novelty of this research. ResNet is a network of residual blocks that has a skip connection which adds the identity mapping from one block to the following (Bouaafia et al.; 2021). Tensorflow library is used in this research to develop the model. Pre-trained weights such as imagenet are not used in this model. GitHub is referenced to develop the model [4]. Figure 7 shows the code definition for the residual block and ReLU activation layer in ResNet.

[4]https://gist.github.com/lazuxd/d7aaba284123bf3340e723701e381e6e#file-res_net-py

Model definition and summary

```
def relu_bn(inputs: Tensor) -> Tensor:
    relu = ReLU()(inputs)
    bn = BatchNormalization()(relu)
    return bn

def residual_block(x: Tensor, downsample: bool, filters: int, kernel_size: int = 3) -> Tensor:
    y = Conv2D(kernel_size=kernel_size,
               strides= (1 if not downsample else 2),
               filters=filters,
               padding="same")(x)
    y = relu_bn(y)
    y = Conv2D(kernel_size=kernel_size,
               strides=1,
               filters=filters,
               padding="same")(y)

    if downsample:
        x = Conv2D(kernel_size=1,
                   strides=2,
                   filters=filters,
                   padding="same")(x)
    out = Add()([x, y])
    out = relu_bn(out)
    return out
```

Figure 7: Residual block

Figure 8 shows the code definition to create a ResNet model. Input shape is (32,32,3) and the output dense layer has 43 nodes for classification. The softmax activation function is used. The learning rate is set to 0.001, and the epoch size is 20. Adam optimizer is used to compile the model with 'categorical_crossentropy' loss function and 'Accuracy' metric.

```
def create_resnet():

    inputs = Input(shape=(32, 32, 3))
    num_filters = 64

    t = BatchNormalization()(inputs)
    t = Conv2D(kernel_size=3,
               strides=1,
               filters=num_filters,
               padding="same")(t)
    t = relu_bn(t)

    num_blocks_list = [2, 5, 5, 2]
    for i in range(len(num_blocks_list)):
        num_blocks = num_blocks_list[i]
        for j in range(num_blocks):
            t = residual_block(t, downsample=(j==0 and i!=0), filters=num_filters)
        num_filters *= 2

    t = AveragePooling2D(4)(t)
    t = Flatten()(t)
    outputs = Dense(43, activation='softmax')(t)

    model = Model(inputs, outputs)

    lr = 0.001
    epochs = 20
    opt = Adam(lr=lr, decay=lr / (epochs * 0.5))

    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy']) # Model compilation

    return model
```

Figure 8: ResNet model

Data augmentation is used to train data while training the model. 'ImageDataGenerator' function is used as shown in figure 9.

Data Augmentation

```
[ ]  aug = ImageDataGenerator(
           rotation_range=12,
           zoom_range=0.26,
           width_shift_range=0.2,
           height_shift_range=0.2,
           shear_range=0.18,
           horizontal_flip=False,
           vertical_flip=False,
           fill_mode="nearest")
```

Figure 9: Data Augmentation

Figure 10 shows the model training with the first five epochs. As shown in the figure, the data augmentation function is called in model.fit to augment the data. Train and validation data are used for model training. Following the model training, model performance is evaluated by plotting the model accuracy and model loss plot for train and validation data.

Model training

```
[ ] history = model.fit(aug.flow(train_x, train_y, batch_size=batch_size), epochs=20, validation_data=(val_x, val_y))

    Epoch 1/20
    868/868 [==============================] - 51s 42ms/step - loss: 2.5281 - accuracy: 0.2667 - val_loss: 3.4587 - val_accuracy: 0.3255
    Epoch 2/20
    868/868 [==============================] - 35s 41ms/step - loss: 1.0262 - accuracy: 0.6653 - val_loss: 0.5992 - val_accuracy: 0.8271
    Epoch 3/20
    868/868 [==============================] - 35s 41ms/step - loss: 0.3999 - accuracy: 0.8735 - val_loss: 0.2380 - val_accuracy: 0.9300
    Epoch 4/20
    868/868 [==============================] - 35s 41ms/step - loss: 0.2228 - accuracy: 0.9329 - val_loss: 0.0884 - val_accuracy: 0.9743
    Epoch 5/20
    868/868 [==============================] - 35s 41ms/step - loss: 0.1588 - accuracy: 0.9554 - val_loss: 0.0465 - val_accuracy: 0.9859
```

Figure 10: Model training

Test data of around 12K images are then imported by using the path from the CSV file. Figure 11 shows the code block for the same. 'read_csv' function is used to get the path from the CSV file. Test data is then converted into a NumPy array by using the NumPy library.

Importing the test data

```
# Reading the test data using csv
test = pd.read_csv(test_path + '/Test.csv')

labels = test["ClassId"].values
imgs = test["Path"].values

data =[]

for img in imgs:
    try:
        image = cv2.imread(test_path + '/' +img)
        image_fromarray = Image.fromarray(image, 'RGB')
        resize_image = image_fromarray.resize((img_height, img_width))
        data.append(np.array(resize_image))
    except:
        print("Error in " + img)
test_x = np.array(data)
test_x = test_x/255
```

Figure 11: Importing the test data

Predict function from the model is used to predict the test data. 'accuracy_score' function from 'sklearn.metrics' library is used to calculate the test data accuracy. Figure 12 shows the code block for the model prediction and test data accuracy.

Model prediction

```
predict_x = model.predict(test_x)
classes_x = np.argmax(predict_x,axis=1)

print('Test data accuracy: ',accuracy_score(labels, classes_x)*100)
```

Figure 12: Model prediction

The confusion matrix is plotted and evaluated to validate the performance of the model on test data. Figure 13 shows the code for the confusion matrix.

Evaluating the confusion matrix

```
cf = confusion_matrix(labels, classes_x)

df_cm = pd.DataFrame(cf, index = classes,  columns = classes)
plt.figure(figsize = (20,20))
sns.heatmap(df_cm, annot=True)
```

Figure 13: Confusion matrix

Few samples from the test data are plotted with the actual and predicted value of the class. The code block for the same is shown in figure 14.

Plotting the test data

```
# Plotting the samples from test data with actual and predicted value
plt.figure(figsize = (25, 25))

start_index = 0
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    prediction = classes_x[start_index + i]
    actual = labels[start_index + i]
    col = 'g'
    if prediction != actual:
        col = 'r'
    plt.xlabel('Actual={} || Pred={}'.format(actual, prediction), color = col)
    plt.imshow(test_x[start_index + i])
plt.show()
```

Figure 14: Plotting the test data

References

Bouaafia, S., Messaoud, S., Maraoui, A., Ammari, A. C., Khriji, L. and Machhout, M. (2021). Deep pre-trained models for computer vision applications: Traffic sign recognition, *2021 18th International Multi-Conference on Systems, Signals Devices (SSD)*, pp. 23–28.