Configuration Manual

MSc Research Project
Data Analytics

Priyanshu Srivastava
Student ID: X21199787

School of Computing
National College of Ireland

Supervisor: Prof. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Priyanshu Srivastava
Student ID:	X21199787
Programme:	Data Analytics **Year:** 2022/2023
Module:	MSc. Research Project
Lecturer:	Prof. Hicham Rifai
Submission Due Date:	14/08/2023
Project Title:	Ocular Disease Detection using Deep Convolution Neural Network

Word Count: 897 **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: ………

Date: ………

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

Priyanshu Srivastava
Student ID: X21199787

1 Introduction

The configuration manual gives an overview of all the requirements needed for the code to run and provides a procedure to get the code executed and obtain results.

2 System Specifications

The code is implemented on Windows 11 operating system, 12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz processor and 16 GB of RAM.

Device name	PRIYANSHU_07
Processor	12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	D173EFC5-AD0F-4EE4-A668-818D58BF399D
Product ID	00342-42622-70416-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

3 Software Specification

We have used Python 3.11.1 version for writing the code and it was implemented on Visual Studio Code.

Python: https://www.python.org/downloads/ This is the link for downloading python

Visual Studio Code: https://code.visualstudio.com/ link for downloading Visual Studio Code.

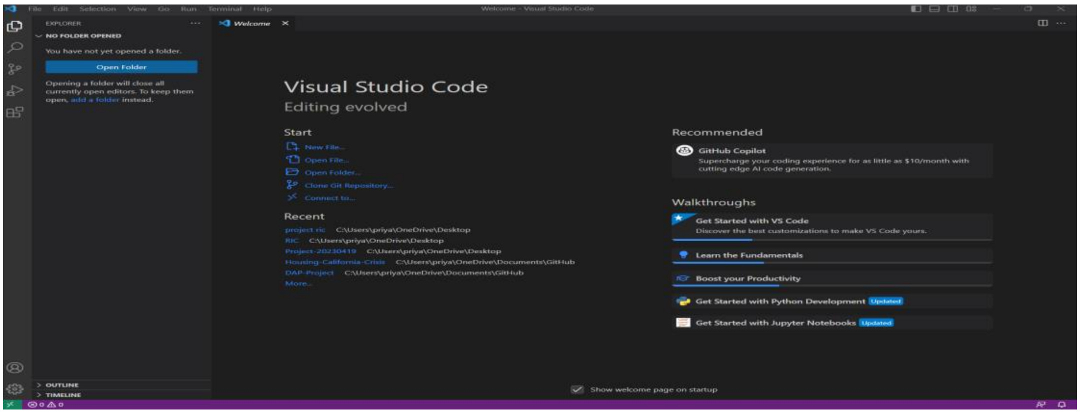

Figure 1: Visual Studio Code

4 Libraries Required for code implementation

Here is the list of all the libraries and packages used to implement the code. The packages were installed using the pip command:

1. os: Standard library for interacting with the operating system.
2. numpy (np): Fundamental package for scientific computing with Python.
3. pandas (pd): Data analysis and manipulation library.
4. tensorflow (tf): Open-source machine learning framework.
5. PIL (Python Imaging Library): Library for opening, manipulating, and saving image files.
6. ImageDataGenerator: Part of the TensorFlow library for real-time data augmentation and preprocessing.
7. matplotlib.pyplot (plt): Data visualization library for creating plots and charts.
8. seaborn (sns): Data visualization library built on top of Matplotlib for statistical graphics.
9. sklearn.metrics: Part of the scikit-learn library for evaluating machine learning models.
10. Conv2D, MaxPooling2D, Flatten, Dense, Layer, Input: Keras layers for building neural network models.
11. Model, Sequential: Keras classes for creating and working with neural network models.
12. Adam: Optimizer for training neural networks using gradient-based optimization.
13. classification_report: Function from scikit-learn for generating classification reports.
14. confusion_matrix: Function from scikit-learn for generating confusion matrices.

5 Dataset:

The dataset was gathered from Mendeley here is the link of the dataset Multi-Label Retinal Diseases (MuReD) Dataset - Mendeley Data. This dataset contains 2208 fundus images of eye and also contains train and validation csv.

6 Data Pre-Processing:

First we have to convert all the .tif file in the dataset to .png file as shown in Figure 2.

```
# Define a function to convert .tif images to .png format
def tif_to_png(input_directory, output_directory):
    # Create the output directory if it doesn't exist
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    # Loop through files in the input directory
    for file_name in os.listdir(input_directory):
        if file_name.lower().endswith('.tif'):
            file_path = os.path.join(input_directory, file_name)
            output_path = os.path.join(output_directory, os.path.splitext(file_name)[0] + '.png')

            # Open the .tif image and save it as .png
            image = Image.open(file_path)
            image.save(output_path, 'png')

# Specify the input directory containing .tif images
input_dir = "C:\\Users\\priya\\OneDrive\\Desktop\\project ric\\images"

# Specify the output directory to save the converted .png images
output_dir = "C:\\Users\\priya\\OneDrive\\Desktop\\project ric\\images"

# Call the tif_to_png function to convert .tif to .png
tif_to_png(input_dir, output_dir)
```

Figure 2 .tif to .png conversion

Then we have to resize all the images into one specific size as you can see in Figure 3.

```
# Directory containing the images
image_dir = "C:\\Users\\priya\\OneDrive\\Desktop\\project ric\\images"

# Target size for resizing
target_size = (256, 256)  # Replace with your desired target size

# Output directory for resized images
output_dir = "C:\\Users\\priya\\OneDrive\\Desktop\\project ric\\resized_images"

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Iterate through all files in the directory
for filename in os.listdir(image_dir):
    if filename.endswith(".png") or filename.endswith(".jpg"):
        file_path = os.path.join(image_dir, filename)
        img = Image.open(file_path)

        # Resize the image
        resized_img = img.resize(target_size, Image.ANTIALIAS)

        # Save the resized image to the output directory
        output_path = os.path.join(output_dir, filename)
        resized_img.save(output_path)

print("All images have been resized and saved.")
```

Figure 3: Resizing of image

Normalizing of image is done as you can see in Figure 4.

```
# Function to load and preprocess an image
def load_image(file_path, target_size):
    image = Image.open(file_path)
    image = image.resize(target_size)
    return np.array(image) / 255.0
```

Figure 4: Normalizing of image

As you can see in Figure 5 we have implemented a code for ensuring all images have same shape.

```
for _, row in train_data.iterrows():
    file_name = row['ID'] + '.png'
    file_path = os.path.join(image_dir, file_name)
    image = load_image(file_path, target_size)

    # Ensure all images have the same shape
    if image.shape == target_size + (3,):
        train_images.append(image)
        labels = np.array([row[column] if not pd.isna(row[column]) else 0.0 for column in class_columns], dtype=np.float32)
        train_labels.append(labels)

train_images = np.array(train_images)
train_labels = np.array(train_labels)  # Convert list of arrays to a single array

print("Train images shape:", train_images.shape)
print("Train labels shape:", train_labels.shape)
```

Figure 5 : Program for ensuring all images have same shape

7 Oversampling and Data Augmentation

In Figure 6 we have implemented oversampling and data augmentation technique

```
# Manually oversample each class based on the count you mentioned
oversample_counts = [396, 395, 135, 211, 125, 126, 209]

oversampled_train_images = []
oversampled_train_labels = []

for i, count in enumerate(oversample_counts):
    indices = np.where(train_labels[:, i] == 1)[0]
    sampled_indices = np.random.choice(indices, count, replace=True)
    oversampled_train_images.extend(train_images[sampled_indices])
    oversampled_train_labels.extend(train_labels[sampled_indices])

oversampled_train_images = np.array(oversampled_train_images)
oversampled_train_labels = np.array(oversampled_train_labels)

# Data Augmentation
train_datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
```

.

Figure 6: Oversampling and Data Augmentation

8 Model Building

In Figure 7 you can see the implementation of capsule network.

```
# Build a Capsule Layer
class CapsuleLayer(Layer):
    def __init__(self, num_capsules, dim_capsule, routings=3, **kwargs):
        super(CapsuleLayer, self).__init__(**kwargs)
        self.num_capsules = num_capsules
        self.dim_capsule = dim_capsule
        self.routings = routings

    # Squash function
    def squash(self, vector):
        norm = tf.norm(vector, axis=-1, keepdims=True)
        norm_squared = norm**2
        return (norm_squared / (1 + norm_squared)) * (vector / (norm + 1e-10))

    # ...

# Load the trained convolutional model
conv_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
])

# Create the Capsule Network model using Functional API
input_image = Input(shape=(256, 256, 3))
encoded_image = conv_model(input_image)
capsule_layer = CapsuleLayer(num_capsules=len(class_columns), dim_capsule=32, routings=5)(encoded_image)
output_capsules = Dense(len(class_columns), activation='sigmoid', kernel_regularizer=tf.keras.regularizers.l2(0.001))(capsule_layer)
capsule_model = Model(inputs=input_image, outputs=output_capsules)
```

Figure 7: Capsule Network implementation

In Figure 8 we implemented learning rate scheduler, training the model with Augmented and oversampled data, evaluating the model on validation set and generation of confusion matrix.

```
# Compile the model with learning rate schedule
optimizer = Adam(learning_rate=0.0001)
capsule_model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Define a learning rate schedule
def lr_schedule(epoch, lr):
    if epoch < 20:
        return lr
    else:
        return lr * tf.math.exp(-0.1)

# Create a learning rate scheduler callback
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lr_schedule)

# Train the model with data augmentation
batch_size = 32
epochs = 50

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

train_generator = train_datagen.flow(oversampled_train_images, oversampled_train_labels, batch_size=batch_size)  # Use oversampled data

capsule_model.fit(train_generator, epochs=epochs, validation_data=(validation_images, validation_labels), callbacks=[early_stopping, lr_scheduler])

# Evaluate the model on the validation set
loss, accuracy = capsule_model.evaluate(validation_images, validation_labels)
print(f"Validation Loss: {loss}, Validation Accuracy: {accuracy}")

# Generate predictions on the validation set
predictions = capsule_model.predict(validation_images)
predictions_binary = (predictions > 0.5).astype(int)

# Print classification report and confusion matrix
print(classification_report(validation_labels, predictions_binary))
print(confusion_matrix(validation_labels.argmax(axis=1), predictions_binary.argmax(axis=1)))
```

Figure 8

9 Evaluation

Figure 9 shows code for visualising Evaluation Metrics by Class.

```
# Generate predictions on the validation set
predictions = capsule_model.predict(validation_images)
predictions_binary = (predictions > 0.5).astype(int)

# Print classification report
class_names = ['DR', 'NORMAL', 'MH', 'ODC', 'TSLN', 'ARMD', 'OTHER']
report = classification_report(validation_labels, predictions_binary, target_names=class_names, output_dict=True)

# Extract precision, recall, and F1-score for each class
precision = [report[class_name]['precision'] for class_name in class_names]
recall = [report[class_name]['recall'] for class_name in class_names]
f1_score = [report[class_name]['f1-score'] for class_name in class_names]

# Create a bar chart
x = range(len(class_names))
plt.bar(x, precision, width=0.2, label='Precision')
plt.bar([i + 0.2 for i in x], recall, width=0.2, label='Recall')
plt.bar([i + 0.4 for i in x], f1_score, width=0.2, label='F1-Score')

plt.xlabel('Classes')
plt.ylabel('Score')
plt.title('Evaluation Metrics by Class')
plt.xticks([i + 0.2 for i in x], class_names)
plt.legend()
plt.show()
```

Figure 9: Evaluation Metrics

Figure 10 shows code for visualising confusion matrix and ROC-AUC curve.

```
# Generate predictions on the validation set
predictions = capsule_model.predict(validation_images)
predictions_binary = (predictions > 0.5).astype(int)

# Compute the confusion matrix
cm = confusion_matrix(validation_labels.argmax(axis=1), predictions_binary.argmax(axis=1))

# Plot the confusion matrix
plt.figure(figsize=(8, 8))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names, rotation=45)
plt.yticks(tick_marks, class_names)
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Compute ROC-AUC curve and micro-average ROC curve
fpr_micro, tpr_micro, _ = roc_curve(validation_labels.ravel(), predictions.ravel())
roc_auc_micro = auc(fpr_micro, tpr_micro)

# Plot ROC-AUC curve
plt.figure()
plt.plot(fpr_micro, tpr_micro, color='darkorange', lw=2, label=f'Micro-average ROC curve (area = {roc_auc_micro:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.legend(loc='lower right')
plt.show()
```

Figure 10: Confusion matrix

Figure 11 shows the code for visualising AUC curve for each class

```
# Generate predictions on the validation set
predictions = capsule_model.predict(validation_images)

# Compute ROC-AUC curve for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(class_columns)):
    fpr[i], tpr[i], _ = roc_curve(validation_labels[:, i], predictions[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Print AUC values for each class
for i, label in enumerate(class_columns):
    print(f'AUC for {label}: {roc_auc[i]:.4f}')

# Plot ROC-AUC curve for each class
plt.figure(figsize=(10, 8))

for i, label in enumerate(class_columns):
    plt.plot(fpr[i], tpr[i], label=f'{label} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve for Each Class')
plt.legend(loc='lower right')
plt.show()
```

Figure 11: AUC curve for each class.

Figure 12 shows the code for visualising sensitivity and specificity for each class.

```
# Calculate sensitivity and specificity for each class
sensitivity = np.diag(cm) / np.sum(cm, axis=1)
specificity = np.diag(cm) / np.sum(cm, axis=0)

class_names = ['DR', 'NORMAL', 'MH', 'ODC', 'TSLN', 'ARMD', 'OTHER']

# Create a DataFrame to display sensitivity and specificity
data = {'Class': class_names, 'Sensitivity': sensitivity, 'Specificity': specificity}
df = pd.DataFrame(data)

# Create a heatmap to visualize sensitivity and specificity
sns.set(font_scale=1.2)
plt.figure(figsize=(10, 6))
sns.heatmap(df.pivot_table(index='Class', values=['Sensitivity', 'Specificity'], aggfunc='mean'),
            annot=True, cmap='YlGnBu', fmt='.3f', linewidths=0.5)
plt.title('Sensitivity and Specificity for Each Class')
plt.show()
```

Figure 12: Sensitivity and Specificity

References

Rodriguez Rivera, Manuel Alejandro; Al-Marzouqi, Hasan; Liatsis, Panos (2022), “Multi-Label Retinal Diseases (MuReD) Dataset”, Mendeley Data, V1, doi: 10.17632/pc4mb3h8hz.1