National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Linda Susan Raju
Student ID: 22134409

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Linda Susan Raju
Student ID:	22134409
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Vladimir Milosavljevic
Submission Due Date:	14/12/2023
Project Title:	Enhancing Low-Light Images using Deep Learning
Word Count:	480
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:	Linda Susan Raju
Date:	31st January 2024

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

Linda Susan Raju
22134409

1 Introduction

The configuration manual serves as a guide for setting up and configuring the project environment for the proper execution of the research project. It provides information on hardware and software requirements and other system configurations. The purpose of this manual is to give step-by-step instructions on the necessary procedures ensuring easy replication of the experiment and also if interested, extend the project for their own research or applications.

2 System Requirements

2.1 Hardware Specifications

Figure 1 shows the hardware specifications

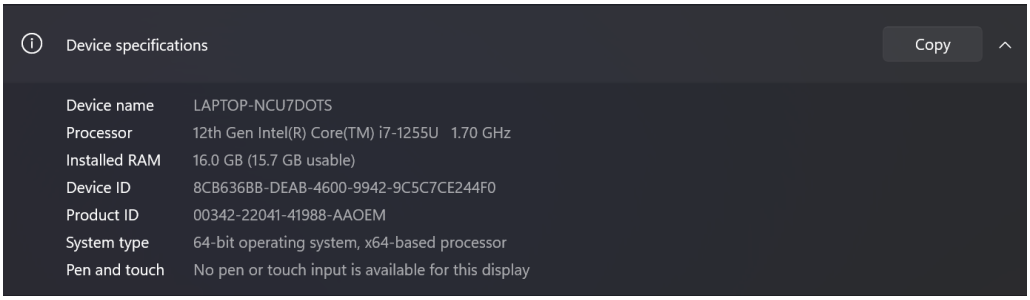


Figure 1: Hardware Specifications

Graphics:

- NVIDIA GeForce MX550
- Intel(R) UHD Graphics

2.2 Software Requirements

The project was executed using Python on Google Colab Pro+.

2.3 Software Dependencies

- Python (Programming Language) : Python 3.10.12
- TensorFlow (Deep Learning Library) : TensorFlow 2.12.0
- Keras (Neural Networks API) : Keras 2.12.0
- Matplotlib (Data Visualization Library) : 3.7.1
- Scikit-learn (Machine Learning Library): 1.2.2

3 Setting Up the Environment

Google Colab Environment:

- Mount Google Drive: To access the dataset which is uploaded in the Google Drive and also save the model checkpoints, we mount Google Drive using the following code snippet in Figure 2

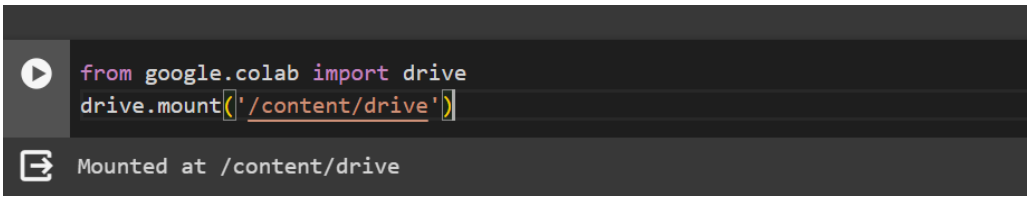


Figure 2: Code to Mount Google Drive

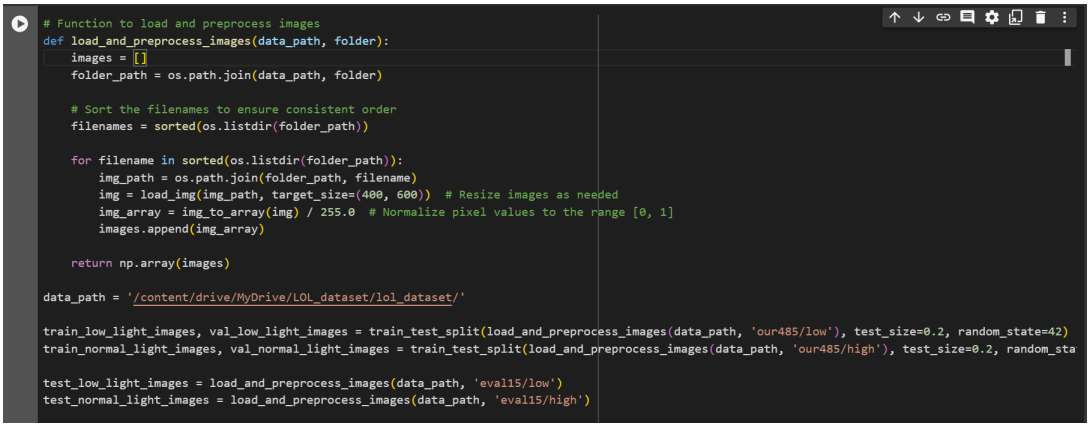
4 Dataset and Preprocessing

4.1 Dataset Acquisition

The LOL dataset is used consists of 500 low-light and normal-light image pairs and is separated into 485 training pairs and 15 testing pairs. The dataset used is public and available in Kaggle repository.
Link: https://www.kaggle.com/datasets/soumikrakshit/lol-dataset/

4.2 Dataset Preprocessing

Figure 3, the preprocessing code defines a function to load and preprocess images for a given folder in the dataset. The images are resized to (400, 600) pixels, and their pixel values are normalized to the range [0, 1]. The preprocessing is done separately for training and evaluation sets.



```python
# Function to load and preprocess images
def load_and_preprocess_images(data_path, folder):
    images = []
    folder_path = os.path.join(data_path, folder)

    # Sort the filenames to ensure consistent order
    filenames = sorted(os.listdir(folder_path))

    for filename in sorted(os.listdir(folder_path)):
        img_path = os.path.join(folder_path, filename)
        img = load_img(img_path, target_size=(400, 600))  # Resize images as needed
        img_array = img_to_array(img) / 255.0  # Normalize pixel values to the range [0, 1]
        images.append(img_array)

    return np.array(images)

data_path = '/content/drive/MyDrive/LOL_dataset/lol_dataset/'

train_low_light_images, val_low_light_images = train_test_split(load_and_preprocess_images(data_path, 'our485/low'), test_size=0.2, random_state=42)
train_normal_light_images, val_normal_light_images = train_test_split(load_and_preprocess_images(data_path, 'our485/high'), test_size=0.2, random_sta

test_low_light_images = load_and_preprocess_images(data_path, 'eval15/low')
test_normal_light_images = load_and_preprocess_images(data_path, 'eval15/high')
```

Figure 3: Dataset Preprocessing

5 Model Architecture

This research has four CNN models and one GAN model, which uses the CNN model as it's generator to produce an enhanced image. Each CNN model along the GAN model was executed in 4 separate files because of limited GPU memory during the computation. So other than CNN models, preprocessing, hyperparameter tuning and training code are the same.

5.1 Model 1: Basic CNN Model

```python
# Model 1: Basic CNN Model

def build_denoising_model():
    model = Sequential()

    # Encoder
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(None, None, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2), padding='same'))

    # Decoder
    model.add(Conv2DTranspose(128, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2DTranspose(64, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(UpSampling2D((2, 2)))

    # Output layer
    model.add(Conv2D(3, (3, 3), activation='sigmoid', padding='same'))

    return model
```

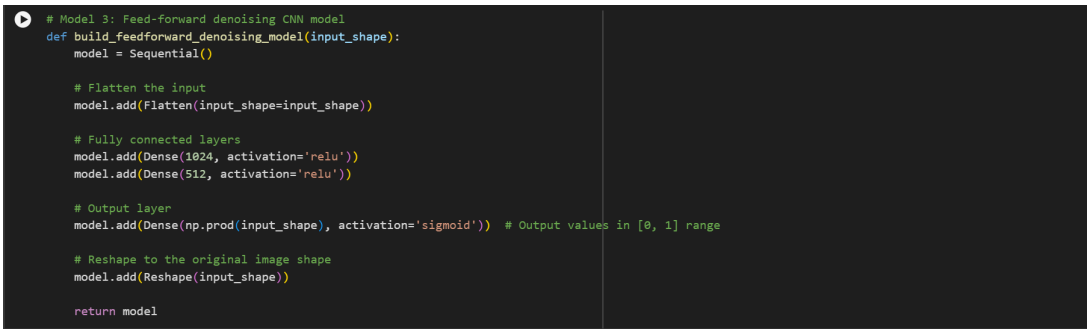
Figure 4: Basic CNN Model

5.2 Model 2: Feature-map Based CNN with Skip Connections

```python
# Model 2: FMBCNN with Skip Connections
def build_fmbcnn_with_skip_connections(input_shape):
    # Input layer
    inputs = Input(shape=input_shape)
    # Encoder (Downsampling)
    conv1 = Conv2D(64, kernel_size=3, strides=1, padding='same')(inputs)
    bn1 = BatchNormalization()(conv1)
    relu1 = Activation('relu')(bn1)
    skip_connection_1 = relu1
    conv2 = Conv2D(128, kernel_size=3, strides=2, padding='same')(relu1)
    bn2 = BatchNormalization()(conv2)
    relu2 = Activation('relu')(bn2)
    skip_connection_2 = relu2
    conv3 = Conv2D(256, kernel_size=3, strides=2, padding='same')(relu2)
    bn3 = BatchNormalization()(conv3)
    relu3 = Activation('relu')(bn3)
    # Feature Map Processing Layer
    up1 = UpSampling2D()(relu3)
    concat1 = Concatenate()([up1, skip_connection_2])
    conv_transpose = Conv2D(128, kernel_size=3, strides=1, padding='same')(concat1)
    bn4 = BatchNormalization()(conv_transpose)
    relu4 = Activation('relu')(bn4)
    # Decoder (Upsampling) with Skip Connections
    up2 = UpSampling2D()(relu4)
    concat2 = Concatenate()([up2, skip_connection_1])
    conv_transpose2 = Conv2D(64, kernel_size=3, strides=1, padding='same')(concat2)
    bn5 = BatchNormalization()(conv_transpose2)
    relu5 = Activation('relu')(bn5)
    # Output layer
    output = Conv2D(3, kernel_size=3, strides=1, padding='same', activation='sigmoid')(relu5)
    # Create the model
    model = Model(inputs=inputs, outputs=output)
    return model
```

Figure 5: Feature-map Based CNN with Skip Connections

5.3 Model 3: Feed-Forward Denoising CNN



```python
# Model 3: Feed-forward denoising CNN model
def build_feedforward_denoising_model(input_shape):
    model = Sequential()

    # Flatten the input
    model.add(Flatten(input_shape=input_shape))

    # Fully connected layers
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))

    # Output layer
    model.add(Dense(np.prod(input_shape), activation='sigmoid'))  # Output values in [0, 1] range

    # Reshape to the original image shape
    model.add(Reshape(input_shape))

    return model
```

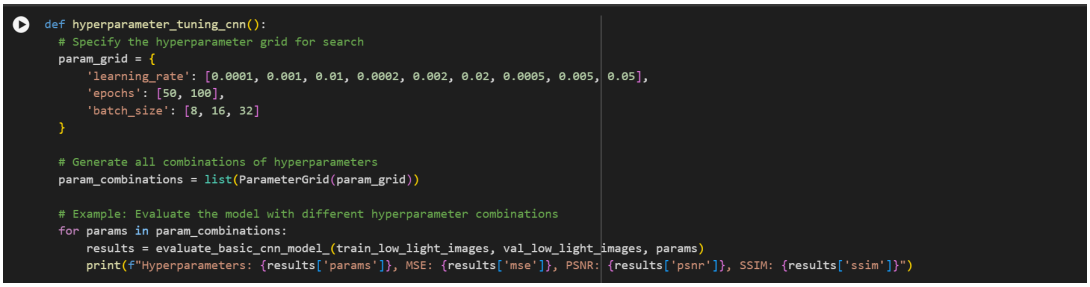
Figure 6: Feed-Forward Denoising CNN

5.4 Model 4: Feed-Forward Denoising CNN with Filtering Stages

```python
# Model 4: Feed-Forward Denoising CNN with Filtering Stages
def build_ffd_cnn_denoising_model(input_shape):
    model = Sequential()

    # Pre-processing stage (you may replace this with your specific pre-processing logic)
    model.add(Conv2D(64, kernel_size=3, strides=1, padding='same', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(ReLU())

    # Four convolution filtering stages
    for _ in range(4):
        model.add(Conv2D(64, kernel_size=3, strides=1, padding='same'))
        model.add(BatchNormalization())
        model.add(ReLU())
        model.add(MaxPooling2D(pool_size=2, strides=2, padding='same'))
        model.add(UpSampling2D(size=(2, 2)))

    # Global Average Pooling
    model.add(GlobalAveragePooling2D())

    # Fully connected layer
    model.add(Dense(256, activation='relu'))

    # Output layer
    model.add(Dense(np.prod(input_shape), activation='sigmoid'))
    model.add(Reshape(input_shape))

    return model
```

Figure 7: Feed-Forward Denoising CNN with Filtering Stages

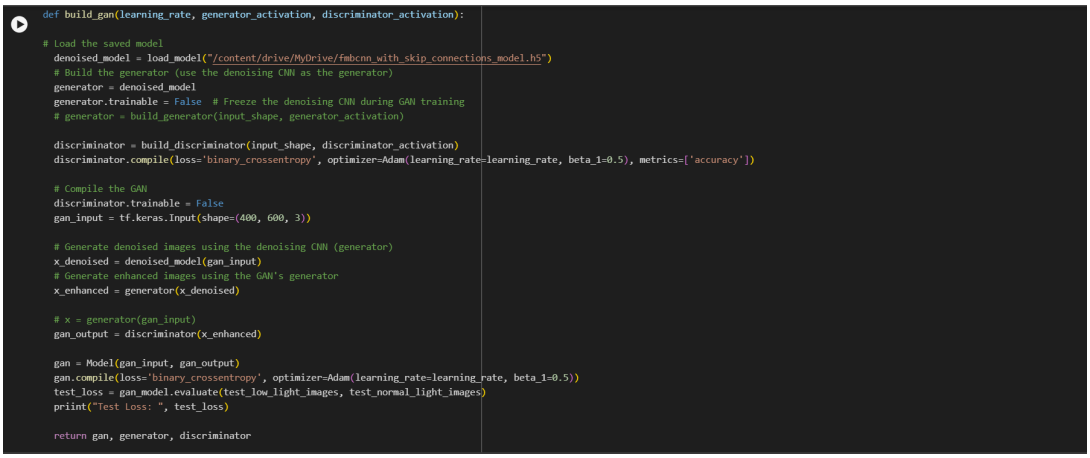
5.5 Hyperparameter Tuning



```python
def hyperparameter_tuning_cnn():
    # Specify the hyperparameter grid for search
    param_grid = {
        'learning_rate': [0.0001, 0.001, 0.01, 0.0002, 0.002, 0.02, 0.0005, 0.005, 0.05],
        'epochs': [50, 100],
        'batch_size': [8, 16, 32]
    }

    # Generate all combinations of hyperparameters
    param_combinations = list(ParameterGrid(param_grid))

    # Example: Evaluate the model with different hyperparameter combinations
    for params in param_combinations:
        results = evaluate_basic_cnn_model_(train_low_light_images, val_low_light_images, params)
        print(f"Hyperparameters: {results['params']}, MSE: {results['mse']}, PSNR: {results['psnr']}, SSIM: {results['ssim']}")
```

Figure 8: Hyperparameter Tuning Function

5.6 GAN Model



```python
def build_gan(learning_rate, generator_activation, discriminator_activation):

    # Load the saved model
    denoised_model = load_model("/content/drive/MyDrive/fmbcnn_with_skip_connections_model.h5")
    # Build the generator (use the denoising CNN as the generator)
    generator = denoised_model
    generator.trainable = False  # Freeze the denoising CNN during GAN training
    # generator = build_generator(input_shape, generator_activation)

    discriminator = build_discriminator(input_shape, discriminator_activation)
    discriminator.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=learning_rate, beta_1=0.5), metrics=['accuracy'])

    # Compile the GAN
    discriminator.trainable = False
    gan_input = tf.keras.Input(shape=(400, 600, 3))

    # Generate denoised images using the denoising CNN (generator)
    x_denoised = denoised_model(gan_input)
    # Generate enhanced images using the GAN's generator
    x_enhanced = generator(x_denoised)

    # x = generator(gan_input)
    gan_output = discriminator(x_enhanced)

    gan = Model(gan_input, gan_output)
    gan.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=learning_rate, beta_1=0.5))
    test_loss = gan_model.evaluate(test_low_light_images, test_normal_light_images)
    priint("Test Loss: ", test_loss)

    return gan, generator, discriminator
```

Figure 9: GAN Model

```python
# Function to evaluate the model with different hyperparameter combinations
def evaluate_basic_cnn_model_(train_images, val_images, params):
    # Build the denoising model
    basic_cnn_denoising_model = fmbcnn_with_skip_connections_model_compile(params)

    # Evaluate on validation set
    denoised_images = basic_cnn_denoising_model.predict(val_images)

    mse = MeanSquaredError()(val_images, denoised_images).numpy()
    psnr = peak_signal_noise_ratio(val_images[0], denoised_images[0], data_range=denoised_images[0].max())
    ssim_index = structural_similarity(val_images[0], denoised_images[0], multichannel=True)

    return {'params': params, 'mse': mse, 'psnr': psnr, 'ssim': ssim_index}
```

Figure 10: Evaluation Function

6 Training

All the models have similar training functions as in Figure 11

```python
enhanced_images_model_1 = []
# Compile CNN model
def basic_cnn_model_compile(params):
  # Build the feed-forward denoising model
  denoising_model = build_denoising_model()
  # Compile the model
  denoising_model.compile(optimizer=Adam(learning_rate=params['learning_rate']), loss='mse', metrics='accuracy')
  checkpoint_basic_cnn = ModelCheckpoint("/content/drive/MyDrive/basic_cnn_denoising_model.h5", save_best_only=True)
  # early_stopping_basic_cnn = EarlyStopping(patience=5, restore_best_weights=True)
  # Train the model
  history = denoising_model.fit(
      train_low_light_images, train_normal_light_images,  # Denoising autoencoder, input and output are the same
      epochs=params['epochs'],
      batch_size=params['batch_size'],
      validation_data=(val_low_light_images, val_normal_light_images),
      callbacks=[checkpoint_basic_cnn]
  )
  # Evaluate the model on the test set
  test_loss = denoising_model.evaluate(test_low_light_images, test_normal_light_images)

  enhanced_images_model_1 = denoising_model.predict(test_low_light_images)
  generate_basic_cnn_model_images(enhanced_images_model_1)

  return denoising_model
```

Figure 11: Training Function

7 Evaluation

The results are interpreted by analyzing the metrics provided, like Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM).

```python
# Function to evaluate the model with different hyperparameter combinations
def evaluate_basic_cnn_model_(train_images, val_images, params):
    # Build the denoising model
    basic_cnn_denoising_model = fmbcnn_with_skip_connections_model_compile(params)

    # Evaluate on validation set
    denoised_images = basic_cnn_denoising_model.predict(val_images)

    mse = MeanSquaredError()(val_images, denoised_images).numpy()
    psnr = peak_signal_noise_ratio(val_images[0], denoised_images[0], data_range=denoised_images[0].max())
    ssim_index = structural_similarity(val_images[0], denoised_images[0], multichannel=True)

    return {'params': params, 'mse': mse, 'psnr': psnr, 'ssim': ssim_index}
```

Figure 12: Evaluation Function

8 Visualization

Figure 13 shows the final enhanced image of the integrated model

Figure 13: Output images of the CNN-GAN Model