Configuration Manual

MSc Research Project
MSc Cloud Computing

Divya Henry
Student ID: x22241540

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Divya Henry		
Student ID:	x22241540		
Programme:	MSc Cloud Computing	**Year:**	2023-2024
Module:	MSc Research Project		
Lecturer:	Shaguna Gupta		
Submission Due Date:	16-09-2024		
Project Title:	Energy-Efficient Data Optimization for Resource-constrained Edge/Fog Computing Devices		
Word Count:	9310	**Page Count:**	23

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: 16 – 09 – 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

Divya Henry
x22241540

1 Introduction

This config manual gives a step-by-step guideline on installation and system operation of the DCSANet-JCF project. This tool is intended to allow users to effectively construct the environment, which they want to work in, so that they can accomplish their tasks in experiments in the best possible manner.

The manual covers the following key aspects: The manual covers the following key aspects:

- System Requirements: General instructions of the specifics of the hardware and software configuration that was used during development and testing.
- Environment Setup: The procedure on how to configure the Google Colab environment and how to look for data by using Google Drive.
- Library Installation: A list of dependencies that consist of an array of libraries and packages, and their versions along with their relevance in the projects.
- Dataset Preparation: The structure of the DIV2K dataset, and general information on its nature plus guiding principles on how to strip it down for use in the project.
- Model Architecture: An example of implementing the architecture of DCSANet, its elements such as ResidualBlock and the encoder-decoder structure.
- Training Configuration: Information are provided about hyperparameters, loss functions as well as the training loop used in order to fine-tune the proposed model.
- Evaluation Metrics: Discussion on Predictor Accuracy comprising of Mean Squared error (MSE), Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM).
- Visualization Tools: Prescriptions for creating and reading graphics as addition to the abstract representations of the model's output.

This way, by reproducing the context described in this manual, researchers and developers will be able to implement and further advance the project for experimentation in the more general area of Edge/Fog computation and Data Compression.

2 System Information

System	Google Colab Notebook with T4 GPU
CPU	Intel Xeon CPU
Memory	12 GB RAM
GPU	NVIDIA Tesla T4
GPU Memory	16 GB GDDR6
CUDA	2,560 Cores
Tensor	320 Cores
Disk	60GB
Python	3.10.12

3 Key Libraries and Packages

The following libraries and packages were utilized in this project.

3.1 TensorFlow

- **Version**: V2.17.0
- **Usage**: For building and training DCSANET architecture, this includes use of layers, models and GPU acceleration

3.2 Numpy

- **Version**: V1.26.4
- **Usage**: Utilized for handling arrays and performing numerical operations

3.3 Google.colab

- **Usage**: Used to mount google drive to the colab environment, enabling access to the datasets and other requirements

3.4 Tensorflow.keras

- **Usage**: High-level API for building and training deep learning models. Used to define DCSANET architecture including custom layers like `ResidualBlock`.

3.5 Matplotlib and Pandas

- **Version**: **matplotlib**: 3.7.1, **Pandas**: 2.1.4
- **Usage**: Data analytics and plotting libraries, utilized for data manipulation and representation.

4 Dataset

- DIV2K is a high-quality image dataset created specifically to enable data reduction and compression techniques. It contains 1000 high-definition high resolution images.
- Link: https://data.vision.ee.ethz.ch/cvl/DIV2K/

5 Detailed Implementation Steps

1. Environment Setup:

- Use Google Colab with access to an NVIDIA T4 GPU.
- Mount Google Drive:

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

2. Install required libraries:

!pip install tensorflow==2.17.0 numpy==1.26.4 matplotlib==3.7.1 pandas==2.1.4 scikit-learn pillow

3. Import necessary modules:

```
import tensorflow as tf
from tensorflow.keras import layers, models, applications
import numpy as np
from sklearn.model_selection import train_test_split
import os
from PIL import Image
```

4. Dataset Preparation:

- Implement the load_div2k_dataset function:

```
def load_div2k_dataset(data_dir, image_size=(256, 256),
num_images=1000):
    images = []
    for filename in os.listdir(data_dir)[:num_images]:
        if filename.endswith(".png"):
            img_path = os.path.join(data_dir, filename)
            img = Image.open(img_path).convert('RGB')
            img = img.resize(image_size)
            img_array = np.array(img) / 255.0
            images.append(img_array)
    return np.array(images, dtype=np.float32)
```

- Load and split the dataset:

```
data_dir =
"/content/drive/MyDrive/Datasets/DIV2K/DIV2K_train_HR/DIV2K_train_HR"
images = load_div2k_dataset(data_dir)
x_train, x_test = train_test_split(images, test_size=0.2,
random_state=42)
x_train = tf.convert_to_tensor(x_train, dtype=tf.float32)
x_test = tf.convert_to_tensor(x_test, dtype=tf.float32)
```

5. Model Architecture:

- Implement the ResidualBlock class:

```
class ResidualBlock(layers.Layer):
    def __init__(self, filters, **kwargs):
        super(ResidualBlock, self).__init__(**kwargs)
        self.conv1 = layers.Conv2D(filters, 3, padding='same')
        self.bn1 = layers.BatchNormalization()
        self.relu = layers.ReLU()
        self.conv2 = layers.Conv2D(filters, 3, padding='same')
        self.bn2 = layers.BatchNormalization()
        self.add = layers.Add()
```

```
    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.add([x, inputs])
        return self.relu(x)
```

- Implement the DCSANet class:

```
class DCSANet(models.Model):
    def __init__(self, latent_dim):
        super(DCSANet, self).__init__()
        self.latent_dim = latent_dim
        self.encoder = tf.keras.Sequential([
            # Encoder layers
        ])
        self.decoder = tf.keras.Sequential([
            # Decoder layers
        ])

    def encode(self, x):
        mean, logvar = tf.split(self.encoder(x), num_or_size_splits=2,
axis=1)
        return mean, logvar

    def reparameterize(self, mean, logvar):
        eps = tf.random.normal(shape=mean.shape)
        return eps * tf.exp(logvar * .5) + mean

    def decode(self, z):
        return self.decoder(z)

    def call(self, inputs):
        mean, logvar = self.encode(inputs)
        z = self.reparameterize(mean, logvar)
        reconstructed = self.decode(z)
        return reconstructed
```

- Loss Function:

```
vgg = applications.VGG19(include_top=False, weights='imagenet')
feature_extractor = models.Model(inputs=vgg.input,
outputs=vgg.get_layer('block3_conv3').output)

def perceptual_loss(y_true, y_pred):
    return tf.reduce_mean(tf.square(feature_extractor(y_true) -
feature_extractor(y_pred)))
```

```
def vae_loss(model, x):
    mean, logvar = model.encode(x)
    z = model.reparameterize(mean, logvar)
    x_recon = model.decode(z)
    reconstruction_loss = tf.reduce_mean(tf.square(x - x_recon))
    perceptual = perceptual_loss(x, x_recon)
    kl_loss = -0.5 * tf.reduce_mean(1 + logvar - tf.square(mean) -
tf.exp(logvar))
    total_loss = reconstruction_loss + 0.1 * perceptual + 0.1 * kl_loss
    return total_loss
```

6. Model Training:

```
latent_dim = 512
model = DCSANet(latent_dim)
optimizer = tf.keras.optimizers.Adam(1e-4)

@tf.function
def train_step(model, x, optimizer):
    with tf.GradientTape() as tape:
        loss = vae_loss(model, x)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    return loss

epochs = 100
batch_size = 16

for epoch in range(epochs):
    total_loss = 0
    num_batches = 0
    for i in range(0, len(x_train), batch_size):
        batch = x_train[i:i+batch_size]
        loss = train_step(model, batch, optimizer)
        total_loss += loss
        num_batches += 1
    avg_loss = total_loss / num_batches
    print(f'Epoch {epoch+1}, Average Loss: {avg_loss.numpy():.4f}')
```

7. Model Evaluation:

```
def evaluate_model(model, x_test):
    mean, _ = model.encode(x_test)
    z = mean
    reconstructed = model.decode(z)
    mse = tf.reduce_mean(tf.square(x_test - reconstructed))
    psnr = tf.image.psnr(x_test, reconstructed, max_val=1.0)
    ssim = tf.image.ssim(x_test, reconstructed, max_val=1.0)
```

```python
        return mse.numpy(), tf.reduce_mean(psnr).numpy(),
    tf.reduce_mean(ssim).numpy()

mse, psnr, ssim = evaluate_model(model, x_test)
print(f"Mean Squared Error: {mse:.4f}")
print(f"Peak Signal-to-Noise Ratio: {psnr:.4f}")
print(f"Structural Similarity Index: {ssim:.4f}")
```

8. Visualization function

```python
def plot_reconstructed(model, x_test):
    n = 5
    mean, _ = model.encode(x_test[:n])
    reconstructed = model.decode(mean)
    fig, axes = plt.subplots(2, n, figsize=(20, 8))
    for i in range(n):
        axes[0, i].imshow(x_test[i])
        axes[0, i].axis('off')
        axes[1, i].imshow(reconstructed[i])
        axes[1, i].axis('off')
    plt.show()

plot_reconstructed(model, x_test)
```

References

Google Research. (2019). TensorFlow Federated: Machine Learning on Decentralized Data. Retrieved from https://www.tensorflow.org/federated

Agustsson, E., & Timofte, R. (2017). NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (pp. 126-135). Retrieved from http://data.vision.ee.ethz.ch/cvl/DIV2K/