National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Carlos Alberto Noyola Sanchez
Student ID: X22232991

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. Carlos Alberto Noyola Sanchez ……

Student ID: …… X22232991 ……

Programme: …… MSc in Cloud Computing …… **Year:** ……2024….

Module: …… MSc Research Project ……

Lecturer: …… Vikas Sahni ……

Submission Due Date: ……12th Aug 2024 ……

Project Title: Optimizing Data Storage through Neural Network Based Adaptive Compression

Word Count: ……1300…… **Page Count:** ……11……

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: ……Carlos Alberto Noyola Sanchez ……

Date: ……10th Aug 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

Carlos Alberto Noyola Sanchez
Student ID: X22232991

1 Introduction

This configuration manual describes the steps to replicate the work submitted for the CNN model to infer compression method from images. This describes the preparation of the environment and the preparation of the dataset prior the CNN training.

2 System Configuration

To run the experiments, the code was run using Google Colab. On Google Colab there is an option to choose Python 3[1] (version 3.10.12) to run the experiments and libraries like TensorFlow (Abadi et al., 2016), Numpy (Harris et al., 2020), Pandas (Mckinney, 2011) and Scikit-learn (Pedregosa et al., 2018). Python provides useful libraries to run Machine learning models. All the libraries required are in Figure 1 and versions used in Table 1.

Library	Version
Pandas	2.1.4
NumPy	1.26.4
TensorFlow	2.17.0
Scikit-learn	1.3.2
Matplotlib[2]	3.7.1
Seaborn[3]	0.13.1
CV2 (OpenCV)[4]	4.10.0

Table 1: Library versions

[1] https://www.python.org
[2] https://matplotlib.org
[3] https://seaborn.pydata.org
[4] https://pypi.org/project/opencv-python

```
# Import necessary libraries
from google.colab import drive
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.metrics import Precision, Recall
import cv2
```

Figure 1: Libraries for CNN model

2.1 Preprocess datasets

Download and extract the following datasets.

https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-00/data
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-01/data
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-02/data
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-03/data
https://www.kaggle.com/datasets/ttahara/birdsong-resampled-train-audio-04/data
https://www.kaggle.com/datasets/miljan/stanford-dogs-dataset-traintest/data
https://www.kaggle.com/datasets/paultimothymooney/cvpr-2019-papers/data

Then, the files need to be compressed by using 5 different compression methods as show in Figure 2. The Python script compress the files in a specific directory in Google Drive. The required libraries are shown in Table 2. The compression methods are LZ78, Deflate, LZMA, Snappy and Zstandard. The function compress_file verify if there is a previous compress version of the file before read and compress the files again. The main function scans the directory and apply all the compression methods to each file. Then the files are stored in a separate directory with the compression method as file extension.

Library	Version
Snappy[5]	0.7.2
Zstandard [6]	0.23.0
Zlib[7]	1.3.1
LZMA[8]	Included in Python

Table 2: File compression libraries

[5] https://pypi.org/project/python-snappy
[6] https://pypi.org/project/zstandard
[7] https://docs.python.org/3/library/zlib.html
[8] https://docs.python.org/3/library/lzma.html

```python
import os
import snappy
import zstandard as zstd
import zlib
import lzma

def compress_lz78(data):
    compressed_data = bytearray()
    dictionary = {}
    next_code_index = 256
    buffer = bytearray()
    i = 0
    while i < len(data):
        buffer.append(data[i])
        if bytes(buffer) not in dictionary:
            dictionary[bytes(buffer)] = next_code_index
            next_code_index += 1
            compressed_data.extend(buffer[:-1])
            buffer = bytearray()
        else:
            i += 1
        i += 1
    if buffer:
        compressed_data.extend(buffer)
    return bytes(compressed_data)
def compress_deflate(data):
    return zlib.compress(data)
def compress_lzma(data):
    return lzma.compress(data)
def compress_snappy(data):
    return snappy.compress(data)
def compress_zstandard(data):
    cctx = zstd.ZstdCompressor()
    return cctx.compress(data)
compression_methods = {
    'lz78': compress_lz78,
    'deflate': compress_deflate,
    'lzma': compress_lzma,
    'snappy': compress_snappy,
    'zstandard': compress_zstandard
}
def compress_file(filepath, method, counter):
    #New folder 'from_kaggle_compressed', where compressed files will be stored
    new_path = filepath.replace('from_kaggle', 'from_kaggle_compressed')
    new_filepath = f"{new_path}.{method}"
    os.makedirs(os.path.dirname(new_filepath), exist_ok=True)
    if os.path.exists(new_filepath):
        filename = os.path.basename(filepath)
        print(f"Skipping {method}")
        print(f"already exists. {new_filepath} - {counter}")
        return
    with open(filepath, 'rb') as f:
        data = f.read()
    if data is None:
        print(f"Skipping {filepath}, no data read.")
        return
    compressed_data = compression_methods[method](data)
    if compressed_data is None:
        print(f"Compression method {method} failed for {filepath}")
        return
    with open(new_filepath, 'wb') as f:
        trimmed = '...'.join(new_filepath.split('/')[-3:])
        print(f"Created {trimmed}, # {counter}")
        f.write(compressed_data)
    return new_filepath

def compress_files_in_directory(directory):
    counter = 0
    for root, _, files in os.walk(directory):
        for file in files:
            counter += 1
            file_path = os.path.join(root, file)
            file_name, file_extension = os.path.splitext(file_path)
            print(f"File {file_name} Extension {file_extension}")
            for method in compression_methods:
                compress_file(file_path, method, counter)
    print(f"Creation finished")

# Specify the directory you want to compress files in
directory = '/content/drive/MyDrive/from_kaggle/pdf/'
compress_files_in_directory(directory)
```

Figure 2: Compression mixed files

The code shown in Figure 3 convert the original files into grayscale images. The grayscale images resolution can be set in the code. The resolutions selected were 48x48 and 120x120. It starts mounting the Google Drive to access the files. The function binary_to_image read the file in binary, converts the stream into pixels and then calculates the dimensions according to the length of the byte stream to create an image. Then an image is created in grayscale a resized -in this research 48x48 and 120x120 pixels where used- and stored as PNG image. The function convert_files_in_directory scans the directory and converts each file into image, when an image is not found already in the destination directory.

```python
from PIL import Image
import os
from google.colab import drive
drive.mount('/content/drive', force_remount=False)
my_path = '/content/drive/My Drive/from_kaggle/'

def binary_to_image(input_file, output_image, counter):
    with open(input_file, 'rb') as f:
        binary_data = f.read()
    # Generate pixel values from binary data
    pixel_values = list(binary_data)
    # Determine image dimensions
    width = int(len(pixel_values) ** 0.5)
    height = (len(pixel_values) + width - 1) // width
    # Create image
    img = Image.new('L', (width, height))
    img.putdata(pixel_values)
    # Resize the image to TARGET SIZE
    img = img.resize((48, 48), Image.Resampling.LANCZOS)
    # Save image
    img.save(output_image)
    print(f"{counter} Image binary {input_file}")

def convert_files_in_directory(input_dir, output_dir):
    counter = 0
    os.makedirs(output_dir, exist_ok=True)
    for root, dirs, files in os.walk(input_dir):
        for file in files:
            counter += 1
            input_file_path = os.path.join(root, file)
            output_file_path = os.path.join(output_dir, file + '.png')
            if os.path.exists(output_file_path):
                trimmed = '...'.join(output_file_path.split('/')[-3:])
                print(f"{counter}. Image {trimmed} already exists, skipping ")
                continue
            try:
                binary_to_image(input_file_path, output_file_path, counter)
            except Exception as e:
                print(f"Failed to convert {input_file_path}: {e}")
    print(f"Finish sequence of images")

# Example usage
input_directory = '/content/drive/My Drive/from_kaggle/'
output_directory = '/content/drive/My Drive/from_kaggle_to_image48/'
convert_files_in_directory(input_directory, output_directory)
```

Figure 3: Files to grayscale images

Then later, a CSV file is created. The CSV contains the name of the files with the size of the compressed file that offers the best compression ratio. The code shown in Figure 4 scans a specific directory and looks for the specific file extension -.lz78, .deflate, .lzma, .snappy and .zstandard- identify the smallest compressed file for every original file and store the information in the CSV file. The function find_samllest_compressed calculates and returns the smallest file based on the size. The function process_directory organize the files according to the original name before compression, check for the smallest file after compression and store that information in the CSV file. The information in the CSV includes the file path, original name file, compressed file name, size and format of the original file.

```
import os
import csv
def get_all_files(directory):
    file_list = []
    for root, _, files in os.walk(directory):
        for file in files:
            file_list.append(os.path.join(root, file))
    return file_list
def find_smallest_compressed(files):
    sizes = {file: os.path.getsize(file) for file in files}
    smallest_file = min(sizes.items(), key=lambda item: item[1])
    return smallest_file
def process_directory(directory, output_csv):
    all_files = get_all_files(directory)
    processed_files = {}
    for file in all_files:
        if any(file.endswith(ext) for ext in ['.lz78', '.deflate', '.lzma', '.snappy', '.zstandard']):
            original_file = os.path.splitext(file)[0]
            if original_file not in processed_files:
                processed_files[original_file] = []
            processed_files[original_file].append(file)
    results = []
    for original_file, compressed_files in processed_files.items():
        if compressed_files:
            smallest_file, size = find_smallest_compressed(compressed_files)
            original_extension = os.path.splitext(original_file)[1]
            file_format = os.path.splitext(smallest_file)[1]
            row = [original_file, os.path.basename(original_file), smallest_file, size, file_format, original_extension]
            results.append(row)
    with open(output_csv, 'w', newline='') as csvfile:
        csvwriter = csv.writer(csvfile)
        csvwriter.writerow([
            'File Path', 'Original File Name', 'Smallest Compressed File', 'Smallest Size',
            'Smallest Compressed Format', 'Original Extension'])
        csvwriter.writerows(results)
    print(f"CSV file '{output_csv}' created successfully.")

# Specify the directory to process and the output CSV file name
directory_to_process = '/content/drive/My Drive/from_kaggle_to_image'
output_csv_file = '/content/drive/My Drive/compressed_filesv2_updated.csv'
# Process the directory and create the CSV file
process_directory(directory_to_process, output_csv_file)
```

Figure 4: Create CSV file for CNN

3 Implementation

After the images are loaded and pre-processed, they are next used to train the CNN. Also, the CSV that contains the image names and labels is loaded as shown in Figure 5. The code starts by loading the CSV file that contains the metadata about the images. The path of the images is constructed by using the image_folder_path and the name of the file in the CSV file. The categorical labels stored in the CSV file then are converted into numeric values. The numeric values are one-hot encoded to make then suitable to train the CNN. Then the images are loaded and converted into binary pixels. Then dataset is divided into training and testing. Finally, the images are loaded and converted into NumPy matrixes with an extra channel expected by the CNN.

```
image_folder_path = '/content/from_kaggle_to_image48'
file_path = '/content/compressed_filesv2_updated.csv'
df = pd.read_csv(file_path, usecols=selected_columns)
#add column combination of column(1) + '.png' AND
#another column that tranform the categories from column(7) into numbers to feed a CNN model
df['image_file'] = df.iloc[:, 0].astype(str) + '.png'
df['category_numeric'] = df.iloc[:, 2].astype('category').cat.codes
df.iloc[:, [2, 4]]
image_names = df.iloc[:, 3].values  # Assuming image names are in the 4th column (index 3)
labels1 = df.iloc[:, 4].values       # Assuming labels are in the 5th column (index 4)
num_classes = len(np.unique(labels1))
labels = to_categorical(labels1, num_classes)
# Function to preprocess images with resizing to a lower resolution
def load_and_preprocess_image(path, index, total):
    image = tf.keras.preprocessing.image.load_img(path, color_mode='grayscale')
    image = tf.keras.preprocessing.image.img_to_array(image)
    image /= 255.0
    if index % 3000 == 0 or index == total - 1:
        print(f'Processed {index+1}/{total} images')
    return image
image_paths = [os.path.join(image_folder_path, image_name) for image_name in image_names]
train_paths, test_paths, train_labels, test_labels = train_test_split(
    image_paths, labels, test_size=0.2, random_state=42, stratify=labels)
train_images = np.array([load_and_preprocess_image(path, i, len(train_paths)) for i, path in enumerate(train_paths)])
test_images = np.array([load_and_preprocess_image(path, i, len(test_paths)) for i, path in enumerate(test_paths)])
train_images = np.expand_dims(train_images, axis=-1)
test_images = np.expand_dims(test_images, axis=-1)
```

Figure 5: Data pre-processing for CNN

Later the 3-layered 48px image CNN model is compiled and trained as shown in Figure 7. Figure 8, Figure 9 and Figure 9 show the 4-layered 48px image, 3-layered 120px image and 4-layered 120px image models.

```
#Model 3 layers
model3layers48 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
model3layers48.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()])
history = model3layers48.fit(train_images, train_labels,
                             epochs=30, validation_data=(test_images, test_labels))
modelname = 'thesis-model3-30epoch-48px.keras'
model3layers48.save('/content/' + modelname)  # Change to your desired save path
```

Figure 6: 3-layered CNN 48px

```
#4 layers 48PX
model4layers = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

Figure 7: 4-layered CNN 48px

```
#3 layers 120PX
model3layers120 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(120, 120, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

Figure 8: 3-layered CNN 120px

```
#4 layers 120px
model4layers120 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(120, 120, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

Figure 9: 4-layered CNN 120px

3.1 Generation of results and data visualization

The results are shown in the confusion matrixes and graphs as shown in **Error! Reference source not found.**. The code evaluates the trained CNN, by calculating the test loss, accuracy, precision and recall. Then generates a classification report by comparing the predictions against the true values of the images, a confusion matrix, and plot the validation accuracy and loss over epochs in order to check the learning process and tackle possibles overfitted scenarios. Note that the name of the model needs to be changed as the 3-layered 48px CNN model is used as example.

```
test_loss, test_accuracy, test_precision, test_recall = model3layers48.evaluate(test_images, test_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
print("Test Precision:", test_precision)
print("Test Recall:", test_recall)
# Predictions
predictions = model3layers48.predict(test_images)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(test_labels, axis=1)
# Classification report
print(classification_report(true_classes, predicted_classes))
# Confusion matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)
# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=range(num_classes), yticklabels=range(num_classes))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()
```

Figure 10: Evaluation of CNNs experiments

4 References

Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016) 'TensorFlow: A system for large-scale machine learning'. doi.org/10.48550/arXiv.1605.08695

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020) 'Array programming with NumPy', 357–362. doi.org/10.1038/s41586-020-2649-2

Mckinney, W. (2011) 'Pandas: a Foundational python library for data analysis and statistics'. Python High Performance Science Computer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2018) 'Scikit-learn: machine learning in python'. doi.org/10.48550/arXiv.1201.0490