Configuration Manual

Research Project
Master of Science in Data Analytics

Shreya Bhattacharya
Student ID: 18185207

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shreya Bhattacharya		
Student ID:	18185207		
Programme:	Master of Science in Data Analytics	**Year:**	2019-2020
Module:	Research Project		
Lecturer:	Research Project		
Submission Due Date:	17-08-2020		
Project Title:	Image Compression using Convolutional Autoencoder		
Word Count:	1232 **Page Count:** 14		

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.
I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature: Shreya Bhattacharya

Date: 17-08-2020

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

Shreya Bhattacharya
Student ID: 18185207

1 Introduction

Configuration manual is a document that has the documented details of the various necessary requirements of the project with respect to the software used, the hardware used and the different phases of the programming for the implementation of the research work in a sequential manner. The research work deals with the development of an autoencoder that consists of 20 layers, 10 layers in the encoder and the 10 layers in the decoder respectively for image compression such that the reconstructed image has the minimum loss of data. Dataset that has been used for the implementation are the MNIST digits dataset and the Fashion MNIST dataset, which are the integrated part of the Keras dataset, originally acquired from NIST and Zalando's article respectively. After the data acquisition is done based on the business understanding, the autoencoder has been constructed having the novel architecture for the image compression and denoising. The performance of the model has been evaluated with the help of various performance matrices and graphically for vivid understanding. The configuration manual provides a clear assistance to the user with respect to evaluation, appropriate usage of the code and it's clear and detailed execution. The objective of the research is to construct a convolutional autoencoder capable of image compression and denoising with high accuracy in the domain of Deep learning (**Image Compression using Convolutional Autoencoder).**

2 System Configurations

The different system configurations that are being used for the project implementation has been illustrated in this section. The system configuration specification provides an aid to the user as a pre-requisite for the conduction of project implementation.

2.1 Google colab specification:

RAM: 12.72 GB
Runtime Types: CPUs, GPUs, and. TPUs
Accelerator: GPU (GPU like the Tesla T4 GPU has been initialized and assigned during runtime)

2.2 Software Requirements

Python 3: The implementation of the project has been done on the python 3 platform that has been documented in the jupyter notebook format. Python is one of the most popularly used languages for web application development, desktop application development, for implementing the projects in the domain of data science and data analytics respectively.

Jupyter Notebook: The project has been implemented that involves the entire programming and execution in the Jupyter notebook. Jupyter notebook is a web application that has been made open source for the user intended for code implementation, programming, execution, perform various visualizations and perform various critical tasks.(*Project Jupyter*, n.d.)

Google colab: The project has been conducted in the google colaboratory also known as 'colab'. Google colab is a part of Google research which is a cloud platform that enables user to program and execute python code via a browser. This is of great advantage for performing data science project as it gives access to many high-end GPUs that leads to higher computational speed and needs no pre-requisites for working.(*Colaboratory – Google*, n.d.)

3 Data Acquisition and Preparation:

3.1 Guidance to Google colab

The following guidelines needs to be performed for using the google colab.

- A google account should be there for signing into the google colab.
- Once the sign up is done, it is necessary to open the link https://colab.research.google.com/notebooks/welcome.ipynb.
- Then one has to go to File and then have to select Python3 from there and then connect to notebook for the working environment.
- As GPU is being used as it aids fast performance, one has to change the runtime type to GPU from the runtime option.
- The file is then being saved to google drive.

3.2 Package Dependencies and Data Acquisition

3.2.1 Package Installation

In google colab many packages are pre-installed, in this project we are using tensorflow version 1, keras, numpy etc. The below snapshot shows the details of the libraries and packages that are installed.

```
#importing all the essential libraraies
import tensorflow as tf

import numpy as np

from sklearn.model_selection import train_test_split

from tensorflow.keras.layers import Input,MaxPooling2D,Conv2D,UpSampling2D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

import sklearn
```

3.2.2 Data Acquisition

The dataset that is being used for the implementation of the project is the Keras inbuilt datasets the MNIST digits dataset and fashion MNIST dataset. Below snapshot shows the acquisition of the data.

```
#import the mnist dataset
from keras.datasets import mnist
from keras.datasets import fashion_mnist

#load the data as the training and testing
(FullTrain,_),(FullTest,_)= mnist.load_data()
```

3.3 Data Preparation

Dividing the dataset into the training and the testing dataset after data acquisition is done then processing of the data, the reshaping of the data and normalization is done by dividing with 255 as the value of the pixel lies between 0 to 255. The below snapshot shows the processing of the data.

```
#number of training images used for the training the model()
#TrainingSet=FullTrain[0:8000]
TrainingSet=FullTrain
#number of testing images used for the testing the model()
#TestingSet=FullTest[0:2000]
TestingSet=FullTest
```

```
#reshape the training and testing dataset also normalising by dividing them by 255.
TrainingSet=TrainingSet.reshape(-1, 28,28, 1)/255

TestingSet=TestingSet.reshape(-1, 28,28, 1)/255
```

```
#divide some data of trainin dataset for validation
T,T1,V,V1 = train_test_split(TrainingSet,TrainingSet,test_size=0.2)
```

```
[ ]  #decide the shape of the imput image
     InImage = Input(shape = (28, 28, 1))
```

4 Implementation

The architecture of the autoencoder has been prepared and the definition of the autoencoder model is being done, along with the defining of the loss function and the optimizer. The below snapshot shows the development of the autoencoder model.

```
#function to define the encoder decoder
def autoencoder(InImage):
    #layers of encoder
    w = Conv2D(128, (3, 3), activation='relu', padding='same')(InImage)
    w = Conv2D(64, (3, 3), activation='relu', padding='same')(w)

    #Applying Maxpooling
    w = MaxPooling2D(pool_size=(2, 2))(w)

    w = Conv2D(32, (3, 3), activation='relu', padding='same')(w)
    w = Conv2D(16, (3, 3), activation='relu', padding='same')(w)
    w = Conv2D(8, (1, 1), activation='relu', padding='same')(w)

    #Applying Maxpooling
    w = MaxPooling2D(pool_size=(2, 2))(w)

    w = Conv2D(4, (3, 3), activation='relu', padding='same')(w)
    w = Conv2D(2, (1, 1), activation='relu', padding='same')(w)

    #bottle neck (1x1)Conv
    w = Conv2D(1, (1, 1), activation='relu', padding='same')(w)
```

```
#layers of decoder
w = Conv2D(2, (1, 1), activation='relu', padding='same')(w)
w = Conv2D(4, (3, 3), activation='relu', padding='same')(w)
w = Conv2D(8, (3, 3), activation='relu', padding='same')(w)

#Upsampling to increase the volume as to get the same size output
w = UpSampling2D((2, 2))(w)

w = Conv2D(16, (3, 3), activation='relu', padding='same')(w)
w = Conv2D(32, (3, 3), activation='relu', padding='same')(w)
w = Conv2D(64, (3, 3), activation='relu', padding='same')(w)

#Upsampling to increase the volume as to get the same size output
w = UpSampling2D((2, 2))(w)

w = Conv2D(128, (3, 3), activation='relu', padding='same')(w)
w = Conv2D(1, (3, 3), activation='tanh', padding='same')(w)

return w
```

```
#define the model and pass the our defined autoencoder function
TrainingAutoencoder = Model(InImage, autoencoder(InImage))
#compile the defined model by specifying the loss function and the optimizer
TrainingAutoencoder.compile(loss='mean_squared_error', optimizer = Adam())
```

```
#summary of autoencoder model
TrainingAutoencoder.summary()
```

4.1 Experiment 1

The model is being trained with the MNIST digits dataset which is run till 20 epochs to reach the optimum loss value, along with the definition of the encoder and the decoder model has been done. The evaluation of the model is being by visual and graphical interpretation. The below snapshots illustrate the process sequentially.

```
#Train the model

Performance = TrainingAutoencoder.fit(T,V,epochs = 20,validation_data = (T1,V1),callbacks = [ModelWeights])
```

```
#defining the encoder model
Autoencoder_encoder = Model(InImage,TrainingAutoencoder.layers[10].output)
#see the encoder summary
Autoencoder_encoder.summary()
```

```
#the layer which has to be find
LayerNo = 11
#the shape of inserted image
DimesionInImage = TrainingAutoencoder.layers[LayerNo].get_input_shape_at(0)[1:]
#input for the 11 layer
Inputs = Input(shape=DimesionInImage)

# introducing the new neuron in each layer
w = Inputs
for L in TrainingAutoencoder.layers[LayerNo:]:
    w = L(w)

# define decoder
Autoencoder_decoder = Model(Inputs, w)
Autoencoder_decoder.summary()
```

```
[ ]  # Encoder make prediction for the testing dataset
     PredictionOfEncoder = Autoencoder_encoder.predict(TestingSet)
```

```
[ ]  # Decoder predictions
     PredictionOfDecoder = Autoencoder_decoder.predict(PredictionOfEncoder)
```

```
]  from matplotlib import pyplot as pt
   M=10
   Img=pt.figure(figsize=(10, 20))
   col = 2
   NoOfRows = 5
   T = 0
   M = 0
   for j in range(1, col*NoOfRows +1):
       if j%2 == 0:
           I = PredictionOfDecoder[T,...,0]#prediction of decoder
           T+=1
       else:
           I = TestingSet[M,...,0]#actual data
           M+=1
       Img.add_subplot(NoOfRows, col, j)
       pt.imshow(I,cmap = 'gray')
   pt.show()
```

```
loss_model= Performance.history['loss']
loss_validation= Performance.history['val_loss']
e= range(len(loss_model))
import matplotlib.pyplot as plt
plt.figure()
plt.plot(e, loss_model, 'r', label='Loss of training data')
plt.plot(e, loss_validation, 'b',label='Loss of validation data')
plt.title('Training and Validation Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```

4.2 Experiment 2

The model is being evaluated against the fashion MNIST dataset as it gives a better visualization of the reconstructed image and the graph is being plotted to see the results generated. The below snapshot shows the training of the model and the evaluation of the result.

```
#number of training images used for the training the model()
#TrainingSet=FullTrain[0:8000]
TrainingSet=FullTrain
#number of testing images used for the testing the model()
#TestingSet=FullTest[0:2000]
TestingSet=FullTest
```

```
#reshape the training and testing dataset also normalising by dividing them by 255.
TrainingSet=TrainingSet.reshape(-1, 28,28, 1)/255

TestingSet=TestingSet.reshape(-1, 28,28, 1)/255
```

```
#divide some data of trainin dataset for validation
T,T1,V,V1 = train_test_split(TrainingSet,TrainingSet,test_size=0.2)
```

```
#function to define the encoder decoder
def autoencoder(InImage):
    #layers of encoder
    w = Conv2D(128, (3, 3), activation='relu', padding='same')(InImage)
    w = Conv2D(64, (3, 3), activation='relu', padding='same')(w)

    #Applying Maxpooling
    w = MaxPooling2D(pool_size=(2, 2))(w)

    w = Conv2D(32, (3, 3), activation='relu', padding='same')(w)
    w = Conv2D(16, (3, 3), activation='relu', padding='same')(w)
    w = Conv2D(8, (1, 1), activation='relu', padding='same')(w)

    #Applying Maxpooling
    w = MaxPooling2D(pool_size=(2, 2))(w)

    w = Conv2D(4, (3, 3), activation='relu', padding='same')(w)
    w = Conv2D(2, (1, 1), activation='relu', padding='same')(w)

    #bottle neck (1x1)Conv
    w = Conv2D(1, (1, 1), activation='relu', padding='same')(w)
```

```
  #layers of decoder
  w = Conv2D(2, (1, 1), activation='relu', padding='same')(w)
  w = Conv2D(4, (3, 3), activation='relu', padding='same')(w)
  w = Conv2D(8, (3, 3), activation='relu', padding='same')(w)

  #Upsampling to increase the volume as to get the same size output
  w = UpSampling2D((2, 2))(w)

  w = Conv2D(16, (3, 3), activation='relu', padding='same')(w)
  w = Conv2D(32, (3, 3), activation='relu', padding='same')(w)
  w = Conv2D(64, (3, 3), activation='relu', padding='same')(w)

  #Upsampling to increase the volume as to get the same size output
  w = UpSampling2D((2, 2))(w)

  w = Conv2D(128, (3, 3), activation='relu', padding='same')(w)
  w = Conv2D(1, (3, 3), activation='tanh', padding='same')(w)

  return w
```

```
#define the model and pass the our defined autoencoder function
TrainingAutoencoder = Model(InImage, autoencoder(InImage))
#compile the defined model by specifying the loss function and the optimizer
TrainingAutoencoder.compile(loss='mean_squared_error', optimizer = Adam())
```

```
#summary of autoencoder model
TrainingAutoencoder.summary()
```

```
#Train the model

Performance = TrainingAutoencoder.fit(T,V,epochs = 20,validation_data = (T1,V1),callbacks = [ModelWeights])
```

```
#defining the encoder model
Autoencoder_encoder = Model(InImage,TrainingAutoencoder.layers[10].output)
#see the encoder summary
Autoencoder_encoder.summary()
```

```
#the layer which has to be find
LayerNo = 11
#the shape of inserted image
DimesionInImage = TrainingAutoencoder.layers[LayerNo].get_input_shape_at(0)[1:]
#input for the 11 layer
Inputs = Input(shape=DimesionInImage)

# introducing the new neuron in each layer
w = Inputs
for L in TrainingAutoencoder.layers[LayerNo:]:
    w = L(w)

# define decoder
Autoencoder_decoder = Model(Inputs, w)
Autoencoder_decoder.summary()
```

4.3 Experiment 3

A new instance has been introduced for denoising the minimum loss that is being encountered and the denoising factor of the model is evaluated graphically and visually. The snapshot shows the execution of the code and the evaluation of the denoising factor that was introduced. TensorBoard has been used to graphically visualize the results, which has been also shown in the trailing snapshots.

```
PredictionOfEncoderTraining = Autoencoder_encoder.predict(TrainingSet)
train_noisy = Autoencoder_decoder.predict(PredictionOfEncoderTraining)
```

```
from matplotlib import pyplot as pt
M=10
Img=pt.figure(figsize=(10, 20))
col = 2
NoOfRows = 5
T = 0
M = 0
for j in range(1, col*NoOfRows +1):
    if j%2 == 0:
        I = TestingSet[T,...,0]#prediction of decoder
        T+=1
    else:
        I = test_noisy[M,...,0]#actual data
        M+=1
    Img.add_subplot(NoOfRows, col, j)
    pt.imshow(I,cmap = 'gray')
pt.show()
```

```
#define the model and pass the our defined autoencoder function
autoencoder = Model(InImage, autoencoder(InImage))
#compile the defined model by specifying the loss function and the optimizer
autoencoder.compile(loss='mean_squared_error', optimizer = Adam())
```

```
from keras.callbacks import TensorBoard
```

```
%load_ext tensorboard
```

```
autoencoder.fit(train_noisy, TrainingSet,
                epochs=100,
                batch_size=128,
                shuffle=True,
                validation_data=(test_noisy, TestingSet),
                callbacks=[TensorBoard(log_dir='/tmp/tb', histogram_freq=0, write_graph=False)])
```

```
# Encoder make prediction for the testing dataset
PredictionOfEncoder = Autoencoder_encoder.predict(test_noisy)
# Decoder predictions
PredictionOfDecoder = Autoencoder_decoder.predict(PredictionOfEncoder)
```

```
from matplotlib import pyplot as pt
M=10
Img=pt.figure(figsize=(10, 20))
col = 2
NoOfRows = 5
T = 0
M = 0
for j in range(1, col*NoOfRows +1):
    if j%2 == 0:
        I = PredictionOfDecoder[T,...,0]#prediction of decoder
        T+=1
    else:
        I = test_noisy[M,...,0]#actual data
        M+=1
    Img.add_subplot(NoOfRows, col, j)
    pt.imshow(I,cmap = 'gray')
pt.show()
```

```
[ ]  %tensorboard --logdir '/tmp/tb'
```

```
[ ]  # Encoder make prediction for the testing dataset
     PredictionOfEncoder = Autoencoder_encoder.predict(TestingSet)
     # Decoder predictions
     PredictionOfDecoder = Autoencoder_decoder.predict(PredictionOfEncoder)
```

```
[ ]  from matplotlib import pyplot as pt
     M=10
     Img=pt.figure(figsize=(10, 20))
     col = 2
     NoOfRows = 5
     T = 0
     M = 0
     for j in range(1, col*NoOfRows +1):
         if j%2 == 0:
             I = PredictionOfDecoder[T,...,0]#prediction of decoder
             T+=1
         else:
             I = TestingSet[M,...,0]#actual data
             M+=1
         Img.add_subplot(NoOfRows, col, j)
         pt.imshow(I,cmap = 'gray')
     pt.show()
```

References

Colaboratory – Google. (n.d.). Retrieved August 17, 2020,

fromhttps://research.google.com/colaboratory/faq.html

Project Jupyter. (n.d.). Retrieved August 17, 2020, from https://www.jupyter.org