**An unstructured MRI compression technique using a convolutional approach**

Debkumar Chowdhury1\*, Tirtharaj Sinha1, Arnobrata Ghosh1, Anurag Unnikannan1, Susmit De1, Devesh Raj1and Siddhartha Bose1

1University of Engineering and Management, Kolkata, India

1[debkumar.cse@gmail.com](mailto:debkumar.cse@gmail.com),1[sinhatirtharaj@gmail.com](mailto:sinhatirtharaj@gmail.com) ,1[ghosharnobrata@gmail.com](mailto:ghosharnobrata@gmail.com),1[anuragunnikannan7@gmail.com](mailto:anuragunnikannan7@gmail.com),1[desusmit101@gmail.com](mailto:desusmit101@gmail.com), 1[rajdevesh437@gmail.com](mailto:rajdevesh437@gmail.com) ,1[bosesiddhart1234@gmail.com](mailto:bosesiddhart1234@gmail.com)

**Abstract.** Currently, different methods are available for the purpose of MRI compression technique. Most of these techniques are well appreciated by society and in response to the demand of society, almost every year different techniques are introduced by different researchers. But it does not satisfy the demand of current requirements. Under such a situation, we are going to propose a new MRI compression and storing technique using a convolutional approach. This technique starts with the MRI preprocessing step. In the next step, the segregation of the dataset into the training and testing phase is performed. Then the convolutional model architecture is trained using the preprocessed MRIs from the dataset. After that our proposed algorithm performs encoding and decoding of preprocessed MRI. We have calculated the performance parameters and found that our proposed algorithm can be used for MRI compression technique using a convolutional approach and has a compression ratio of 1.4593%. Novel preprocessing steps and modifications in the convolutional architecture make the proposed methodology unique. Due to high performance, novelty, ease of use, our proposed method is useful to develop any mobile or web application in the future.

**Keywords:** Preprocessed image, Convolutional, Compression Ratio, Image Compression

# Introduction

In today’s world with the advancement of medical procedures, and the new researches in this field has helped save hundreds of thousands of more patients with every passing year. Medical equipment such as a Computed Tomography (CT scanner), Magnetic Resonance Imaging (MRI scanner), Positron Emission Tomography (PET scanner), Ultrasound scanner, etc., are used to generate images of the organs in the body. These images in turn are analyzed by specialists and experts in the medical field to provide a proper diagnosis of the patient. These images are very crucial as they can show if the patient is suffering from any ailment, or a disease, or has a tumor, etc., and can thus help provide proper treatment in time.

Therefore, these images need to be in high resolution, as every little detail has to be visible for it to be analyzed. This means that every image has a large size, and there are hundreds of images alone for every patient from each test alone. But space is limited, especially in medical facilities that don’t have enough resources or funds to sustain the storage of so many large sized images.

Our project aims to provide a remedy to this problem. We originally take all the original images and store it in a dataset. Then we use an Autoencoder to compress the images, hence producing an encoded image, and then we decode it and then check how close it is to the original image.

Our proposed methodology aims to improve on existing technologies and make healthcare better and more accessible to more individuals across the globe. It displays the classified output using convolutional model architecture along with the satisfactory performance parameters (SSIM [8], PRD, PSNR, Compression Ratio [1][2], BPP, etc.).

Our paper is divided into various sections. In section 2, we explain the advantages and disadvantages of some pre-existing techniques, in section 3 we explain our proposed methodology which consists of the main architecture of the method and the algorithm, whereas the experiment result and analysis are explained in section 4.

1. **Literature Survey**

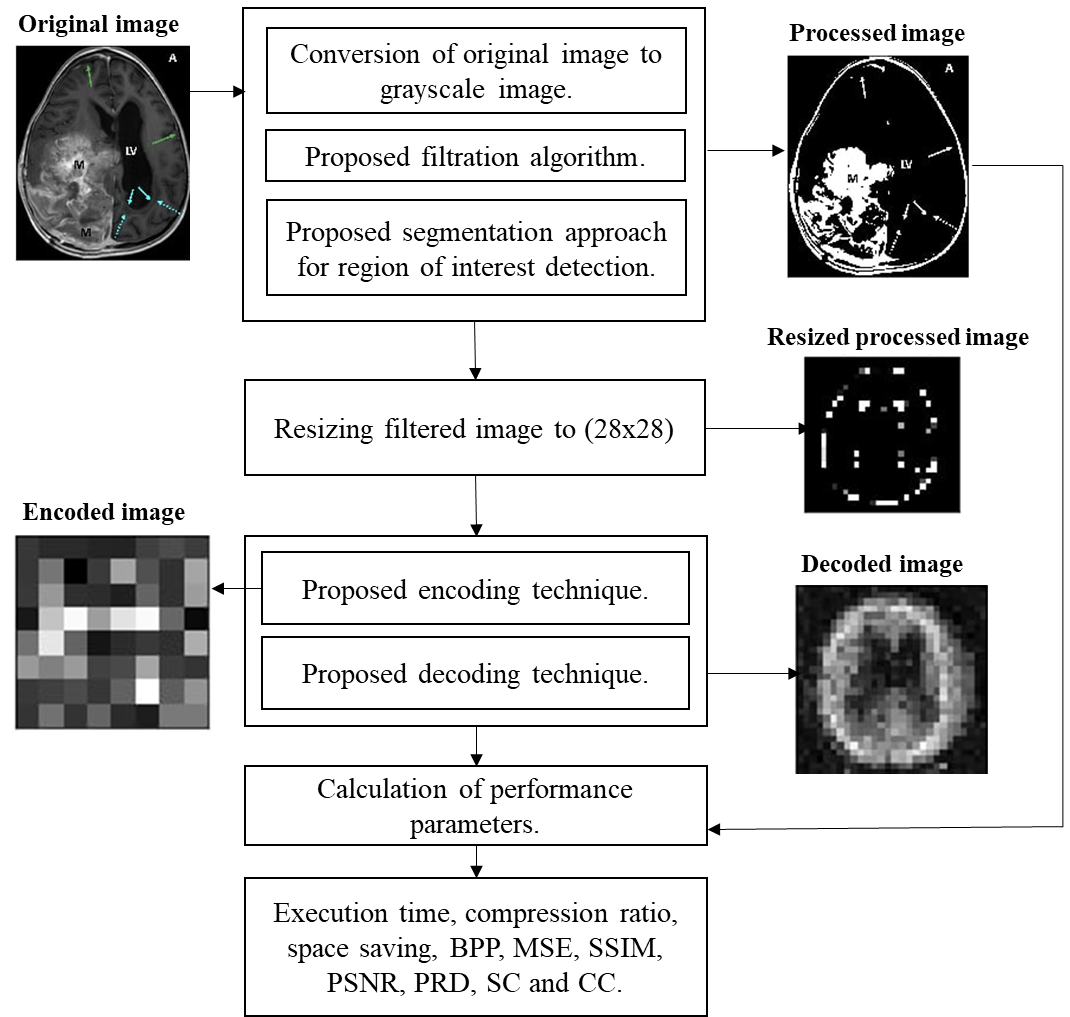
This section represents various pre-existing methods [1] [2] [3][5], along with their advantages and disadvantages as shown in the analysis Table (**Table. 1**). All these methods are well appreciated but in context with our problem the results can be improved. In-depth analysis of these methodologies has proven to be very competent to identify the downsides. Identification of these drawbacks helps us to update, modify our algorithm and code and to calculate the performance parameters. The Survey Table is given below:

**Table. 1.** Existing Methodology Analysis Table

|  |  |  |
| --- | --- | --- |
| **Methodology** | **Advantages** | **Disadvantages** |
| LZW Compression [1][3] | It provides greater compression ratio and appropriate for larger data. | Requires more compression  and decompression time and inappropriate for smaller data. |
| Huffman Coding [1][3] | Requires less compression  and decompression time and appropriate for smaller data. | It provides less compression ratio and inappropriate for larger data. |
| Embedded Zerotree Wavelet (EZW) [2] | User can control desired bit rate. | Properties of the image affects the performance. |
| Shannon–Fano Encoding [3][4] | For Shannon Fano coding procedure we do not need to build the entire codebook. | Shannon–Fano Encoding sometime fails to produce an optimal tree. |
| Run-length encoding [3] | It works well when an image contains long runs of identical samples that usually do not appear in an authentic image. | It’s inappropriate for larger data.[9] |
| Arithmetic coding [3][5] | provides a better compression ratio. | It can corrupt the whole image for a minute error because it has very poor error resistance. |

1. **Proposed Methodology**

Our proposed methodology focused on medical image compression from a custom hydrocephalus dataset using image compression algorithm using a convolutional approach. We have proposed a block diagram to show the main concept of the methodology at a glance as shown in **Fig. 1.**



**Fig. 1.** The built-in architecture of MRI compression technique using a convolutional approach

* 1. **Algorithm**

Our algorithm is divided into 8 steps. The algorithm takes a custom hydrocephalus dataset, as an input from the dataset [10] and produces an encoded image and a decoded image. The algorithm is as follows:

**Algorithm:** An unstructured MRI compression technique using a convolutional approach

**Input:**

**Output:**  encoded image and decoded image

1. Read an image from the dataset.
2. Convert the image to a grayscale image,.
3. Preprocessed using the following formula and 5x5 kernel:

(1)

Where,

= Processed Image

= Standard Deviation

= th row in the grayscale image

= th column in the grayscale image

1. Perform segmentation on using the following equation:

If pixel then,

,

Otherwise,

, provided maximum pixel value = 255

Where,

= grayscale image

= a pixel of an image in position

= selected threshold value in our algorithm. It is tested after the trial-and-error method.

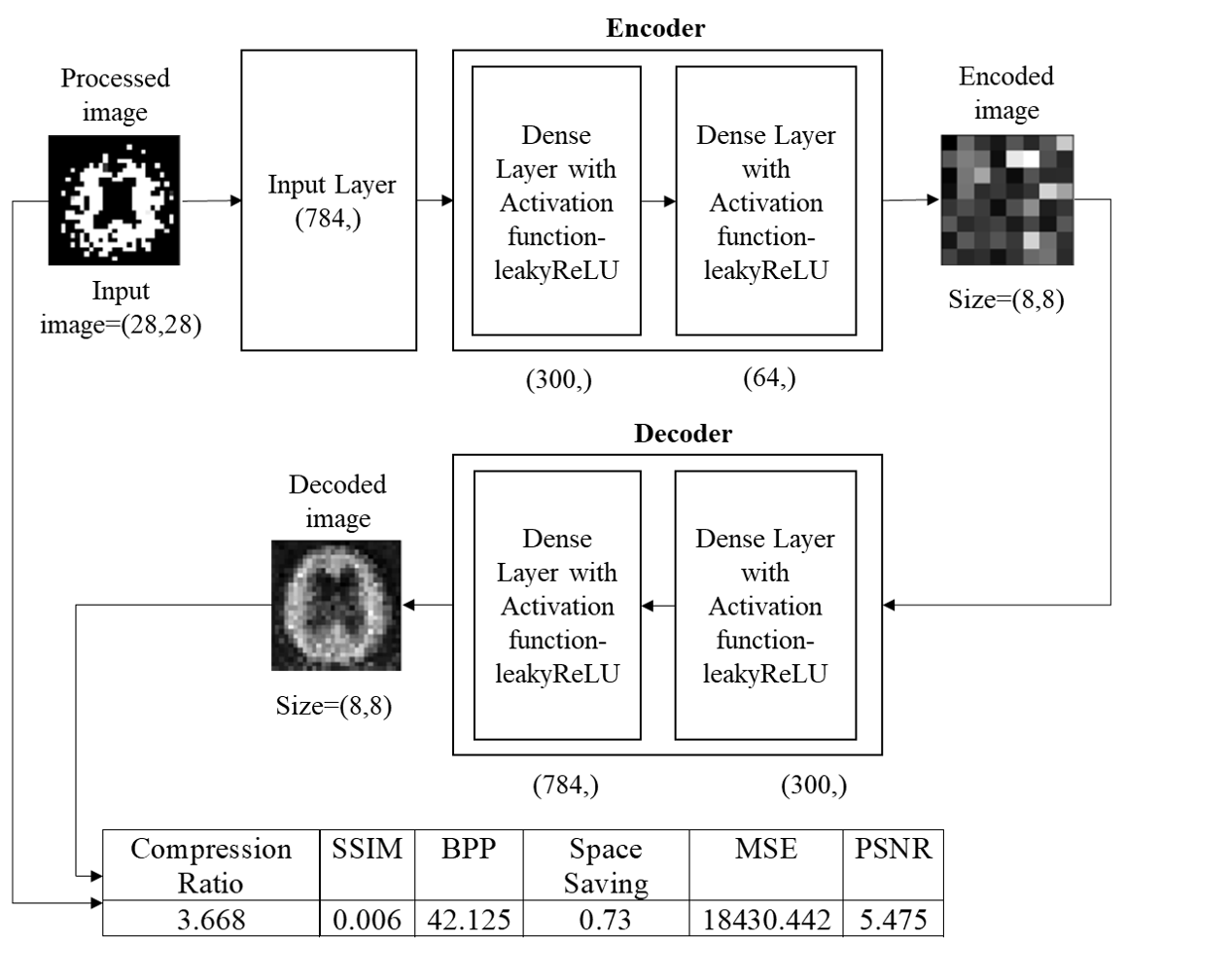
1 = Light

0 = Dark

1. Resize into a 28 X 28 image.
2. The proposed convolutional architecture each has an input x of size 28x28x3 is feed into it. In the convolutional architecture following layers are observed:
   1. Encoder Architecture
   2. One input layer which accepts input of size 784.
   3. One dense layer of output size 300 with an activation leakyReLU function.
   4. One dense layer of output size 64 with an activation leakyReLU function.
   5. Decoder Architecture
      1. One input layer which accepts input of size 64.
      2. One dense layer of output size 300 with an activation leakyReLU function.
      3. One dense layer of output size 784 with an activation leakyReLU function.
3. Segregate the whole dataset [10] into two sections. We select 80% MRI from the dataset [10] for training purposes and 20% MRI from the dataset [10] for testing purposes.
4. We feed the training and testing data into the designed convolutional model architecture to generate result.

**3.2 Model Architecture**

Training and testing phase can be explained with the help of a model architecture as shown in **Fig.2.** the architecture accepts preprocessed MRI image and displays encoded and decoded images along with performance parameters for the same.

~~~~

**Fig.2.** Training and testing phase using proposed model architecture

**3.3 Code**

# # Autoencoder for image compression using Deep learning

# install python libraries requires to run the code in your workspace

# !pip install -r requirements.txt

# to subpress warning

import tensorflow.keras.datasets

import tensorflow.keras.optimizers

import tensorflow.keras.models

import tensorflow.keras.layers

import warnings

warnings.filterwarnings("ignore")

try:

    # importing librarys

    # cv2 is a python extarnal package to do image processing and manipulation related stuff.

    import cv2

    # imutils is a series of convenience functions to make basic image processing easy.

    import imutils

    # tensorflow is a python library for machine learning and artificial intelligence related work.

    import tensorflow as tf

    # NumPy is a Python library used for working with arrays

    import numpy as np

    # pandas offers data structures and operations for manipulating numerical tables and time series.

    import pandas as pd

    # Keras is a library that provides a Python interface for artificial neural networks.

    # Keras acts as an interface for the TensorFlow library.

    import keras

    # 1. Keras layers are the building blocks of the Keras library that can be stacked together for creating neural network models.

    # 2. Keras Conv2D creates a 2D convolution kernel that is wind with layers input which helps produce a tensor of outputs.

    # 3. maxpooling2D Downsamples the input along its spatial dimensions by taking the maximum value over an input window for each channel of the input.

    # 4 .Flattening is converting the data into a 1-dimensional array for inputting it to the next layer.

    # 5 .Dropout regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data.

    # 6. Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch.

    from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

    #  Model groups layers into an object with training and inference features.

    from tensorflow.keras.models import Model

    # tensorflow.keras.callbacks is used to visualize training of a model.

    from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint

    # used to split dataset(features and target) into test and test

    from sklearn.model\_selection import train\_test\_split

    # F1/F Score is a measure of how accurate a model is by using Precision and Recall following the

    # formula of: F1\_Score = 2 \* ((Precision \* Recall) / (Precision + Recall))

    # Precision is commonly called positive predictive value.

    from sklearn.metrics import f1\_score

    # shuffle the dataset for a even mixture of each type of feature and target.it gives better result.

    from sklearn.utils import shuffle

    # A one hot encoding allows the representation of categorical data to be more expressive.

    from sklearn.preprocessing import OneHotEncoder

    # confusion matrix is used to evaluate the accuracy of a classification.[[TP,FP],[FN,TN]]

    # A classification report is a performance evaluation metric in machine learning.

    # It is used to show the precision, recall, F1 Score, and support of your trained classification model.

    from sklearn.metrics import confusion\_matrix, classification\_report

    # importing prebuild structure similarity index

    from skimage.metrics import structural\_similarity as SSIM

    # The OS module in Python provides functions for interacting with the operating system.

    import os

    # Matplotlib is a data visualization and graphical plotting library for Python.

    import matplotlib.pyplot as plt

    # seaborn is alse a data visualization and graphical plotting library for Python.

    import seaborn as sn

    # used to display markdown,image,control (frontend utilities)

    from IPython import display

    import time

    # time package

    from math import log10, sqrt

    # for mathematical operations

    import requests

except:

    get\_ipython().system('pip install -r requirements.txt')

# dataset path

path = r"DATASET"

# make sure in this dir two folder named "yes" and "no" is present.

# In[3]:

# GPU Info

try:

    import GPUtil

except:

    get\_ipython().system('pip install GPUtil')

    import GPUtil

# check physical computing devices

device = tf.config.experimental.list\_physical\_devices()

for i in device:

    print(i)

if len(device) > 1:

    # find GPU details

    print("="\*20, "GPU Details", "="\*20)

    gpus = GPUtil.getGPUs()

    for gpu in gpus:

        print(gpu\_id, gpu.name, gpu.driver, gpu.memoryTotal, gpu.temperature)

# In[4]:

# this function crop out the unnecessary part of a image.

# it takes two parameter -> image : numpy/cv2 image array , plot : binary (You want to plot after before effect of the image)

def image\_threshholder(image, plot=False):

    # grayscalling the image

    # applying gausionBlur operation on grayscaled image.

    # binary threshholding the image to clean the gray scale range and makes the image binary colored based on range.

    # to remove thresh holding comment the next line

    grayscale = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    blurred = cv2.GaussianBlur(grayscale, (5, 5), 0)

    new\_image = cv2.threshold(

        blurred, 100, 255, cv2.THRESH\_BINARY, cv2.CHAIN\_APPROX\_SIMPLE)[1]

    if plot:

        # plots the after before effect on the image based on parameter plot

        # plot if plot=true, ignore the if block if plot=False

        plt.figure()

        plt.figure(figsize=(12, 7))

        plt.subplot(1, 4, 1)

        plt.imshow(image)

        plt.tick\_params(axis="both", which="both",

                        top=False, bottom=False, left=False, right=False,

                        labelbottom=False, labeltop=False, labelleft=False,

                        labelright=False)

        plt.title("(a)")

        plt.subplot(1, 4, 2)

        plt.imshow(grayscale, cmap='gray')

        plt.tick\_params(axis="both", which="both",

                        top=False, bottom=False, left=False, right=False,

                        labelbottom=False, labeltop=False, labelleft=False,

                        labelright=False)

        plt.title("(b)")

        plt.subplot(1, 4, 3)

        plt.imshow(blurred, cmap='gray')

        plt.tick\_params(axis="both", which="both",

                        top=False, bottom=False, left=False, right=False,

                        labelbottom=False, labeltop=False, labelleft=False,

                        labelright=False)

        plt.title("(c)")

        plt.subplot(1, 4, 4)

        plt.imshow(new\_image, cmap="gray")

        plt.tick\_params(axis="both", which="both",

                        top=False, bottom=False, left=False, right=False,

                        labelbottom=False, labeltop=False, labelleft=False,

                        labelright=False)

        plt.title("(d)")

        plt.show()

    return new\_image

# In[5]:

# testing the crop\_contour\_brain\_img() function

example\_image = cv2.imread(path+"/4.jpg")

img = image\_threshholder(example\_image, True)

# plt.imshow(img)

print()

# In[6]:

# view directory tree

def tree\_printer(root):

    try:

        if ":" not in root:

            root = os.getcwd().replace("\\", "/")+"/"+root

            print(root)

        if not os.path.isdir(root) and root != "":

            print(root, ": path not exists....")

            return

    except:

        print("🛠️ set path of the data set from your local mechine")

        return

    print("🗁", root, "-->", len(os.listdir(os.path.join(root))), "Items present.")

    for name in os.listdir(root):

        try:

            print(" |- 🗁", name, " "\*2+"🏴",

                  len(os.listdir(os.path.join(root, name))), "items")

        except:

            continue

tree\_printer(path)

# In[7]:

image\_size = (28, 28)  # defining the image size

hotencoder = OneHotEncoder()  # calling the OneHotEncoder

# using binary crossentropy over catagorical crossentropy

hotencoder.fit([[0], [1]])

# load the images from the local mechine and process before putting the resultent list.

def load\_images(path, type=["jpg", "jpeg"], target=0, images=[], orgimages=[], result=[]):

    for filepath in os.listdir(path):

        if filepath.split(".")[-1].lower() in type:

            img = cv2.imread(path+"/"+filepath)

            try:

                # filtering the image

                img = image\_threshholder(img)

                # resizing the image in defined size

                img = cv2.resize(img, image\_size)

            except:

                print("Excluded image :", path+"/"+filepath)

                continue

            # 3d image to 2D

            img = np.expand\_dims(img, 2)

            # 2D to 1D convertion

            img = img.reshape(-1)

            # returning to 2d image

            orgimg = np.reshape(img, image\_size)

            # putting in image array

            img = np.array(img)

            # inserting flatten image to list

            images.append(img)

            # inserting 2D image to list

            orgimages.append(orgimg)

            # appening the terget 0 for tumor,1 for no tumor

            result.append(target)

    return [images, orgimages, result]

# plot images

def show\_image(datasets, num=4):

    for i in range(num\*\*2):

        plt.subplot(num, num, i+1)

        plt.imshow(datasets[i], cmap="gray")

        plt.axis('off')

# In[8]:

# load the image having brain cancer

images, orgimages, result = load\_images(

    path, target=0, images=[], orgimages=[], result=[])

# print(images.shape)

show\_image(orgimages, num=5)

# print(result)

# In[9]:

# # loading the images not having brain cancer

# images,orgimages,result=load\_images(path+"/no",target=1,images=images,orgimages=orgimages,result=result)

# # print(no\_images.shape)

# # show\_image(images)

# print("total images :",len(result),"| total targets :",len(images))

# # validation of data,original data and target

# In[10]:

# image dataset is stored in variable data

data = np.array(images)

print(data.shape)

data = data/255

# In[11]:

# unnecessary part for this project

orgdata = np.array(orgimages)

print(orgdata.shape)

# In[12]:

target = np.array(result)

print(target.shape)

# ## spliting of data,orgdata,target

# In[13]:

# customized test train splitter

def test\_train\_splitter(data, ratio=0.2):

    range = int(data.shape[0]\*(1-0.2))

    x\_train = data[:range]

    x\_test = data[range:]

    return x\_train, x\_test

x\_train, x\_test = test\_train\_splitter(data)

print(x\_train.shape, x\_test.shape)

# # model starts here -----------:)

# In[14]:

# importing necessary packages

# view explanation in 2nd cell

# ## encoder model

# In[15]:

# encoder model func

# encode the image in 64bit image(8X8)

def get\_encoder():

    # Encoder input shape (784,)

    x = tensorflow.keras.layers.Input(shape=(784), name="encoder\_input")

    # neurons in a Dense layer is now 300

    encoder\_dense\_layer1 = tensorflow.keras.layers.Dense(

        units=300, name="encoder\_dense\_1")(x)

    encoder\_activ\_layer1 = tensorflow.keras.layers.LeakyReLU(

        name="encoder\_leakyrelu\_1")(encoder\_dense\_layer1)

    # neurons in a Dense layer is decreased to 64

    encoder\_dense\_layer2 = tensorflow.keras.layers.Dense(

        units=64, name="encoder\_dense\_2")(encoder\_activ\_layer1)

    encoder\_output = tensorflow.keras.layers.LeakyReLU(

        name="encoder\_output")(encoder\_dense\_layer2)

    # containerized the model

    encoder = tensorflow.keras.models.Model(

        x, encoder\_output, name="encoder\_model")

    return encoder

encoder = get\_encoder()

encoder.summary()

# ## decoder model

# In[16]:

# decoder model func

# encode the encoded image(64bit) back to original size(784 bit).

def get\_decoder():

    decoder\_input = tensorflow.keras.layers.Input(

        shape=(64), name="decoder\_input")

    # neurons in a Dense layer is now 300

    decoder\_dense\_layer1 = tensorflow.keras.layers.Dense(

        units=300, name="decoder\_dense\_1")(decoder\_input)

    decoder\_activ\_layer1 = tensorflow.keras.layers.LeakyReLU(

        name="decoder\_leakyrelu\_1")(decoder\_dense\_layer1)

    # neurons in a Dense layer is increased to 300.

    decoder\_dense\_layer2 = tensorflow.keras.layers.Dense(

        units=784, name="decoder\_dense\_2")(decoder\_activ\_layer1)

    decoder\_output = tensorflow.keras.layers.LeakyReLU(

        name="decoder\_output")(decoder\_dense\_layer2)

    # containerized the model

    decoder = tensorflow.keras.models.Model(

        decoder\_input, decoder\_output, name="decoder\_model")

    return decoder

decoder = get\_decoder()

decoder.summary()

# ## autoencoder model

# In[17]:

# Autoencoder model

# combine and sync the encoder and decoder model.

# parameter encoder model and decoder model.

def autoencoder(encoder, decoder):

    ae\_input = tensorflow.keras.layers.Input(shape=(784), name="AE\_input")

    # encoding the input images

    ae\_encoder\_output = encoder(ae\_input)

    # decoding the encoded image back to original shape

    ae\_decoder\_output = decoder(ae\_encoder\_output)

    ae = tensorflow.keras.models.Model(ae\_input, ae\_decoder\_output, name="AE")

    return ae

ae = autoencoder(encoder, decoder)

ae.summary()

# In[18]:

# calculating RMSE of autoencoder

# RMSE is a measure of how spread out these residuals are.

# It tells you how concentrated the data is around the line of best fit.

def rmse(y\_true, y\_predict):

    return tensorflow.keras.backend.mean(tensorflow.keras.backend.square(y\_true-y\_predict))

# AE Compilation

ae.compile(loss="mse", optimizer=tensorflow.keras.optimizers.Adam(

    learning\_rate=0.0005))

# In[19]:

# preparing brain tumor dataset spliting into test train default ratio is 0.2

x\_train, x\_test = test\_train\_splitter(data)

print(x\_train.shape, x\_test.shape)

x\_train\_org, x\_test\_org = test\_train\_splitter(orgdata)

print(x\_train\_org.shape, x\_test\_org.shape)

y\_train, y\_test = test\_train\_splitter(target)

print(y\_train.shape, y\_test.shape)

# In[21]:

# Training Autoencoder

ae.fit(x\_train, x\_train, epochs=20, batch\_size=10,

       shuffle=True, validation\_data=(x\_test, x\_test))

# In[22]:

get\_ipython().run\_cell\_magic('time', '',

                             'start\_time = time.time()\n\nencoded\_images = encoder.predict(x\_test)\nencode\_time=time.time() - start\_time\ndecoded\_images = decoder.predict(encoded\_images)\n# calculating execution time\nexecution\_time=time.time() - start\_time\ndecode\_time=execution\_time-encode\_time\n# reshaping the image of (784,) to plotable image(28,28)\ndecoded\_images\_orig = np.reshape(decoded\_images, newshape=(decoded\_images.shape[0], 28, 28))\ndecoded\_images\_orig.shape')

# # Analyzing compression ratio

# In[23]:

print("encoder image shape", encoded\_images[0].shape)

print("decoder image shape", decoded\_images[0].shape)

# ploting the comparition between original,encoded and decoded image.

plt.figure()

plt.subplot(1, 3, 1)

original\_sample = x\_test\_org[2]

plt.imshow(original\_sample, cmap="gray")

plt.tick\_params(axis="both", which="both",

                top=False, bottom=False, left=False, right=False,

                labelbottom=False, labeltop=False, labelleft=False,

                labelright=False)

plt.title("original image")

plt.subplot(1, 3, 2)

encoded\_samples = np.reshape(

    encoded\_images, newshape=(decoded\_images.shape[0], 8, 8))

encoded\_sample = np.reshape(

    encoded\_images, newshape=(decoded\_images.shape[0], 8, 8))[0]

plt.imshow(encoded\_sample, cmap="gray")

plt.tick\_params(axis="both", which="both",

                top=False, bottom=False, left=False, right=False,

                labelbottom=False, labeltop=False, labelleft=False,

                labelright=False)

plt.title("encoded image")

plt.subplot(1, 3, 3)

decoded\_sample = decoded\_images\_orig[0]

plt.imshow(decoded\_sample, cmap="gray")

plt.tick\_params(axis="both", which="both",

                top=False, bottom=False, left=False, right=False,

                labelbottom=False, labeltop=False, labelleft=False,

                labelright=False)

plt.title("decoded image")

plt.show()

# ### performance parameter

# 1. execution time

# 2. compression ratio

# 3. space saving

# 4. Bits per pixel (BPP)

# 5. Mean squared error (MSE)

# 6. Structure Similarity Index (SSIM)

# 7. Peak Signal to Noise Ratio (PSNR)

# 8. Percent rate of distortion (PRD)

# 9. Structural Content (SC)

# 10. Correlation Coefficient (CC)

# In[24]:

# avg execution time=total execution time/no of testing sample

print("Total execution time = {} seconds".format(execution\_time))

print("Total encoding time = {} seconds".format(encode\_time))

print("Total decode Time = {} seconds".format(decode\_time))

print("Avarage Execution time = {} MilliSeconds".format(

    execution\_time/encoded\_images.shape[0]\*1000))

print("Avarage Encoding time = {} MilliSeconds".format(

    encode\_time/encoded\_images.shape[0]\*1000))

print("Avarage Decoding time = {} MilliSeconds".format(

    decode\_time/encoded\_images.shape[0]\*1000))

print()

def compression\_ratio(encoded\_sample, original\_sample, imgnum="test"):

    if not os.path.isdir(r"static"):

        os.system("mkdir static")

    # saving the image .jpg format.

    cv2.imwrite('static/comp\_sample\_'+str(imgnum)+'.jpg', encoded\_sample)

    cv2.imwrite('static/org\_sample\_'+str(imgnum)+'.jpg', original\_sample)

    # getting the size of the image

    size\_bytes\_comp = os.path.getsize('static/comp\_sample\_'+str(imgnum)+'.jpg')

    size\_bytes\_org = os.path.getsize('static/org\_sample\_'+str(imgnum)+'.jpg')

    # number of pixel in compressed image for image sample 0

    comp\_pixel = encoded\_sample.shape[0]

    # converting the bytes to bit by multiplying by 8

    size\_bits\_comp = size\_bytes\_comp\*8

    size\_bits\_org = size\_bytes\_org\*8

    # Bits Per Pixel (BPP): BPP is defined as the ratio of the total size of the compressed image to the total number

    # of the pixel in the image.

    BPP = size\_bits\_comp/comp\_pixel

    # compression\_ratio = orizinal image size / compressed image size

    compression\_ratio = round(size\_bits\_org / size\_bits\_comp, 3)

    # defined as the reduction in size relative to the uncompressed size:

    space\_saving = round(1-(1/compression\_ratio), 2)

    return compression\_ratio, BPP, space\_saving, size\_bits\_comp, size\_bits\_org

# In[25]:

# Mean Square Error (MSE): MSE is the description of the cumulative squared error between the

# compressed image and the original image

def MSE(imageA, imageB):

    # the 'Mean Squared Error' between the two images is the

    # sum of the squared difference between the two images;

    err = np.sum((imageA.astype("float") - imageB.astype("float")) \*\* 2)

    err /= float(imageA.shape[0] \* imageA.shape[1])

    return round(err, 3)

def PSNR(original, compressed):

    mse = np.mean((original - compressed) \*\* 2)

    if(mse == 0):  # MSE is zero means no noise is present in the signal .

        # Therefore PSNR have no importance.

        return 100

    max\_pixel = 255.0

    psnr = 20 \* log10(max\_pixel / sqrt(mse))

    return round(psnr, 3)

def PRD(original, decoded):

    sum\_diff = np.sum((original - decoded))\*\*2

    sum\_org = np.sum(original)\*\*2

    prd = sqrt(sum\_diff/sum\_org)\*100

    return round(prd, 3)

def SC(original, decoded):

    sum\_org = np.sum(original\*\*2)

    sum\_dec = np.sum(decoded\*\*2)

    sc = sum\_org/sum\_dec

    return round(sc, 3)

def correlation\_coeff(orgimage, decoded):

    cc = (np.sum(orgimage\*decoded)) / \

        ((sqrt((np.sum(orgimage)\*\*2)) \* (sqrt((np.sum(decoded)\*\*2)))))

    return round(cc, 4)

# In[26]:

comp\_ratio, BPP, space\_saving, size\_bits\_comp, size\_bits\_org = compression\_ratio(

    encoded\_sample, original\_sample)

print("File size of Compressed image :", size\_bits\_comp, "bits")

print("File size of Original image :", size\_bits\_org, "bits")

print("BPP for sample image :", BPP, "bits/pixel")

print("Compression ratio :", comp\_ratio)

print("Space saving :", space\_saving)

# calculating MSE of original and decompressed image

print("Mean Squared Error of original and decompressed image :",

      MSE(original\_sample, decoded\_sample))

# calculating SSIM of original and decompressed image

# Structure Similarity Index (SSIM): SSIM is used to measure the tendency of similarity between the

# original image and the compressed image

print("Structure Similarity Index(SSIM) of original and decompressed image :",

      round(SSIM(original\_sample, decoded\_sample), 3))

# Peak Signal to Noise Ratio (PSNR): PSNR is defined as the ratio of the maximum pixel intensity to the

# mean square error

print("Peak Signal to Noise Ratio (PSNR) of original and decompressed image :",

      PSNR(original\_sample, decoded\_sample))

print("PRD of original and decompressed image :",

      PRD(original\_sample, decoded\_sample), "%")

print("Structural content of original and decompressed image :",

      SC(original\_sample, decoded\_sample))

print("Correlation Coefficient (CC) of original and decompressed image :",

      correlation\_coeff(original\_sample, decoded\_sample))

# In[27]:

warnings.filterwarnings("ignore")

def performance\_monitor(num\_images\_to\_show=5):

    # comparing some more images between original and decode image.

    performance = []

    columns = ['Id', 'encoded image size (bits)', 'original image size (bits)',

               "compression ratio", "BPP", "space saving", "MSE", "SSIM", "PSNR", "PRD", "SC", "CC"]

    for im\_ind in range(num\_images\_to\_show):

        plot\_ind = im\_ind\*2 + 1

        rand\_ind = np.random.randint(low=0, high=x\_test.shape[0])

        original\_sample = x\_test\_org[rand\_ind, :, :]

        plt.figure(figsize=(7, round(num\_images\_to\_show\*1.5)))

        plt.subplot(num\_images\_to\_show, 2, plot\_ind)

        plt.imshow(original\_sample, cmap="gray")

        plt.subplot(num\_images\_to\_show, 2, plot\_ind+1)

        encoded\_sample = encoded\_samples[rand\_ind]

        # convert 2D image to 3D

        decodeds = cv2.cvtColor(

            decoded\_images\_orig[rand\_ind], cv2.COLOR\_GRAY2BGR)

        decoded\_sample = decoded\_images\_orig[rand\_ind]

        # Clipping data to the valid range for imshow with RGB data ([0..1]

        plt.imshow(np.clip(decodeds, 0, 1))

        comp\_ratio, BPP, space\_saving, size\_bits\_comp, size\_bits\_org = compression\_ratio(

            encoded\_sample, original\_sample, im\_ind)

        mse = MSE(original\_sample, decoded\_sample)

        ssim = round(SSIM(original\_sample, decoded\_sample), 3)

        psnr = PSNR(original\_sample, decoded\_sample)

        prd = PRD(original\_sample, decoded\_sample)

        sc = SC(original\_sample, decoded\_sample)

        cc = correlation\_coeff(original\_sample, decoded\_sample)

        performance.append([im\_ind+1, size\_bits\_comp, size\_bits\_org,

                           comp\_ratio, BPP, space\_saving, mse, ssim, psnr, prd, sc, cc])

    plt.figure()

    plt.scatter(encoded\_images[:, 0], encoded\_images[:, 1], c=y\_test)

    plt.colorbar()

    table = pd.DataFrame(performance, columns=columns)

    table.set\_index('Id', inplace=True)

    return table

# In[33]:

num\_images\_to\_show = 4

table = performance\_monitor(num\_images\_to\_show)

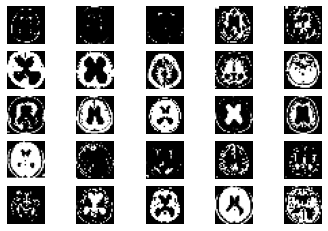
table.head(num\_images\_to\_show)

1. **Experimental results**

We consider a custom hydrocephalus dataset, which contains 132 images. The size, colour, and format of images in the dataset are similar in nature, whereas the resolutions of the images are different. The format of the images is ‘.jpg’ by nature.

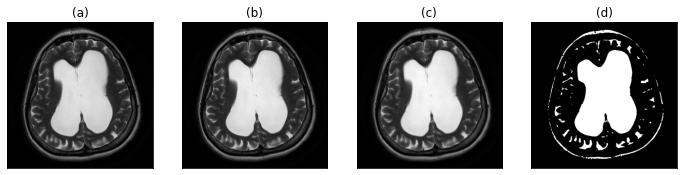
We have applied our algorithm in the python environment, version 3.8, with the hardware configuration of the Intel Core i3 5th Generation processor,4GB DDR3 primary memory (RAM), and an integrated graphics card. Anaconda as a distributor of Python version 3.8 is used. jupyter notebook version 6.3.0 as an open web interface is used as a programming platform for the implementation of our algorithm.

After reading the images from the local machine we have achieved the following results as shown in **Fig. 3.**



**Fig. 3.** Samplesof hydrocephalus dataset

After reading all the images from the dataset each image will pass through the various steps of the algorithm as shown in **Fig. 5.**

****

**Fig. 5.** (a) original image (b) gray-scale image (c) image after applying proposed filtration method (d) proposed segmentation approach for the region of interest detection

We are splitting our dataset [10] into training and testing sets. 80% of the MRIs are used for training and the rest is used for testing. After the execution of our proposed algorithm, we observed that the number of training samples are 104, number of testing samples are 26 whereas training shape values are (104, 28, 28). Our proposed convolutional layered architecture is trained. It is used to train on n number of samples (104 in our case). With the changes of several iterations or epochs, the total time consumed by the algorithm is calculated.

**Table 2:** An example of the training phase.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SNo.** | **Epoch No.** | **Time Taken (s)** | **Loss** | **Value Loss** |
| 1. | 1 | 1 | 0.2193 | 0.1604 |
| 2. | 2 | 0 | 0.1407 | 0.1219 |
| 3. | 3 | 0 | 0.1159 | 0.1086 |
| . | . | . | . | . |
| . | . | . | . | . |
| . | . | . | . | . |
|  |  |  |  |  |
| 18. | 18 | 0 | 0.0517 | 0.0842 |
| 19. | 19 | 0 | 0.0494 | 0.0835 |
| 20. | 20 | 0 | 0.0473 | 0.0831 |

* 1. **Performance parameters**

In this work, the compression techniques use a wide number of performance measures to compute their efficiency and performance.

**Table 3:** Various performance parameters

|  |  |  |
| --- | --- | --- |
| **Serial No**. | **Name of the performance parameter** | **Equation** |
| 1. | Compression Ratio [10][1] | CR = |
| 2. | Mean Square Error [10] | MSE= |
| 3. | Bits per Pixel [10] | CR = |
| 4. | Structure Similarity Index [10] | SSIM = |
| 5. | Correlation coefficient [10] | CC = |
|  | Peak Signal to Noise Ratio [10] | PSNR=20 |
| 7. | Percent rate of distortion [10] | PRD = |
| 8. | Structural Content [10] | SC = |

**Table 4:** Calculation ofperformance parameters for each image

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SNo** | **BPP** | **Compression Ratio** | **MSE** | **SSIM** | **PSNR** | **PRD** | **Structural Content** | **CC** |
| 1. | 338 | 1.538 | 17142.986 | 0 | 5.79 | 99.665% | 10.081 | 0.0040 |
| 2. | 333 | 2.261 | 1421.007 | 0 | 16.605 | 99.55% | 475.408 | 0.0065 |
| 3. | 333 | 1.967 | 3585.557 | 0 | 12.585 | 99.541% | 129.134 | 0.0072 |
| . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . |
| 25. | 341 | 1.757 | 37420.317 | 0 | 2.4 | 99.659% | 4.243 | 0.0020 |
| 26. | 339 | 1.664 | 35073.048 | 0 | 2.681 | 99.625% | 3.669 | 0.0023 |

**Table 4** displays performance parameters of 26 randomly selected original samples from the dataset. At the end, we have found that our proposed algorithm can be used for medical image compression from MRIs. As the data is balanced, we are considering the results to be satisfied. Comparing it with other existing methodologies a satisfactory result is observed as shown in **Table 5.** We can conclude that our proposed methodology overpowers the existing compression algorithms.[1][2][3][5]

**Table 5:** Comparison chart

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Serial No**. | **Name of the compression algorithm** | **Compression ratio** |
| 1. | LZW [1] | 5.319% |
| 2. | Huffman coding [1] | 2.203 % |
| 3. | EZW [2] | 1.6647% |
| 4. | RLE [3] | 1.5766% |
| 5. | Shannon–Fano [3] | 1.9825% |
| 6. | Arithmetic coding [3][5] | 2.2059% |
| **7.** | **Proposed methodology** | **1.4593%** |

1. **Conclusion**

In this paper, we have proposed an Autoencoder algorithm. This algorithm is capable of compressing and storing generated images from medical equipment contained in the Custom hydrocephalus dataset, collected from the web resource. It is responsible for encoding the original image, and decoding the image, and finally comparing the decoded and original image. In addition, we have also generated BPP, Compression ratio, MSE, SSIM, PSNR, PRD, Structural content and CC values, through which we can compare our proposed method with existing methods [1] [2] [3] [5]. The experimental result shows that after applying the proposed and existing methods [1] [2] [3] [5] on the Custom hydrocephalus dataset, our technique is producing a Compression ratio of 1.4593%. This result is considered to be satisfactory, and based on this result we can say that the proposed algorithm overpowers the efficiency of the existing method. Due to high performance, novelty, ease of use, our proposed method is useful to develop any mobile or web applications in the future. Our method can be tested on various medical equipment generated images datasets to identify the generic performance of the proposed method in the future. The performance of our method may be increased by making necessary modifications in the algorithm.

1. **Publications**

Title: An unstructured mammogram analysis for feasible classification and detection of breast cancer using a convolutional approach.

Domain: Medical Image Processing

Subdomain: Disease detection

Tools used: Multi layer CNN

Guide: Debkumar Chowdhury

Member: Tirtharaj Sinha1, Arnobrata Ghosh2, Anurag Unnikannan3, Susmit De4

Status: Accepted and presented

Outcome: Published in the book chapter.

Lecture Notes in Network and Systems (Springer)

Print ISSN: 2367-3370

1. **References**
2. Gajendra Sharma, “Analysis of Huffman Coding and Lempel–Ziv–Welch (LZW) Coding as Data Compression Techniques,” International Journal of Scientific Research in Computer Science and Engineering, Vol.8, Issue.1, pp.37-44, 2020.
3. A.M.Raid, W.M.Khedr, M. A. El-dosuky and Wesam Ahmed, “Image compression using embedded zerotree wavelet” Signal & Image Processing : An International Journal (SIPIJ) Vol.5, No.6, December 2014, pp. 33-39, doi : 10.5121/sipij.2014.5603
4. Rahman, Md. A., and Mohamed Hamada. 2019. "Lossless Image Compression Techniques: A State-of-the-Art Survey" Symmetry 11, no. 10: 1274. doi : 10.3390/sym11101274
5. S. S. Yu, M. N. Wernick and N. P. Galatsanos, "Lossless compression of multi-dimensional medical image data using binary-decomposed high-order entropy coding," Proceedings of 1st International Conference on Image Processing, 1994, pp. 351-355 vol.2, doi: 10.1109/ICIP.1994.413590.
6. G. Langdon and J. Rissanen, "Compression of Black-White Images with Arithmetic Coding," in IEEE Transactions on Communications, vol. 29, no. 6, pp. 858-867, June 1981, doi: 10.1109/TCOM.1981.1095052.
7. A. K. Jain, "Image data compression: A review," in Proceedings of the IEEE, vol. 69, no. 3, pp. 349-389, March 1981, doi: 10.1109/PROC.1981.11971.
8. P. Roos, M. A. Viergever, M. C. A. van Dijke and J. H. Peters, "Reversible intraframe compression of medical images," in IEEE Transactions on Medical Imaging, vol. 7, no. 4, pp. 328-336, Dec. 1988, doi: 10.1109/42.14516.
9. Z. Xiao and C. Zheng, "Medical Image Fusion Based on the Structure Similarity Match Measure," 2009 International Conference on Measuring Technology and Mechatronics Automation, 2009, pp. 491-494, doi: 10.1109/ICMTMA.2009.558.
10. Amin Mubarak Alamin Ibrahim\* et al. ,(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH, Volume No.3, Issue No.1, December – January 2015, 1808 – 1812.
11. Pardeep Kumar, Ashish Parmar, Versatile Approaches for Medical Image Compression: A Review, Procedia Computer Science, Volume 167, 2020, Pages 1380-1389, ISSN 1877-0509, doi : 10.1016/j.procs.2020.03.349