Image Compression and Decompression using Convolutional Autoencoders and K-means Clustering

Team - 5

Shriya Baskaran	185001149
Sowmya R	185001159
Swetha P	185001182
Swetha Saseendran	185001183

Abstract

Image compression is a task of increasing importance, due to the pervasive nature of images in the applications we use today along with the need to transmit these images in an inexpensive way while maintaining reasonable quality. We investigate the problem of image compression, experimenting with several different models. Our baselines are a Convolutional autoencoder and an ML model that uses K-means clustering to perform image compression. We aim to compare the performance of the aforementioned models to understand why Deep Learning is the industry standard for image compression.

Introduction

The most basic problem faced by everyone in the 21st century is a time and space(memory) manageable. Digitizing the image consumes more bandwidth which ultimately leads to more cost for the transmission of data. Hence, neural networks are used for efficient use of bandwidth. Image compression is used for faster transmission in-order to provide better services to the user or society in general.

Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data and then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible. Autoencoder is one of the simplest generative models. The output of a generative model is an image of the same size and similar appearance as the input image. . For example, The original image of size (let's say 28 * 28) is reduced to 28 * 1 using the encoder. Hence the image is compressed after the input to the neural network data points. They are usually data specific, lossy and automatically learn from examples.
K-Means algorithm is a centroid based clustering technique. This technique clusters the dataset into k different clusters. Each cluster in the k-means clustering algorithm is represented by its centroid point. The k-Means Clustering algorithm takes advantage of the visual perception of the human eye and uses few colors to represent the image. Colors having different values of intensity that are RGB values seem the same to the human eye. The K-Means algorithm takes this advantage and clubs similar looking colors (which are close together in a

cluster) and reconstruct the image using the centroids of these clusters.

Applications of our Project:

Image compression is minimizing the size in bytes of a graphics file without degrading the quality of the image to an unacceptable level. There are several uses to this. The reduction in file size allows more images to be stored in a given amount of disk or memory space. Also transmission of files to another device is made much easier through this process. Further on, communication cost becomes much lesser when compression is used. Eventually, communication bandwidth is saved without creating much impact on the image quality.

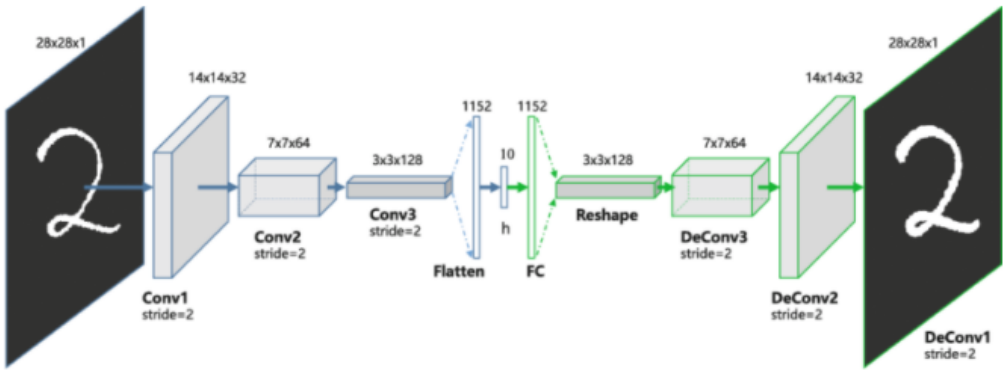
Henceforth, image compression is applied in several fields:

- Broadcast television
- Remote sensing via satellite
- Military communication via radar, sonar etc
- Tele-conferencing
- Computer communications
- Facsimile transmission
- Medical images (computer tomography)
- Magnetic resonance imaging
- Satellite images, geological surveys, weather maps

Models and Metrics Used:

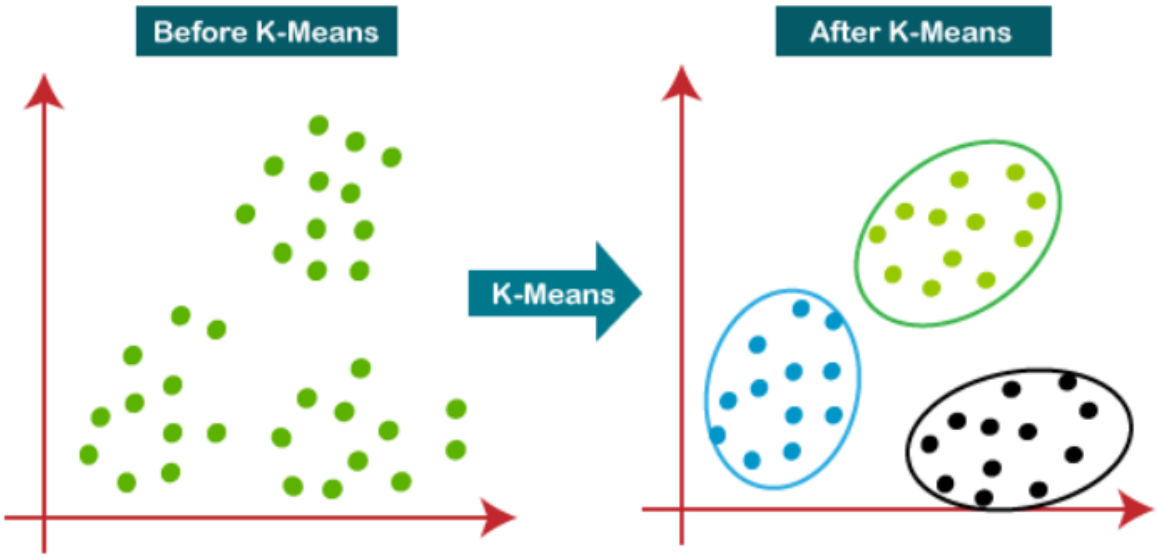
Convolutional Autoencoder:

Autoencoders in their traditional formulation do not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders use the convolution operator to exploit this observation. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them, modify the geometry or the reflectance of the image.



K-Means:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. K-means algorithm identifies **k** number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. K defines the number of predefined clusters that need to be created in the process, if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.



PSNR:

The PSNR block computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is used as a quality measurement between the original and a compressed image.
The higher the PSNR, the better the quality of the compressed, or reconstructed image.The mean-square error (MSE) and the peak signal-to-noise ratio (PSNR) are used to compare image compression quality. The MSE represents the cumulative squared error between the compressed and the original image, whereas PSNR represents a measure of the peak error. The lower the value of MSE, the lower the error.
To compute the PSNR, the block first calculates the mean-squared error using the following equation:

$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M * N}$$

In the previous equation, *M* and *N* are the number of rows and columns in the input images.
Then the block computes the PSNR using the following equation:

$$PSNR = 10 \log_{10}\left(\frac{R^2}{MSE}\right)$$

Here, *R* is the maximum fluctuation in the input image data type.

Main Objective

The main objective of this project is to compare the performances of image compression performed by a convolutional autoencoder (deep learning model) and a k-means clustering algorithm (ML model). In the convolutional encoder, the image is passed through an encoding layer which converts the image into a one dimensional vector. The model is trained in such a way that the decoding layer can reconstruct the image by reducing the size of the image significantly and

retaining the clarity of the content at the same time. Whereas, the K-means model clusters similar looking pixels in the image. The centroid of each cluster is considered and reconstructed into a compressed image. This project essentially compares the outputs of both these models for various conditions and proves that convolutional autoencoders are the better option.

Limitations

- The autoencoder is highly data-specific so it can only compress images of the same type as the training data, i.e, MNIST dataset.
- It results in a lossy compression and the image cannot be reconstructed without some amount of loss.
- The K-means algorithm encountered a memory overload problem for the whole dataset and had to be significantly reduced and the number of clusters had to be reduced as well.

Dataset

This phase involves collection of data. Many procedures are consecutively executed to understand this collected data. Rigorous examination of the various datasets collected from different data sources are done to ensure if the collected data is adequate to solve the business problem. This stage can be further divided into

Data Collection:

Here the data relevant to the problem statement is collected in various forms from different data sources. MNIST digits dataset consisting of digits that are handwritten originally sourced from a much larger dataset named NIST is used. This made a part of the Keras inbuilt dataset, which are used directly using Keras.

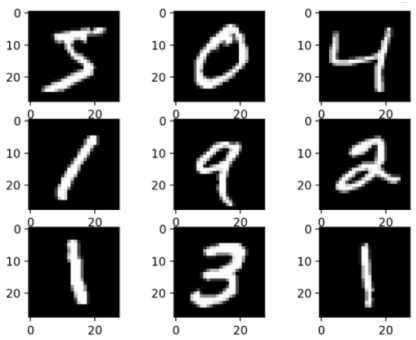
Data Description:

➔ Gray-scale images of hand-drawn digits, from zero through nine.
➔ Each image is 28 pixels in height and 28 pixels in width, for a total of 784

pixels in total.

➔ The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

➔ The size of each image in the MNIST dataset (which we'll use in this project) is 28x28.That is, each image has 784 elements. If each image is compressed so that it is represented using just 28x1 which is 28 elements and thus : **(756/784)*100=96.42% of the data!**



Data Pre-processing

In this stage, data being spread through different files are collated from multiple data sources to form an integrated dataset. In the last stage after the dataset is being studied, the outliers and other anomalies that are observed are handled to make suitable for the model fit. This is done through various data cleaning steps, transformations, and pre-processing after the abstraction of the data, depending on the characteristics of the model. This phase can be divided into certain sub steps:

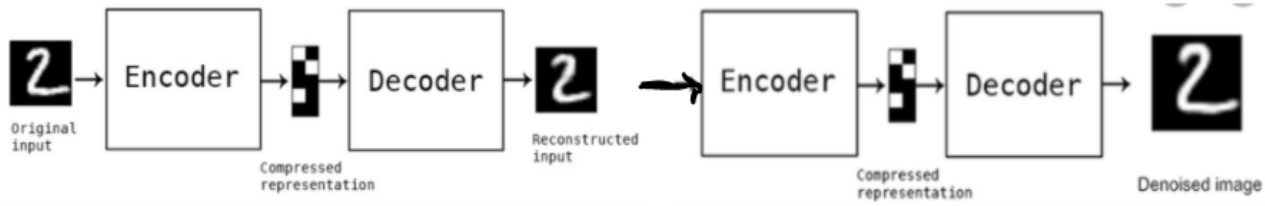
DATA SELECTION: This step involves the selection of the dataset, suitable for the implementation of the project. MNIST digits dataset consisting of 70,000 images handwritten digits in black and white from 10 different classes, both having dimensions suitable for performing the image compression with the architecture has been selected. Thus, it shows that the image datasets are suitable for the model fit.

DATA CLEANING: Data cleaning is performed by removing the abnormalities and doing suitable changes to make the data ready for use. There is no step

involved in the cleaning of the data as these images unlike numeric or string data have no chances of presence of any abnormalities or outliers. So, the step involving the data cleaning is not required in our case.

DATA CONSTRUCTION AND FORMATTING: In this stage sometimes from the existing data new characteristics are derived and introduced. This helps in gaining better insights of the data. Formatting of the data is consecutively done for efficient fitting of the model.

Design Specification for Autoencoder:

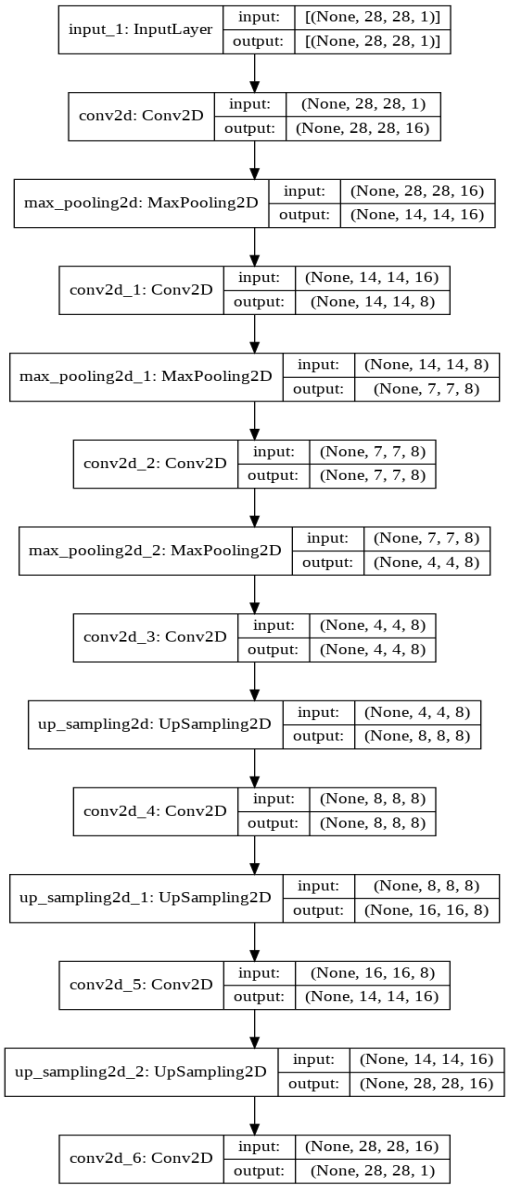


The autoencoder is made up of layers that make up the encoder and the decoder and it has been trained on the MNIST dataset for 50 epochs. The input layer of the encoder takes the preprocessed image as input. Preprocessing is done by first converting all the values in the image array to a floating point value between 0 and 1 and normalizing the values. The numpy array then moves through a set of convolutional and max pooling layers and gets resized according to the dimensions specified during the creation of layers. The decoder is a replica of the encoder in the sense that it tries to perform the exact opposite operation of that of the encoder layers. It has a series of convolutional and upsampling layers to slowly increase the dimensions and reconstruct the original image.

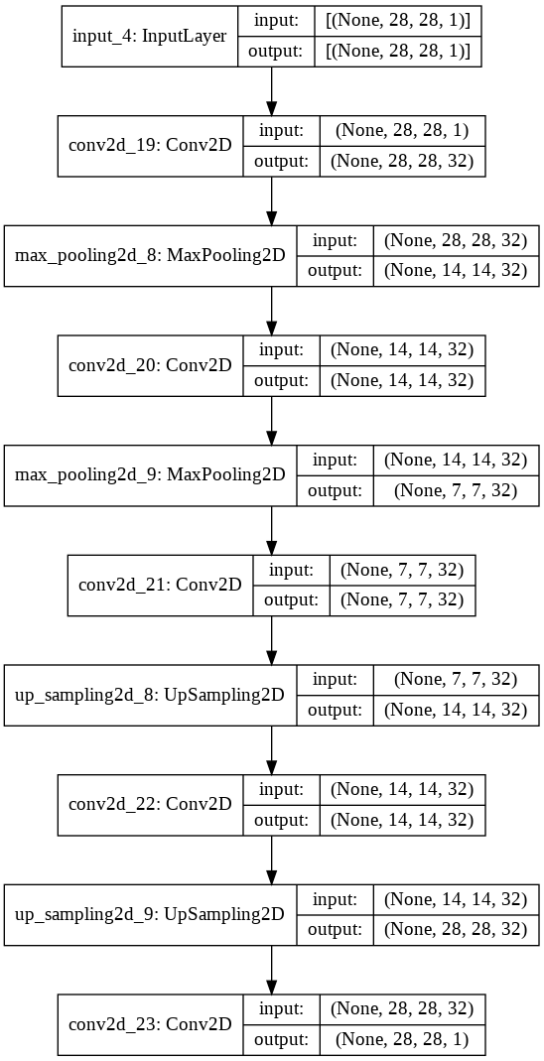
The denoising model has been trained on a slightly modified version of the MNIST dataset where noise has been introduced into the training data intentionally. It has also been trained for 50 epochs and the model is fed the reconstructed image of the autoencoder and it tries to make the image look sharp and much closer to the original image. Adding this additional model to the architecture significantly improves the results of the experiment which can be seen in the significantly high PSNR value that has been calculated.

The overall architecture of the above mentioned models along with the input and output shape required for each layer has been summarized in the section below.

Autoencoder Model Plot for Compression and Decompression of Image:



Autoencoder Model Plot for Denoising the Image:



Working Environment

Processor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
RAM: 8.00 GB
Operating System: Windows 10
System Type: 64-bit operating system, x64-based processor

Tools and libraries required

IDE: Visual Studio Code
Tools: virtualenv, Google Collab
Python Libraries:

- tensorflow
- keras
- numpy
- os
- flask
- matplotlib
- pillow
- scikit-image
- opencv

Implementation Outcomes:

- **Convolutional Autoencoders**
  Working:

  The project consists of two parts. First, using Autoencoders, we're able to decompress this image and represent it as a flattened vector. Using it, we can reconstruct the image. Of course, this is an example of lossy compression, as we've lost quite a bit of info.
  If we take into consideration that the whole image is encoded in the extremely small vector we lose a lot of data. Hence,we use the autoencoders for denoising the decompressed image to restore as much information as possible.
  Steps of Implementation:
  - ➔ Downloading the data and preprocessing it.
  - ➔ Creating the Autoencoder and training it for compression and decompression.
  - ➔ Testing the trained model and comparing it with the original data.
  - ➔ Visualising the states of a image through the autoencoder
  - ➔ Creating the autoencoder model for denoising the decompressed images for restoration.
  - ➔ Visualising the results of denoising the decompressed data.
  - ➔ Analyzing the quality metrics of the images and the convolutional layers.

Original Image

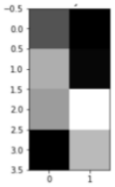
Decompressed Image

Denoised Image

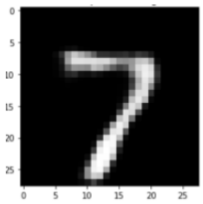
States of the image through the autoencoder:

Original Image

Compressed Image



Decompressed Image



K-means clustering

Working:

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs to only one group that has similar properties.

We exploit this property of the K-Means algorithm to classify similar pixels. In a colored image, each pixel can have intensity values from 0 to 255. Following combinatorics, the total number of colors which can be represented is 256*256*256 which is a huge number! The K-Means algorithm takes this advantage and clubs similar looking colors which are close together in a cluster with closer pixel values.

We will be using k-Means clustering to find k number of colors which will be representative of its similar colors. These k-colors will be centroid points from the algorithm. Then we will replace each pixel value with its centroid points. Thus, the color combination formed using only k values will be very less compared to the total color combination. We will try different values of k and observe the output image.

Steps of Implementation:

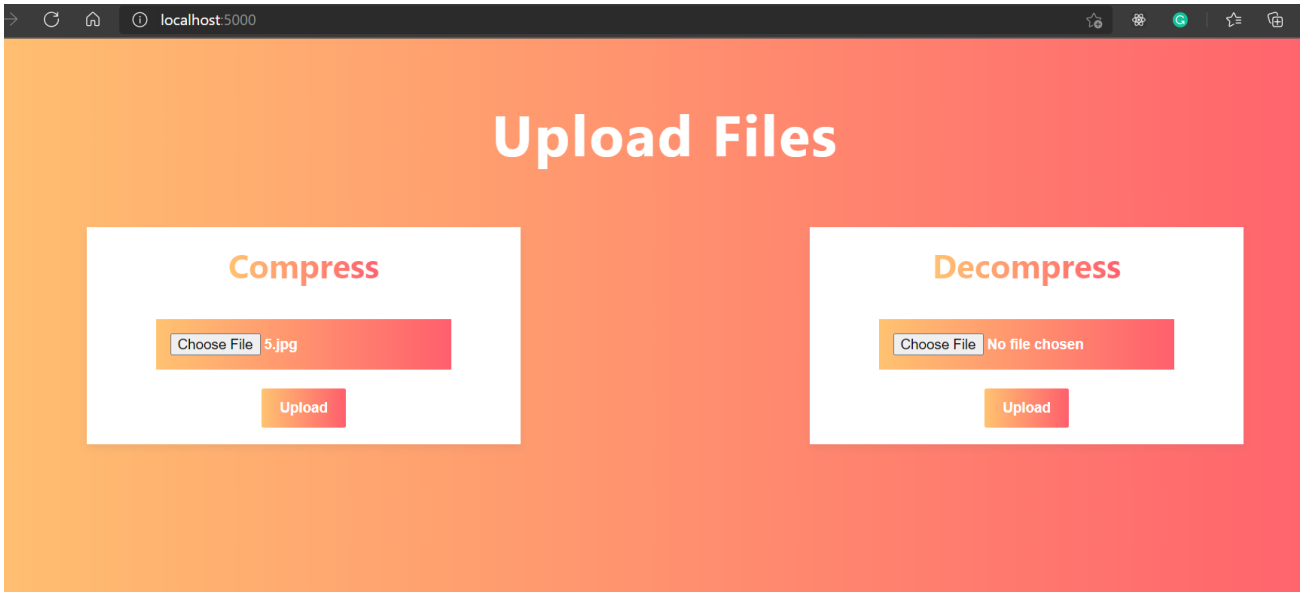
- ➔ Importing dataset and preprocessing the data
- ➔ Visualising training data image
- ➔ Resizing the image to flatten it.
- ➔ Analysing the relationship between the number of clusters and the inertia to find the optimum number of clusters
- ➔ Clustering the pixel points as per the pixel values.
- ➔ Constructing the Compressed image
- ➔ Compare the original and the reconstructed image
- ➔ Visualising the compressed image for all the clusters

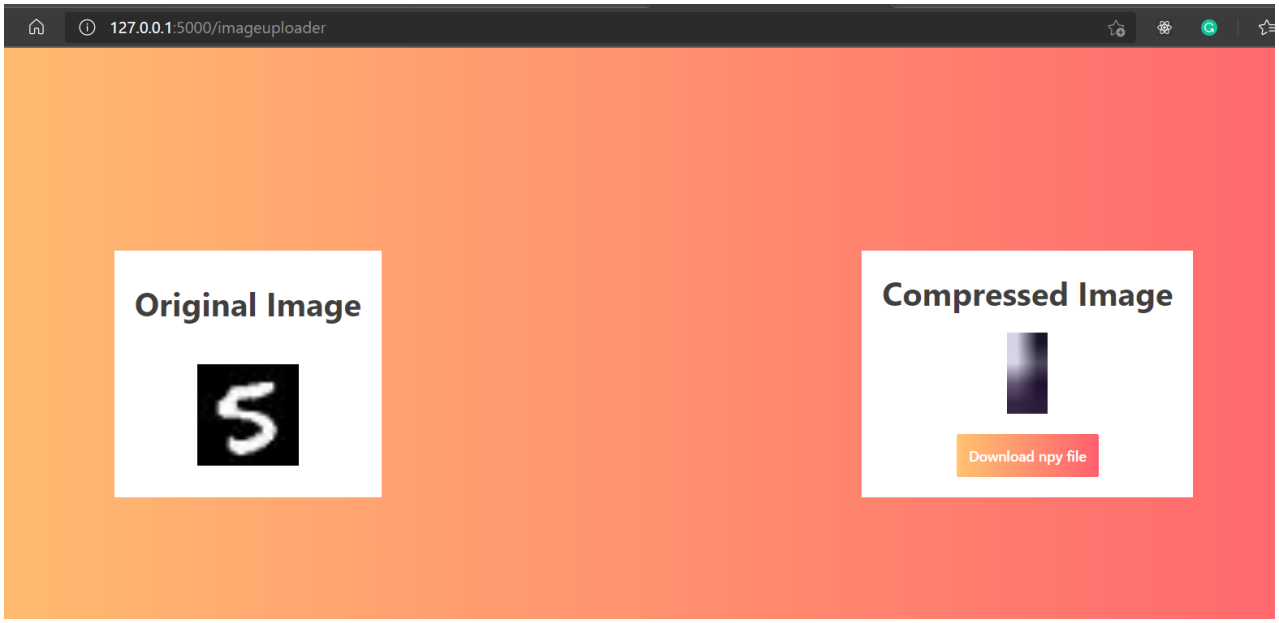
- **Deployment of models using Flask**

Autoencoder:

We built a web application using Flask which will allow the user to upload images to be encoded (i.e., compressed) using a convolutional autoencoder. Once encoded, the user gets a vector of elements representing the entire image which will be available for download as a .npy file.

This application can be used by an online storage provider to store the customer's data in a compressed vector format and reconstruct the data based on the user's demand. It could also be used by a layman to compress a very large image file and send it to their intended recipient over the internet as a npy file, thus consuming much lesser bandwidth. The recipient can then use the web application's decompression feature to reconstruct the original image with minimal loss and download the same.


localhost:5000
Upload Files
Compress
Choose File 5.jpg
Upload
Decompress
Choose File No file chosen
Upload


127.0.0.1:5000/imageuploader
Original Image
Compressed Image
Download npy file

The autoencoder model has been trained for 50 epochs and has been split into 2 separate models- encoder and decoder, to cater to the needs of the web application's user. These saved models are loaded in the backend of the web application and the user's image uploaded via the website is preprocessed and changed into the format specified by the input layer of the encoder model. The output of the encoder model is saved as a .npy file and is made available for download on the website. In addition to that, the website also displays the intermediate compressed image.

K-Means:

The Flask web application allows the user to upload images to be encoded (i.e., compressed). The user get's the output compressed image.
This application can also be used by an online storage provider to store the customer's data in a compressed vector format and reconstruct the data based on the user's demand based on the cluster number input. Needless to say, lesser the cluster number, there is more decrease in the size, although there is a tradeoff between the size and the quality of the image. One important thing to note is that this application can compress an image of any dimension (RBG) as per the user's choice of cluster number i.e the number of colours he'd like to preserve in the compressed image.

Original Image:

Image Compression Using K-Means Visualizer

Compress

Choose File aot.jpg

Enter the no. of clusters:

3

Upload

Original Image

Compressed Image

Download compressed file

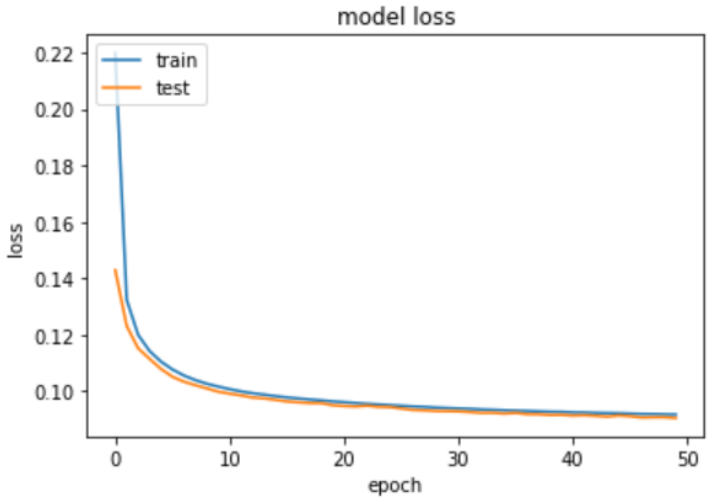
Compressed Image:

K = 3 **K=12**

Analysis performed on the models:

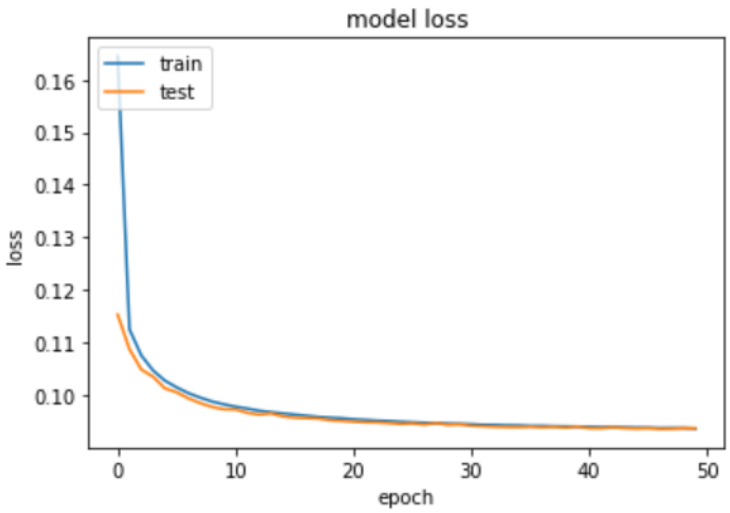
Convolutional Autoencoder:

Loss Vs. Epoch (Autoencoder for Compression and Decompression)



The above graph graphically visualizes the reconstruction loss that is the mean squared error loss for the validation and training dataset. The x-axis represents the epochs that are run for training the model, and the y-axis shows the loss value for the validation and training data. Both the training and the test data is assessed against the same number of epochs run.
It can be observed initially that the loss of the training data is very high as at the initial stage the model has just started to learn. While starting to train, it can be observed that the training loss has undergone a steep fall, and the rate of decrease of the loss is slowly getting reduced from the **20th** epoch. The reconstruction loss for the training data is continuously decreasing and also gets stable at the minimum loss reaching the value of 0.12 reconstruction loss value. The loss of the validation data also decreases and gets stable.

Loss Vs. Epoch (Autoencoder for Denoising)



A good fit is the goal of the learning algorithm and exists between an overfit and underfit model. The loss of the model will almost always be lower on the training dataset than the validation dataset. This means that we should expect some gap between the train and validation loss learning curves. This gap is referred to as the “generalization gap.”

The above plot shows a good fit as:

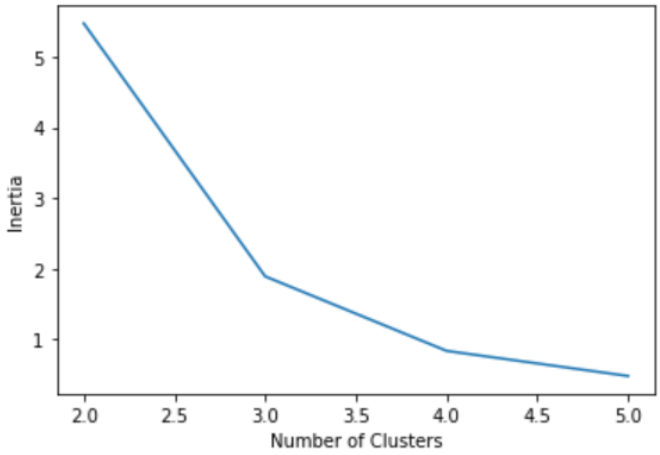
- The plot of training loss decreases to a point of stability.
- The plot of validation loss decreases to a point of stability and has a small gap with the training loss.

Quality Metrics: PSNR value

The term peak signal-to-noise ratio (PSNR) is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation.
The PSNR value of a given random input image is found to be: `75 dB`
PSNR high means good quality and low means bad quality. The PSNR value of the model is high, due to denoising. But there is a tradeoff between huge decrease in size of the image and image quality.

K-Means:

Inertia Vs. Clustering

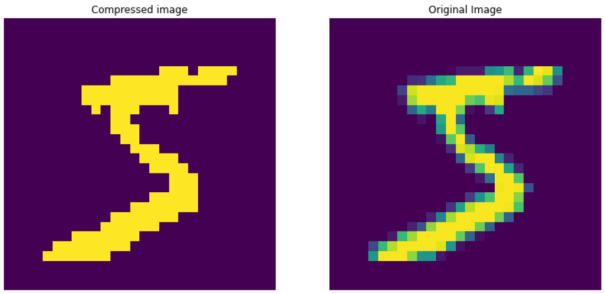


A fundamental step for any unsupervised algorithm is to determine the optimal number of clusters into which the data may be clustered. The Elbow Method is one of the most popular methods to determine this optimal value of k.

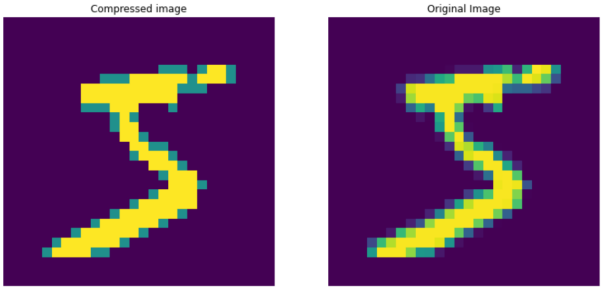
To determine the optimal number of clusters, we have to select the value of k at the “elbow” i.e. the point after which the distortion/inertia start decreasing in a linear fashion. Thus for the given data, we conclude that the optimal number of clusters is **4**.

Compressed Images for varying number of clusters:

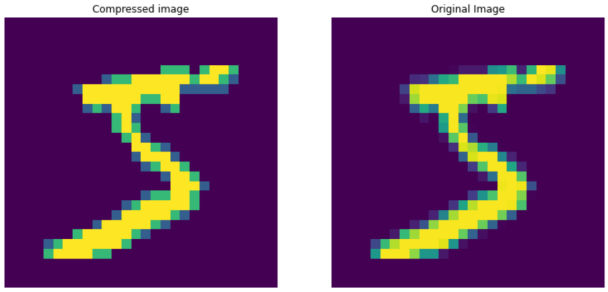
No. of clusters = 2



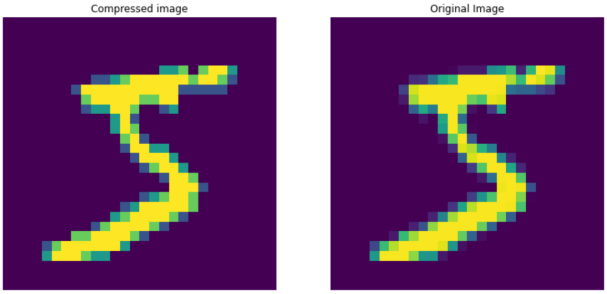
No. of clusters = 3



No. of clusters = 4



No. of clusters = 5

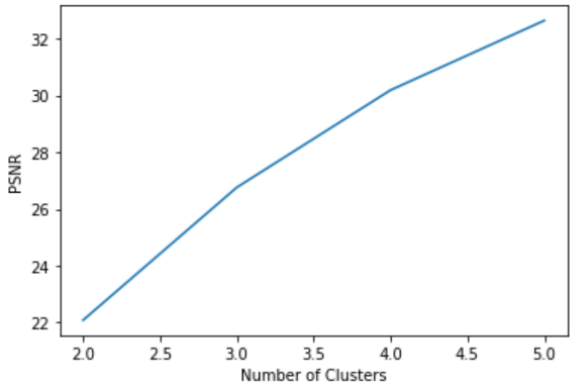


Observations from the above image :

- Dimensions of all the compressed images are the same as that of input images.
- The size of the compressed image decreases as k decreases.
- For the value of k=4,5 the output compressed images seem reasonably good and the loss of colors are not visible to a human eye. The size of the compressed image decreases by almost 3 times compared to the original image.
- For the value of k=2,3, the output compressed images lose a lot of colors and the lossy compression is visible to a human eye.

Quality Metrics: PSNR value

The term peak signal-to-noise ratio (PSNR) is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation.



The value of PSNR increases with the increase in the number of clusters. This means that there is less and less loss of information with increase in clusters but lesser difference in reduction of the size of the image.

A comparative study:

Image compression has been tested using two different techniques - an unsupervised clustering algorithm (K-means clustering) and an artificial neural network (Autoencoder). In this study, the K-means algorithm actually provided slightly better results than the autoencoder, which is quite surprising. The inferior performance of the autoencoder can be attributed to it's high data specificity and since the model has been trained on the MNIST dataset where the image sizes are considerably small (28x28), the compressed npy file's size turns out to be in KB. This upscaling in size would not occur if the image was huge because the npy file's size will be in the order of the same magnitude and will not change.

The K-means algorithm provided very satisfactory results and it could compress all types of images and is not restricted to any particular class of images like the autoencoder. The downside of this method is that the K value (number of clusters) will have to be selected through some visualization and trials. It produces images with different shades and colors while compressing and if we want the image to retain its color, we will have to use a technique like Principal Component Analysis.

Scope for future work

Image compression has become the most significant problem in almost every domain like the medical sciences, entertainment, and IT field as with the rapid digitization, the rate of growth of data generation has become very high. This has led to the necessity of the development of compression of images for optimum storage of these large numbers of generated images and their transmission. Various methods have been used for image compression, but those methods had their own glitches. Hence the objective of this project was to handle image compression with the deep learning approach.

A convolutional autoencoder has been constructed that has successfully compressed the image of higher to lower dimension where the compression and the decoder could successfully reconstruct the compressed image with minimum possible reconstruction loss. Though the performance of the model was very good but there persists a future scope of optimizing the model further, so that it can compress much higher dimensional data with almost no loss and also the compression factor can further be optimized. Along with this, a comparative study of the performance of this model can be done with other methods like PCA, which are also renowned for image compression so that the model can be further optimized by identifying those optimizing factors, by studying those competitive existing models.

References

1. *Image Compression Using Autoencoders in Keras | Paperspace Blog*

2. *Tutorials | TensorFlow Core*

3. *A. Sento, "Image compression with auto-encoder algorithm using deep neural network (DNN)," 2016 Management and Innovation Technology International Conference (MITicon), Bang-San, Thailand, 2016, pp. MIT-99-MIT-103, doi: 10.1109/MITICON.2016.8025238.*

4. *Z. Cheng, H. Sun, M. Takeuchi and J. Katto, "Deep Convolutional AutoEncoder-based Lossy Image Compression," 2018 Picture Coding Symposium (PCS), San Francisco, CA, USA, 2018, pp. 253-257, doi: 10.1109/PCS.2018.8456308.*

5. *A. Baviskar, S. Ashtekar and A. Chintawar, "Performance evaluation of high quality image compression techniques," 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 2014, pp. 1986-1990, doi: 10.1109/ICACCI.2014.6968643.*

6. *Song Zebang and Kamata Sei-ichiro. 2019. Densely connected AutoEncoders for image compression. In Proceedings of the 2nd International Conference on Image and Graphics Processing ICIGP '19 Association for Computing Machinery, New York, NY, USA, 78–83. DOI:https://doi.org/10.1145/3313950.3313965*