Indian Institute of Information Technology(IIIT), Design and Manufacturing, Jabalpur

CS314b : Machine Learning
Project Report

Reverse-Oldification Of Images

Manav Goyal *2017138*
Aditya Baurai *2017307*
Rakshak *2017207*

We want to thank our course instructor Dr. Kusum Bharti(CSE faculty at IIIT Jabalpur) for giving us the opportunity to work on this project.

(I) Abstract :

Here we aim to study and implement a deep learning architecture revolving around the application of a neural network in order to rejuvenate black and white images, that is by colorizing them, and hence making them *'alive'* again. Image restoration cum reconstruction has always been a topic of interest, with applications such as extracting useful information from the images of ancient historical artifacts(*after reverse-oldifying it to increase the color channels and hence, the amount of information encapsulated*), or even bringing a black and white snapshot from the 90s to this century(*applications in entertainment industry*), or colorizing the popular Mangas(Japanese comics), which are drawn without colors(mostly). The heavy process has expedited with the advent of the modern deep-learning/Big Data era, where GPUs and TPUs are getting more and more powerful as time progresses, along with a massive surge in the amount of data available to learn from.

(II) Dataset used :

The dataset is a result of seven researches from the website **flickr** containing real world photos :

- Landscapes(900+ pictures)
- Landscapes mountains(900+ pictures)
- Landscapes desert(100+ pictures)
- Landscapes sea(500+ pictures)
- Landscapes beach(500+ pictures)

- Landscapes island(500+ pictures)
- Landscapes Japan(500+ pictures)

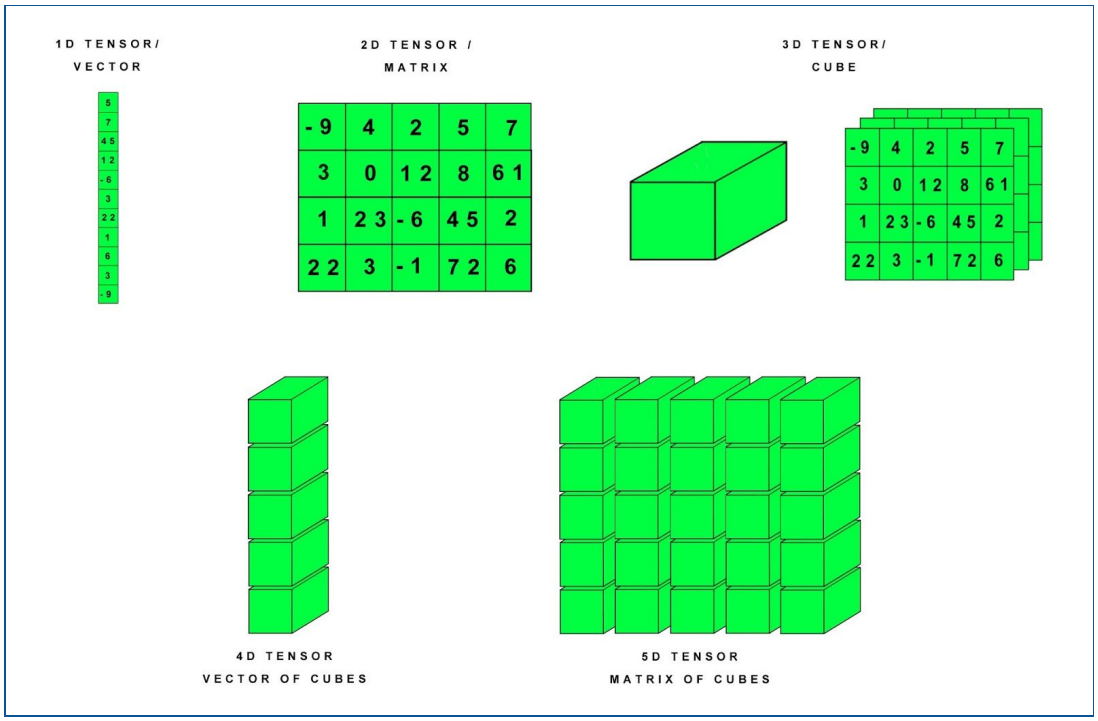
(Fig 1 || *Some sample images from the landscape dataset*)

This dataset containing 3000-4000 images is publicly hosted on **Kaggle** from where it's downloaded and subsequently uploaded(after cleaning) on the following google drive path:

https://drive.google.com/drive/folders/10FpGEaeEM2AROcP0zAJ8Oz1QmERD7ifB?usp=sharing

(III) Libraries used :

- **Numpy :** Fundamental package for scientific computing in Python3, helping us in creating and managing n-dimensional tensors. A vector can be regarded as a 1-D tensor, matrix as 2-D, and so on.

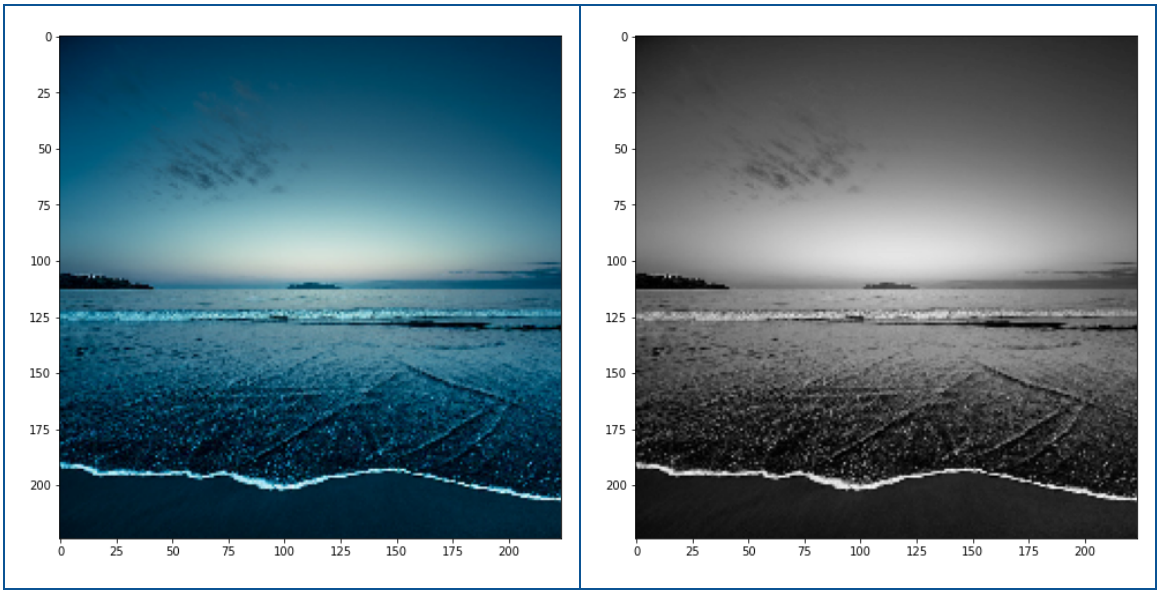


- **Matplotlib :** A Python3 plotting library used for data visualization.
- **SkImage :** Is an open source Python3 package designed for image processing.
- **Tensorflow :** Is an open source deep learning framework for dataflow and differentiable programming. It's created and maintained by Google.
- **Tqdm :** Is a progress bar library with good support for nested loops and Jupyter notebooks.

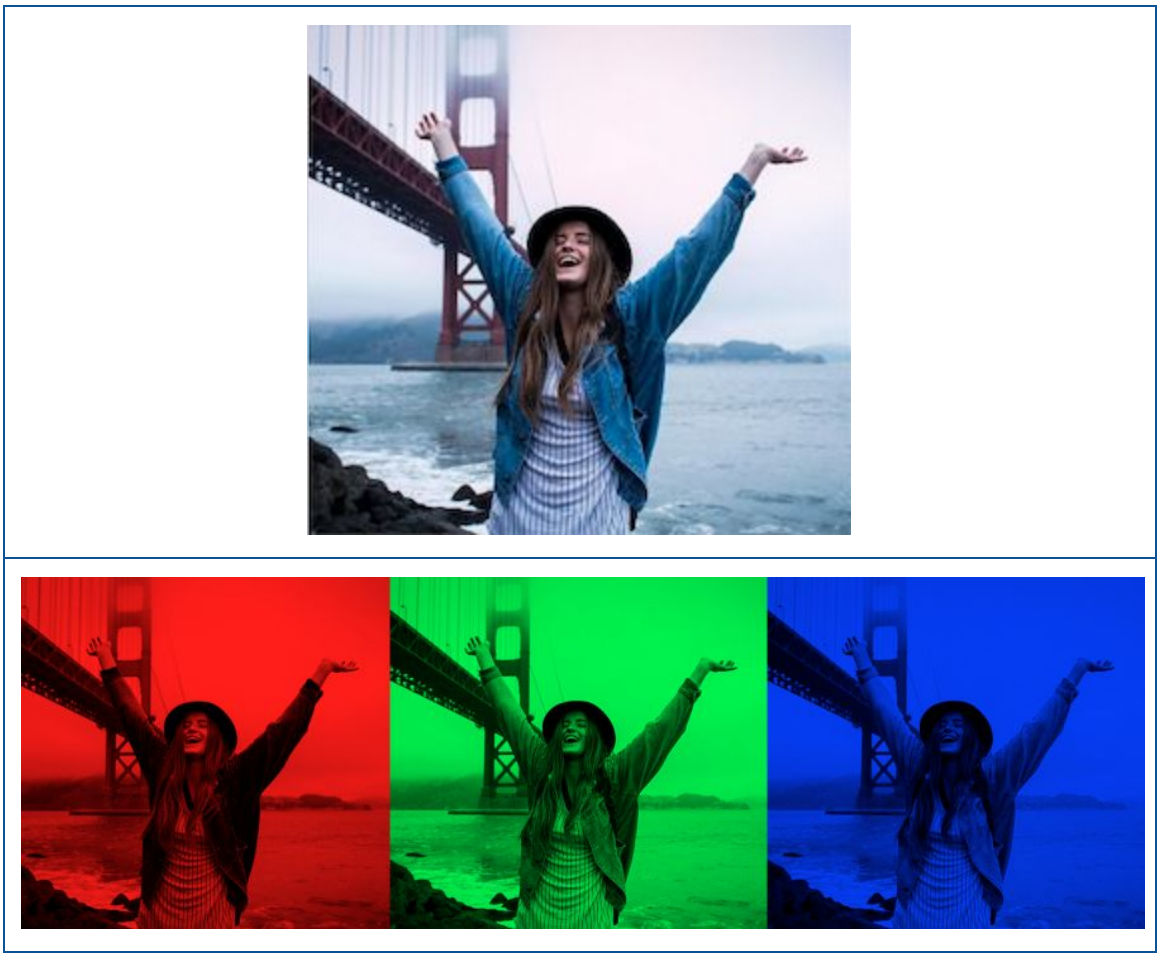
(IV) Image pre-processing :

- **Rescaling -** All the images of the data set will be resized to 224 X 224 dimensions. This is in accordance with the input size expected by the neural network architecture(encoder).

- **Batch Size -** Instead of feeding the whole set of 3000+ images to our deep learning model in one go, images are fed in batch of 16; so that our Jupyter notebook doesn't try to load data that may exceed the RAM allotted by Google Colab(~ 12 GB).
- **Image augmentation -** ***ImageDataGenerator***, a Tensorflow image pre-processing deep learning module is used for this cumbersome task. Using ImageDataGenerator, we **rescale** each image in our data set, so that the value of each pixel lies in the range of 0-1, instead of 0-255. This rescaling aids our subsequent deep learning model to converge faster as the **skewness** of the overall data distribution is tremendously reduced, thereby expediting the gradient descent operation. Moreover, an additional parameter known as '**validation_split**' is also passed, which segregates a small fraction of images for cross-validation(after the model is trained on the training data set).
- **Helper function design :** Helper functions such as **convert_lab**(for converting an RGB image into LAB format using skimage library), **convert_rgb**(for converting the constructed LAB image, after A and B channels are predicted, into RGB format using skimage image pre-processing library), and **plot_image**(for displaying an image) are designed.



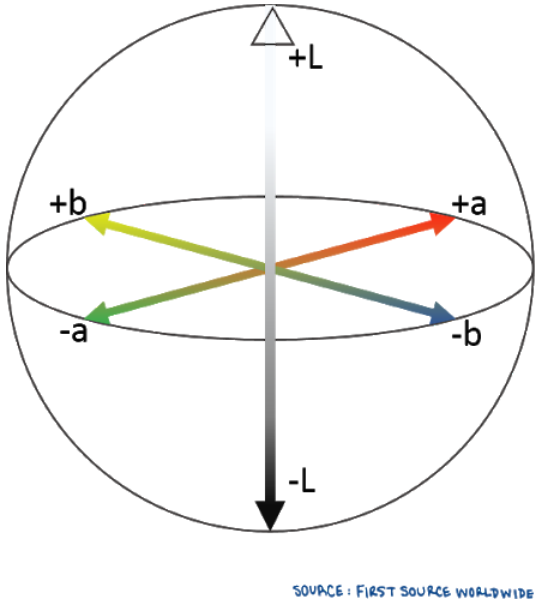
(Fig 2 || *The* ***convert_lab*** *function converts RGB to Lab format.* ***Left Image :*** *RGB image from training set,* ***right image :*** *The corresponding '****L****' channel from the Lab. Like R,G,B channels in an RGB image; the Lab also comprises 'L', 'a' and 'b' channels. 'a' and 'b' channel regulates the color tradeoff whereas L channel has information of luminosity in an image*).



(Fig 3 || *An RGB image shown above is decomposed into its channels - R, G and B*)

- **Preparing the Numpy arrays :** The training and validation numpy array are created. All the images in the training set are loaded one after the other, and are subsequently converted into LAB format. The **x_train** list contains the 'L' channel of our LAB image, whereas **y_train** contains the 'A' and 'B' channels. Same goes for **x_val** and **y_val.** Finally the list is converted into a numpy array, as they are the input type desired by Tensorflow.
    - The shape of x_train observed is (3680, 224, 224) signifying that this 3-D tensor contains 3680 images, where each image has a dimension of 224 X 224. This makes sense as earlier we rescaled all the images to 224 X 224. Moreover, as the training set contains only the 'L' channel, therefore each image has only dimension(*depth wise*).

- The shape of x_val observed is (3680, 224, 224, 2) signifying that this 3-D tensor contains 3680 images, where each image has a dimension of 224 X 224 along with depth 2. This is because both 'a' and 'b' channels of the Lab image are incorporated in the x_val tensor. All the images in a and b are 224 X 224, and they are stacked together in x_val, imparting the shape an overall depth of two.

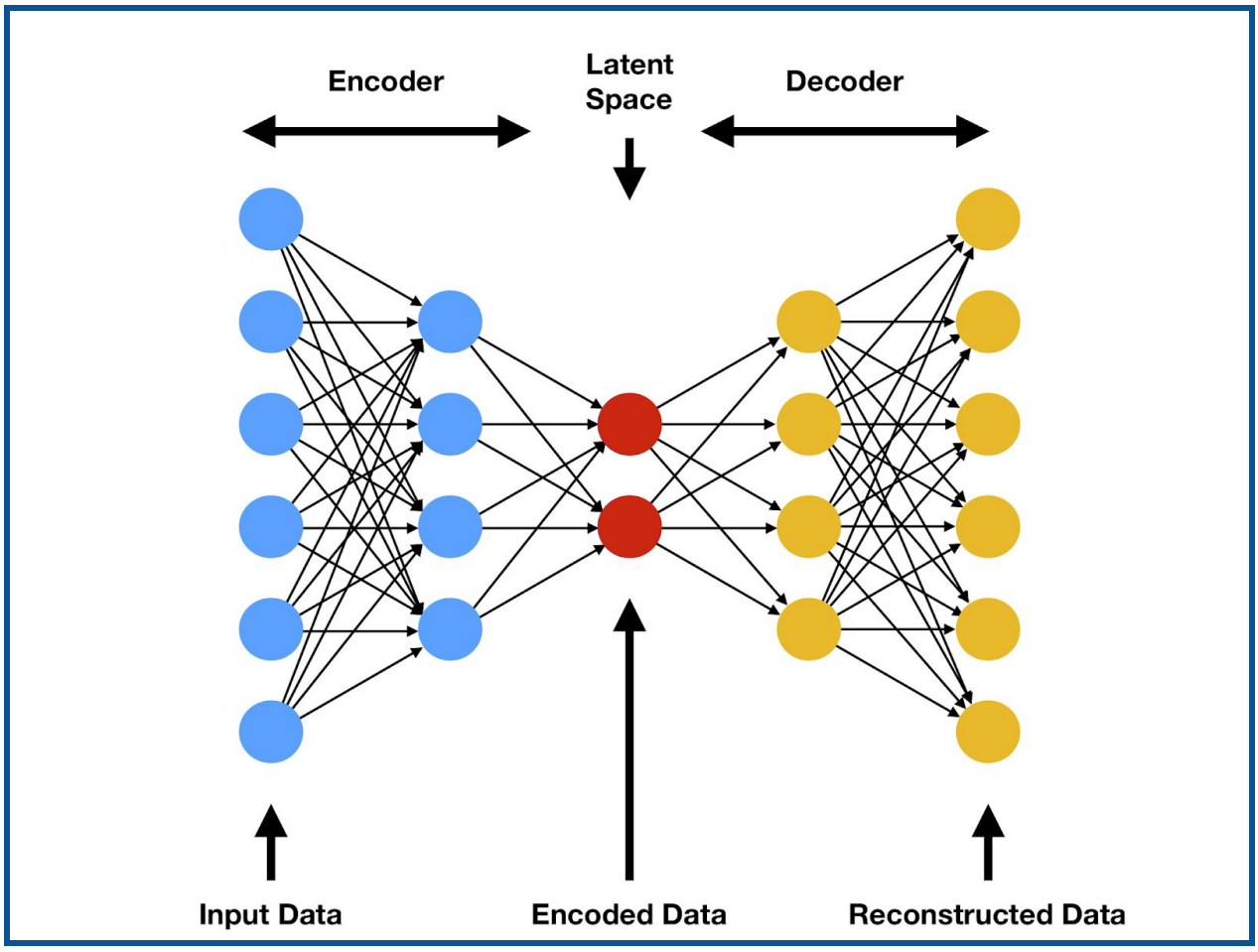


(Fig 4 || *Lab color space depicted. L controls the luminosity/brightness in an image, whereas 'a' and 'b' controls the color tradeoffs).*

(Fig 5 || *L and ab components of an Lab(also called CIELab) shown*).

(V) Deep Convolutional Auto Encoder :

Auto encoders are deep neural networks used to determine a compressed version of the input data with the **lowest amount of loss** in information. The concept of PCA(Principal Component Analysis) is to find the best and relevant parameters for training of a model where the dataset has a huge number of parameters. An autoencoder works in a similar fashion. The encoder part of the architecture **breaks down the input data into a compressed version ensuring that important data is not lost but the overall size of the data is significantly reduced.** This concept is called **Dimensionality reduction.**

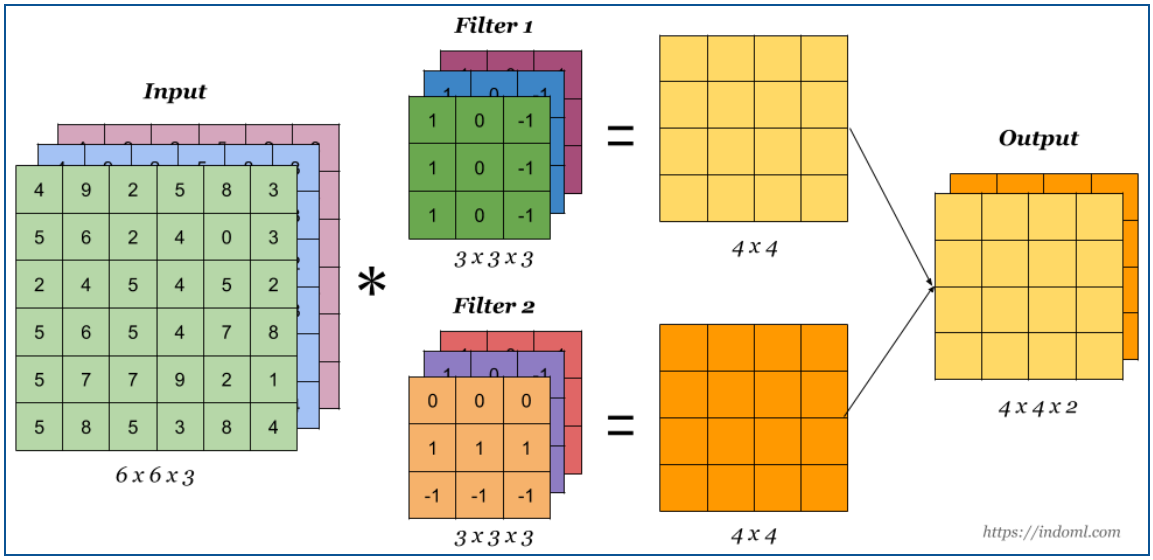


(Fig 6 || *An AutoEncoder*)

One important property of an autoencoder is that they are **data-specific**, which means that they will only be able to compress data similar to what they have been trained on. This is

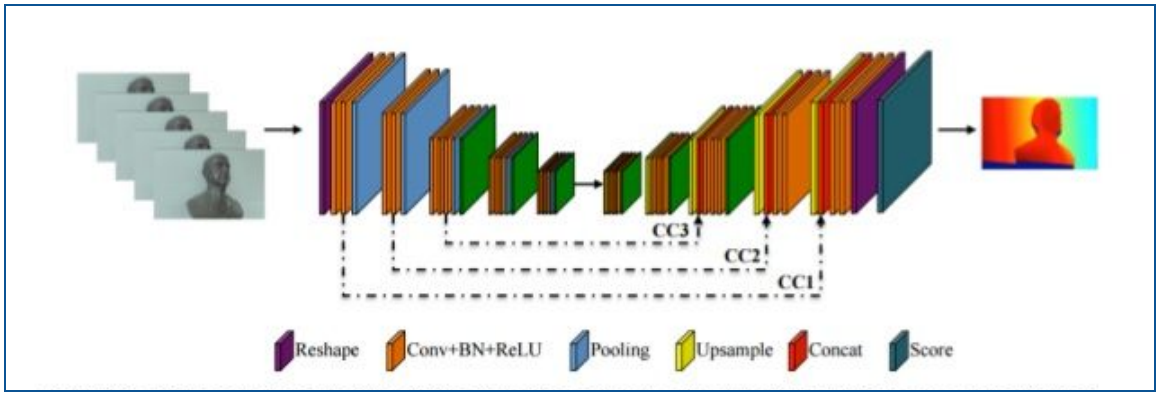
why our hold-out cross validation set consists only of **landscape images**, as those are the ones our convolutional autoencoder had trained on.
Since we are dealing with images, we have used a deep convolutional autoencoder having two main layers :

- **Convolution layer :** A convolution is a simple application of a filter to an input that results in an activation. Using convolving filters of the size (3,3), we extract useful features from an image to be colored. By adding an 'additional' **stride** component, the images are downsampled from their original dimensions.



(Fig 7 || *2-D Convolution operation shown on an image*)

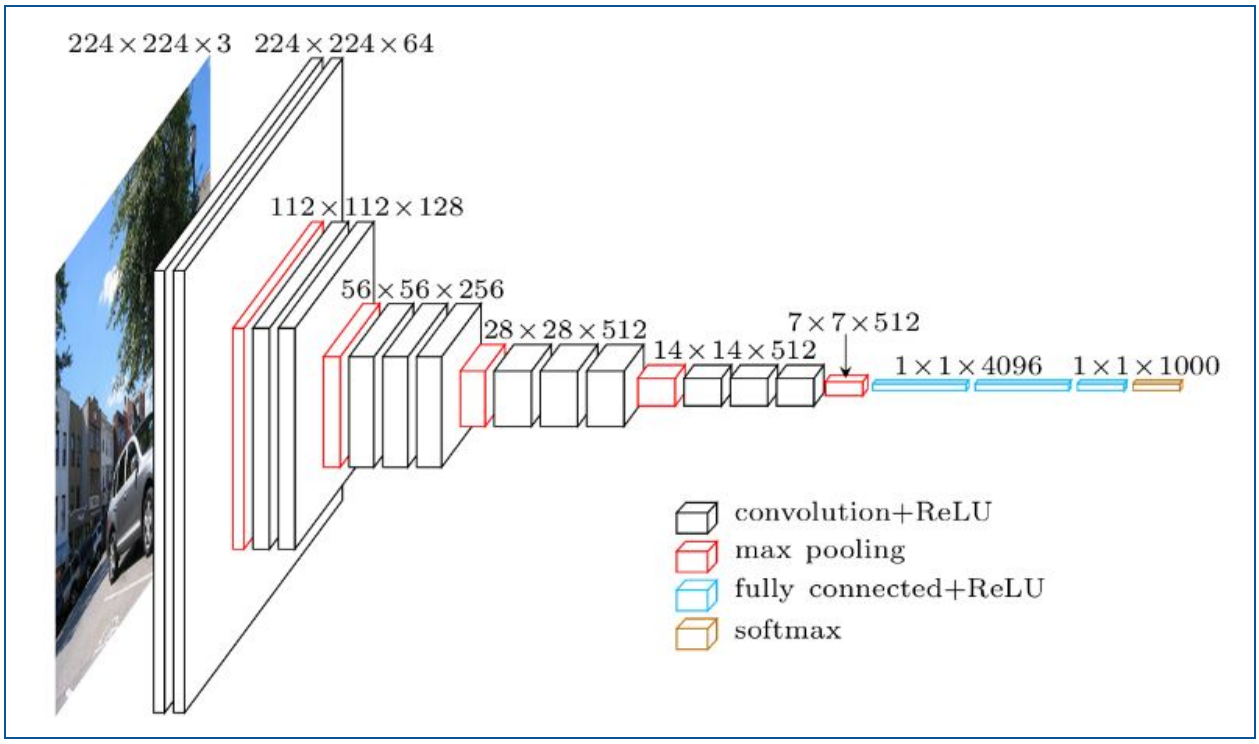
- **Upsampling layer :** This is applied in the decoder section once we start reconstructing the image, after the encoder has extracted all the useful features. Upsampling simply results in doubling the dimensions of the input.



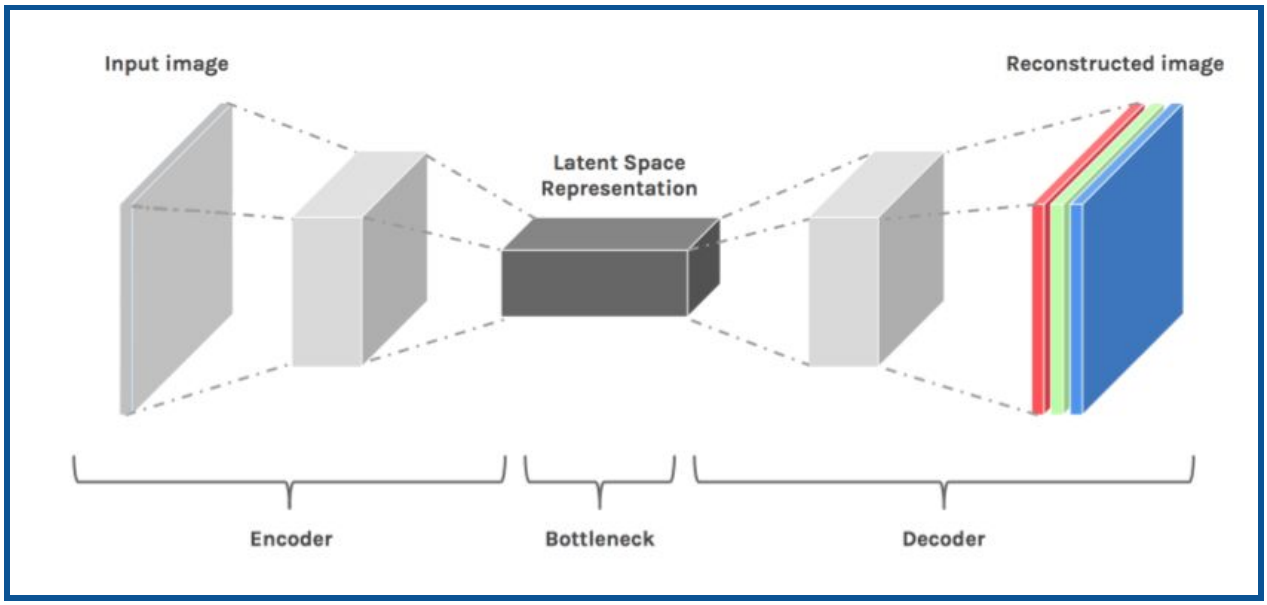
(Fig 8 || *Upsampling leading to an overall increase in dimensions in the latter half of the structure*)

VGG net 16 has been used for our encoder to obtain high accuracy in the feature extraction process. The architecture of VGG-16 is such that it takes an input of size 224 X 224 X 3(hence, the image resizing in pre-processing), and outputs a softmax prediction over a thousand classes. As discussed above, this network also has immense stacking of convolving layers, and the last feature extraction layer results in an output of 7 X 7X 512. Hence, we will use these layers for feature extraction for our own encoder.

Following is the architecture of VGG-16 :

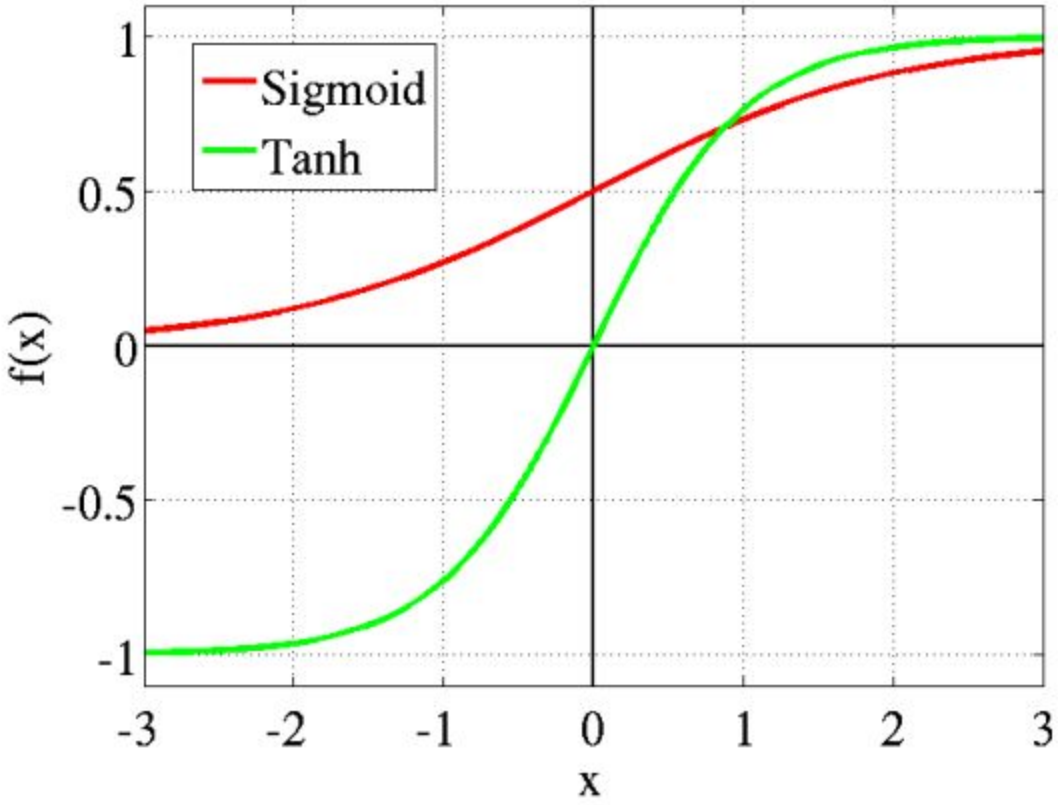


*(*Fig 9 || *Note that VGG 16 takes a three dimensional input, whereas we have only our L channel as input, which is one dimensional. Hence, in order to satisfy the need of this architecture,* ***we will make our input 3-D by stacking the 'L' layer behind itself two more times, forming a total of three dimensions****).*



(Fig 10 || *In short what's happening with our autoencoder model. The input image has only one channel, that is 'L'(which is later stacked behind itself to give an impression of depth 3 in order to satisfy our encoder model), and output comprises two sets of predictions, that is 'a' and 'b' channel; which are then combined with the input 'L' to form a reconstructed Lab image. This Lab image is subsequently converted into an RGB image, by the* ***convert_rgb()*** *function earlier defined and displayed*).

The "**relu**" activation has been used throughout the network's architecture, however for the final layer, we have used **'tanh'** activation function. The reason is before preprocessing the 'a' and 'b' channel values were divided by 128 as the **range of both channels is in (-127, 128)**. **Hence, by dividing the values, each pixel lies between (-1,1)**. The mathematical "tanh" activation function outputs in the range of (-1, 1), therefore making it the ideal function of choice.



(Fig 11|| *The **tanh** Vs **logistic/sigmoid** activation function*)

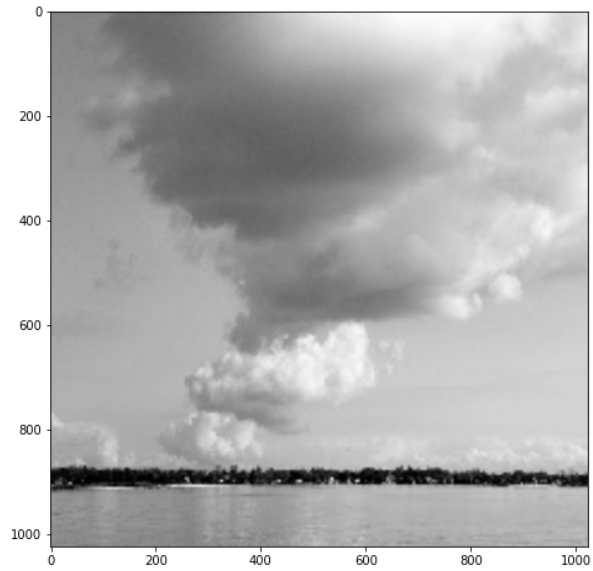
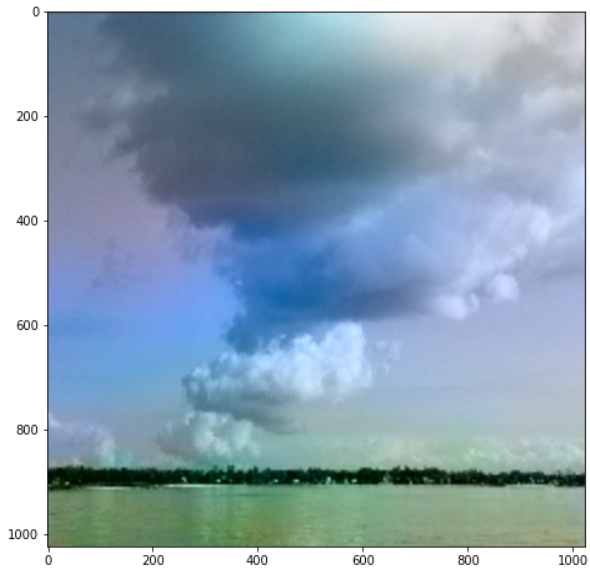
(VI) Results obtained :

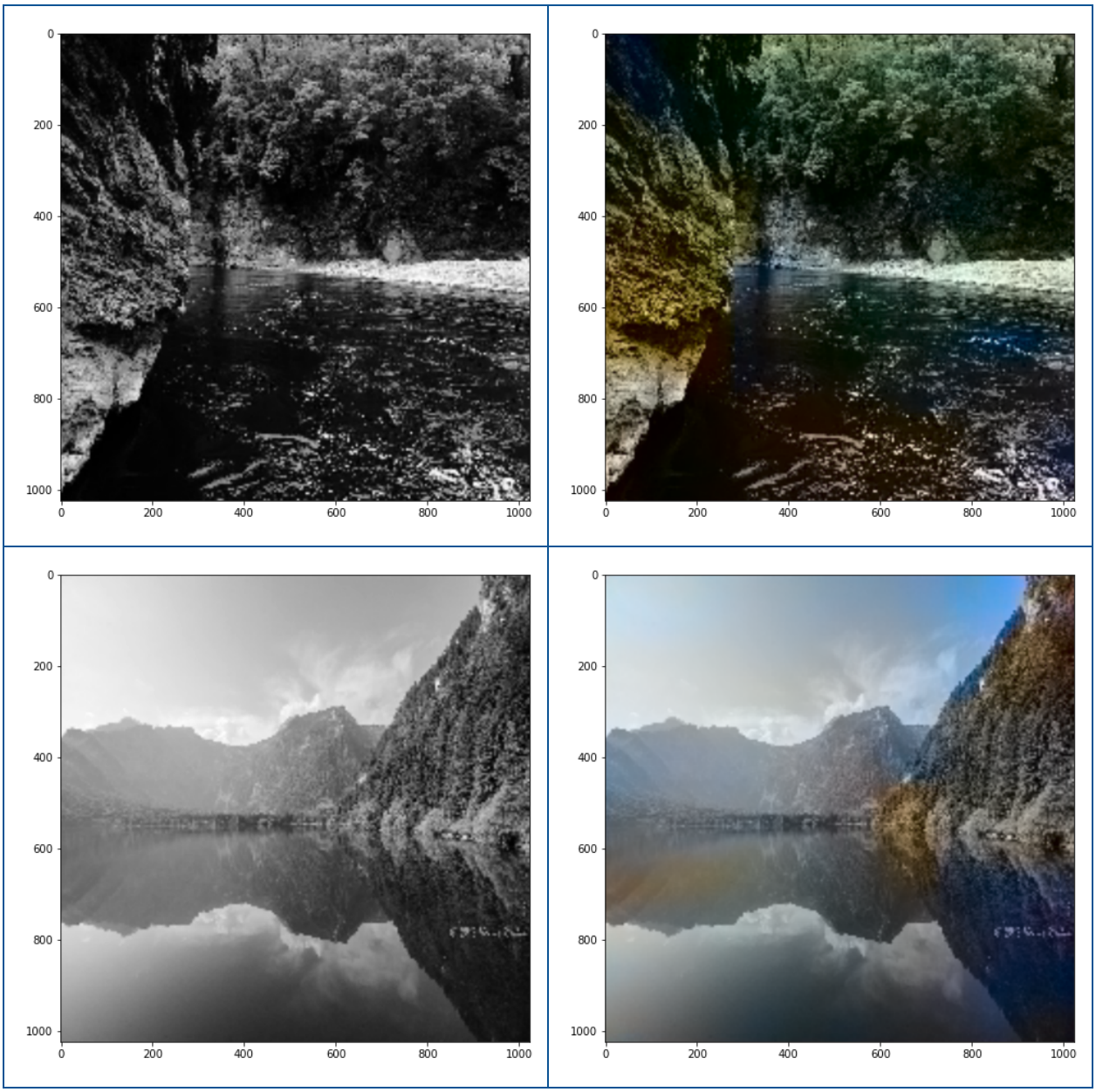
The model was trained on **3,680 images** for **1000** epochs. An overall accuracy of **86.12%** was achieved by the model in coloring the images of dimension : (224, 224)

The training process took roughly four- five hours to complete. The entire model was trained using **Google Collaboratory notebooks**, which are powerful GPU-enabled Jupyter notebooks supported by Google cloud.

The RAM availability for the whole project was roughly around **12 GB** using a free **Tesla K-80 GPU**.

Following were the results obtained on the hold-out cross validation set :

L image fed to our model	Colored image obtained
	
	



(Fig 12 || *Results obtained on the hold-out cross validation set, after the model trained on the training set*).