The Red-Green-Blue(RGB) space is only one way to represent a color image. Going from a traditional RGB representation of an image to grayscale is a trivial matter, in which the grayscale image is just a linear combination of the red, green, and blue channels. However, going the opposite way has many challenges,

that is we must predict a three dimensional object from a one dimensional input. Before we construct our neural network, we change the problem. Since going from one channel to three is very difficult, we change our bases into something more managable

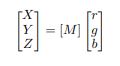
- the LAB color space.

Here L is the lightness, and is analogous to the grayscale image.

A and B, is a two dimensional mapping of RGB.

We can transition between RGB and LAB using the following method:

First transition to the XYZ specific illuminants and observers and then construct the "Hunter-Lab" colors pace:



where [M] is calculated from the RGB reference white primaries (Xr, Yr, Zr) of the computer system. After obtaining the XYZ space from the original RGB image, the LAB space is calculated via the relationship

L = 116fy − 16

A = 500(fx − fy)

B = 200(fy − fz)

We do this transformation to reduce the dimension of our problem. In the original situation, the grayscale image is used to predict the RGB channels. After the transormation, we use the L channel, analagous to grayscale, to predict only the A and B channels. This is a common strategy used in the literature. The opencv library offers routines to translate RGB to LAB using the formalism above, we used this library in our preprocessing and post processing stages in our algorithm.

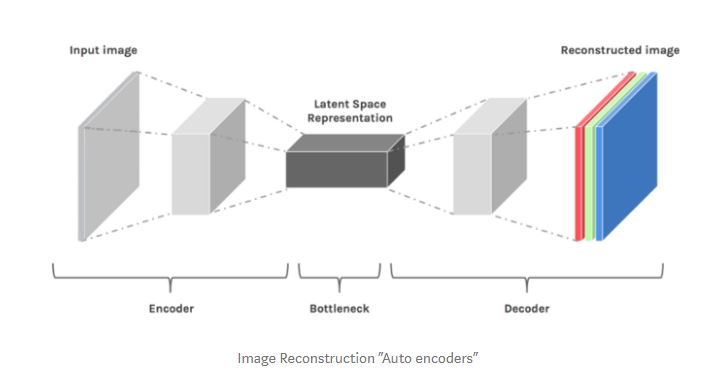
**Dataset**

you must choose a dataset that is consistent, for example, many of the datasets I found have many black and white images and that can maximize your error in the training part because in this case instead of showing a grayscale image and its colored version for your model to learn the mapping from grayscale to colored image, you are showing a grayscale version and a grayscale image too as a target, nothing useful for your model to learn in this case, so you should remove any grayscale images from your dataset and choose all of them to be colored as **RGB**. After filtering your data, now you ready for the preprocessing part.

## Preprocessing

Now, you have a dataset of **RGB** images. as you figured out in the **general explanation** section that choosing **Lab**instead of **RGB** is a better choice because you are only interested in generating colors and the colors information are embedded in the **ab** channels instead of **RGB**. in preprocessing you will need to convert all the dataset from **RGB**to **Lab.**another thing to do before converting to **Lab**is to normalize your dataset (divide by the maximum value that **RGB** pixel can reach) because the **RGB** range is between **0–255** for each color channel, so maximum value that **RGB** pixel can reach is **255**. This “normalization” enables us to compare the error from our prediction and converge faster. Also, all the images should be of the same size, so we will resize them to be “**256** x **256**”

## Train your network



As you see, we can use Auto-encoder for the reconstruction of the image, in other words, we would say that it has the ability to generate and that’s exactly what we want to do, we want to generate the three channels of ***RGB***.

One approach is to make two copies of your image, one to be a grayscale image and it will act as your input to the encoder which is responsible for extracting the features of the image *“Latent Space Representations”*that can be used as input for the decoder to reconstruct the image, the other copy will be the same image but colorized as your target to the decoder ([Supervised Learning](https://en.wikipedia.org/wiki/Supervised_learning)) so that it can minimize the error between the original colored image and the generated one. Auto-encoder architecture would be something like in the figure above.

There will be a [convolutional neural network](https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148)([**CNN**](http://becominghuman.ai/)) through the encoder part for extracting features. in the decoder part, there will be convolutional layers like those in the encoder (with different filters) but followed by upsampling layers for the reconstruction part.

you can control the number of the filters and layers in each layer, of course, the last layer should contain three filters that will be the ***RGB*** channels of the reconstructed image but, there is something smarter you can do. What if you have another color space instead of ***RGB*** that can isolate the color information from the image? This means you will get pure black and white information of an image in one single channel and the other two channels will contain the color information embedded in those two channels. that seems a very good idea and that what the ***LAB*** color space does for you.

Continuing to the previous section, we are going to change model architecture and use transfer learning instead of training the network from scratch, we are going to use VGG16 pre-trained model as an encoder, VGG16 is a convolutional neural network architecture. To this day is it still considered to be an excellent vision model. it was learned to classify between 1000 class of ImageNet dataset

The figure below illustrates the architecture of **VGG16**: the input layer takes an image in the size of (224 x 224 x 3), and the output layer is a softmax prediction on 1000 classes. From the input layer to the last max-pooling layer (labeled by 7 x 7 x 512) is regarded as **the feature extraction part**of the model, while the rest of the network is regarded as **the classification part**of the model.

