*A Project Report*

on

# Image colorization using a Deep convolutional neural network Auto-encoder without image compression

*to be submitted in partial fulfilling of the requirements for the course on*

**Artificial Intelligence – CSE 3013**

**(E2)**

by

| | |
|---|---|
| **Shaik Haseeb Ur Rahman** | **18BCE0646** |
| **Naman Sood** | **18BCI0168** |
| **Lakshit Dua** | **18BCE0824** |

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Fall Semester 2020-2021

**TABLE OF CONTENTS**

## ABSTRACT

This paper tends to the issue of producing a conceivable hued photo given a grayscale picture. Past ways to deal with tackle this issue have either depended on human inputs in the form of scribbles and hints or resulted in desaturated colorizations. Motivated by [20], we present a completely programmed approach that utilizes profound neural systems to image colorization. We investigate the convolutional neural network space, the optimizers, regularization techniques and learning rate scheduling to comprehend the viable procedures to acquire acceptable colorized images as our output. We train a convolutional neural system and provide an image colourization solution that retains the dimensions of the image. Our model aims to have the same input and output image size. We additionally examine other cutting-edge research done in the field that led us to generating satisfying images with low error rates, including conditional generative adversarial networks [19]. At last, we give a subjective furthermore, visual examination of our outcomes and plots to accompany them, finishing up with roads for future exploration.

## Objectives:

- Conversion of black and white images to colored images without human intervention using a unique convolutional neural network approach retaining the image properties and dimensions.
- Our secondary objectives involve improving the efficiency of the model through learning rate schedulers, different optimizers, dropouts and other techniques such as lasso and ridge regression.

## Introduction

In the past years, CNNs have emerged as the dominant technology for a host of image related tasks such as classification, labeling, image generation and style transfer, achieving error rates below 5% on the ImageNet challenge. Recently, Convolutional neural network models have been emerging as the state-of-the-art image classifiers and solving problems around it.. These characteristics make it a clear natural choice to explore the auto-colorization task at hand.

They have the ability to discern different patterns, hints and associate them back to the target classes. **We aim to use the robust nature of Convolutional neural networks with image colorization to provide an efficient and accurate colorization of the given input image while retaining the image size** (no form of compression or image property loss). We would be combining this robust property with auto-encoders that are usually used for image classification, data compression and other minor tasks such as image denoising.

This auto-colorization model can further be extended into a variety of different industries. Many large animation companies, sketching and comic book artists use these auto-colorization tools to efficiently color similar panels. This technology can also be incorporated in medical industries where the majority of machines work on black and white sensors (ultra-sound, MRI scans etc.). It can also be used as a preliminary proxy task for data visualization and analysis in data-sets collected from sensors.

**REVIEW-1 (Survey & Analysis)**

## Literature Review

There has been a great deal of progress in colorization of grayscale pictures without human mediation [20, 19, 18]. This issue isn't just fascinating from an aesthetic and materialistic perspective however it additionally has critical applications in significant exercises, for example, Video Restoration, Image Enhancement for better interpretability, colorization of representations done by craftsmen and artists and so on. The cycle can likewise be considered as a basic pre-handling task for highlight learning in a self-administered mode, carrying on as a cross-channel encoder [20]. As of not long ago, colorization has consistently been a semi-mechanized cycle that depends on indications and highlights from the client. Levin et al. [14], for example, suggested that neighboring pixels in space with comparable pixel weights ought to have comparable hues. Through his work, these hints/indications are spread out in the types of harsh and incorrect scrawls on a grayscale picture. The algorithm can recognize these indications and create excellent colorization dependent on it. A few others have improved this algorithm further, including Huang et al. [13] (by tending to shading and draining issues) and Qu et al. [12] (by changing the cost function to account for the continuity of similar colors over matching textures in addition to similar intensities). Welsh et al. [11], then again, utilized a full shading model of comparative structure, in this manner decreasing human reliance significantly more.

One of the pioneers to propose a convolutional neural system which was trained for characterization of pictures to create full shading channels for the pictures provided as input was Dahl [17]. Fitting on the ImageNet information base with a L2 regularization loss applied on the chrominance esteems and resulting pixels, his methodology brought about normal to below average yields and predicted results. The anticipated hues were consistently sensible and lied in the same root shade as the given picture but they were consistently desaturated, had muted colours or were tinted, an after-effect of the "averaging impact" of the L2 regularization. Later work in this space has moved toward this issue in a myriad of ways, the most well-known strategies comprising of regression onto a constant shading space [18, 17, 10] or order of quantized color esteems through classification [9].

There has additionally been progressions in present regularization techniques and objective functions to fold over and accurately order loss values for a multi-modular issue such as this. Hwang et al. [20] made an objective function exceptionally custom fitted for the issue of picture colorization. The methodology predicts an appropriation of potential hues for each given pixel and re-gauges the given misfortunes losses at the time of fitting to represent/accentuate on the uncommon hues (class weighting for rebalancing). Larsson et al. [8] and Iizuka et al. [7] have created comparable frameworks with negligible changes in the neural system architecture, techniques and models. While Hwang et al. [20] utilized a class weight classification model metric with rebalancing uncommon classes (pixels relating to uncommon hues), Larsson et al. utilized an un-rebalanced classification function, and Iizuka et al. utilized a regression objective function for the equivalent. Larsson et al. used hyper-sections [6] on a VGG architecture [5], Iizuka et al. utilize a two-stream design in which intertwined global and neighborhood weights and features.

GANs fill in as a great method to gain proficiency with the mappings between the sources of info and predictions just as the comparing loss and regularized values. It permits the model to be utilized in a more adaptable way for a wide assortment of circumstances. There has been a lot of analysts who have utilized GANs in a conditional setting too, polishing them on discrete labels [4] and text [3]. Isola et al. [2] expanded the utilization of GAN for picture to picture interpretation over a few areas, for example, blending photographs from name maps, recreating objects from edge maps, and colorizing pictures. All the more as of late, cycle GANs are being utilized for colorization [1] which diminishes the requirement for input-output sets. There has additionally been progress of picture colorization for craftsmen and comic essayists through the assistance of DRNNs (Deep residual neural systems) [22] as well as the blend of Vanilla neural systems with Adaptive picture clustering predictions [21]

## Preliminaries

The working environment would be Google Colaboratory for inherent GPU access and cloud storage. The language for modeling our auto-encoder would be python3. The various python3 libraries included in the project to obtain necessary outputs are as follows:

**1. NumPy** – A general purpose array-processing package. It provides high-performance multidimensional array objects and tools supporting operations on these arrays. It is the fundamental package for scientific computing with Python.

**2. Matplotlib** – It is a plotting library for Python and its numerical mathematics extension NumPy

**3. OS** – The OS module provides functions for interacting with the underlying operating system and its dependent functionality. It comes under Python's standard utility modules.

**4. cv2/OpenCV** – It is a library of Python bindings designed to solve computer vision problems. All the underlying OpenCV array structures are converted to and from NumPy arrays. It also makes integration with other libraries much easier such as SciPy and Matplotlib.

**5. Keras with Tensorflow backend** – It's open-source neural-network library which can run on top of a series of other libraries and software such as Theano, Tensorflow, PlaidML etc.

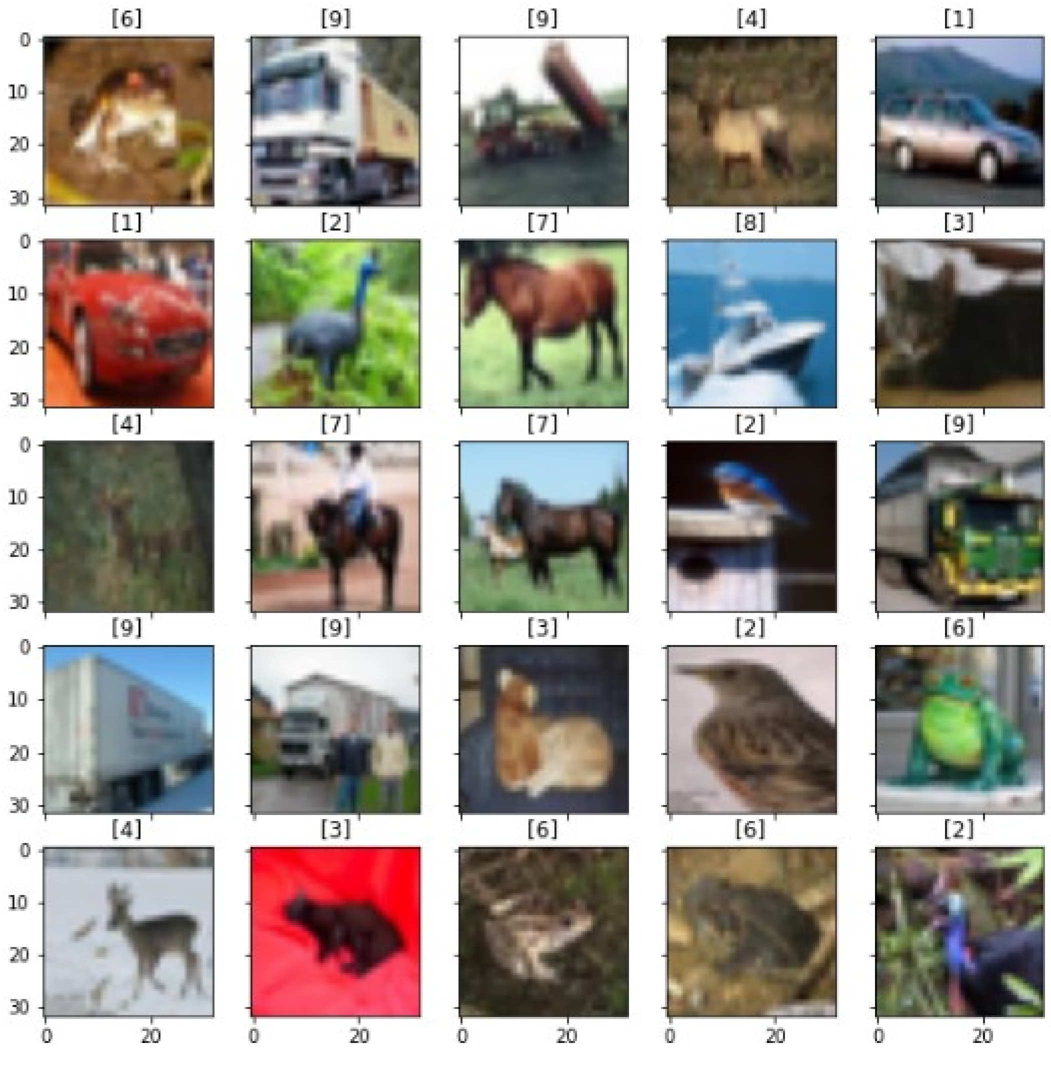
**Neural Network Components**

1. **Layers** – It refers to the neural network layers, there are different layers used in an auto-encoder model.
    1. **Dense** – It builds layers such that each node in an input layer is connected to the output layer.
    2. **Conv2D** – It develops kernels that extract features from the images
    3. **Flatten** – Converts the 2D matrix into 1D matrix. Reshapes the matrix layers to a list
    4. **Conv2DTranspose** – It is the transpose operation of Conv2D focused on building back the image from the filter aggregation.
2. **Model** – The object wrapped around the layer

## Dataset

The CIFAR-10 dataset is one of the most diverse and rich data sets for image training models. It consists of 60,000 32×32 color images with 10 classes, computing over to 6000 images per class. The model would be using 50,000 training images and 10,000 test images. The data set is divided into five training batches and one test batch, each with 10,000 images. The test batch consists of exactly 1,000 randomly selected images form each class. The training set contains exactly 5,000 images for each class. As the CIFAR-10 data set is coloured, we proceed with manually converting our training data set into Black and white through the help of computer vision tools like OpenCV. We will then proceed to use them in the training process.

*Sample CIFAR-10 Images with Accompanying classes*



## Proposed Methodology

### Input/Output

Our approach consists of training a CNN auto-encoder model with Black-and-white images and allowing it to learn coloured features without human supervision. To be more specific, we will convert the given coloured images into single-channel Black and white color space images that can provide us a grayscale input to our model. We then pass this model to our auto-encoder and it passes through the encoder schema first. The encoder will apply the convolution operation multiple times and return a vector of embeddings. These embeddings would be then passed to our decoder model which will apply the transpose convolution operation and return an output of the same height and width but with 3 different channels – RGB. The image dimensions and other properties would be retained throughout.

Input Image size – 32×32 pixel, single channel black and white images

Output Image size – 32×32 pixel, three channel image (RGB)

Encoding generation – [256] embedding per image generated in a latent vector

## Algorithm – Architecture and Flow

The model consists of an auto-encoder model which is a horizontal stack of the encoder and decoder models. The black and white image would first flow through the encoder model to be aggregated in an embedding vector. The process is as follows:

1. The image is passed to the encoder model

    1. The image undergoes convolution by 64 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 16×16×64
    2. The image undergoes convolution by 128 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 8×8×128
    3. The image undergoes convolution by 256 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 4×4×256
    4. The resultant vector is spread out into a 1D vector (dimension – [256]).

2. The embeddings vector (latent_vector) is passed to the decoder model to apply the transpose convolution operation:

    1. A dense neural network layer is applied onto the latent vector. It results in a 1D vector (dimension [4096])
    2. The vector is reshaped into a three dimensional vector – (4×4×256)
    3. The image undergoes convolution by 256 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 8×8×256
    4. The image undergoes convolution by 128 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 16×16×128
    5. The image undergoes convolution by 64 filters (dimension – 3×3) with a stride of 2, no padding and ReLU activation. It results in dimensions – 32×32×64
    6. The image undergoes convolution by 3 filters (dimension – 3×3) with a stride of 2, no padding and sigmoid activation. It results in dimensions – 32×32×3

The resulting image from the encoder model is displayed as the output image. We successfully retain the height and width of the image (32×32) throughout the colorization process.

1. The architecture of the image passed in step 1 (the encoder model):

| **Layer** | **Output Shape** | **Trainable Parameters** |
|---|---|---|
| encoder_input (Input Layer) | (None, 32, 32, 1) | 0 |
| Conv2d_1 (Conv2D) | (None, 16, 16, 64) | 640 |
| Conv2d_2 (Conv2D) | (None, 8, 8, 128) | 73856 |
| Conv2d_3 (Conv2D) | (None, 4, 4, 256) | 295168 |
| Flatten_1 (Flatten) | (None, 4096) | 0 |
| Latent_vector (Dense) | (None, 256) | 1048832 |

2. The architecture of the image passed in step 2 (the decoder model):

| **Layer** | **Output Shape** | **Trainable Parameters** |
|---|---|---|
| decoder_input (Input Layer) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 4096) | 1052672 |
| Reshape_1 (Reshape) | (None, 4, 4, 256) | 0 |
| Conv2d_transpose_1 (Conv2DTranspose) | (None, 8, 8, 256) | 590080 |
| Conv2d_transpose_2 (Conv2DTranspose) | (None, 16, 16, 128) | 295040 |
| Conv2d_transpose_3 (Conv2DTranspose) | (None, 32, 32, 64) | 73792 |
| decoder_output (Conv2DTranspose) | (None, 32, 32, 3) | 1731 |

The final aggregated model:

| **Layer** | **Output Shape** | **Trainable Parameters** |
|---|---|---|
| encoder_input (Input Layer) | (None, 32, 32, 1) | 0 |
| encoder_model (Model) | (None, 4096) | 1418496 |
| decoder_model (Model) | (None, 32, 32, 3) | 2013315 |

## Data Preprocessing

### Processing images to single channel black-and-white

Since there is no proper image dataset for the tasks of automatic colorization, we will be using the processed CIFAR-10 dataset where each and every image would be converted into black and white. The black-and-white single channel images would be the features for our training data while the coloured image would be the target.

### Normalization

An 8 bit image is represented by 255 shades of colours i.e. pixel values. A broad range of pixel values can hurt the training of a model and cause it to over-fit and take much longer to learn parameter values. In order to prevent unwanted interference, the image should be normalized by dividing it with 255 so that the final values lie between the range of [0-1]. Each pixel here will be given equal preference

### Activation Function

ReLU stands for rectified linear unit and it is the main activation function used in the hidden layers of our model. It is defined by the following mathematical equation.

$$\mathbf{y = max(0, x)}$$

## Learning Rate

The learning rate is scheduled with the plateau function. After a set number of epochs, if the model does not show any improvement in validation accuracy, it will divide the learning rate by a set value. There is also a lower limit set to the learning rate decliner function so that the learning rate doesn’t fall below a certain value. It may give rise to problems such as vanishing gradients and would also result in the model taking up too much time to train effectively.

## Optimizer

**The Adam optimizer** [26] is one of the optimizers experimented with. Adam stands for **Adaptive moment estimation** combines the ideas of momentum optimization and RMSProp: just like momentum optimization, it keeps track of an exponentially decaying average of past gradients; and just like RMSProp, it keeps track of an exponentially decaying average of past squared gradients.

Here, Beta1 refers to the momentum decay hyperparameter while Beta2 refers to the scaling decay hyperparameter. The epsilon parameter is the smoothing terms which is usually initialized to ($10^{-7}$).

As Adam is an adaptive learning rate algorithm like AdaGrad and RMSProp. It requires less tuning of the learning rate hyperparameter (n). The default value n=0.001 works for most cases, making it easier to use than gradient descent.

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
3. $\widehat{\mathbf{m}} \leftarrow \dfrac{\mathbf{m}}{1 - {\beta_1}^t}$
4. $\widehat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1 - {\beta_2}^t}$
5. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \, \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}} + \varepsilon}$

*The Adam Algorithm for updating parameters (Theta) [26]*

The default gradient descent algorithm which is the mini-batch gradient descent or the Stochastic Gradient descent is almost always outperformed by other gradient descent algorithms. Adam optimization has two sub-categories – Nadam and AdaMax. These are for more specific cases so we will be using the parent optimization algorithm.

Another optimizer we will be looking at is RMSProp which is another versatile gradient descent algorithm.

The **RMSProp algorithm [33]** fixes other gradient descent algorithms such as AdaGrad. RMSProp has an interesting background in that it was introduced by Geoff Hinton on Coursera first and was spread widely after. AdaGrad runs the risk of slowing down a bit too fast and never converges to the global optimum. It accumulates gradients from the most recent iterations (instead of all gradients since the beginning of training). It scales down the gradient vectors along the steepest dimension. It does so by the process of exponential decay.

1. $\mathbf{s} \leftarrow \beta \mathbf{s} + (1-\beta)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s} + \varepsilon}$

The decay rate is Beta and it is typically set to the value of 0.9 (default value). It is another hyperparameter but the default value works for most cases. This optimizer usually works better than AdaGrad. It was the most preferred algorithm for a long time until Adam optimization came along and worked better than RMSProp in some cases.

## Dropout

Dropout is an amazingly compelling regularization method presented by Srivastava et al. [27]. Dropout is a method where essentially each neuron has a probability to be put to sleep during the training step/epoch.

It may be active during the next phase of training. We add another hyperparameter p which is the dropout rate – the probability of a neuron to be put to sleep during the next epoch. Neurons trained with dropout cannot adapt with their neighboring neurons, they cannot rely excessively on just a few neurons and their weights. They end up being less sensitive to slight changes in the input.

By the end of the dropout process, the neural network is much more robust. A unique model is generated at each training step. The neural networks are not independent because they share many weights but the resulting network can be seen as an averaging ensemble of all the smaller neural network.

**Here, we are working with the dropout rate of 0.2 for all convolution layers**

## Objective Function

As brought up in [20, 7, 8], one of the most significant difficulties in auto-colorization is a cost function that represents the multi-modal idea of the issue. To explore this further, loss functions were tested.

The L1 Loss function can be represented in the following method. The parameters are the same as above. It was was established by Girshick [28].

$$\frac{1}{n}\sum_{n}\begin{pmatrix} 0.5 \times (x_i - y_i)^2 \; if \; |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5 \; otherwise \end{pmatrix}$$

This loss penalizes outliers less and it can result in more blurry images as the pixel values could be completely inaccurately predicted. This could lead to the predicted value being far away from the ground truth value and can result in wrong coloring. We observe that this function would be much more harmful than the L2 Loss function.

**The L2 Loss function**: $x_i$ represents the predicted pixel values and $y_i$ the ground truth value. N is the total number of pixels across all input images to the model.

$$\frac{1}{n}\sum_{n}(x_i - y_i)^2$$

This is our primary loss function and we would be using this as our means for the final prediction accuracy and validation loss functions. This loss is also known as the MSE (mean squared error) or the quadratic loss function. This loss metric is not robust to outliers and penalizes them hard enough to capture the finer details in the final predictions made by our auto-encoder model.

We will also experiment with the cross_entropy loss function to compare the predicted image colorization, hues and saturation levels to cross-verify and examine the best loss metric for our current task of autocolorization.

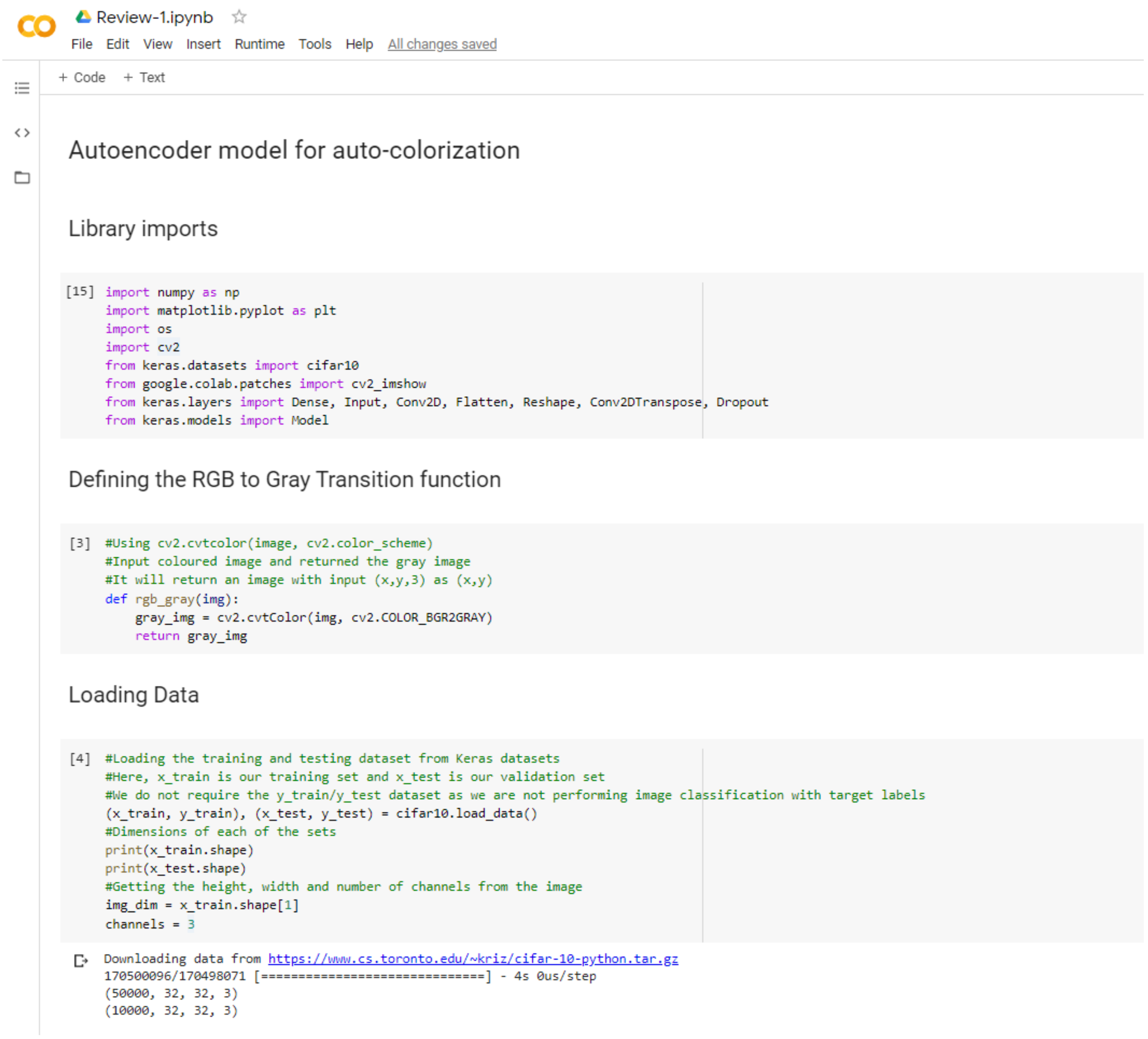
## Code implementation

### 20% code implementation:

```
#imports
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from keras.datasets import cifar10
from google.colab.patches import cv2_imshow
from keras.layers import Dense, Input, Conv2D, Flatten, Reshape,
Conv2DTranspose, Dropout
from keras.models import Model

#Using cv2.cvtcolor(image, cv2.color_scheme)
#Input coloured image and returned the gray image
#It will return an image with input (x,y,3) as (x,y)
def rgb_gray(img):
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return gray_img


#Loading the training and testing dataset from Keras datasets
#Here, x_train is our training set and x_test is our validation set
#We do not require the y_train/y_test dataset as we are not performing image
classification with target labels
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
#Dimensions of each of the sets
print(x_train.shape)
print(x_test.shape)
#Getting the height, width and number of channels from the image
img_dim = x_train.shape[1]
channels = 3

#Our lists to store the converted images
x_train_Gray = []
x_test_Gray = []
#Iterate over number of images for training set
for i in range(x_train.shape[0]):
 img = x_train[i]
 #convert and append
 x_train_Gray.append(rgb_gray(img))
print(len(x_train_Gray))

#Same process for the test images
for i in range(x_test.shape[0]):
 img = x_test[i]
 x_test_Gray.append(rgb_gray(img))
print(len(x_test_Gray))
```

```
#Convert the given list to a numpy array for input
x_train_Gray = np.asarray(x_train_Gray)
x_test_Gray = np.asarray(x_test_Gray)

#Reshape the given data into (m, height, width, channels)
#We will reshape both the training coloured and gray images - Features and
targets
#Reshaping (x,y) into (x,y,1) for our grayscale images
x_train = x_train.reshape(x_train.shape[0], img_dim, img_dim, channels)
x_test = x_test.reshape(x_test.shape[0], img_dim, img_dim, channels)
x_train_Gray = x_train_Gray.reshape(x_train_Gray.shape[0], img_dim,
img_dim, 1)
x_test_Gray = x_test_Gray.reshape(x_test_Gray.shape[0], img_dim,img_dim,1)
input_shape = (img_dim, img_dim, 1)

#Display sample images from the dataset
fig, ax = plt.subplots(5,5,sharex=True,sharey=True,figsize=(10,10))
c=0
for i in range(0,5):
    for j in range (0,5):
        ax[i,j].imshow(x_train[c])
        ax[i,j].set_title(y_train[c])
        c=c+1

#Display sample image RBG and black and white
plt.imshow(x_train[0])
cv2_imshow(x_train_Gray[0])

#Dividing each pixel by 255 to get a normalized value
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
x_train_Gray = x_train_Gray.astype('float32')/255
x_test_Gray = x_test_Gray.astype('float32')/255


#Encoder part of the auto-encoder
inputs = Input(shape=input_shape, name='encoder_input')
x = inputs
x = Conv2D(64, (3, 3), strides=2, activation='relu', padding='same')(x)
x = Dropout(0.2)(x)
x = Conv2D(128, (3, 3), strides=2, activation='relu', padding='same')(x)
x = Dropout(0.2)(x)
x = Conv2D(256, (3, 3), strides=2, activation='relu', padding='same')(x)
x = Dropout(0.2)(x)
shape = K.int_shape(x)
x = Flatten()(x)
latent = Dense(lat_dim, name='latent_vector')(x)
encoder = Model(inputs, latent, name='encoder_model')
encoder.summary()
```

## Screenshots of code execution



### Autoencoder model for auto-colorization

#### Library imports

```
[15] import numpy as np
     import matplotlib.pyplot as plt
     import os
     import cv2
     from keras.datasets import cifar10
     from google.colab.patches import cv2_imshow
     from keras.layers import Dense, Input, Conv2D, Flatten, Reshape, Conv2DTranspose, Dropout
     from keras.models import Model
```

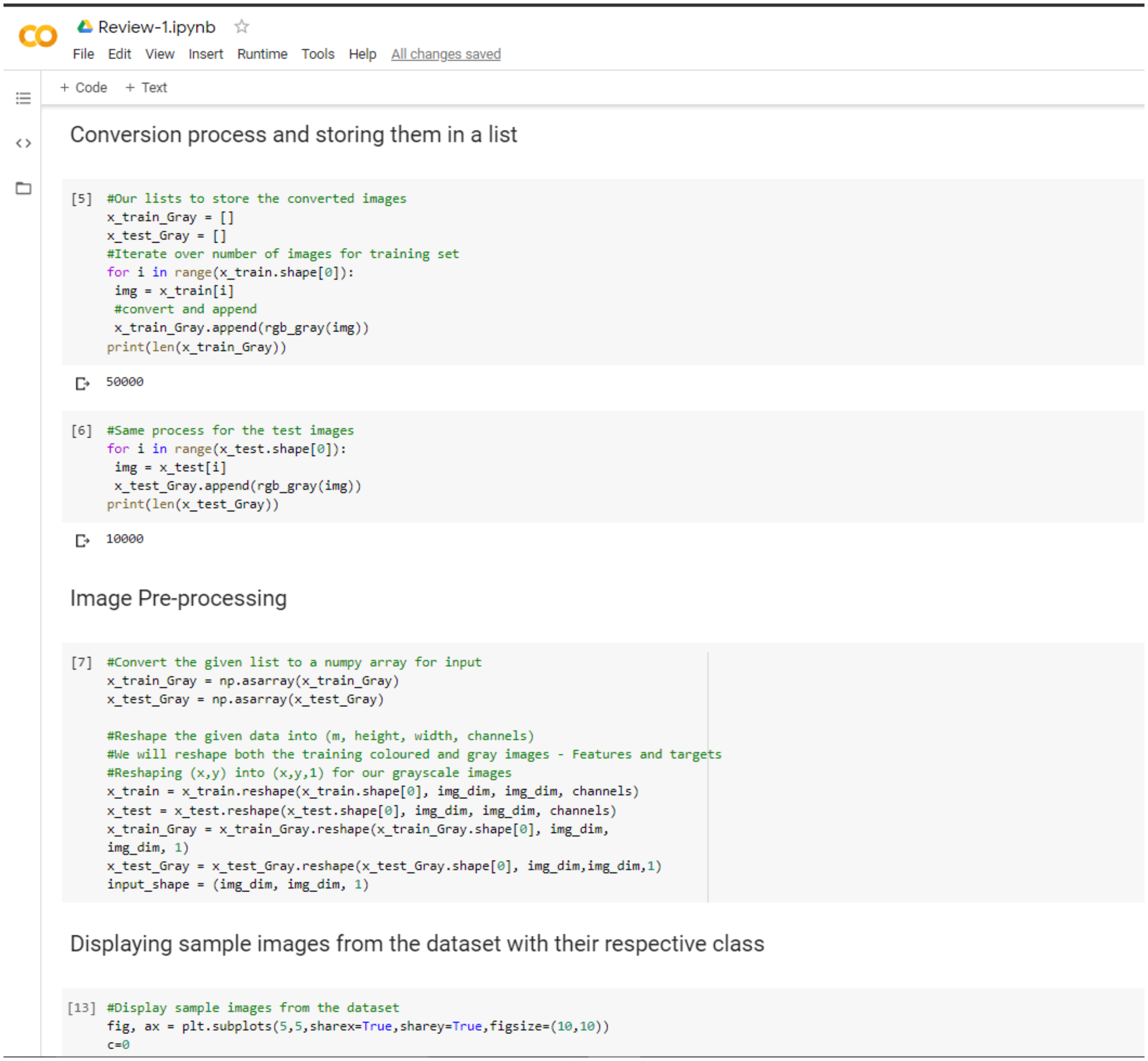
#### Defining the RGB to Gray Transition function

```
[3]  #Using cv2.cvtcolor(image, cv2.color_scheme)
     #Input coloured image and returned the gray image
     #It will return an image with input (x,y,3) as (x,y)
     def rgb_gray(img):
         gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
         return gray_img
```

#### Loading Data

```
[4]  #Loading the training and testing dataset from Keras datasets
     #Here, x_train is our training set and x_test is our validation set
     #We do not require the y_train/y_test dataset as we are not performing image classification with target labels
     (x_train, y_train), (x_test, y_test) = cifar10.load_data()
     #Dimensions of each of the sets
     print(x_train.shape)
     print(x_test.shape)
     #Getting the height, width and number of channels from the image
     img_dim = x_train.shape[1]
     channels = 3
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```



Review-1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

## Conversion process and storing them in a list

```
[5] #Our lists to store the converted images
    x_train_Gray = []
    x_test_Gray = []
    #Iterate over number of images for training set
    for i in range(x_train.shape[0]):
     img = x_train[i]
     #convert and append
     x_train_Gray.append(rgb_gray(img))
    print(len(x_train_Gray))
```

```
50000
```

```
[6] #Same process for the test images
    for i in range(x_test.shape[0]):
     img = x_test[i]
     x_test_Gray.append(rgb_gray(img))
    print(len(x_test_Gray))
```

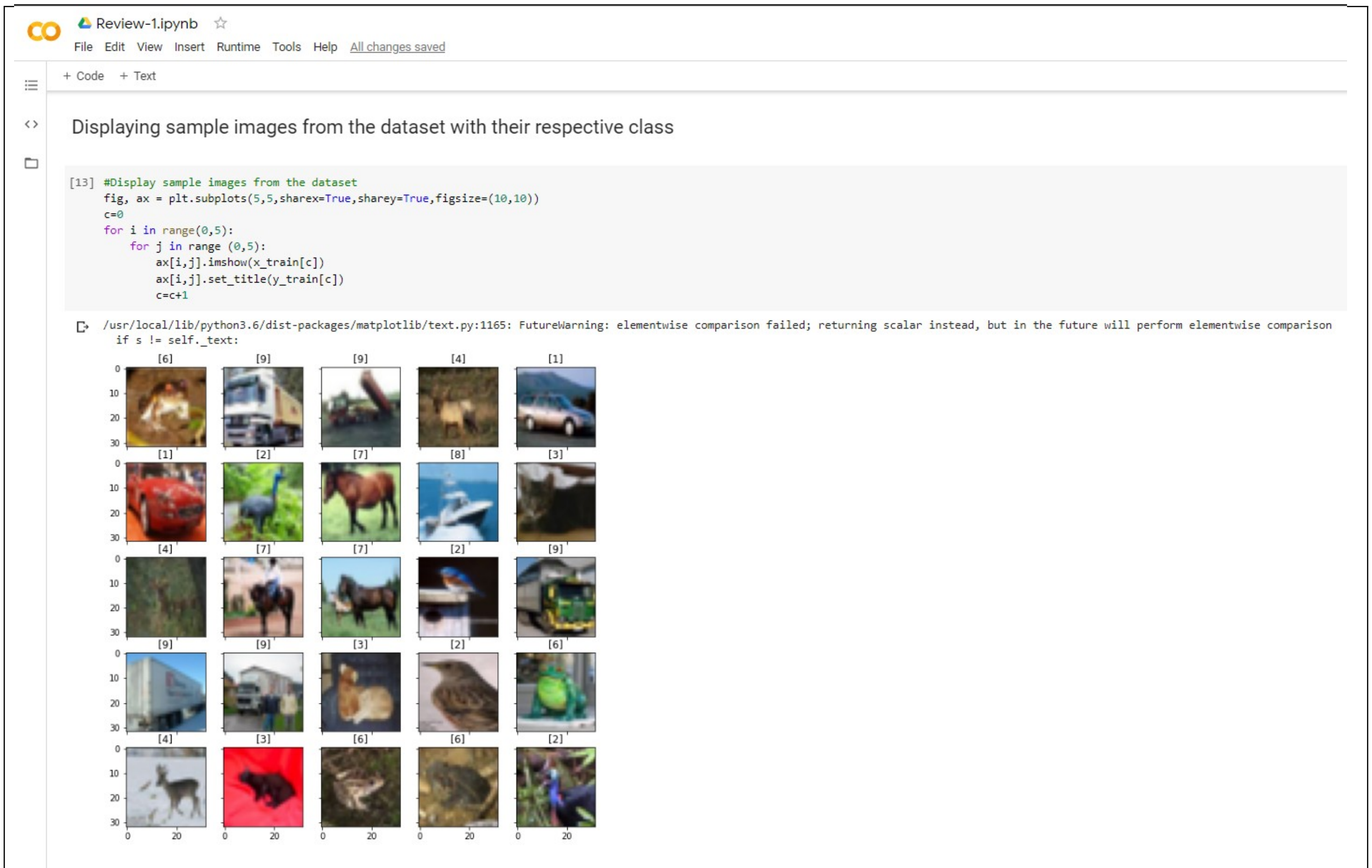
```
10000
```

## Image Pre-processing

```
[7] #Convert the given list to a numpy array for input
    x_train_Gray = np.asarray(x_train_Gray)
    x_test_Gray = np.asarray(x_test_Gray)

    #Reshape the given data into (m, height, width, channels)
    #We will reshape both the training coloured and gray images - Features and targets
    #Reshaping (x,y) into (x,y,1) for our grayscale images
    x_train = x_train.reshape(x_train.shape[0], img_dim, img_dim, channels)
    x_test = x_test.reshape(x_test.shape[0], img_dim, img_dim, channels)
    x_train_Gray = x_train_Gray.reshape(x_train_Gray.shape[0], img_dim,
    img_dim, 1)
    x_test_Gray = x_test_Gray.reshape(x_test_Gray.shape[0], img_dim,img_dim,1)
    input_shape = (img_dim, img_dim, 1)
```

## Displaying sample images from the dataset with their respective class

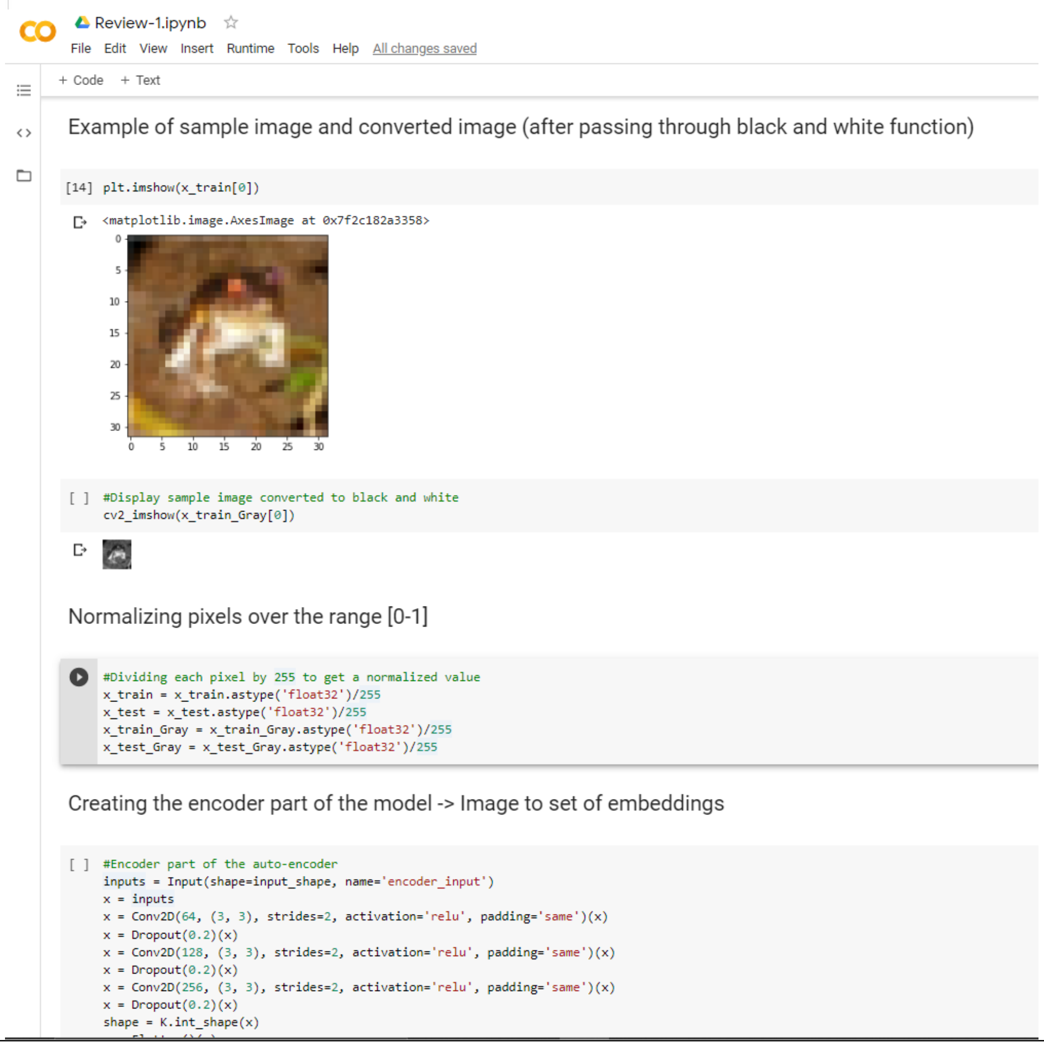
```
[13] #Display sample images from the dataset
     fig, ax = plt.subplots(5,5,sharex=True,sharey=True,figsize=(10,10))
     c=0
```



Displaying sample images from the dataset with their respective class

```
[13] #Display sample images from the dataset
     fig, ax = plt.subplots(5,5,sharex=True,sharey=True,figsize=(10,10))
     c=0
     for i in range(0,5):
         for j in range (0,5):
             ax[i,j].imshow(x_train[c])
             ax[i,j].set_title(y_train[c])
             c=c+1
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  if s != self._text:
```



Example of sample image and converted image (after passing through black and white function)

```
[14] plt.imshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f2c182a3358>
```

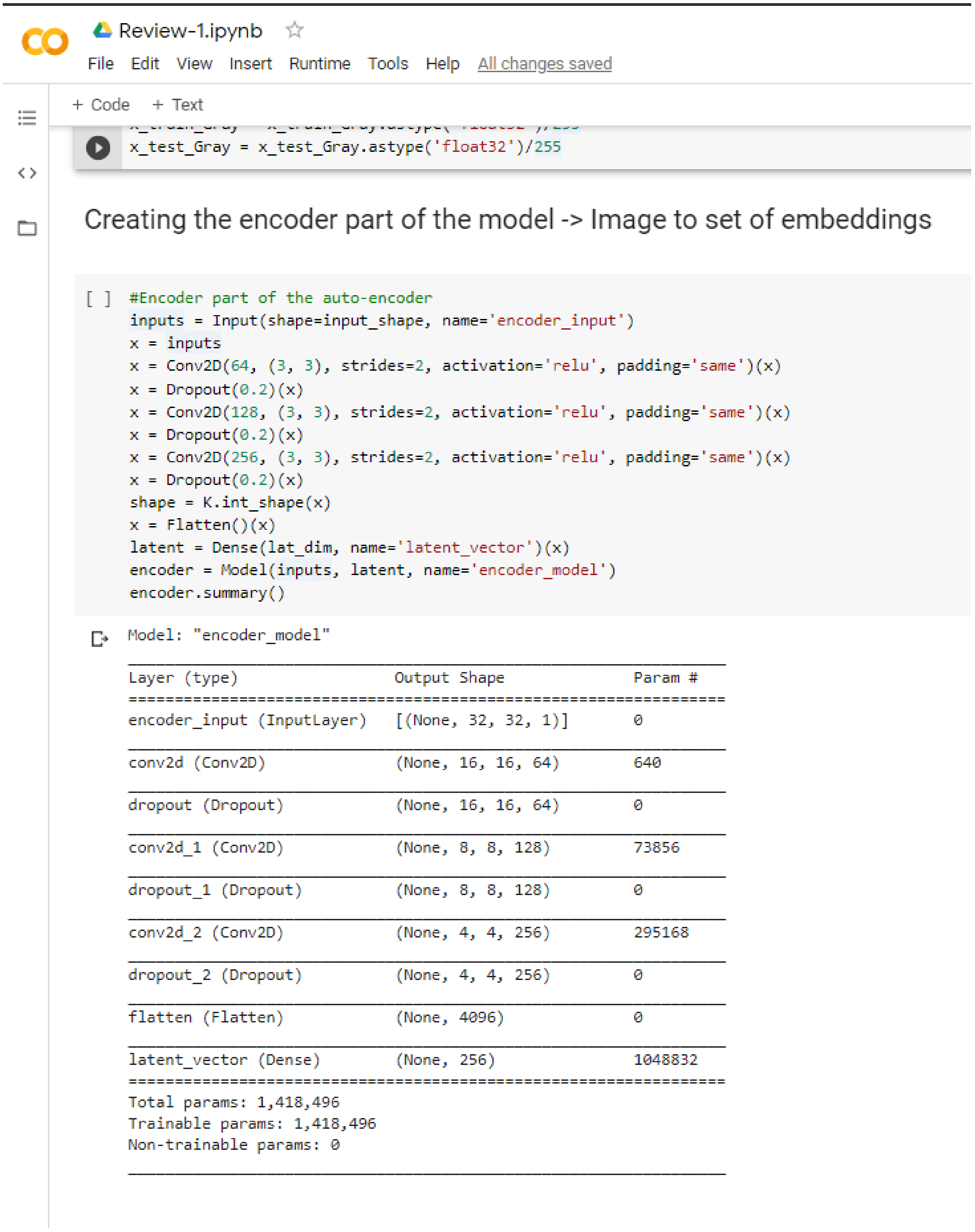
```
[ ] #Display sample image converted to black and white
    cv2_imshow(x_train_Gray[0])
```

Normalizing pixels over the range [0-1]

```
#Dividing each pixel by 255 to get a normalized value
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
x_train_Gray = x_train_Gray.astype('float32')/255
x_test_Gray = x_test_Gray.astype('float32')/255
```

Creating the encoder part of the model -> Image to set of embeddings

```
[ ] #Encoder part of the auto-encoder
    inputs = Input(shape=input_shape, name='encoder_input')
    x = inputs
    x = Conv2D(64, (3, 3), strides=2, activation='relu', padding='same')(x)
    x = Dropout(0.2)(x)
    x = Conv2D(128, (3, 3), strides=2, activation='relu', padding='same')(x)
    x = Dropout(0.2)(x)
    x = Conv2D(256, (3, 3), strides=2, activation='relu', padding='same')(x)
    x = Dropout(0.2)(x)
    shape = K.int_shape(x)
```


Review-1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
x_test_Gray = x_test_Gray.astype('float32')/255
```

## Creating the encoder part of the model -> Image to set of embeddings

```
#Encoder part of the auto-encoder
inputs = Input(shape=input_shape, name='encoder_input')
x = inputs
x = Conv2D(64, (3, 3), strides=2, activation='relu', padding='same')(x)
x = Dropout(0.2)(x)
x = Conv2D(128, (3, 3), strides=2, activation='relu', padding='same')(x)
x = Dropout(0.2)(x)
x = Conv2D(256, (3, 3), strides=2, activation='relu', padding='same')(x)
x = Dropout(0.2)(x)
shape = K.int_shape(x)
x = Flatten()(x)
latent = Dense(lat_dim, name='latent_vector')(x)
encoder = Model(inputs, latent, name='encoder_model')
encoder.summary()
```

```
Model: "encoder_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   [(None, 32, 32, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 16, 16, 64)        640
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
latent_vector (Dense)        (None, 256)               1048832
=================================================================
Total params: 1,418,496
Trainable params: 1,418,496
Non-trainable params: 0
_________________________________________________________________
```

## References

[1] Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." arXiv preprint arXiv:1703.10593 (2017).

[2] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." arXiv preprint arXiv:1611.07004 (2016).

[3] Reed, Scott, et al. "Generative adversarial text to image synthesis." 33rd International Conference on Machine Learning. Vol. 3. 2016.

[4]Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).

[5] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014).

[6] Hariharan, Bharath, et al. "Hypercolumns for object segmentation and fine-grained localization." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015

[7] Iizuka, Satoshi, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification." ACM Transactions on Graphics (TOG) 35.4 (2016): 110.

[8] Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Learning representations for automatic colorization." European Conference on Computer Vision. Springer International Publishing, 2016.

[9] Charpiat, Guillaume, Matthias Hofmann, and Bernhard Schölkopf. "Automatic image colorization via multimodal predictions." Computer Vision–ECCV 2008 (2008): 126-139.

[10] Deshpande, Aditya, Jason Rock, and David Forsyth. "Learning large-scale automatic image colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[11] Welsh, Tomihisa, Michael Ashikhmin, and Klaus Mueller. "Transferring color to greyscale images." ACM Transactions on Graphics (TOG). Vol. 21. No. 3. ACM, 2002.

[12] Qu, Yingge, Tien-Tsin Wong, and Pheng-Ann Heng. "Manga colorization." ACM Transactions on Graphics (TOG). Vol. 25. No. 3. ACM, 2006.

[13] Huang, Yi-Chin, et al. "An adaptive edge detection based colorization algorithm and its applications." Proceedings of the 13th annual ACM international conference on Multimedia. ACM, 2005.

[14] Levin, Anat, Dani Lischinski, and Yair Weiss. "Colorization using optimization." ACM Transactions on Graphics (ToG). Vol. 23. No. 3. ACM, 2004.

[15] Li, Jiwei, et al. "Adversarial learning for neural dialogue generation." arXiv preprint arXiv:1701.06547 (2017).

[16] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.

[17] Dahl, Ryan. "Automatic colorization." (2016).

[18] Cheng, Zezhou, Qingxiong Yang, and Bin Sheng. "Deep colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[19] Liang, Xiangguo, et al. "Deep patch-wise colorization model for grayscale images." SIGGRAPH ASIA 2016 Technical Briefs. ACM, 2016.

[20] Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." European Conference on Computer Vision. Springer International Publishing, 2016.

[21] Deep Colorization Zezhou Cheng, Student Member, IEEE, Qingxiong Yang, Member, IEEE, Bin Sheng, Member, IEEE, arXiv:1605.00075v1

[22] Interactive Anime Sketch Colorization with Style Consistency via aDeep Residual Neural Network Ru-Ting Ye,Wei-Li Wang,Ju-Chin Chen Kawuu W. Lin

[23] Learning-Based Colorization of Grayscale Aerial Images Using Random Forest Regression Dae Kyo Seo, Yong Hyun Kim , Yang Dam Eo and Wan Yong Park

[24] Automatic Image Colorization Via Multimodal Predictions Guillaume Charpiat, Matthias Hofmann, and Bernhard Sch¨olkopf

[25] Semi-Auto Sketch Colorization Based on Conditional Generative Adversarial Networks - Zijuan Cheng Fang Meng Jingbo Mao

[26] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[27] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1

[28] Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[29] M. M. Hadhoud, N. A. Semary and A. M. Abbas, "Fully automated black and white movies colorization system

[30]Y. Xiao, A. Jiang, C. Liu and M. Wang, "Single Image Colorization Via Modified Cyclegan," 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 2019

[31] https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

[32] H. Brendan Mcmahan and Matthew Streeter. Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning. Advances in Neural Information Processing Systems (Proceedings of NIPS), pages 1–9, 2014.

[33] Larsson, Gustav & Maire, Michael & Shakhnarovich, Gregory. (2017). Colorization as a Proxy Task for Visual Understanding.