B.Tech. Project Report

IT-414

on

Image Colorization and Enhancement

BY

LAKSHAY HURRIA (1140444)

SATYENDER KUMAR (1140887)

MUJAHID OMER (2140003)

Under the Supervision of

Miss Mamtesh, Assistant Professor

DEPARTMENT OF COMPUTER ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
KURUKSHETRA – 136119, HARYANA (INDIA)

Dec 2017 – April 2018

CERTIFICATE

I hereby certify that the work which is being presented in this B. Tech. Major Project (IT-414) report entitled **"Image Colorization and Enhancement",** in partial fulfilment of the requirements for the award of the **Bachelor of Technology in Information Technology** is an authentic record of my own work carried out during a period from December 2017 to April 2018 under the supervision of **Miss Mamtesh**, Assistant Professor Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

Signature of Candidate

LAKSHAY HURRIA (1140444)

SATYENDER KUMAR (1140887)

MUJAHID OMER (2140003)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Signature of Supervisor

Miss Mamtesh

Assistant Professor

TABLE OF CONTENTS

ABSTRACT

Colorization is the process of coloring monochrome images. This is used in various fields like processing of images and scientific works. Traditionally, colorization process was a tedious and take lot of time and for adding colors to image required artistic skills. Previously, editing software and tools such as Photoshop and such were used to add colours to black and white pictures. The task in image colourization is in specifying which parts of the image should be colourized by what colours. There are many modern deep learning techniques which can colorize photos. One such technique is CNN (Convolutional Neural Network), which this project is based on. Since training data is easy to get, this problem is easily automated. The coloured images can be de-saturated, which is then used to train data. In this project, the data of Pokémon is used to work with and to colour black and white Pokémon images. For this, a video is used as training data after extracting frames from it one by one.

The colorization of image automatically can be of making image as colours full without any user intervention using training dataset. The above thing, as mentioned, is going above head, that one cannot automatic give the colours to a black and white image without any prior knowledge regarding colours. It may be that many things can have different colours, not only artificial objects but like tree leaves in different season can have different nuisances of green color and in autumn it will be brown, with the constant shape of leaves.

I. INTRODUCTION

The colorization of grayscale images automatically is very keen topic of research within the different communities of machine learning and computers. Beyond taking consideration from an artificial intelligence perspective, such concept has broad practical applications ranging from video re-establishment to enhancement of image for better accountably. Here approach with learning dataset taken for this. For this designed a convolutional neural network (CNN) which takes a black and white image as an input and give a colorized image as output. This colorizing process of black and white images does assigning of 3D (RGB) pixel weights to input which range along only 1D (brightness or intensity). There are various colors which may have same brightness value but can be different in intensity. Due to these redundancy, a significant role is played by human interaction in colorization process.

The CNN makes use of back-propagation. The training of dataset comprises of feed forward, loss function, learning parameters and back pass. In this way with the help of batch the whole dataset is trained. Image enhancement is also a wide topic where various enhancement techniques like brightness enhancement, sharpness, contrast enhancement, histogram equalization. So these techniques can be used to get the enhanced image of colored image.

II. LITERATURE SURVEY

A few related works in this field are following

II.1 Scribble Based Colorization

Levin et al. [3] introduced a related colorization technique which can be applied on still images and sequences of images. The user provides the training data sets using the color scribbles in the image. The information of scribble is propagated to remaining pixels of the target image. This algorithm was later improved by Huang et al. for reducing the color blending on image edges. Yatziv et al. [5] also gave a method which concatenate the info of colors of multiple scribble to determine the color of a pixel. The main disadvantage of scribble based colorization is that the performance is heavily dependent on human intervention.

II.2 Example Based Colorization

Reinhard et al gave a color transform algorithm which is based on statistical analysis to impose one image's color characteristics on the other image. In the same manner, Welsh et al [4] applied intensity of pixels and neighborhood statistics to find similar pixels in a reference image and then transfer the color info to a matched pixels of target image. Irony et al [6] first determined each pixel of the target image with example segment that it should learn its colors from, the authors treated them as user scribble input for colorization. Example based colorization works in a good manner only if it is able to find appropriate color image that is given as one of the inputs to algo.

II.3 Learning Based Colorization

Before CNN approach Bugeau and Ta [8] gave a method which was patch based image colorization which have input of square patches around each pixels. Cheng et al. [7] gave a method of colorization of image automatically based on three layer deep neural network. CNN based approach was suggested by Ryan Dahl [9]. He used it as a feature extractor of an image with the help of pre-trained model. In machine learning, a large set of guideline to solve the issue should be avoided, rather than this a model should provide to system with which it can evaluate examples and instructions should be there to make change in the model whenever it gives a wrong result. For example: Think about kids, they may not able to recognise things at their childhood. So parents used to correct them whenever they made a mistake. This is how things are reinforced and things are recognize correctly.

The major disadvantage of Scribble based approach is that it requires a lot of human intervention and the Example based approach suits where we are able to get the perfect image which is input to the algorithm.

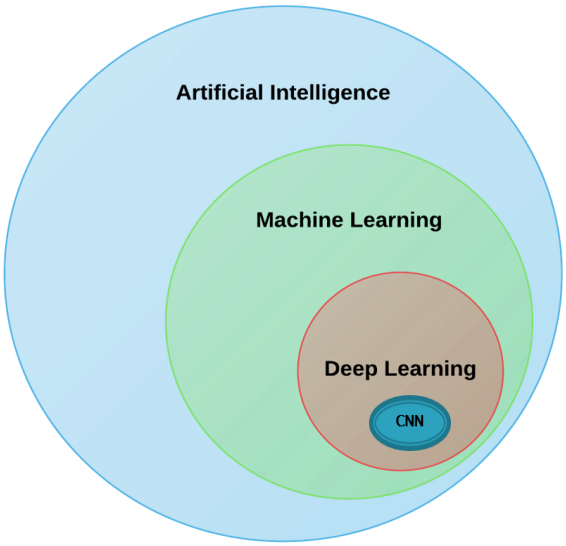

Fig 1: CNN as a subset of Deep Learning

The training makes use of back propagation neural network. It consists of 18 hidden layers along with Adam optimizer which has a learning rate of 10^4. **Rectified Linear Unit (ReLU)** activation function is used to extract the features in the input images to train the convolution layers and the data is stored in checkpoint models. The difference between the actual input image and the output of the CNN layers is marked and then the error is back propagated.

Backpropagation consist of 4 different sections: the forward pass, the loss function, the backward pass, and the weight update.

Forward Pass

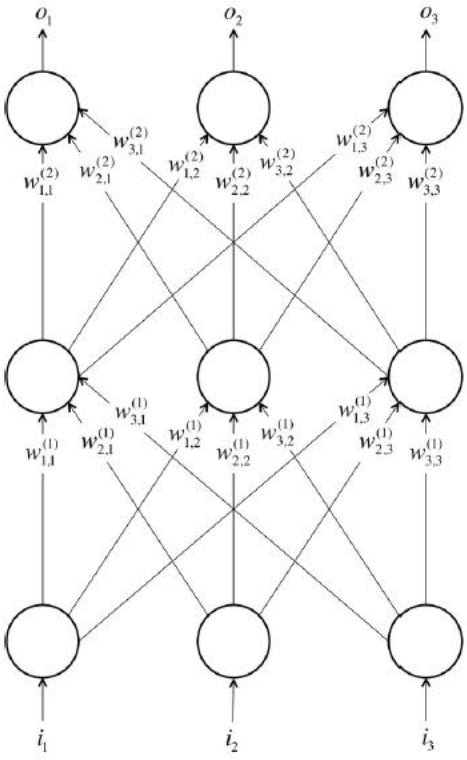

Fig 2: Forward Pass

In this neural network, there exists no link between Neurons at same level and Neurons that transmit data from higher level to lower-level. Hence, called **feed-forward** neural networks.

i) **Bottom Layer-** It pulls in the input data.
ii) **Middle Layer-** Also called the hidden layer where most of the magic or complex computations takes place.
iii) **Output Layer**-It gives the output in vector-form.

Loss Function

Mathematical Model: Any continuous target function ***(x)*** can be approximated by SFLNs. Also it can be stated that small positive value ***y*** is given, for SLFNs with enough number of hidden nodes ***(L)*** we have

$$||f_L(x) - f(x)|| < y$$

Backward Pass & Weight Update

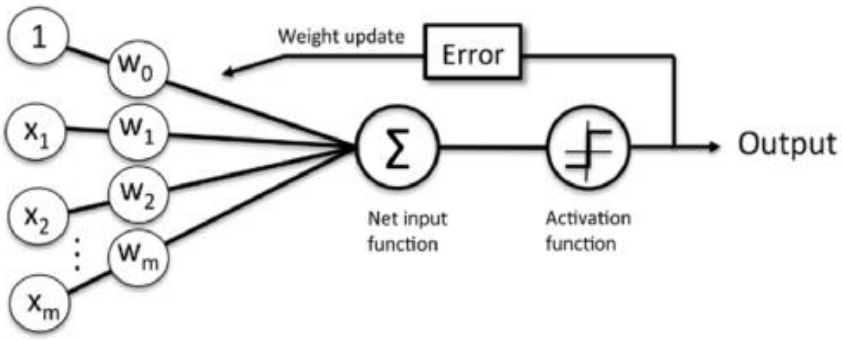

Fig 3: Backpropagation

The curve is generated by approximating the input weight and biases and back propagating the error. Once the training is complete, the same curve stored as checkpoints in the models is used to determine the color of the greyscale image.

Variety of activation functions

Sigmoid function	$G(\mathbf{a}, b, \mathbf{x}) = \frac{1}{1+\exp(-(\mathbf{a}\cdot\mathbf{x}+b))}$
Hyperbolic tangent function	$G(\mathbf{a}, b, \mathbf{x}) = \frac{1-\exp(-(\mathbf{a}\cdot\mathbf{x}+b))}{1+\exp(-(\mathbf{a}\cdot x+b))}$
Gaussian function	$G(\mathbf{a}, b, \mathbf{x}) = \exp(-b\|\mathbf{x}-\mathbf{a}\|)$
Multiquadric function	$G(\mathbf{a}, b, \mathbf{x}) = (\|\mathbf{x}-\mathbf{a}\| + b^2)^{1/2}$
Hard limit function	$G(\mathbf{a}, b, \mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{a}\cdot\mathbf{x}+b \le 0 \\ 0, & \text{otherwise} \end{cases}$
Cosine function/Fourier basis	$G(\mathbf{a}, b, \mathbf{x}) = \cos(\mathbf{a}\cdot\mathbf{x}+b)$

The activation function used in this project is Rectified Linear Unit (ReLU) activation function. It has become famous in past few years. It has 5 times improvement in convergence from tanh function.

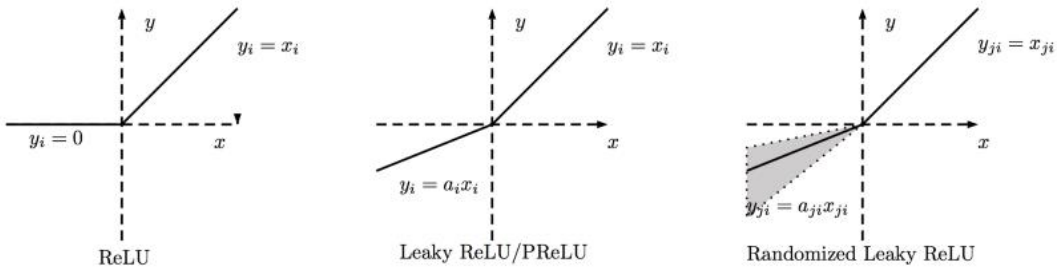

Fig 4: ReLU Types

II.4 Enhancement

Histogram Equalization

This method suggested by Y. Wang et al. [10]. The global contrast of many images can be increased with this technique especially when the adjacent pixels have low contrast differences. The intensities can be distributed using this technique. By this, lower contrast areas can gain higher contrast. The most frequent intensity values are spread on the image.

Implementation

A discrete greyscale image ***{x}*** is considered and n_i be the no. of occurrences of gray level ***i***. The occurrence of a pixel of level ***i*** probability is:

$$p_x(i) = p(x = i) = \frac{n_i}{n}, \quad 0 \leq i < L$$

L is image's total gray levels (typically 256)

n is the total number of pixels

$p_x(i)$ is the image's histogram for pixel value ***i***

Cumulative Distribution function:

$$cdf_x(i) = \sum_{j=0}^{i} p_x(j)$$

Now a transformation of the form $y'=T(x)$ is introduced to give a new image $\{y'\}$with a flat histogram.

$$cdf_y(y') = cdf_y(T(k)) = cdf_x(k)$$

$$T_k = \frac{L-1}{MN} \sum_{j=0}^{k} n_j \quad \text{Where,}$$

M : number of rows

N : number of columns

K are all the available pixel values

Histogram Equalization for Colored Images

It is a non-linear process. Channel splitting and equalizing each channel separately is incorrect. It doesn't involve color components instead the intensity values of the image, not the. So histogram equalization cannot be applied directly on the channels for a RGB color image. It needs to be applied in such a way without disturbing the color balance of the image while the intensity values are equalized. So, the first step is to convert one of the color spaces that separates intensity values from color components from the color space of the RGB image. The possible options are HSV/HLS, YUV etc.

Other Enhancements

There are many other enhancement techniques and models available. This projects encompasses the main and useful techniques like brightness, sharpness and color enhancement techniques.

III. CONTRIBUTION TECHNICAL

This project works on coloring black and white images using Convolutional Neural Networks.

Training Data

Firstly, the frames are extracted from a Pokémon video to get the training data set. After that, the images are resized, cropped and normalized to make them suitable for training and to be passed as input to Convolutional Neural Networks. For this purpose, a black and white image is taken initially and passed to CNN, which then colorize it. Then, the actual colored image is compared with the output image of CNN. The difference between actual image and output of CNN layers is marked and the error is back-propagated. In this way, the training is performed to get the best results.

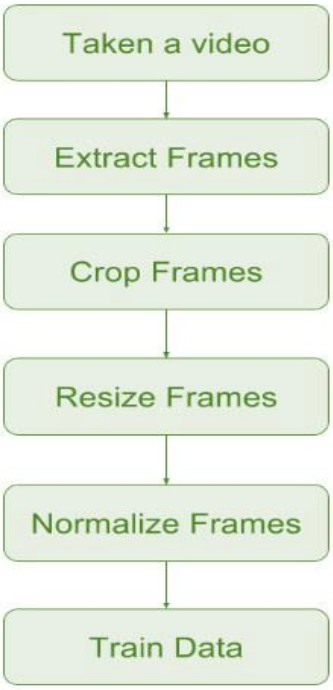

Fig 5: Training of Data

Colorizing Black and White Image

A black and white image is given as an input to the system which is then cropped, resized and normalized to get it into the standard size suitable for training. The image is then passed to 18 Convolutional Neural Network layers to get the colorized image as the output.

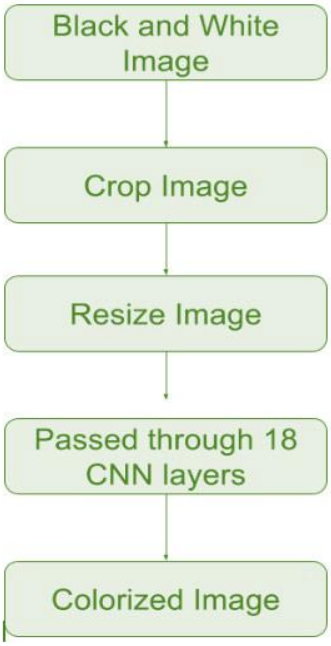

Fig 6: Colorizing a Black and White Image

Software/Packages/Libraries Used

The following libraries and softwares are used throughout the project

1. Python Libraries

i) ***NumPy*** used to add support for large, multi-dimensional arrays and matrices, along with a large collection of high level mathematical functions to operate on these arrays.

ii) ***TensorFlow*** used for machine learning applications such as neural networks.

iii) ***ImageMagick*** used to convert an image from colored to grayscale.

iv) ***OpenCV*** used to read, write and convert an input image from one color space to another.

2. Anaconda (Python distribution)

Anaconda is a freemium open source distribution of the Python programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

Enhancement Technique: Histogram Equalization

In this method of enhancement colored image is used as input. OpenCV and NumPy need to import for this method. OpenCV have a method equalizeHist() which can be applied on grayscale image directly but here for each pixel there is 3 channels (RGB) value and can't apply this method on three channel in separate manner. So RGB image need to convert into YUV channel space using cvtColor(). Now equalize the Y channel and apply this method on Y channel using equalizeHist() and then convert it to RGB to get desired result.

IV. DATA FLOW DIAGRAMS

IV.1 Level 0 DFD

Fig 7: Level 0 DFD

IV.2 Level 1 DFD

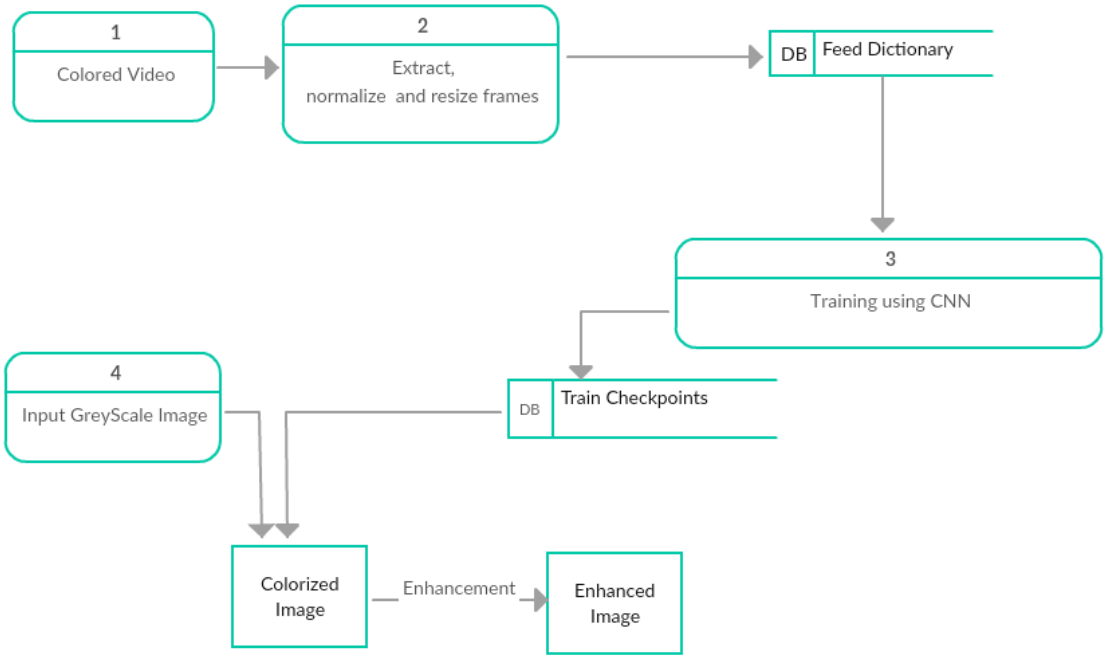

Fig 8: Level 1 DFD

IV.3 Level 2 DFD

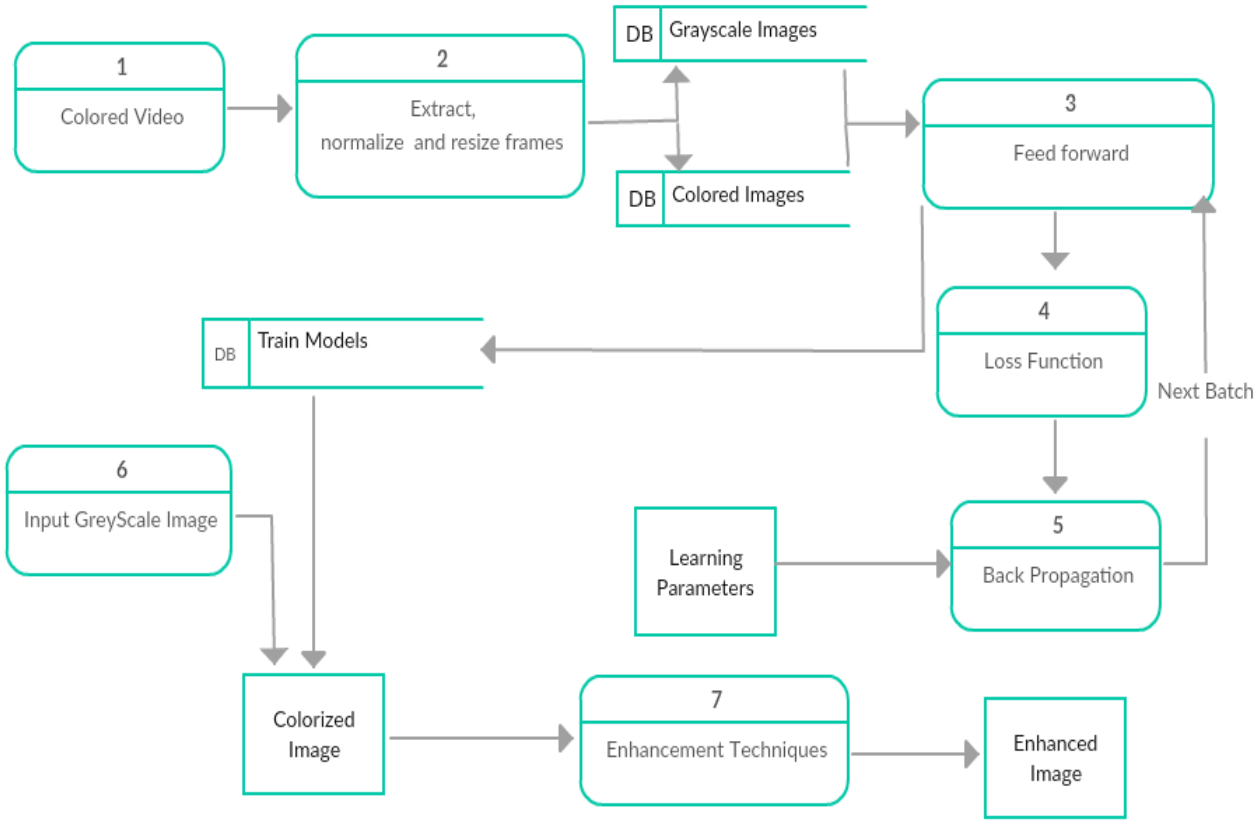

Fig 9: Level 2 DFD

V. RESULTS/ OBSERVATIONS

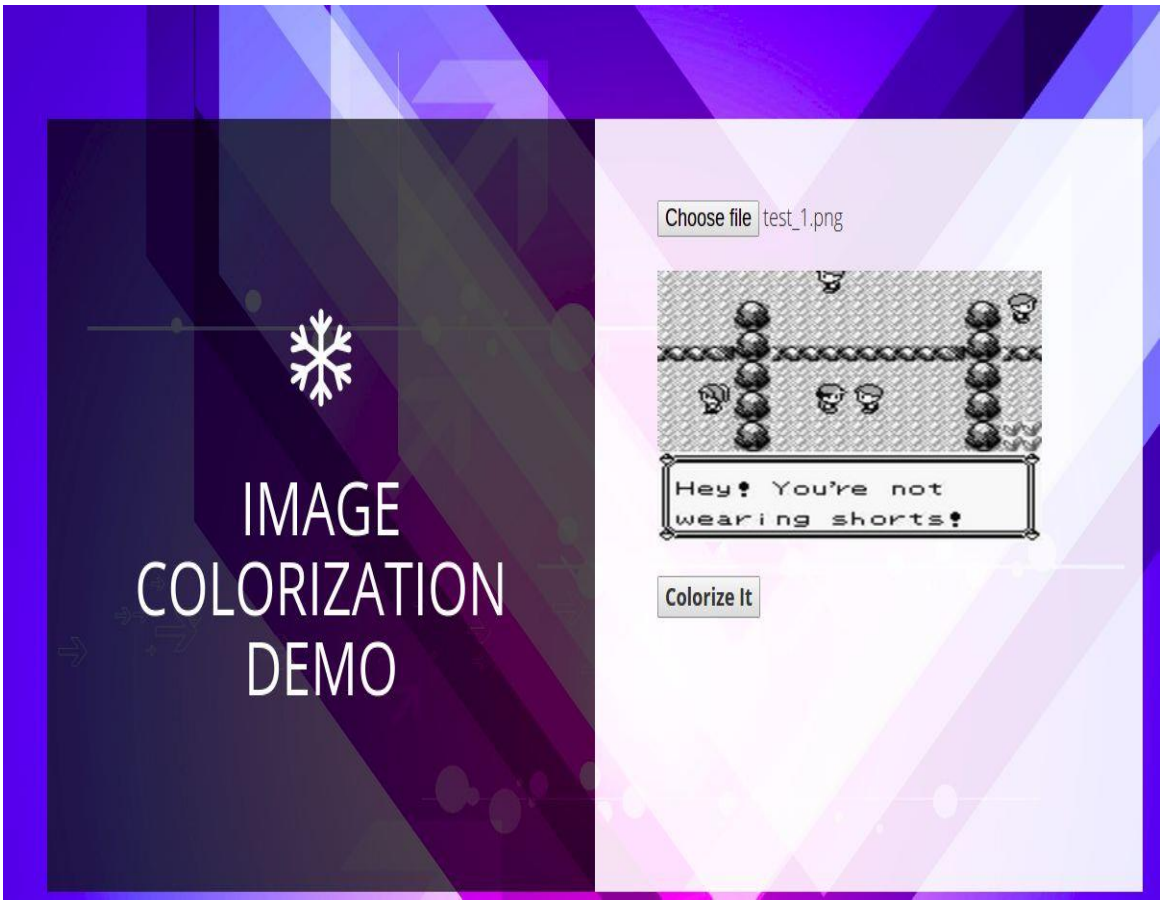

Fig 10: UI and Frontend

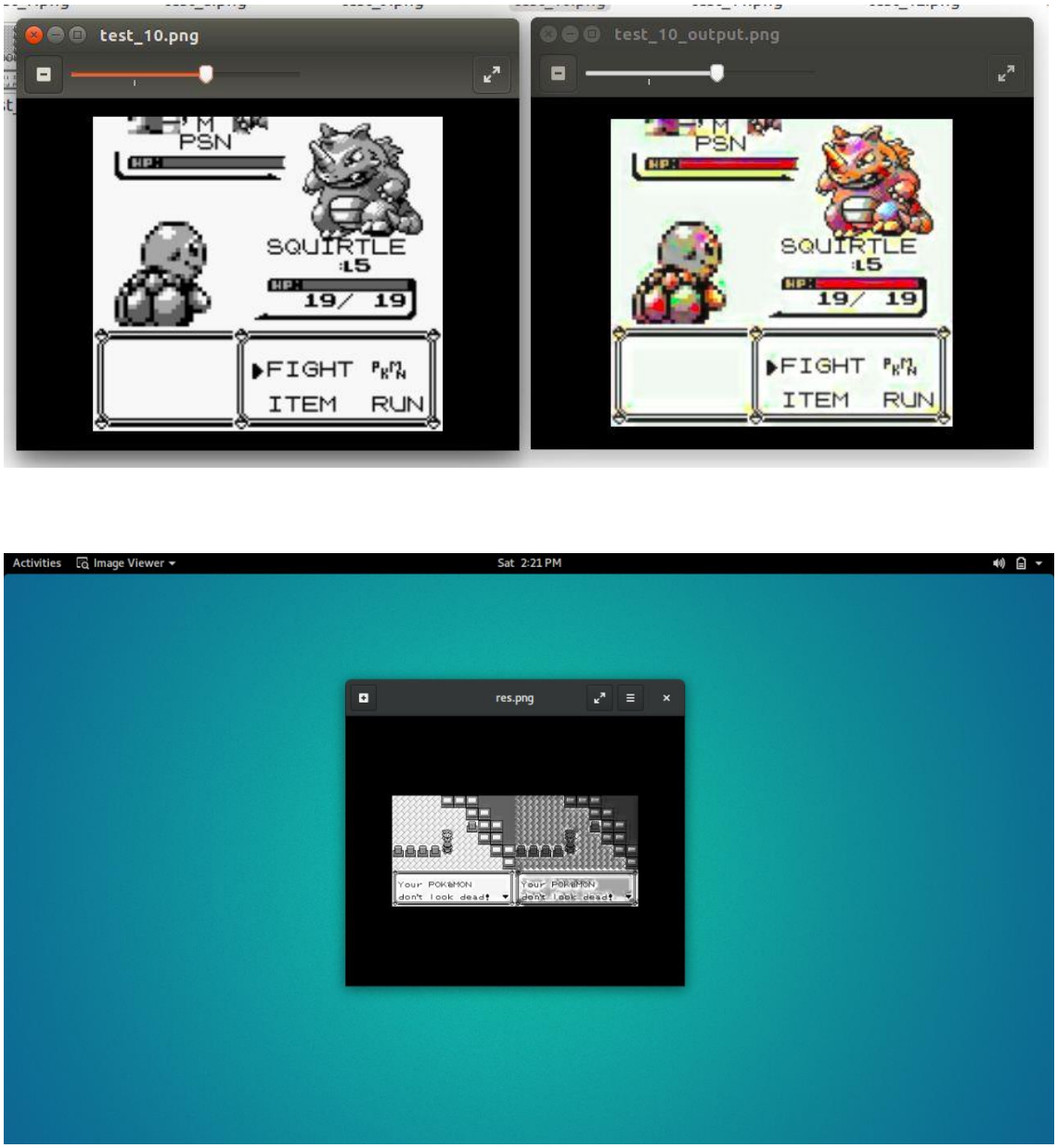

Fig 11: Results of Our Application

VI. CONCLUSION AND FUTURE PLAN

The project aims to colorize black and white images of Pokémon's with CNN (Convolutional Neural Networks). The dataset used for our project is of Pokémon's video frames. The approach used is easy and efficient as no extraction of features is being required. Directly the input is given and it is passed to many convolutional layers and we get the colorized output. The accuracy is good in case of Pokémon's images as here only Pokémon's images taken into consideration for training.

There are different methods like deep belief network and deep Boltzmann machine but the training with these methods are slow. So, the training part is done with the help of CNN and after that the colorized images are enhanced with the help of histogram equalization method. Also other methods of enhancement can be used like brightness, neural enhancement, sharpness, inverse of image etc. The future plan is to optimize the enhancement techniques.

REFERENCES

[1]Anat Levin ,Dani Lischinski, Yair Weiss, "Colorization using Optimization", School of Computer Science and Engineering ,The Hebrew University of Jerusalem, IEEE, 731-737, 2001

[2]Qing Luan, Fang Wen, Daniel Cohen,Lin Liang, Ying-Qing Xu, Heung-Yeung Shum, "Natural Image Colorization", University of Science and Technology of China, IEEE, 123-129, 2007

[3] A. Levin, D. Lischinski and Y. Weiss., "Colorization using optimization" ACM transaction on graphics, 689-694, 2004.

[4]Yatziv, L., Sapiro, G., "Fast image and video colorization using chrominance blending*"* IEEE Transactions on Image Processing, 1120–1129, 2016.

[5] T. Welsh, M. Ashikhmin, and K. Mueller, "Transferring color to grayscale images", ACM transaction on graphics, 277-280, 2002.

[6]Domonkos Varga, Tamas Sziranyi, "Fully automatic image colorization based on Convolutional Neural Networks", International Conference on Pattern Recognition (ICPR), 3691-3696, Dec. 2016

[7]Zezhou Cheng, Qingxiong Yang, and Bin Sheng, "Deep colorization", Proceedings of the IEEE International Conference on Computer Vision, 415–423, April 2015.

[8]A. Bugeau et al., "Patch-based image colorization", Proceedings of the IEEE International Conference on Pattern Recognition, 3058–3061, 2012.

[9]Ryan Dahl. http://tinyclouds.org/colorize/

[10]Y. Wang, Q. Chen, B. Zhang, "Image enhancement based on equal area dualistic sub-image histogram equalization method", IEEE Transactions on consumer electronics, Vol. 45, no. 1, 68-75, February 1999.

APPENDIX

COMPLETE SOURCE CODE

server.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

import os

from colorize import eval_one


@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        file = request.files['file']
        file_name = file.filename
        ROOT_PATH =
os.path.normpath(os.path.join(os.path.abspath(__file__), os.pardir))
        model_path = ROOT_PATH + '/models/generation_2/'
        image_path = ROOT_PATH + '/images/testing/' + file.filename
        print image_path
        eval_one.eval(model_path, image_path)
        return render_template('output.html', image=file_name)
    return render_template('index.html')


# server running on port 3000
app.run(port=5000, debug=True)
```

architecture.py

```
import tensorflow as tf
import numpy as np
import sys

FLAGS = tf.app.flags.FLAGS

num_epochs = 100

tf.app.flags.DEFINE_float('weight_decay', 0.0005,""" """)
tf.app.flags.DEFINE_float('alpha', 0.1,"""Leaky RELU param""")

def _variable_on_cpu(name, shape, initializer):
   with tf.device('/cpu:0'):

      var = tf.get_variable(name, shape, initializer=initializer)
   return var


def _variable_with_weight_decay(name, shape, stddev, wd):
  var = _variable_on_cpu(name, shape,

tf.truncated_normal_initializer(stddev=stddev))
  if wd:

    weight_decay = tf.multiply(tf.nn.l2_loss(var), wd,
name='weight_loss')
    weight_decay.set_shape([])
    w=weight_decay
    tf.add_to_collection('losses', weight_decay)

  #print 'find'
  #print var
  #print 'inhere'
  return var
```

```
def _conv_layer(inputs, kernel_size, stride, num_features, idx):
   with tf.variable_scope('{0}_conv'.format(idx)) as scope:
      print (scope)
      print 'hi'
      input_channels = inputs.get_shape()[3]
      #print (input_channels)
      weights = _variable_with_weight_decay('weights',
shape=[kernel_size, kernel_size, input_channels, num_features],
stddev=0.1, wd=FLAGS.weight_decay)
      #print weights.get_shape()
      biases = _variable_on_cpu('biases', [num_features],
tf.constant_initializer(0.1))
      #print biases.get_shape()

      conv = tf.nn.conv2d(inputs, weights, strides=[1, stride, stride,
1], padding='SAME')
      conv_biased = tf.nn.bias_add(conv, biases)

      #Leaky ReLU
      conv_rect = tf.maximum(FLAGS.alpha*conv_biased, conv_biased,
name='{0}_conv'.format(idx))
      #print 'hola'
      #print conv_rect
      #print 'redi'
      return conv_rect



def inference(images, name):
   print '\n \n'
   print images
   print
   conv1 = _conv_layer(images, 3, 1, 32, 1)
   conv2 = _conv_layer(conv1, 3, 1, 32, 2)
   conv3 = _conv_layer(conv2, 3, 1, 64, 3)
```

```
    #print conv3
    conv4 = _conv_layer(conv3, 3, 1, 64, 4)
    #print conv4
    conv5 = _conv_layer(conv4, 3, 1, 128, 5)
    #print conv5
    conv6 = _conv_layer(conv5, 3, 1, 128, 6)
    #print conv6
    conv7 = _conv_layer(conv6, 3, 1, 256, 7)
    #print conv7
    conv8 = _conv_layer(conv7, 3, 1, 256, 8)
    #print conv8
    conv9 = _conv_layer(conv8, 3, 1, 128, 9)
    #print conv9
    conv10 = _conv_layer(conv9, 3, 1, 128, 10)
    #print conv10
    conv11 = _conv_layer(conv10, 1, 1, 64, 11)
    #print conv11
    conv12 = _conv_layer(conv11, 1, 1, 64, 12)
    #print conv12
    conv13 = _conv_layer(conv12, 1, 1, 32, 13)
    #print conv13
    conv14 = _conv_layer(conv13, 1, 1, 32, 14)
    #print conv14
    conv15 = _conv_layer(conv14, 1, 1, 16, 15)
    #print conv15
    conv16 = _conv_layer(conv15, 1, 1, 16, 16)
    #print conv16
    conv17 = _conv_layer(conv16, 1, 1, 8, 17)
    #print conv17
    conv18 = _conv_layer(conv17, 1, 1, 3, 18)
    #print conv18
    return conv18


def loss (input_images, predicted_images):
    error = tf.nn.l2_loss(input_images - predicted_images)
    return error
```

eval_one.py

```
import os
import tensorflow as tf
import cv2
import sys
import numpy as np
import fnmatch
sys.path.append('/home/Imagecolor/')
sys.path.insert(0, '../architecture/')
sys.path.insert(0, '../model/')
from architecture import architecture

def eval(checkpoint_dir, image):
   with tf.Graph().as_default() as graph:
      input_image = tf.placeholder(tf.float32, shape=(1,144,160,3))
      logit = architecture.inference(input_image, 'test')
      variables = tf.all_variables()
      init      = tf.initialize_all_variables()
      sess      = tf.Session()
      saver     = tf.train.Saver(variables)
      #print 'jj'
      print((input_image))
      #print 'kk'
      tf.train.start_queue_runners(sess=sess)

      ckpt = tf.train.get_checkpoint_state(checkpoint_dir)

      if ckpt and ckpt.model_checkpoint_path:
         print 'Trying to Restoring model...'
         try:
            saver.restore(sess, ckpt.model_checkpoint_path)
         except:
            print 'Could not restore model'
            raise
            exit()
```

```
      graph_def = sess.graph.as_graph_def(add_shapes=True)

      img = cv2.imread(image)
      print img.shape
      print len(img.shape)
      if  len(img.shape) != 3:  #just for fun :p
         img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
         img = np.expand_dims(img, axis=2)

      if img.shape[0] != 160:
         img = cv2.resize(img, (160,144))

      img = img.astype('float')
      print img
      #print 'cc'
      fake = np.zeros((1,144,160,3))
      #print fake
      fake[0,:,:,:] = img
      print fake
      gen_img = sess.run([logit], feed_dict={input_image:fake})[0]
      #gen_img = gen_img*255
      #print (sess.run(input_image))
      image_name = image.split('.png')[0]+'.png'
      image_name = image_name.split('/')[-1]
      try:
         print 'Writing image ', image_name
         cv2.imwrite('./static/processed_images/' + image_name,
gen_img[0,:,:,:])
      except:
         raise

def main(argv=None):
   eval(sys.argv[1], sys.argv[2])

def land():
   print architecture.inference
```

```
if __name__ == "__main__":
   if len(sys.argv) < 2:
      print "Usage: python eval.py /path/to/model/ images.png"
      exit()
   tf.app.run()
```

train.py

```
import tensorflow as tf
import numpy as np
import os
import sys
import numpy as np
import cv2
from optparse import OptionParser
import fnmatch

sys.path.insert(0, '../utils/')
sys.path.insert(0, '../architecture/')

import architecture
import time
import feed_dict as fd

def get_feed_dict(batch_size, original_images_placeholder,
gray_images_placeholder, image_list, normalize):

   original_images, gray_images = fd.get_batch(batch_size, image_list,
normalize)
   feed_dict = {
      original_images_placeholder: original_images,
      gray_images_placeholder: gray_images
   }
   return feed_dict
```

```
def train(checkpoint_dir, image_list, batch_size, normalize):
   with tf.Graph().as_default():

      global_step = tf.Variable(0, name='global_step', trainable=False)

      original_images_placeholder = tf.placeholder(tf.float32,
shape=(batch_size, 144, 160, 3))
      gray_images_placeholder     = tf.placeholder(tf.float32,
shape=(batch_size, 144, 160, 3))

      # image summary for tensorboard
      tf.summary.image('original_images', original_images_placeholder)
      tf.summary.image('gray_images', gray_images_placeholder)

      logits = architecture.inference(gray_images_placeholder, "train")
      loss   = architecture.loss(original_images_placeholder, logits)

      tf.summary.scalar('loss', loss)

      train_op = tf.train.AdamOptimizer(learning_rate=1e-
2).minimize(loss, global_step=global_step)
# summary for tensorboard graph

      summary_op = tf.summary.merge_all()

      variables = tf.all_variables()
      init      = tf.initialize_all_variables()
      sess      = tf.Session()

      try:
         os.mkdir(checkpoint_dir)
      except:
         pass

      sess.run(init)
      print "\nRunning session\n"
```

```
        # saver for the model
        saver = tf.train.Saver(tf.all_variables())

        tf.train.start_queue_runners(sess=sess)

        # restore previous model if one
        ckpt = tf.train.get_checkpoint_state(checkpoint_dir+"training")
        if ckpt and ckpt.model_checkpoint_path:
            print "Restoring previous model..."
            try:
                saver.restore(sess, ckpt.model_checkpoint_path)
                print "Model restored"
            except:
                print "Could not restore model"
                pass

        # Summary op
        graph_def = sess.graph.as_graph_def(add_shapes=True)
        summary_writer = tf.summary.FileWriter(checkpoint_dir+"training",
graph_def=graph_def)

        # Constants
        step = int(sess.run(global_step))
        #epoch_num = step/(train_size/batch_size)

        while True:
            step += 1
            feed_dict = get_feed_dict(batch_size,
original_images_placeholder, gray_images_placeholder, image_list,
normalize)
            loss_value = sess.run([train_op, loss], feed_dict=feed_dict)

            if step % 1 == 0:
                print " Step: " + str(sess.run(global_step)) + " Loss: " +
str(loss_value)
            if step%100 == 0:
```

```
            print "Saving model"
            print
            saver.save(sess, checkpoint_dir+"training/checkpoint",
global_step=global_step)
            print


def main(argv=None):
   parser = OptionParser(usage='usage')
   parser.add_option('-c', '--checkpoint_dir',          type='str')
   parser.add_option('-b', '--batch_size', default=100, type='int')
   parser.add_option('-d', '--data_dir', type='str')
   parser.add_option('-n', '--normalize', type='str')

   opts, args = parser.parse_args()
   opts = vars(opts)

   checkpoint_dir = opts['checkpoint_dir']
   batch_size     = opts['batch_size']
   data_dir       = opts['data_dir']
   normalize      = opts['normalize']

   if normalize == 'y':
      normalize = True
   else:
      normalize = False

   if checkpoint_dir is None:
      print "checkpoint_dir is required"
      exit()

   print
   print 'checkpoint_dir: ' + str(checkpoint_dir)
   print 'batch_size:     ' + str(batch_size)
   print 'data_dir:       ' + str(data_dir)
   print
```

```
    #answer = raw_input("All correct?\n:")
    #if answer == "n":
    #    exit()

    pattern = "*resized.png"
    image_list = list()
    for d, s, fList in os.walk(data_dir):
        for filename in fList:
            if fnmatch.fnmatch(filename, pattern):
                image_list.append(os.path.join(d,filename))

    print str(len(image_list)) + ' images...'
    train(checkpoint_dir, image_list, int(batch_size), normalize)


if __name__ == "__main__":

    if sys.argv[1] == "--help" or sys.argv[1] == "-h" or len(sys.argv) <
2:
        print
        print "-c --checkpoint_dir <str> [path to save the model]"
        print "-b --batch_size     <int> [batch size]"
        print "-d --data_dir       <str> [path to root image folder]"
        print '-n --normalize      <str> [y/n normalize training images]'
        print
        exit()
    tf.app.run()
```

bright.py

```
import cv2
import sys
from scipy import misc
from PIL import Image, ImageEnhance
im = Image.open(sys.argv[1])

#image = Image.open('/home/satyendra/Enhance/index.jpg')
enhancer_object = ImageEnhance.Brightness(im)
out = enhancer_object.enhance(1.7)
im.show()
```

hist.py

```
import cv2
import sys
from scipy import misc
from PIL import Image, ImageEnhance
from scipy.misc.pilutil import Image

img = cv2.imread(sys.argv[1])# image taken as input on run time
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)# the input image is
converted to YUV channel as Histogram Equilisation can't applied on
RGB format of image and also it can't be applied on seperate field of
RGB.

# equalize the histogram of the Y channel
img_yuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0])

# convert the YUV image back to RGB format
img_output = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)

#it will save the resultant image as res.png
cv2.imwrite('res.png',img_output)

#it will show the both images as pop up
cv2.imshow('Color input image', img)
cv2.imshow('Histogram equalized', img_output)

#programme will wait for the key if ESC pressed the programme will
terminate
p = cv2.waitKey(0)
if p == 27:          # the programme will terminate on pressing ESC
    cv2.destroyAllWindows()
```

Inverse.py

```
import cv2
import sys
from scipy import misc
from PIL import Image, ImageEnhance

im = Image.open(sys.argv[1])
#im = Image.open('/home/satyendra/Enhance/index.jpg')
im_array = scipy.misc.fromimage(im)
im_inverse = 255 - im_array
im_result = scipy.misc.toimage(im_inverse)
misc.imsave('result.jpg',im_result)

print('Your image has been processed by Inverse Method')
```

PowerLog.py

```
import cv2
import sys
from scipy import misc
from PIL import Image, ImageEnhance
#im = Image.open(sys.argv[1])
print('4.Your image has been processed by Power Log Transform')

im = cv2.imread('/home/satyendra/Enhance/image_1244_resized_gray.png')
im = im/255.0
im_power_law_transformation = cv2.pow(im,0.6)
cv2.imshow('Original Image',im)
cv2.imshow('Power Law Transformation',im_power_law_transformation)
k = cv2.waitKey(0)
if k == 27:         # the programme will terminate on pressing ESC
    cv2.destroyAllWindows()
```

RGB.py

```
import cv2
import sys
from scipy import misc
from PIL import Image, ImageEnhance

im = Image.open(sys.argv[1])
#im = Image.open('/home/satyendra/Enhance/index.jpg')
rgb_im = im.convert('RGB')
wt, ht = im.size
for w in range(wt):
    for h in range(ht):
        print rgb_im.getpixel((w,h))

print('RGB values shown above')
```