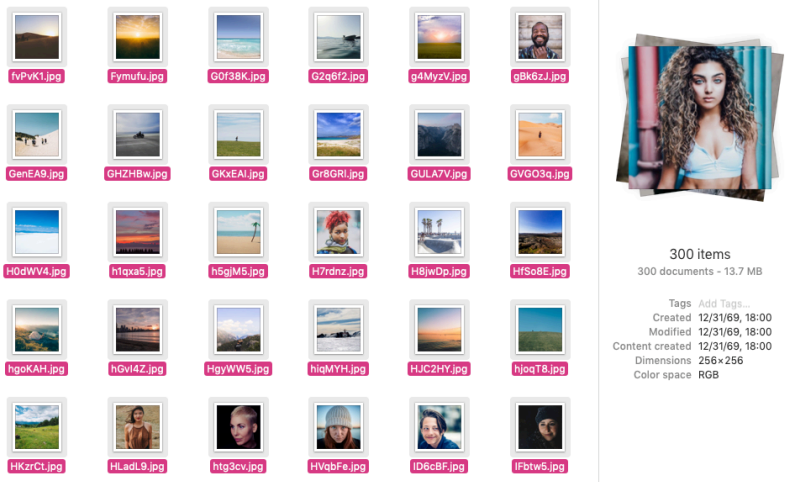
CSCE 636 Spring 2019
Neural Network Project Demo (Updated)
Jiatai Han
UIN: 726009450

1. **Topic**
   This project is called “Colorizing Grey-scale Photos using Neural Network”. The neural network model used in this project is built with Keras on TensorFlow, a simple GUI that can be run on Linux is provided to demonstrate the DNN. User will open a grey scale image (any size and even a colored image to simply test the colorizing performance), by clicking the button “Colorize”, the GUI will use the trained model to colorize the input image.

2. **Dataset**
   There were multiple trials to decide what will be the best dataset, to be able to train the model correctly, but also be time-efficient since this is a class project that has certain deadlines. The original dataset contains 9,294 images, obtained from “unsplash.com”, cropped and converted into resolution of 256x256. However, it is realized that by using a dataset with such size, each epoch consumes unacceptably long time, and if the total epoch amount is small, the model does not perform well as expected. Furthermore, the original dataset contains images with multiple types of objects, therefore the model appears to be underfit, even when total epoch amount is very large. As a result, it was decided that, two categories of images (portraits and landscape) were extracted from the original dataset, and the total number of images are limited into 300 (explained later in hyperparameters sections), therefore the model can be better trained to achieve expected functions, within reasonable time.



Size: 300, Resolution: 256x256

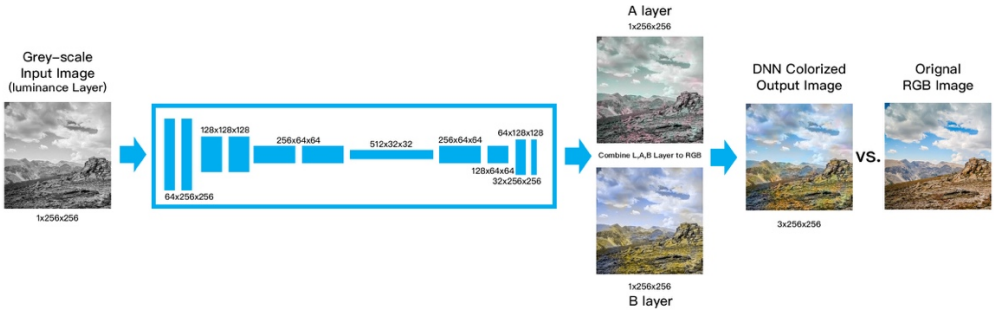
3. DNN Model

There are two versions of the architectures that were tested, a simple one constructed only with Keras, and another more complicated one using a pre-trained classifier as a fusion layer (mentioned earlier in Project Assignment 2), to help identify the objects inside the images, and learn to colorize them. However, the final decision was to use the simple DNN.

The reason of not using the complicated DNN is because, it takes longer time to train, but the result with same parameters are worse than the simple DNN. Also, when implementing them into the GUI, the complicated DNN requires downloading a pre-trained classifier model (InceptionResnet-v2) of size 225 MB. Furthermore, when processing the image, the model takes a lot of system memory when running (crushed on 4 GB ram, worked on 8 GB). The processing time is also much longer. The possible reason is that, with limited computing power and time, also a limited dataset with only two categories, the complicated DNN cannot outperform the simple DNN with its advantages. However, it is expected that, if the training dataset becomes more diverse and larger, the complicated DNN will begin outperforming the simple one.

3.1. Architecture

This model used 1 input layer as entry, and then 12 of 2D convolution layers, there are also 3 up-sampling layers for 2D inputs




```
model = Sequential()
model.add(InputLayer(input_shape=(256, 256, 1)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))
model.add(UpSampling2D((2, 2)))
```

The total 300 input image was split into training and validation set as 95% : 5%, so 285 images were used for training, 15 images are used for validation.

3.2. Input: Shape of Tensor

A grey-scale single image (one channel as luminance)

```
Input data:      (1, 256, 256, 1)
```

3.3. Output: Shape of Tensor

Two image layers (A and B)

```
Output:          (1, 256, 256, 2)
```

3.4. Shape of Output Tensor for Each Layer

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 256, 256, 1)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 64)      640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 64)      36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 128)     73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 256)       295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 256)       590080
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 512)       1180160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 256)       1179904
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 128)       295040
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 64, 64)        73792
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 128, 128, 32)      18464
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 128, 128, 2)       578
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 256, 256, 2)       0
```

4. Hyperparameters

The hyperparameters that were modified and tested in this project is the size of input images from the dataset, the batch size and the epochs. This model was

trained on Google's Colab using GPU (it is tested to be the fasted runtime method for training), which provides 12 GB GPU memory but limited the session time-out as 12 hours, which means the whole training process must be executed within 12 hours, otherwise the session will be reset and all progress will be lost.

4.1. List of Hyperparameters

size of input
batch size
epochs

4.2. Range of Value of Hyperparameters Tried

size of input: 10, 50, 100, 200, 300
batch size: 10, 50, 100
epochs: 10, 100, 500, 1000, 1500, 3250, 5000

4.3. Optimal Hyperparameters Found

size of input: 300
batch size: 100
epochs: 3250

When size of input is more than 300, the time spent for each epoch is too long, so the dataset is reduced to 300 images. The batch size can be as much as 100 to fit into the given memory, but more than that, the Colab crushes. The size of steps depends on total input size and the batch size, so in this case, if the total input is 300 images, and batch size is 100, the steps will be 3. For the epochs, the accuracy keeps improving as it gets larger, however due to the limitation of Colab, this is the largest epochs that balances the Colab's limitation and the performance of the model.

5. Annotated Code

```
# Get resources dataset from Jiatai's Git
! git clone https://github.com/hjt486/DeepColorizer

# Get and confirm path
#! pwd

# Force to install Keras 2.1.5
# (To resolve issue of saving\loading .h5 model)
!pip install keras==2.1.5
# Tensorflow
import tensorflow as tf
# Keras
from keras.layers import Conv2D, UpSampling2D, InputLayer, Conv2DTranspose
from keras.layers import Activation, Dense, Dropout, Flatten
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential
from keras.models import load_model
from keras.callbacks import ModelCheckpoint
```

```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from keras.callbacks import TensorBoard
# Image pre/post processing
from skimage.color import rgb2lab, lab2rgb, rgb2gray, xyz2lab
from skimage.io import imsave
# Misc
import math
import numpy as np
import os
import random
# Display result in CoLab
from IPython.display import Image, display

# Define resource root path
path = '/content/DeepColorizer/datasets/selected/'

# Load images
# Mixed contains 300 images, 200 landscapes, 100 portraits
X = []
file_limit = os.listdir(path + 'Train/Mixed/')[:100]
for filename in file_limit:
    X.append(img_to_array(load_img(path + 'Train/Mixed/' + filename)))
X = np.array(X, dtype=float)

# Split loaded images into train and test data set with 95:5
split = int(0.95*len(X))
Xtrain = X[:split]
Xtrain = 1.0/255*Xtrain
Xval = X[split:]
Xval = 1.0/255*Xval

# Image transformer
datagen = ImageDataGenerator(
        shear_range=0.2,
        zoom_range=0.2,
        rotation_range=20,
        horizontal_flip=True)

# Training parameters
# total = how many traing images
total = 300
batch_size = 100
epochs = 3250
# steps is automatically decided based on total and batch_size
steps = math.ceil(total/batch_size)

# Generate training data
def image_a_b_gen(batch_size):
    for batch in datagen.flow(Xtrain, batch_size=batch_size):
        lab_batch = rgb2lab(batch)
        X_batch = lab_batch[:,:,:,0]
        Y_batch = lab_batch[:,:,:,1:] / 128
        yield (X_batch.reshape(X_batch.shape+(1,)), Y_batch)

# Generate validation data
def image_a_b_gen2(batch_size):
    for batch in datagen.flow(Xval, batch_size=batch_size):
        lab_batch = rgb2lab(batch)
        X_batch = lab_batch[:,:,:,0]
        Y_batch = lab_batch[:,:,:,1:] / 128
        yield (X_batch.reshape(X_batch.shape+(1,)), Y_batch)

# Building the neural network
model = Sequential()
model.add(InputLayer(input_shape=(256, 256, 1)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same', strides=2))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
```

```
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))
model.add(UpSampling2D((2, 2)))
model.compile(optimizer='rmsprop', loss='mse', metrics=['acc'])

# Train model
model.fit_generator(image_a_b_gen(batch_size),
                    validation_data=image_a_b_gen2(batch_size),
                    epochs=epochs, steps_per_epoch=steps, validation_steps=1)

# Save Model
model.save('mixed.h5')
print('Model saved 300x3x3250')
! ls
# Download model
# files.download('mixed.h5')

# Load test images
image_path = path + 'Validate/1dZTRG.jpg'
# Process test image to grey scale
image = img_to_array(load_img(image_path))
color_me = np.array(image, dtype=float)
color_me = rgb2lab(1.0/255*image)[:,:,0]
color_me = color_me.reshape(1, 256, 256, 1)

# Apply model to generate A B channels
output = model.predict(color_me)
output = output * 128

# Merge LAB space and convert to RGB, save
cur = np.zeros((256, 256, 3))
cur[:,:,0] = color_me[0][:,:,0]
cur[:,:,1:] = output[0]
imsave("img_result.png", lab2rgb(cur))
imsave("img_gray_version.png", rgb2gray(lab2rgb(cur)))
Image('img_result.png')
```

6. Training Performance

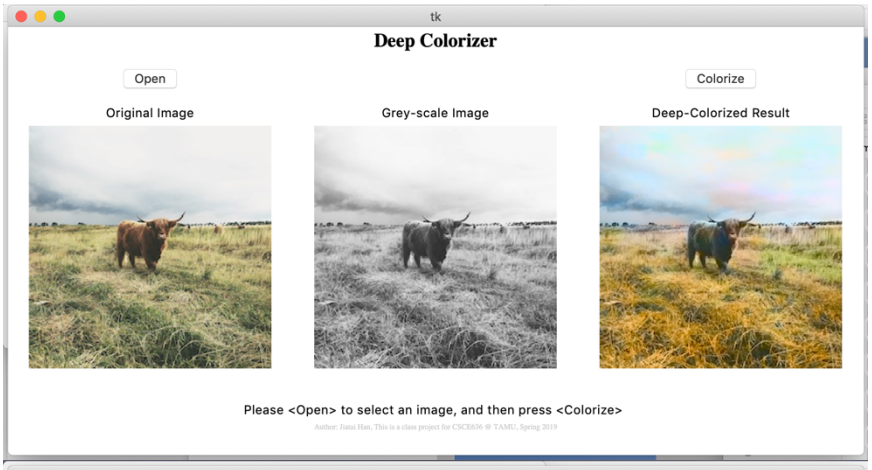
```
Epoch 1/3250
3/3 [==============================] - 29s 10s/step - loss: 0.6956 - acc: 0.5057 - val_loss: 1.1945 - val_acc: 0.3653
Epoch 2/3250
3/3 [==============================] - 4s 1s/step - loss: 1.0032 - acc: 0.4998 - val_loss: 0.0443 - val_acc: 0.3542
Epoch 3/3250
3/3 [==============================] - 12s 4s/step - loss: 0.0212 - acc: 0.5041 - val_loss: 0.0967 - val_acc: 0.3566
Epoch 4/3250
3/3 [==============================] - 13s 4s/step - loss: 0.0262 - acc: 0.5042 - val_loss: 0.0270 - val_acc: 0.3674
Epoch 5/3250
3/3 [==============================] - 13s 4s/step - loss: 0.0120 - acc: 0.5048 - val_loss: 0.0273 - val_acc: 0.3797
.
.
.
Epoch 3245/3250
3/3 [==============================] - 12s 4s/step - loss: 8.6250e-04 - acc: 0.8891 - val_loss: 0.0275 - val_acc:
0.4998
Epoch 3246/3250
3/3 [==============================] - 12s 4s/step - loss: 9.1122e-04 - acc: 0.8862 - val_loss: 0.0318 - val_acc:
0.5411
Epoch 3247/3250
3/3 [==============================] - 12s 4s/step - loss: 0.0013 - acc: 0.8728 - val_loss: 0.0288 - val_acc: 0.5203
Epoch 3248/3250
3/3 [==============================] - 12s 4s/step - loss: 0.0017 - acc: 0.8493 - val_loss: 0.0281 - val_acc: 0.5469
Epoch 3249/3250
3/3 [==============================] - 12s 4s/step - loss: 0.0018 - acc: 0.8542 - val_loss: 0.0341 - val_acc: 0.4802
Epoch 3250/3250
3/3 [==============================] - 13s 4s/step - loss: 0.0025 - acc: 0.8175 - val_loss: 0.0278 - val_acc: 0.5157
```

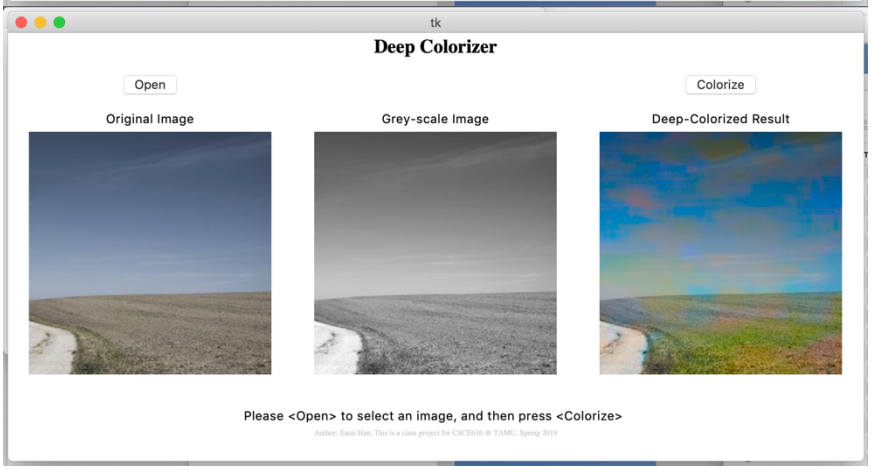
The accuracy is keep increasing during the training, but the validation accuracy only increased from 35% to around 50%, this is expected since the coloring process can never be as same as original image, but it is also possible as epochs increase, it could have a better validation accuracy due to better training result.

7. Test Result

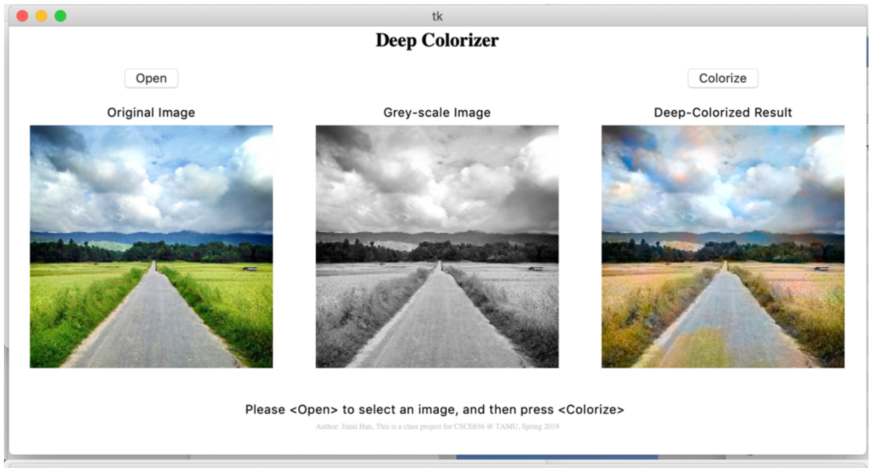
Several images that are either portraits or landscape or combined, are tested, and the result are subjectively classified as “Good”, “Fair” and “Bad”. Since whatever the input image is, the program will convert it into grey-scale, so all the input images in this part are colored, to better demonstrate the comparison between original image and colorized result.

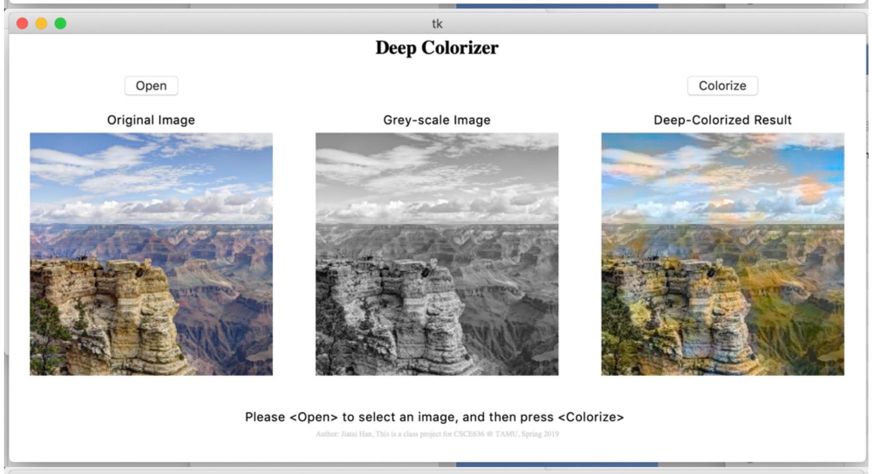
7.1. Good

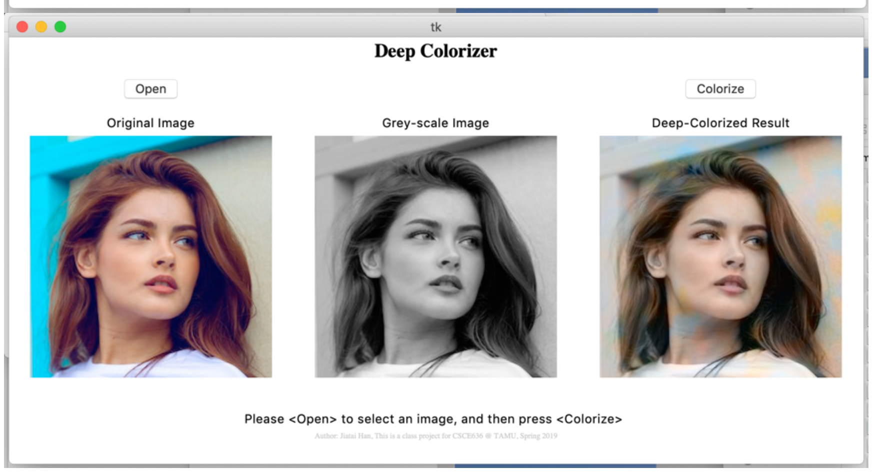

tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>
Author: Jintai Han, This is a class project for CSCE636 @ TAMU, Spring 2019

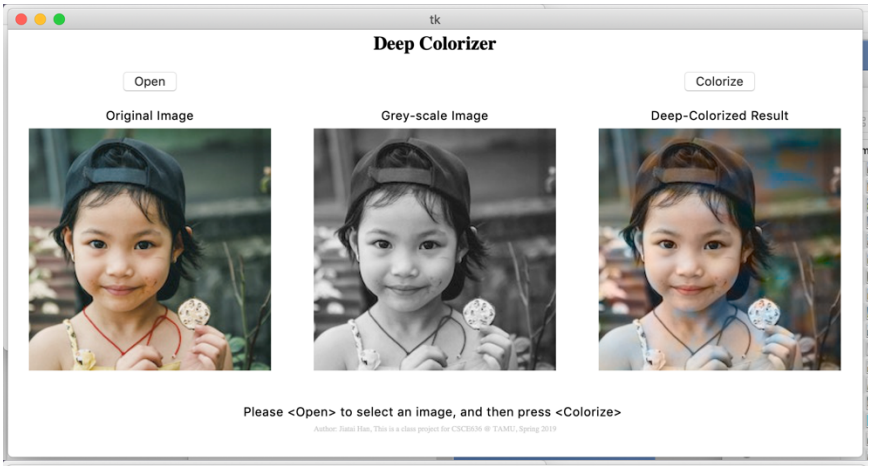

tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>
Author: Jintai Han, This is a class project for CSCE636 @ TAMU, Spring 2019

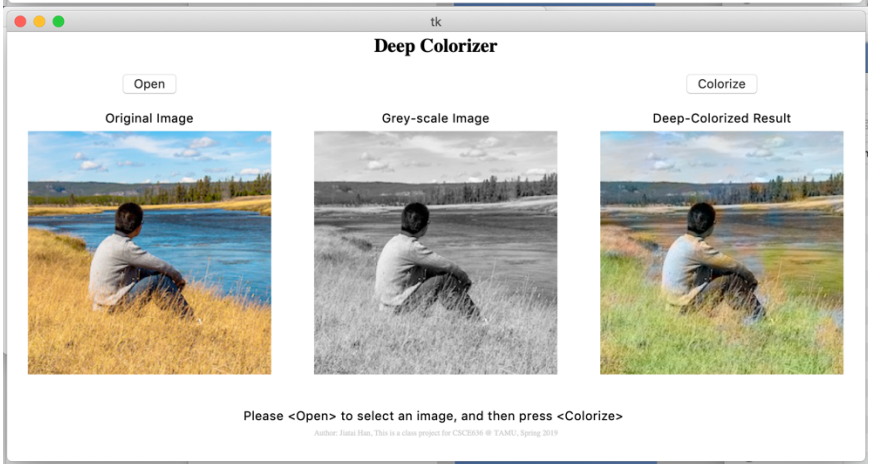
7.2. Fair


tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>
Author: Jiatai Han, This is a class project for CSCE636 @ TAMU, Spring 2019

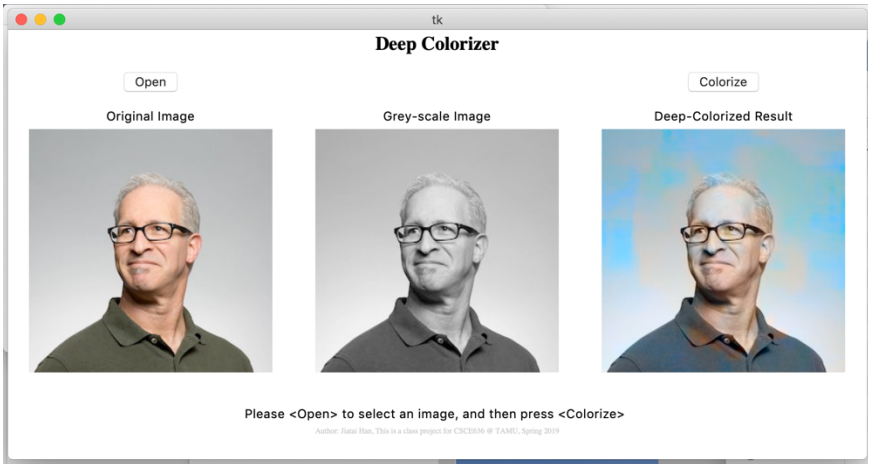

tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>

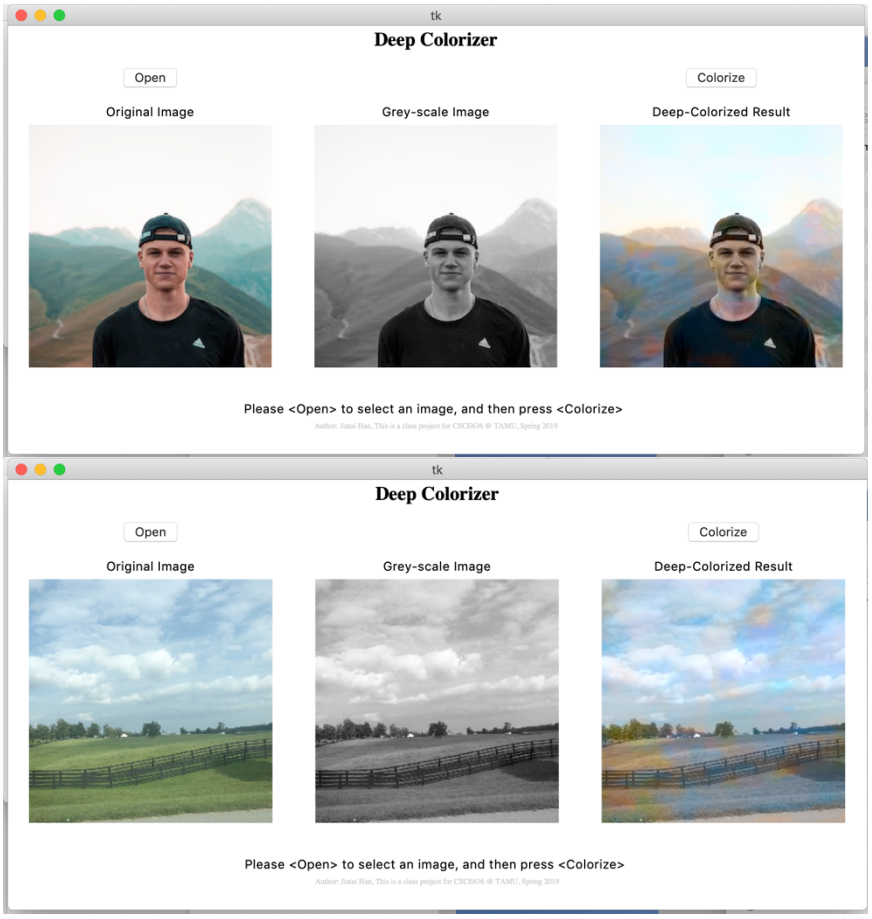

tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>


tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>


tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>

7.3. Bad


tk
Deep Colorizer
Open
Colorize
Original Image
Grey-scale Image
Deep-Colorized Result
Please <Open> to select an image, and then press <Colorize>



8. Instruction

Please use any Linux distribution or Unix-like OS (tested on Ubuntu 18.04.1 and macOS Mojave).

8.1. Install Dependencies

Python 3
Tkinter
Keras
Tensorflow
Numpy
Scikit-image

8.2. Execution

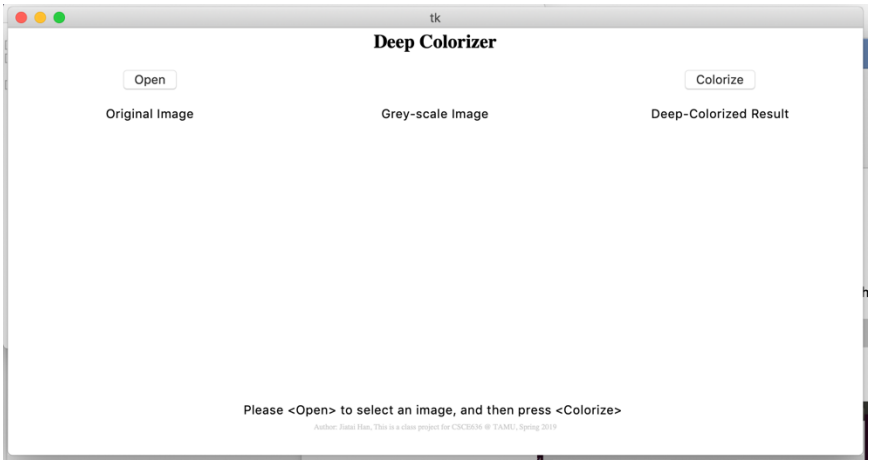
Please go to:
https://github.com/hjt486/DeepColorizer

and clone the entire repository.

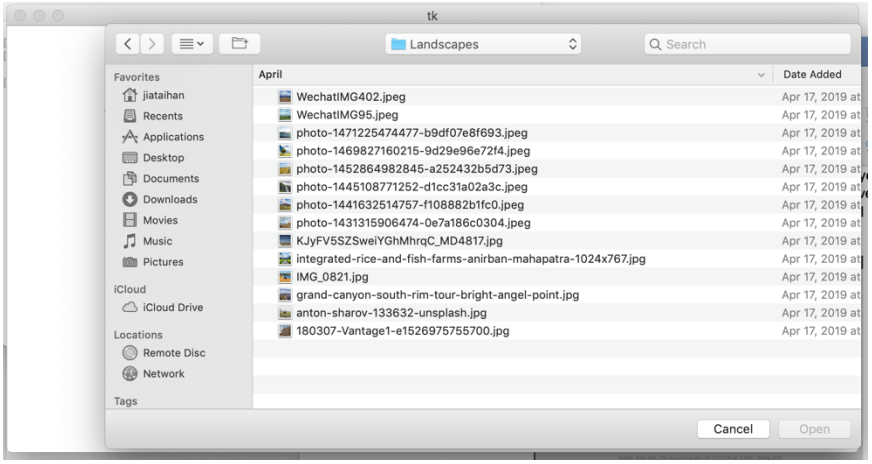
Once the repository is on your local disk, open the terminal and go to the repository, go to the “GUI” folder and execute the ‘colorizer_python3.py’:

```
cd GUI
python3 colorizer_python3.py
```

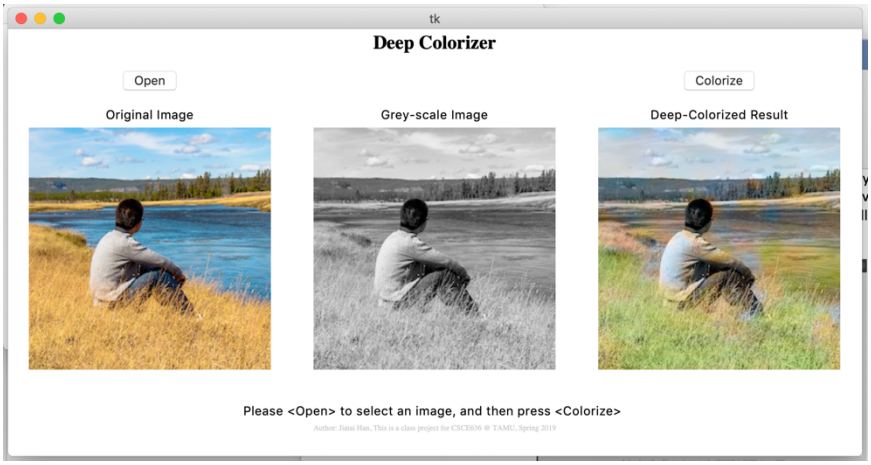
the GUI will pop up:



Click “Open”, and then click “test_images” folder under the “GUI” folder, then please select an image file from it, click “OPEN” to finish selection.
(You can try any photo you like, but it is recommended to use a square ratio (1:1) image with resolution of 256x256, but the program is able to process with image of any size, but changing the ratio may influence the colorizing performance. Also, you can select image that is not grey-scale, as mentioned above, the program will convert the image to grey-scale no matter it is with color or not, so using a colored image will be easy to make comparison)



The input image will be displayed on the left side, click “Colorize” to perform the colorizing procedure, and the colorized image will be displayed on the right.



8.3. Code

https://github.com/hjt486/DeepColorizer

Repository folders explanation:

“datasets”
contains folder name “Train”, which is a datasets used for training (300 images of both portraits and landscape), also contains a folder called “full-dataset” with 9294 images, you can further utilize this dataset to train the model.

“DNN”
contains a .ipynb file that used to train the model, if you would like to train you own model, you can use this on Google Colab or any platform that support Jupyter Notebook.

“GUI”
as explained earlier, this contains a h5 model and a simple GUI application that can be run to test the DNN.

“models”
contains two models:
“mixed_100_1_5000_better_portraits.h5”
“mixed_300_3_3250_better_landscape.h5”

e.g: mixed_300_3_3250_better_portraits means it used mixed dataset (both portraits and landscape) of 300 images to train, steps are 3 and total 3250 epochs with tested result that shows it has better performance on landscape, you can use the model here to replace the one in GUI, to see the difference.

8.4. Demo Link on YouTube:

https://youtu.be/9azzM7n1aCc