Black and White image colorization

-With OpenCV and deep learning

Problem Identification

By

Kannaji Vamsi Krishna

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGENEERING

SUBMITED TO

FEYNN LABS

March 2022

Abstract

Deep learning is a subfield of machine learning, which aims to learn a hierarchy of features from input data. Nowadays, researchers have intensively investigated deep learning algorithms for solving challenging problems in many areas such as image classification, speech recognition, signal processing, and natural language processing.

Introduction

Colorization of black and white images manually is a time consuming, difficult and the resultant image is mostly inaccurate. To overcome this our project focuses on using openCV and Deep Learning to predict the color of each pixel with the help of the model trained using ImageNet dataset.

Methodology

The input black and white image is converted from RGB color space to Lab color space. Lab color space has 3 channels

- L channel encodes lightness intensity
- A channel encodes green-red
- B channel encodes blue-yellow

Fig1: L A B channels of an image

The input black and white image is given as the L channel. With the help of the model, values of a and b channels will be predicted. Lab image will be converted back to RGB image with the altered a and b values. The RGB image is shown as the output. A video file can also be colorized by following the same methodology by extracting each and every frame from video using the vs.read() function of the videostream package.

Models used

ImageNet

ImageNet is formally a project aimed at (manually) labeling and categorizing images into almost 22,000 separate object categories for the purpose of computer vision research.

Models are trained on ~1.2 million training images with another 50,000 images for validation and 100,000 images for testing.

Models implemented

- VGG16
- VGG19
- ResNet50

VGG16 and VGG19

VGG16 and VGG 19 are the variants of the VGGNet. VGGNet is a Deep Convolution Neural Network that was proposed by Karen Simonyan and Andrew Zisserman of the University of Oxford in their research work 'Very Deep Convolutional Neural Networks for Large-Scale Image Recognition'. The VGG16 has 16 layers in its architecture while the VGG19 has 19 layers.

ResNet50

ResNet is the short name for Residual Networks and ResNet50 is a variant of this having 50 layers. It is a deep convolutional neural network used as a transfer learning framework where it uses the weights of pre-trained ImageNet.

Performance analysis of VGG16 and VGG19 and ResNet50

VGG19

We used the VGG10 as a transfer learning framework, pre-trained ImageNet weights will be used with this model.

After adding all the layers, we check the model's summary.

```
#VGG19 Model Summary

model_vgg19.summary()
```

Accuracy

```
#Accuracy of VGG19

from sklearn.metrics import accuracy_score

accuracy_score(y_true, y_pred1)
```

VGG16

```
#Checking the final VGG16 model summarymodel_vgg16.summary()
```

Accuracy

```
#Accuracy of VGG16from sklearn.metrics import accuracy_scoreaccuracy_score(y_true, y_pred2)
```

ResNet50 Transfer Learning Model

```
#Summary of ResNet50 Model
```

```
model_resnet.summary()
```

Accuracy of ResNet50

```
#ResNet50 Classification accuracy

from sklearn.metrics import accuracy_scoreaccuracy_score(y_true, y_pred3)
```

External Search

1. Introduction

Deep learning is a subfield of machine learning, which aims to learn a hierarchy of features from input data. Nowadays, researchers have intensively investigated deep learning algorithms for solving challenging problems in many areas such as image classification, speech recognition, signal processing, and natural language processing.

2. Deep learning methods.

Deep learning methods are a group of machine learning methods that can learn features hierarchically from lower level to higher level by building a deep architecture. The deep learning methods have the ability to automatically learn features at multiple levels, which makes the system be able to learn complex mapping function directly from data, without the help of the human-crafted features. The most characterizing feature of deep learning methods is that their models all have deep architectures. A deep architecture means it has multiple hidden layers in the network. In contrast, a shallow architecture has only a few hidden layers (1 to 2 layers). Deep Convolutional Neural networks are successfully applied in various areas. Regression, Classification, dimensionality reduction, modeling motion, modeling textures, information retrieval, natural language processing, robotics, fault diagnosis, and road crack detection.

3. Deep Learning Algorithms

Deep learning algorithms have been extensively studied in recent years. As a consequence, there are a large number of related approaches. Generally speaking, these algorithms can be grouped into two categories based on their architectures:

- Restricted Boltzmann machines (RBMs).
- Convolutional neural networks (CNNs)

Restricted Boltzmann Machines. (RBMs).

RBM is an energy-based probabilistic generative model. It is composed of one layer of visible units and one layer of hidden units. The visible units represent the input vector of a data sample and the hidden units represent features that are abstracted from the visible units. Every visible unit is connected to every hidden unit, whereas no connection exists within the visible layer or hidden layer.

Convolutional Neural Network. (CNNs).

During the last seven years, the quality of image classification and object detection has been dramatically improved due to the deep learning method. Convolutional neural networks (CNNs) brought a revolution in the computer vision area. It not only have been continuously advancing the image classification accuracy, but also play an important role for generic feature extraction such as scene classification, object detection, semantic segmentation, image retrieval, and image caption.

Convolutional neural network (CNNs) is one of the most powerful classes of deep neural networks in image processing tasks. It is highly effective and commonly used in computer vision applications. The convolution neural network contains three types of layers convolution layers, subsampling layers, and full connection layers. The whole architecture of the convolutional neural network is shown in Figure 2. A brief introduction to each type of layer is provided in the following paragraphs.

Convolution Layer.

As Figure 3 shows, in convolution layer, the left matrix is the input, which is a digital image, and the right matrix is a convolution matrix. The convolution layer takes the convolution of the input image with the convolution matrix and generats the output image.

Usually, the convolution matrix is called filter and the output image is called filter response or filter map. An example of convolution calculation is demonstrated in Figure **4.** Each time, a block of pixels is convoluted with a filter and generates a pixel in a new image.

Subsampling Layer.

The subsampling layer is an important layer to the convolutional neural network. This

layer is mainly to reduce the input image size in order to give the neural network more

invariance and robustness. The most used method for subsampling layer in image processing

tasks is max pooling. So the subsampling layer is frequently called max pooling layer. The max pooling method is shown in Figure 5. The image is divided into blocks and the maximum value of each block is the corresponding pixel value of the output image. The reason to use subsampling layer is as follows. First, the subsampling layer has fewer parameters and it is faster to train. Second, a subsampling layer makes convolution layer tolerate translation and rotation among the input pattern

Full Connection Layer.

Full connection layers are similar to the traditional feed-forward neural layer. They make

the neural network fed forward into vectors with a predefined length. We could fit the vector

into certain categories or take it as a representation vector for further processing.

4. Applications.

Deep learning has been widely applied in various fields, such as computer vision, signal

processing, and speech recognition.

4.1. CNN-Based Applications in Visual Computing.

As we know, convolutional neural networks are very powerful tools for image

recognition and classification. These different types of CNNs are often tested on well-known

ImageNet LargeScale Visual Recognition Challenge (ILSVRC) dataset and achieved state-of

the-art performance in recent years. After winning the ImageNet competition in 2012, the CNN based methods have brought about a revolution in computer vision. CNNs have been applied with great success to the object detection, object segmentation, and recognition of objects and regions in images. Compared with hand-crafted features, for example, Local Binary Patterns 5(LBP) and Scale Invariant Feature Transform (SIFT), which need additional classifiers to solve vision problems, the CNNs can learn the features and the classifiers jointly and provide superior performance.

4.2. CNN for Face Recognition.

Face recognition has been one of the most important computer vision tasks since the

1970s. Face recognition systems typically consist of four steps. First, given an input image with one or more faces, a face detector locates and isolates faces. Then, each face is pre-processed and aligned using either 2D or 3D modeling methods. Next, a feature extractor extracts features from an aligned face to obtain a low-dimensional representation (or embedding). Finally, a classifier makes predictions based on the low-dimensional representation. The key to getting good performances for face recognition systems is obtaining an effective low-dimensional Representation. Face recognition systems using hand-crafted features include. Lawrence et al. first proposed using CNNs for face recognition. Currently, the state-of-the-art performance of face recognition systems, that is, Facebook's DeepFace and Google's FaceNet, are based on CNNs. Other notable CNN-based face recognition systems are lightened convolutional neural networks and Visual Geometry Group (VGG) Face Descriptor.

Instead of using handcrafted features, CNNs are directly applied to RGB pixel values and used as a feature extractor to provide a low-dimensional representation characterizing a person's face. In order to normalize the input image to make the face robust to different view angles, DeepFace models a face in 3D and aligns it to appear as a frontal face. Then, the normalized input is fed to a single convolution-pooling-

convolution filter. Next, 3 locally connected layers and 2 fully connected layers are used to make final predictions. Though DeepFace achieves the best performance on face recognition up to date, its representation is difficult to interpret and use because the faces of the same person are not clustered necessarily during the training process. In contrast, FaceNet defines a triplet loss function directly on the representation, which makes the training procedure learn to cluster face representation of the same person. It should also be noted that OpenFace uses a simple 2D affine transformation to align face input.

Nowadays, face recognition in mobile computing is a very attractive topic. While DeepFace and FaceNet remain private and are of large size, OpenFace offers a lightweight, real-time, and open source face recognition system with competitive accuracy, which is suitable for mobile computing.

Implementation

Colorizing B/W image:

Importing required packages

```
import numpy as np

import argparse

import cv2
```

Setting up argparse to take arguments from command line

```
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str, required=True,
        help="path to input black and white image")
ap.add_argument("-p", "--prototxt", type=str, required=True,
        help="path to Caffe prototxt file")
ap.add_argument("-m", "--model", type=str, required=True,
        help="path to Caffe pre-trained model")
```

```
ap.add_argument("-c", "--points", type=str, required=True,
        help="path to cluster center points")
args = vars(ap.parse_args())
```

Loading models into the program

```
print("Loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
pts = np.load(args["points"])
print(pts)
```

Adding cluster centers as 1x1 convolutions to the model

```
class8 = net.getLayerId("class8_ab")
conv8 = net.getLayerId("conv8_313_rh")

pts = pts.transpose().reshape(2, 313, 1, 1)
net.getLayer(class8).blobs = [pts.astype("float32")]
net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]
```

Reading input image and converting it from RGB to LAB

```
image = cv2.imread(args["image"])
scaled = image.astype("float32") / 255.0
lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
```

Resizing the image to 224x224 and extracting the L channel of the image

```
resized = cv2.resize(lab, (224, 224))

L = cv2.split(resized)[0]

L -= 50
```

Predicting values of a and b channels

```
print("[INFO] colorizing image...")

print("Press q to exit preview")

net.setInput(cv2.dnn.blobFromImage(L))

ab = net.forward()[0, :, :, :].transpose((1, 2, 0))
```

```
ab = cv2.resize(ab, (image.shape[1], image.shape[0]))
```

Concatenating values of ab channel to the input image

```
L = cv2.split(lab)[0]
colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)
```

Converting the image from lab space to RGB color space

```
colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
colorized = np.clip(colorized, 0, 1)
```

Original and colorized image is displayed and the colorized image is saved.

```
colorized = (255 * colorized).astype("uint8")
cv2.imshow("Original", image)
cv2.imshow("Colorized", colorized)
key=cv2.waitKey(0)
if key==ord("q"):
   cv2.destroyAllWindows()
print("Enter name of the file")
name=input()
path="color/"
name=path+name+".png"
cv2.imwrite(name,colorized)
cv2.waitKey(0)
```

Colorizing B/W video:

Importing required packages

```
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
import cv2
```

Setting up argparse to take arguments from command line

```
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", type=str,required=True,
	help="path to optional input video")
ap.add_argument("-p", "--prototxt", type=str, required=True,
	help="path to Caffe prototxt file")
ap.add_argument("-m", "--model", type=str, required=True,
	help="path to Caffe pre-trained model")
ap.add_argument("-c", "--points", type=str, required=True,
	help="path to cluster center points")

args = vars(ap.parse_args())

print("[INFO] opening video file...")
```

```
vs = cv2.VideoCapture(args["input"])
```

Loading models into the program

```
print("[INFO] loading model...")

net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

pts = np.load(args["points"])
```

Adding cluster centers as 1x1 convolutions to the model

```
class8 = net.getLayerId("class8_ab")

conv8 = net.getLayerId("conv8_313_rh")

pts = pts.transpose().reshape(2, 313, 1, 1)

net.getLayer(class8).blobs = [pts.astype("float32")]

net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]
```

Looping through each frame from the video

```
while True:

    frame = vs.read()

    frame = frame[1]

    if frame is None:

        break

    frame = imutils.resize(frame, width=500)
```

Colorizing each frame of the video by converting them to lab colour space and then predicting a,b values and converting them back to BGR colour space

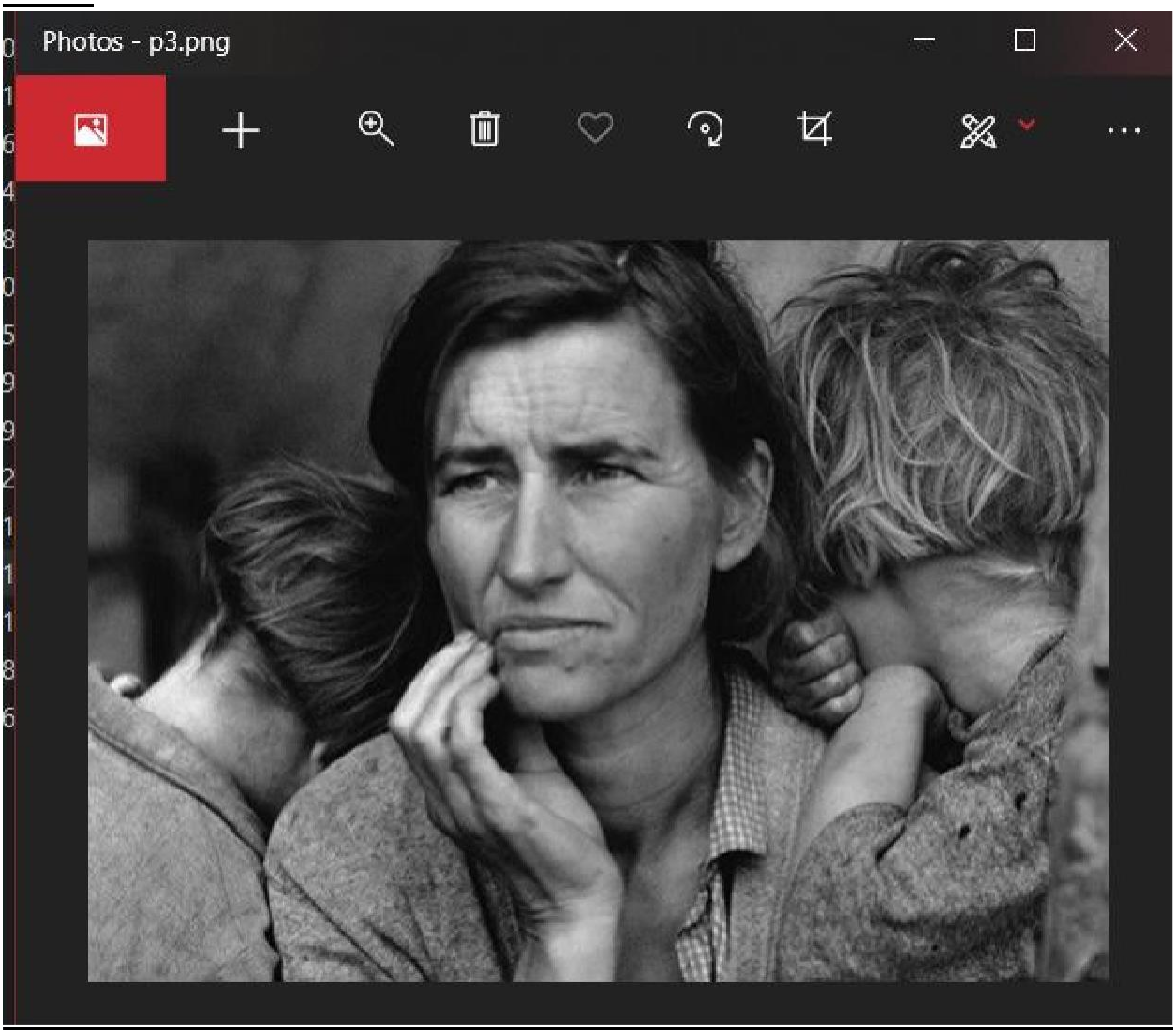
```
    scaled = frame.astype("float32") / 255.0
```

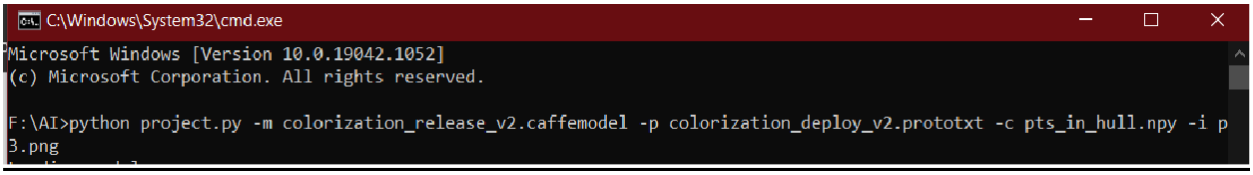
```
            lab = cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
            resized = cv2.resize(lab, (224, 224))
            L = cv2.split(resized)[0]
            L -= 50
            net.setInput(cv2.dnn.blobFromImage(L))
            ab = net.forward()[0, :, :, :].transpose((1, 2, 0))
            ab = cv2.resize(ab, (frame.shape[1], frame.shape[0]))
            L = cv2.split(lab)[0]
            colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)
            colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2BGR)
            colorized = np.clip(colorized, 0, 1)
            colorized = (255 * colorized).astype("uint8")
            cv2.imshow("Orignal", frame)
            cv2.imshow("Colorized", colorized)
```

Assigning "q" to break the loop and close all open windows.

```
            key = cv2.waitKey(1) & 0xFF
            if key == ord("q"):
                        break
    vs.release()
    cv2.destroyAllWindows()
```

Input:

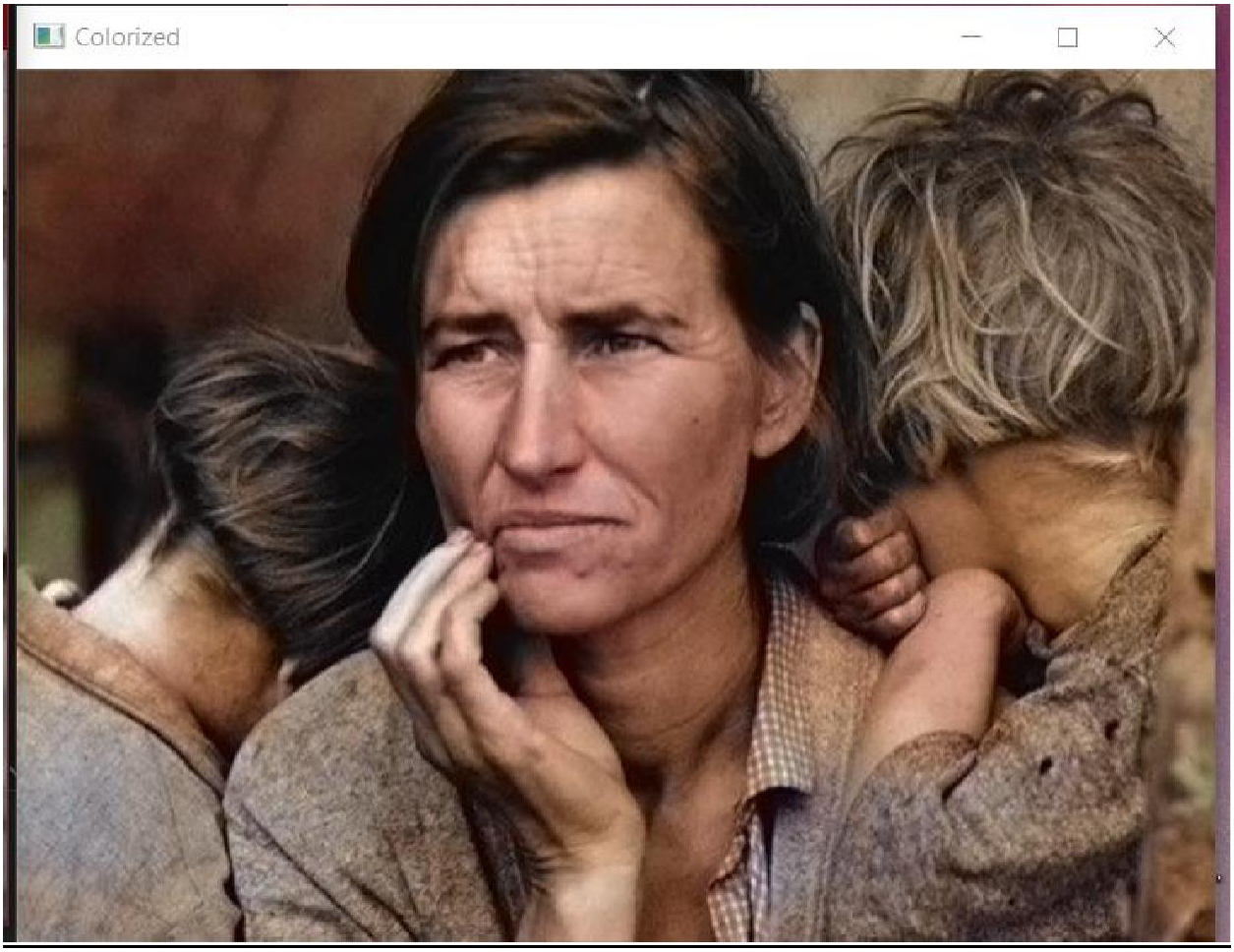

Photos - p3.png


C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.
F:\AI>python project.py -m colorization_release_v2.caffemodel -p colorization_deploy_v2.prototxt -c pts_in_hull.npy -i p
3.png

Output:


[INFO] colorizing image...
Press q to exit preview
Enter name of the file
sample1

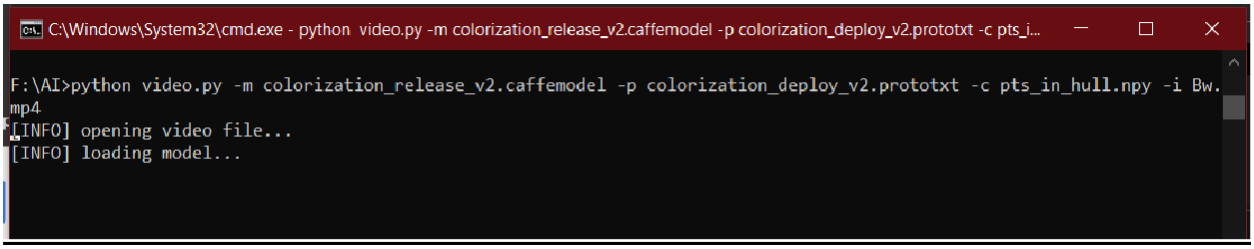

Original


Colorized

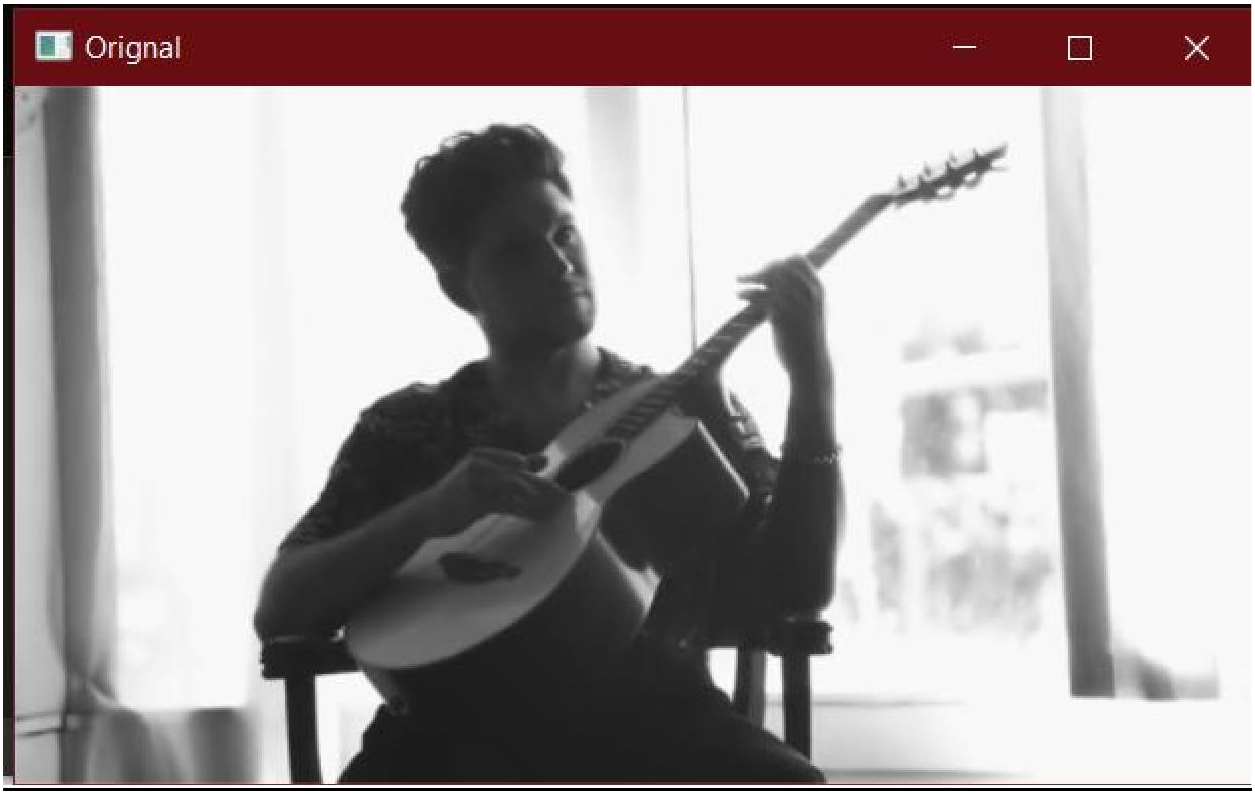

This PC
New Volume (F:)
AI
Color
sample1.png

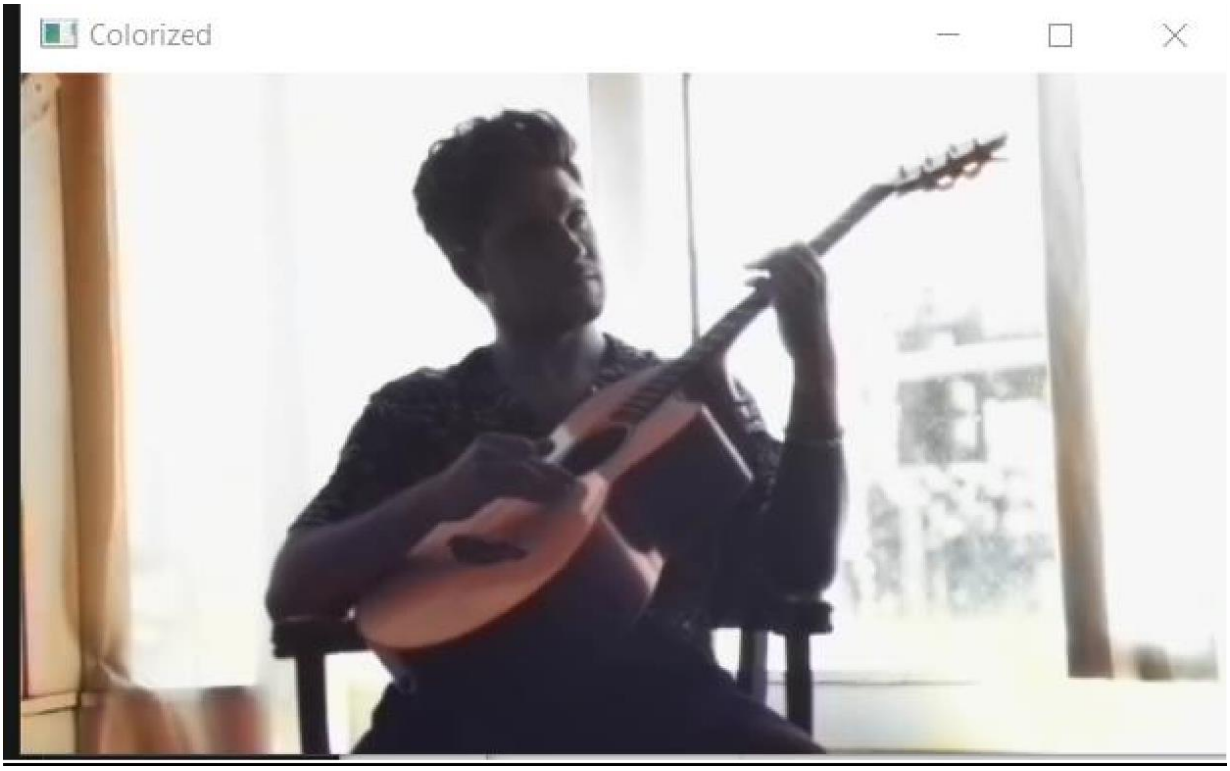
Input:


Bw.mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
00:00
-03:13
90%


C:\Windows\System32\cmd.exe - python video.py -m colorization_release_v2.caffemodel -p colorization_deploy_v2.prototxt -c pts_i...
F:\AI>python video.py -m colorization_release_v2.caffemodel -p colorization_deploy_v2.prototxt -c pts_in_hull.npy -i Bw.mp4
[INFO] opening video file...
[INFO] loading model...

Output:





References

[1]. G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” Neural Computation, vol. 18, no. 7, pp. 1527–1554, 2006.

[2]. D. Yu and L. Deng, “Deep learning and its applications to signal and information processing,” IEEE Signal Processing Magazine, vol. 28, no. 1, pp. 145–154, 2011.

[3]. Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, “Deep learning for visual understanding: a review,” Neurocomputing, vol. 187, pp. 27–48, 2016.

[4]. P. Smolensky, “Information processing in dynamical systems: foundations of harmony theory,” Tech. Rep. DTIC Document, 1986.

[5]. Y. Bengio, “Learning deep architectures for AI,” Foundations and Trends in Machine Learning, vol. 2, no. 1, pp. 1–27, 2009.

[6]. Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” The Handbook of Brain Theory and Neural Networks, vol. 3361, no. 10, p. 1995, 1995