Research on LAB Color Space

Lab(L+ab)

Basically they created all the images which are of the form (RGB) to (LAB)

Lab format i==> L—>Lightness/Intensity(black and white image) ,

A—>Green-Red , B—>Blue-Yellow

Process Being

1.Convert all training images from the RGB color space to the Lab color space

2.Use the L channel as the input to the network and train the network to predict the ab channels.

3.Combine the input L channel with the predicted ab channels.

4.Convert the Lab image back to RGB.

The LAB colour space is a three-dimensional model that encapsulates Lightness (L) and two colour-opponent dimensions: Green-Red (A) and Blue-Yellow (B). Unlike RGB, which is device-dependent and lacks perceptual uniformity, LAB is designed to approximate human vision more closely

The L\*a\*b is a 3D space

(The numbers here represent the points on the 3d plane and how there colour changes with the axis length from the origin)

Researched 2 Ways of Approaching this problem

L = Lightness

1.From the Research Paper/Github Repo of RichZhang

a)Using the pretrained networks and loading them using the dnn function in cv2 library

b)Processing the points(Part of the code wasn’t able to understand )

c)loading and processing the black and white image (converting to grayscale) and LAB color space

d)Resizing the L channel

e}Processing the L image further

f)Resize the ab Channels and mix with L channel

g)Finally convert the LAB 2 BGR using cv2 and display the colored and b/w images side by side

import cv2 as cv

import numpy as np

protoxt\_path = 'Models/colorization\_deploy\_v2.prototxt'

model\_path = 'Models/colorization\_release\_v2.caffemodel'

kernel\_path = 'models/pts\_in\_hull.npy'

img\_path = 'Photos/bw.jpg'

# Load the pre-trained network

network = cv.dnn.readNetFromCaffe(protoxt\_path, model\_path)

points = np.load(kernel\_path)

# Process the points

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

network.getLayer(network.getLayerId("class8\_ab")).blobs = [points.astype(np.float32)]

network.getLayer(network.getLayerId('conv8\_313\_rh')).blobs = [np.full([1, 313], 2.606, np.float32)]

# Load and process the black and white image

bw\_img = cv.imread(img\_path)

normalized = bw\_img.astype("float32") / 255.0

lab = cv.cvtColor(normalized, cv.COLOR\_BGR2LAB)

# Resize the L channel and subtract 50

resized = cv.resize(lab, (224, 224))

L = cv.split(resized)[0]

L -= 50

# Set the input for the network and forward pass

network.setInput(cv.dnn.blobFromImage(L))

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

# Resize the ab channels and concatenate with the L channel

ab = cv.resize(ab, (bw\_img.shape[1], bw\_img.shape[0]))

L = cv.split(lab)[0]

colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)

colorized = cv.cvtColor(colorized, cv.COLOR\_LAB2BGR)

colorized = (255 \* colorized).astype("uint8")

# Display the original black and white and the colorized images

cv.imshow("BW Image", bw\_img)

cv.imshow("Colorized", colorized)

cv.waitKey(0)

2.Created Just by the thought of using LAB color space from base

a)Taking Colored Images and converting them to L and ab channels and creating two Lists putting L in one and ab channels in other and this being our training data

b)Training the model to output the ab channels given an L channel as input

c)Using CNN to train the model and finally doing the remaining part of converting to BGR and all same as the above

d)Finally Using this trained model to some TEST Data and testing on some b/w images

For Generating Data

def create\_training\_data(image\_paths):

L\_channels = []

ab\_channels = []

for image\_path in image\_paths:

L\_channel, ab\_channel = extract\_lab\_channels(image\_path)

L\_channels.append(L\_channel)

ab\_channels.append(ab\_channel)

# Convert lists to numpy arrays

L\_channels = np.array(L\_channels)

ab\_channels = np.array(ab\_channels)

return L\_channels, ab\_channels

##For creating the Model

import torch

import torch.nn as nn

import torch.optim as optim

class ColorizationNet(nn.Module):

def \_\_init\_\_(self):

super(ColorizationNet, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1, 64, kernel\_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1)

self.conv3 = nn.Conv2d(128, 128, kernel\_size=3, stride=1, padding=1)

self.conv4 = nn.Conv2d(128, 256, kernel\_size=3, stride=1, padding=1)

self.conv5 = nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1)

self.conv6 = nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1)

self.conv7 = nn.Conv2d(256, 512, kernel\_size=3, stride=1, padding=1)

self.conv8 = nn.Conv2d(512, 256, kernel\_size=3, stride=1, padding=1)

self.conv9 = nn.Conv2d(256, 128, kernel\_size=3, stride=1, padding=1)

self.conv10 = nn.Conv2d(128, 64, kernel\_size=3, stride=1, padding=1)

self.conv11 = nn.Conv2d(64, 32, kernel\_size=3, stride=1, padding=1)

self.conv12 = nn.Conv2d(32, 2, kernel\_size=3, stride=1, padding=1)

self.relu = nn.ReLU()

def forward(self, x):

x = self.relu(self.conv1(x))

x = self.relu(self.conv2(x))

x = self.relu(self.conv3(x))

x = self.relu(self.conv4(x))

x = self.relu(self.conv5(x))

x = self.relu(self.conv6(x))

x = self.relu(self.conv7(x))

x = self.relu(self.conv8(x))

x = self.relu(self.conv9(x))

x = self.relu(self.conv10(x))

x = self.relu(self.conv11(x))

x = self.conv12(x)

return x

# Initialize the model

model = ColorizationNet()

#For Training the Model

# Define loss function and optimizer

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# Convert data to PyTorch tensors

L\_train, ab\_train = torch.from\_numpy(L\_channels).unsqueeze(1).float(), torch.from\_numpy(ab\_channels).float()

# Training loop

num\_epochs = 10

for epoch in range(num\_epochs):

# Forward pass

outputs = model(L\_train)

# Compute loss

loss = criterion(outputs, ab\_train)

# Backward pass and optimization

optimizer.zero\_grad()

loss.backward()

optimizer.step()

print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}')

Some Basics I had to Learn from Open cv as being a beginner

**OpenCV Python ML coding**

import cv2 as cv

img = cv.imread('Photos/images.jpeg')

def rescaleFrame (frame, scale=0.75):

width = int(frame.shape [1] \* scale)

height = int(frame.shape[0] \* scale)

dimensions = (width,height)

return cv.resize (frame, dimensions, interpolation=cv.INTER\_AREA)

resized\_image = rescaleFrame(img)

cv.imshow('image' , resized\_image)

cv.waitKey(0)

(This is for displaying and capturing the image form vs code using open cv)

1)Data Type of an image is uint8

2)This code is for creating a blank black page/image

3)Here 500 are for width and height and 3 represents the colour RGB

Code==>

**blank = np. zeros ((500,500,3), dtype='uint8' )**

**cv. imshow( 'Blank', blank).**

**#1.Paint the image a certain colour**

**blank[:]= 0,255,0**

**cv. imshow(‘Green’,blank) // for a green image**

**// blank[200:300, 300:400] can use this to get the certain part only blank with any rub type**

import cv2 as cv

import numpy as np

img = cv.imread('Photos/image2.jpg')

# converting to graysclae

gray = cv.cvtColor(img , cv.COLOR\_BGR2GRAY)

cv.imshow('Gray' , gray)

#Blur

blur = cv.GaussianBlur(img,(7,7) , cv.BORDER\_DEFAULT)

cv.imshow('Blur' , blur)

1)This Canny Edges one below is important for the current ps with edge detection.

2)And if we input blur instead go img in canny below we will find way less edges(will only find the bigger ones).

#Edge Cascade(Edge Detector)

canny = cv.Canny(img, 125, 175)

cv.imshow('Canny Edges',canny)

cv.waitKey(0)

#Dilating the image

dilated = cv.dilate(canny, (7,7), iterations=3)

cv.imshow('Dilated' , dilated)

#Eroding (This is to remove the dilation)

eroded = cv.erode(dilated ,(3,3), iterations=1)

cv.imshow('Eroded', eroded)

#Resize

resized = cv.resize(img , (500,500) , interpolation=cv.INTER\_CUBIC)

cv.imshow('Resized' , resized)

#Cropping

cropped = img[50:200 , 200:400]

cv.imshow('Cropped' , cropped)

Using Contours Detection

# converting to grayscale

gray = cv.cvtColor(img , cv.COLOR\_BGR2GRAY)

blur = cv.GaussianBlur(gray , (7,7) , cv.BORDER\_DEFAULT)

# blur is done so as to reduce the number of contours

canny = cv.Canny(blur,125,175)

cv.imshow('Canny Edges' , canny)

contours, hierarchies = cv.findContours (canny, cv.RETR\_LIST, cv.CHAIN\_APPROX\_NONE)

print(f'{len(contours)} contour(s) found!')

50

a = Cyan to Magenta

- 100 = Cyan

0 = Neutral

100 = Magenta

b = Blue to Yellow

- 100 = Blue

0 = Neutral

100 = Yellow