Introduction

Image colourization is an interesting topic in image-to-image translation. Nowadays, many cameras are still capturing greyscale images, like surveillance cameras and satellite cameras. With colourization techniques, we can recover plausible coloured images for safety and research purpose. Besides, image colourization techniques can help us to colourize legacy images and videos to better understand the history. Generative Adversarial Network is a powerful model to produce plausible coloured images.

Colourizing black and white photos is currently a painstaking and labour-intensive process. It must be done manually in photoshop by a skilled graphic designer and the whole process can take a long time because it relies on the designer's imagination and efficiency to produce a realistic colourization. GAN's can circumvent this by developing their own "intuition" over thousands of training iterations. This "intuition" helps them recognize patterns in images and apply the correct colourization.

What is a GAN

A generative adversarial network or "GAN" is a neural network consisting of two sub models. These models work together to generate something, could be an image or even music, that to humans seems like the "real" thing.
The first sub model is the "Generator" and the second is the "Discriminator." After being pre-trained on what is real and what is noise, the Discriminator trains the Generator by revealing to it when it has created something realistic and when it hasn't. At first, the Generator will produce mostly noise, but eventually it will generate realistic results.

There are lots of types of GANs that researchers have given creative names to, such as DCGANs, HyperGans, CycleGANs, and S^2-GANs. Each are tweaked in certain ways to be more suitable to a specific task. However, all share the core principle of one net training the other to generate novel content.

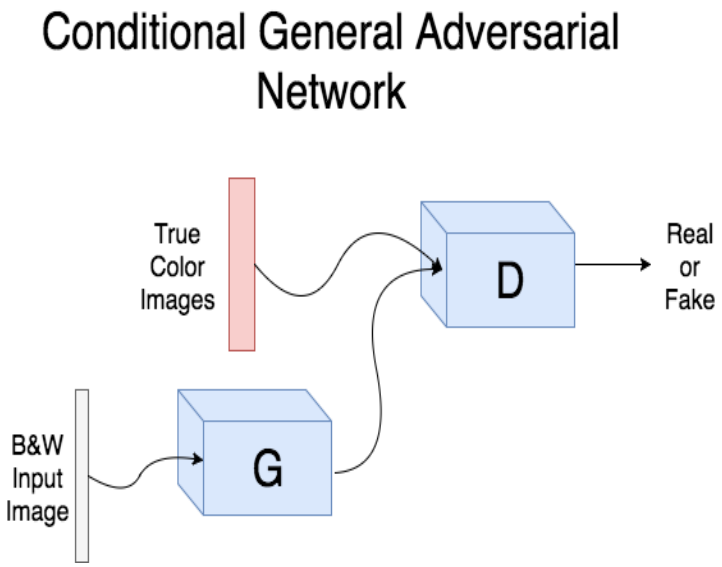


Figure 1: Conditional GAN

Model Architecture

Generator

The goal of the Generator is to create content so indistinguishable from the training set that the Discriminator cannot tell the difference. Below is the structure of the Generator in gan_32.py. The structure used in gan_256.py is similar, but much deeper to account for the images being 256x256 instead of 32x32.

Each encoding layer is a Convolution layer with stride 2 followed by a Batch Normalization layer and then a Leaky ReLU activation layer of slope .2. Each decoding layer is a an up-sampling layer followed by a convolution layer, Batch Normalization, and finally the concatenation layer.

The arrows illustrate how the early layers are concatenated with the later layers. These concatenations help preserve the structure of prominent edges that the decoding layers identified. They are called skip connections and are prevalent when using a neural network that finds the mapping between an input image and the output image.

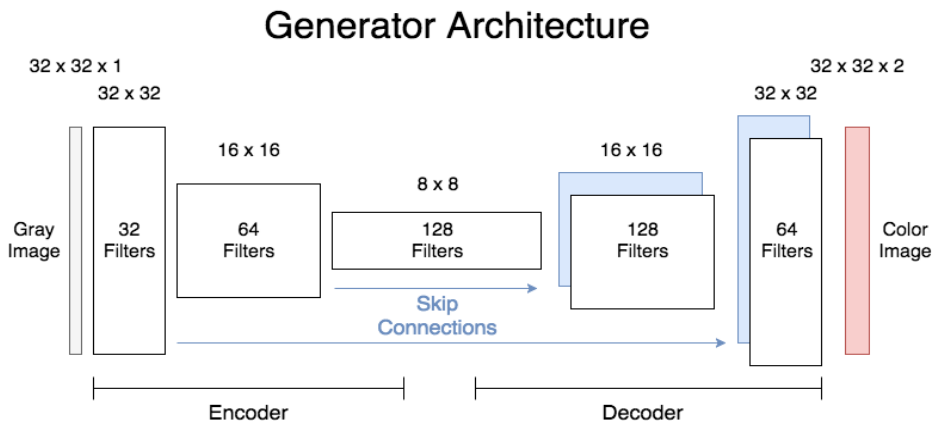


Figure 2: GAN Generator Architecture

Discriminator

The goal of the Discriminator is to be the expert on what a true image looks like. If it is fooled by the Generator too early then it is not doing its job well enough and as a result, will not be able to train the Generator well.

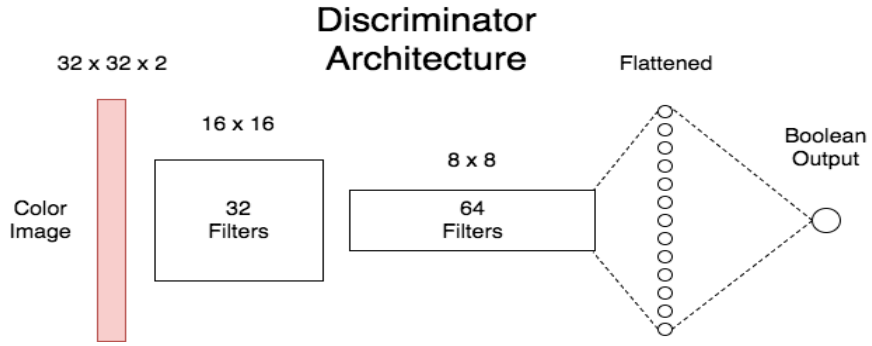


Figure 3: GAN Discriminator Architecture

GAN

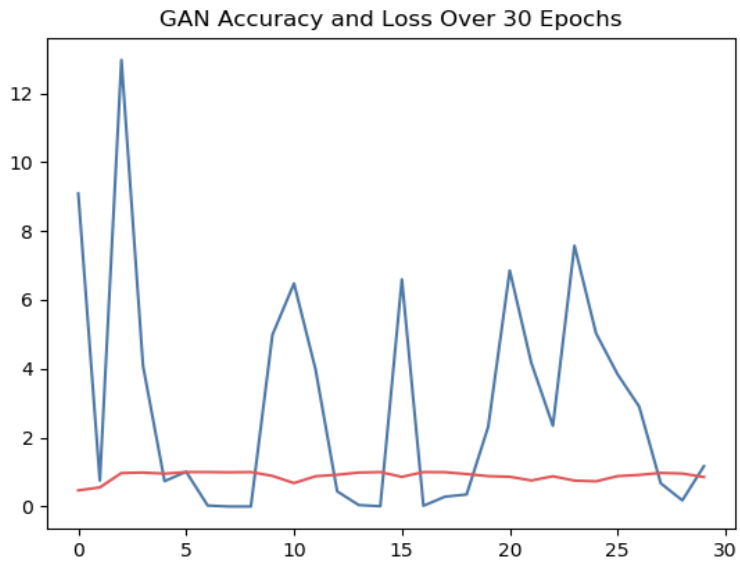
Here is a summary of the overall GAN architecture.

Layer	Output Shape	Params
Input	(256, 256, 1)	0
Generator	(256, 256, 2)	4,729,922
Discriminator	(1)	455,457

Total params: 5,185,379
Trainable params: 4,726,082
Non-trainable params: 459,297

How to Train a GAN

Training GANs is a complex operation and there are ongoing debates about the best methods to accomplish this. This is because the neural networks work together and so often you can see when training is going wrong rather than when training can be stopped because the colourizations are accurate. Below is a graph showing generator loss in blue and discriminator accuracy in red.



The goal of training is to keep the discriminator accuracy near 100% and make sure that the generator loss doesn't drop to 0. If the generator loss drops to 0 then it is fooling the discriminator with bad colourizations.

Colour Spectrums

RGB Colour Space

RGB colour space is a very common representation of colourful images in computer graphics. Specifically, a colourful image is represented by a 3-channel array. Each channel represents the chromaticity of Red, Green and Blue respectively.

Chromaticity means the quality of the colour regardless of its luminance. Red, green and blue are the additive primary colours. RGB colour map is capable to any chromaticity defined by these three primary colours. In computer graphics, every data point in every channel can take the value from 0 to 255, normally as integers. If this value is closer to 0, the colour of the corresponding channel looks darker. Several examples include Black: [R; G; B] = [0; 0; 0], White: [R; G; B] = [255; 255; 255], and Red: [R; G; B] = [255; 255; 255].

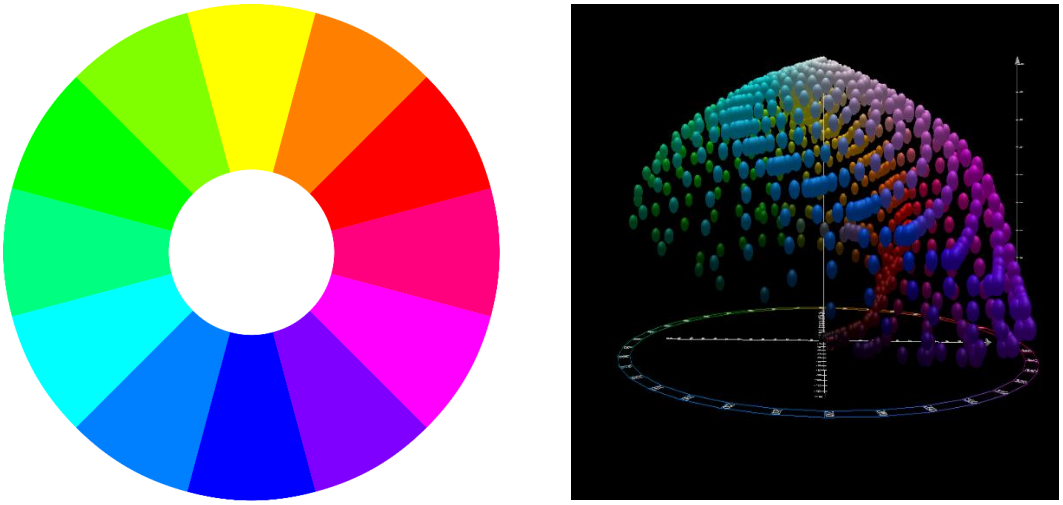


Figure 4: RGB Colour Space (Left), Lab Colour Space (Right)

Lab Colour Space

Lab colour space is another famous colour space. There are several variants of Lab colour space. The Lab in this paper refers to CIE Lab colour space. In Lab colour space, a colourful image is also represented as 3-channel data array.

Comparing with RGB, the difference is the meaning of each channel. The first channel L in Lab colour space represents lightness, which is a black/white colour intensity. It represents black at L = 0, and white at L = 100, and takes value from [0; 100]. The second channel a 2 [128; 127] is the red/green channel, with green at negative a value and red at positive a value. The third channel b 2 [128; 127] is the yellow/blue channel, with blue at negative values and yellow at positive values. When a = 0 or b = 0, both channels represent neutral grey colour.

Conditional GAN

Generative Adversarial Network (GAN) proposed by Goodfellowet al. [2] is a famous generative model. The traditional GAN has two parts: a generator (G) and a discriminator (D). For image generation, the generator is trained to generate images from noise to fool the discriminator. And the discriminator is trained to discriminate whether a image is real (from dataset) or fake (generated by G). To handle image colourization problem, we modify the traditional GAN into a Conditional GAN [5] to generate images from input images, instead of from noise.

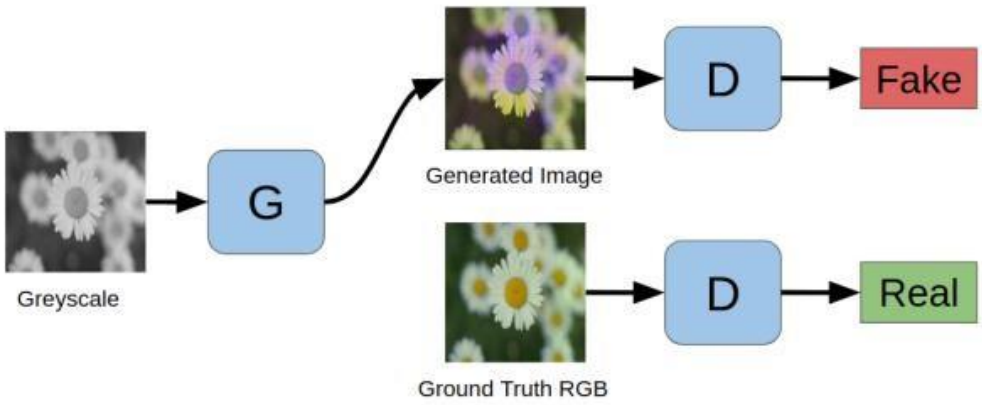


Figure 5: CGAN

Specifically, the generator takes greyscale images (lightness channel from Lab colour space) as inputs and generates colourized 3-channel RGB images. The discriminator is trained both on ground-truth RGB im-ages and the generated images, to determine whether the given image is generated. The overall objective for this Conditional GAN is a min-max game:

$$\min_{\theta_g} \max_{\theta_d} \Big[\mathbb{E}_{y \sim p_{data}} \log D_{\theta_d}(y) + \mathbb{E}_{x \sim p_{data}} \log\left(1 - D_{\theta_d}(G_{\theta_g}(x))\right) \Big]$$

where x represents the greyscale image and y represents the coloured image. g and d are the parameters for generator G and discriminator D respectively. The objective of generator is to minimize the function; and the objective of discriminator is to maximize the same function. That is the adversarial identity of GAN.

In order to constrain the modification made by GAN, an extra regularization term is added to the generator. Without this regularization term, GAN can still be trained but the generator will learn to fool the discriminator using some weird patterns which does not look plausible. Specifically, apart from maximizing the existing term, a L1 distance between generated image and ground truth is added to generator's loss function with a weight.

$$\min_{\theta_g}\left[\mathbb{E}_{x\sim p_{data}}\log\left(1-D_{\theta_d}\left(G_{\theta_g}(x)\right)\right)\right] + \lambda\left\|G_{\theta_g}(x)-y\right\|_1$$

There are many small tricks to train a stable GAN. For the architecture of generator, we still use a encoder-decoder scheme. But in the decoder layers, we add skip-connections between its mirror encoder layers by simply adding the feature map before moving on to the next layer. This technique is called U-Net [7] and can retain the details from the original images much easier. In encoder layers, I use convolutional layers with stride 2 to shrink the height and width. In decoder layers, similarly I use transpose-convolutional layers with stride 2 to expand the height and width.

Each block in encoder contains a convolutional layer, a batch normalization layer and a Leaky ReLU activation function. Each block in decoder contains a transpose-convolutional layer, a batch normalization layer and a Leaky ReLU activation except the last one. The last decoder layer has a transpose-convolutional layer and a tanH activation function to gives [1; 1] values in the prediction. The ground truth RGB images and input images are also normalized to [1; 1].

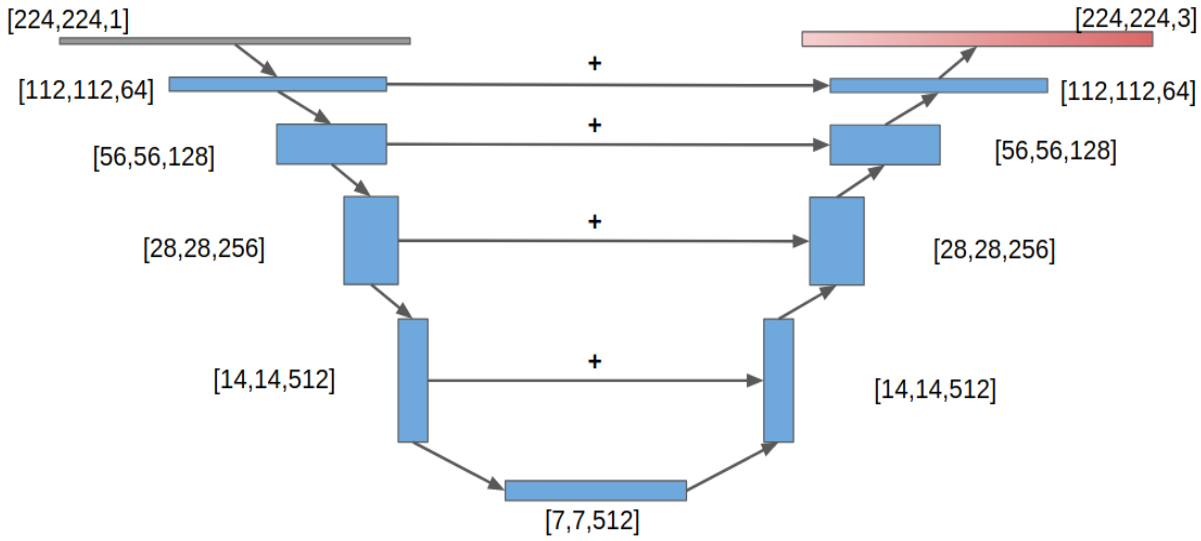
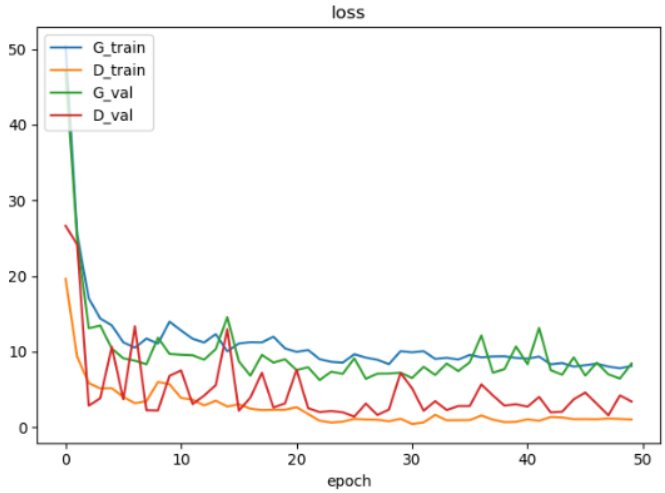


Figure 6: GAN Architecture

The architecture of discriminator only has the en-coder path. And in the final layer discriminator gives a 1-channel label indicating whether the given input is real or fake. Thus in the final layer a sigmoid activation function is used to bound the output to [0; 1] range. For other blocks in the encoder path, similarly I use a convolutional layer with stride 2, a batch normalization layer and a Leaky ReLU activation function. The value for the regularization weight is determined empirically.

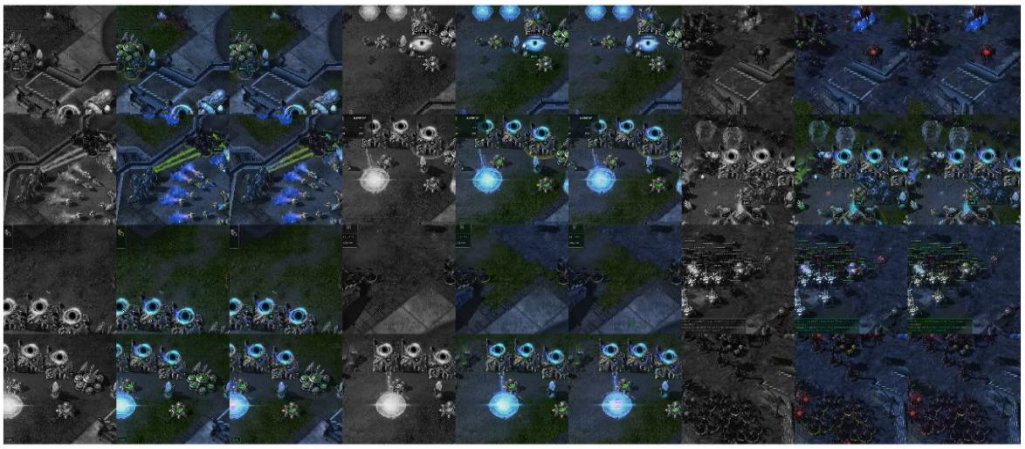
Results

For Conditional GAN, I train both generator and discriminator together in each batch, with same Adam optimizers and same 10^3 learning rates for both. The batch size is 32. After several experiments, I choose the regularization weight = 100, which means I put a large constrain on generating similar images. Without a large input, the model will generate very fake images.
An example training and validation curve of Conditional GAN is shown in the result images. We can see after several epochs, the loss for both generator and discriminator get decreased. Note the generator loss keeps being larger than discriminator loss, which is mainly because of the regularization term with weight 100.



This Conditional GAN obtains very plausible out-put on artificial dataset like SpongeBob and SC2. The reason might be that the colour of units in cartoon series and video games are pre-determined by human. Such that the colour distribution complies some pre-determined rules, which is easier to be captured by neural networks. A large region of mono-colour, GAN generates some checkerboard pattern coming from its convolutional kernels. This problem can be solved by applying further smoothing.

SC2Replay with 480x480 image size:



SpongeBob SquarePants with 224x224 image size:

OxFlower with 224x224 image size:

Future Work

There are several future research directions on image colorization. First is to increase the variety of colours. I verified that Euclidean distance will decreases the variety of colours. Zhang et al. [9] provide colourization methods, which can improve the variety. Besides, they proposed more methods like re-balancing the colour distribution, in order to encourage rare colours to appear. Second, most of the published dataset has labels. For example, the OxFlower dataset contains labels indication the category of flower.

Utilizing these models can be helpful for colorization. Intuitively, if the model can learn what the flower is be-fore colorization, the results are expected to be bet-ter. For example, a multi-task learning model might be very helpful in this case. Thirdly, we can explore the model performance on larger dataset to check the generalizability. Currently for efficiency purpose, my experiments are run on datasets with several thousands of images, which is quite small. All

the colourful images and videos can be used as datasets for colorization tasks. So theoretically, it is possible to verify colourization methods on huge datasets. Ideally, creating a generalized colorization model will be very attractive.

References

[1] Z. Cheng, Q. Yang, and B. Sheng. Deep colorization. CoRR, abs/1605.00075, 2016.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. arXiv, 2014.

[3] G. B. Huang, V. Jain, and E. Learned-Miller. Unsupervised joint alignment of complex images. In ICCV, 2007.

[4] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. CoRR, abs/1603.06668, 2016.

[5] M. Mirza and S. Osindero. Conditional generative adversarial nets. CoRR, abs/1411.1784, 2014.

[6] M.-E. Nilsback and A. Zisserman. A visual vocabulary for flower classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, volume 2, pages 1447–1454, 2006.

[7] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convo-lutional networks for biomedical image segmentation. CoRR, abs/1505.04597, 2015.

[8] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Kuttler,¨ J. Aga-piou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. Star-craft II: A new challenge for reinforcement learning. CoRR, abs/1708.04782, 2017.

[9] R. Zhang, P. Isola, and A. A. Efros. Colourful image colorization. CoRR, abs/1603.08511, 2016.

Project Implementation

Training (and validation)
Example command to train SC2 dataset:
python gan_main.py my_path --dataset sc2 --batch_size 32 --lr 1e-3 --num_epoch 50 --lamb 100 -s --gpu 0
Replace my_path by the root path of SC2 dataset.
Image samples created during validation will be saved in img/; and the model will be saved in model/ if -s option is used.

Testing
You can prepare a testing dataset and run command like:
python gan_main.py my_path --dataset sc2 --test my_path --gpu 0
to test the model with unseen images. Replace my_path by the path of the model that was saved during the training process.

Code
The project has 6 Python files which contain the bulk of the code. These are:

1. utils.py
2. transform.py
3. gan_model.py
4. gan_main.py
5. loss.py
6. load_data.py

utils.py

```
import os
import os.path as osp
import sys
import numpy as np
import pickle
from PIL import Image
import matplotlib.pyplot as plt
# plt.switch_backend('agg')
import time

import torch
import torchvision
from torchvision import transforms
from torch.utils import data
import scipy.io as io
import scipy.misc as misc
import glob
import csv
from skimage import color
from transform import ReLabel, ToLabel, ToSP, Scale

def pil_loader(path):
with open(path, 'rb') as f:
with Image.open(f) as img:
```

```
23. return img.convert('RGB')


24. class lfw_Dataset(data.Dataset):
25. def __init__(self, root,
26. shuffle=False,
27. small=False,
28. mode='test',
29. transform=None,
30. target_transform=None,
31. types='',
32. show_ab=False,
33. loader=pil_loader):

34. tic = time.time()
35. self.root = root
36. self.loader = loader
37. self.image_transform = transform
38. self.imgpath = glob.glob(root + 'lfw_funneled/*/*')
39. self.types = types
40. self.show_ab = show_ab # show ab channel in classify mode

41. # read split
42. self.train_people = set()
43. with open(self.root + 'peopleDevTrain.txt', 'r') as f:
44. reader = csv.reader(f, delimiter='\t')
45. for i, row in enumerate(reader):
46. if i == 0:
47. continue
48. self.train_people.add(row[0])
49. assert self.train_people.__len__() == 4038

50. self.test_people = set()
51. with open(self.root + 'peopleDevTest.txt', 'r') as f:
52. reader = csv.reader(f, delimiter='\t')
53. for i, row in enumerate(reader):
54. if i == 0:
55. continue
56. self.test_people.add(row[0])
57. assert self.test_people.__len__() == 1711

58. self.path = []
59. if mode == 'train':
60. for item in self.imgpath:
61. if item.split('/')[-2] in self.train_people:
62. self.path.append(item)
63. elif mode == 'test':
64. for item in self.imgpath:
65. if item.split('/')[-2] in self.test_people:
66. self.path.append(item)

67. np.random.seed(0)
68. if shuffle:
69. perm = np.random.permutation(len(self.path))
70. self.path = [self.path[i] for i in perm]

71. if types == 'classify':
72. ab_list = np.load('data/pts_in_hull.npy')
73. self.nbrs = NearestNeighbors(n_neighbors=1, algorithm='ball_tree').fit(ab_list)

74. print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))
```

```
75. def __getitem__(self, index):
76. mypath = self.path[index]
77. img = self.loader(mypath) # PIL Image
78. img = np.array(img)[13:13+224, 13:13+224, :]

79. img_lab = color.rgb2lab(np.array(img)) # np array
80. # img_lab = img_lab[13:13+224, 13:13+224, :]

81. if self.types == 'classify':
82. X_a = np.ravel(img_lab[:,:,1])
83. X_b = np.ravel(img_lab[:,:,2])
84. img_ab = np.vstack((X_a, X_b)).T
85. _, ind = self.nbrs.kneighbors(img_ab)
86. ab_class = np.reshape(ind, (224,224))
87. # print(ab_class.shape, ab_class.dtype, np.amax(ab_class), np.amin(ab_class))
88. ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)

89. img = (img - 127.5) / 127.5 # -1 to 1
90. img = torch.FloatTensor(np.transpose(img, (2,0,1)))
91. img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))

92. img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
93. img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

94. if self.types == 'classify':
95. if self.show_ab:
96. return img_l, ab_class, img_ab
97. return img_l, ab_class
98. elif self.types == 'raw':
99. return img_l, img
100.       else:
101.       return img_l, img_ab

102.       def __len__(self):
103.       return len(self.path)

104.       class Flower_Dataset(data.Dataset):
105.       def __init__(self, root,
106.       shuffle=False,
107.       small=False,
108.       mode='test',
109.       transform=None,
110.       target_transform=None,
111.       types='',
112.       show_ab=False,
113.       loader=pil_loader):

114.       tic = time.time()
115.       self.root = root
116.       self.loader = loader
117.       self.image_transform = transform
118.       self.imgpath = glob.glob(root + 'jpg/*.jpg')
119.       self.types = types
120.       self.show_ab = show_ab # show ab channel in classify mode

121.       # read split
122.       split_file = io.loadmat(root + 'datasplits.mat')

123.       self.train_file       =       set([str(i).zfill(4)       for       i       in
   np.hstack((split_file['trn1'][0], split_file['val1'][0]))])
124.       self.test_file = set([str(i).zfill(4) for i in split_file['tst1'][0]])
125.       assert self.train_file.__len__() == 1020
```

```
126.       assert self.test_file.__len__() == 340

127.       self.path = []
128.       if mode == 'train':
129.       for item in self.imgpath:
130.       if item.split('/')[-1][6:6+4] in self.train_file:
131.       self.path.append(item)
132.       elif mode == 'test':
133.       for item in self.imgpath:
134.       if item.split('/')[-1][6:6+4] in self.test_file:
135.       self.path.append(item)

136.       self.path = sorted(self.path)

137.       np.random.seed(0)
138.       if shuffle:
139.       perm = np.random.permutation(len(self.path))
140.       self.path = [self.path[i] for i in perm]

141.       if types == 'classify':
142.       ab_list = np.load('data/pts_in_hull.npy')
143.       self.nbrs                    =                    NearestNeighbors(n_neighbors=1,
    algorithm='ball_tree').fit(ab_list)

144.       print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))

145.       def __getitem__(self, index):
146.       mypath = self.path[index]
147.       img = self.loader(mypath) # PIL Image
148.       img = np.array(img)
149.       img = misc.imresize(img, (224, 224))

150.       img_lab = color.rgb2lab(np.array(img)) # np array
151.       # img_lab = img_lab[13:13+224, 13:13+224, :]

152.       if self.types == 'classify':
153.       X_a = np.ravel(img_lab[:,:,1])
154.       X_b = np.ravel(img_lab[:,:,2])
155.       img_ab = np.vstack((X_a, X_b)).T
156.       _, ind = self.nbrs.kneighbors(img_ab)
157.       ab_class = np.reshape(ind, (224,224))
158.       ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)

159.       img = (img - 127.5) / 127.5 # -1 to 1
160.       img = torch.FloatTensor(np.transpose(img, (2,0,1)))
161.       img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))

162.       img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
163.       img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

164.       if self.types == 'classify':
165.       if self.show_ab:
166.       return img_l, ab_class, img_ab
167.       return img_l, ab_class
168.       elif self.types == 'raw':
169.       return img_l, img
170.       # if self.show_ab:
171.       #     return img_l, img_ab, None
172.       else:
173.       return img_l, img_ab

174.       def __len__(self):
```

```
175.        return len(self.path)

176.        class Spongebob_Dataset(data.Dataset):
177.        def __init__(self, root,
178.        shuffle=False,
179.        small=False,
180.        mode='test',
181.        transform=None,
182.        target_transform=None,
183.        types='',
184.        show_ab=False,
185.        large=False,
186.        loader=pil_loader):

187.        tic = time.time()
188.        self.root = root
189.        self.loader = loader
190.        self.image_transform = transform
191.        if large:
192.        self.size = 480
193.        self.imgpath = glob.glob(root + 'img_480/*.png')
194.        else:
195.        self.size = 224
196.        self.imgpath = glob.glob(root + 'img/*.png')
197.        self.types = types
198.        self.show_ab = show_ab # show ab channel in classify mode

199.        # read split
200.        self.train_file = set()
201.        with open(self.root + 'train_split.csv', 'r') as f:
202.        reader = csv.reader(f, delimiter='\t')
203.        for i, row in enumerate(reader):
204.        if i == 0:
205.        continue
206.        self.train_file.add(str(row[0]).zfill(4))

207.        assert self.train_file.__len__() == 1392

208.        self.test_file = set()
209.        with open(self.root + 'test_split.csv', 'r') as f:
210.        reader = csv.reader(f, delimiter='\t')
211.        for i, row in enumerate(reader):
212.        if i == 0:
213.        continue
214.        self.test_file.add(str(row[0]).zfill(4))
215.        assert self.test_file.__len__() == 348

216.        self.path = []
217.        if mode == 'train':
218.        for item in self.imgpath:
219.        if item.split('/')[-1][6:6+4] in self.train_file:
220.        self.path.append(item)
221.        elif mode == 'test':
222.        for item in self.imgpath:
223.        if item.split('/')[-1][6:6+4] in self.test_file:
224.        self.path.append(item)

225.        self.path = sorted(self.path)

226.        np.random.seed(0)
227.        if shuffle:
228.        perm = np.random.permutation(len(self.path))
```

```
229.       self.path = [self.path[i] for i in perm]

230.       if types == 'classify':
231.       ab_list = np.load('data/pts_in_hull.npy')
232.       self.nbrs                  =                  NearestNeighbors(n_neighbors=1,
    algorithm='ball_tree').fit(ab_list)

233.       print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))

234.       def __getitem__(self, index):
235.       mypath = self.path[index]
236.       img = self.loader(mypath) # PIL Image
237.       img = np.array(img)
238.       if (img.shape[0] != self.size) or (img.shape[1] != self.size):
239.       img = misc.imresize(img, (self.size, self.size))

240.       img_lab = color.rgb2lab(np.array(img)) # np array
241.       # img_lab = img_lab[13:13+224, 13:13+224, :]

242.       if self.types == 'classify':
243.       X_a = np.ravel(img_lab[:,:,1])
244.       X_b = np.ravel(img_lab[:,:,2])
245.       img_ab = np.vstack((X_a, X_b)).T
246.       _, ind = self.nbrs.kneighbors(img_ab)
247.       ab_class = np.reshape(ind, (self.size,self.size))
248.       # print(ab_class.shape, ab_class.dtype, np.amax(ab_class), np.amin(ab_class))
249.       ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)

250.       img = (img - 127.5) / 127.5 # -1 to 1
251.       img = torch.FloatTensor(np.transpose(img, (2,0,1)))
252.       img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))

253.       img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
254.       img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

255.       if self.types == 'classify':
256.       if self.show_ab:
257.       return img_l, ab_class, img_ab
258.       return img_l, ab_class
259.       elif self.types == 'raw':
260.       return img_l, img
261.       # if self.show_ab:
262.       #     return img_l, img_ab, None
263.       else:
264.       return img_l, img_ab

265.       def __len__(self):
266.       return len(self.path)

267.       class SC2_Dataset(data.Dataset):
268.       def __init__(self, root,
269.       shuffle=False,
270.       small=False,
271.       mode='test',
272.       transform=None,
273.       target_transform=None,
274.       types='',
275.       show_ab=False,
276.       large=False,
277.       loader=pil_loader):

278.       tic = time.time()
```

```
279.        self.root = root
280.        self.loader = loader
281.        self.image_transform = transform
282.        if large:
283.        self.size = 480
284.        self.imgpath = glob.glob(root + 'img_480/*.png')
285.        else:
286.        self.size = 224
287.        self.imgpath = glob.glob(root + 'img/*.png')
288.        self.types = types
289.        self.show_ab = show_ab # show ab channel in classify mode

290.        # read split
291.        self.train_file = set()
292.        with open(self.root + 'train_split.csv', 'r') as f:
293.        reader = csv.reader(f, delimiter='\t')
294.        for i, row in enumerate(reader):
295.        if i == 0:
296.        continue
297.        self.train_file.add(str(row[0]).zfill(4))
298.        assert self.train_file.__len__() == 1383

299.        self.test_file = set()
300.        with open(self.root + 'test_split.csv', 'r') as f:
301.        reader = csv.reader(f, delimiter='\t')
302.        for i, row in enumerate(reader):
303.        if i == 0:
304.        continue
305.        self.test_file.add(str(row[0]).zfill(4))
306.        assert self.test_file.__len__() == 345

307.        self.path = []
308.        if mode == 'train':
309.        for item in self.imgpath:
310.        if item.split('/')[-1][6:6+4] in self.train_file:
311.        self.path.append(item)
312.        elif mode == 'test':
313.        for item in self.imgpath:
314.        if item.split('/')[-1][6:6+4] in self.test_file:
315.        self.path.append(item)

316.        self.path = sorted(self.path)

317.        np.random.seed(0)
318.        if shuffle:
319.        perm = np.random.permutation(len(self.path))
320.        self.path = [self.path[i] for i in perm]

321.        if types == 'classify':
322.        ab_list = np.load('data/pts_in_hull.npy')
323.        self.nbrs                    =                    NearestNeighbors(n_neighbors=1,
    algorithm='ball_tree').fit(ab_list)

324.        print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))

325.        def __getitem__(self, index):
326.        mypath = self.path[index]
327.        img = self.loader(mypath) # PIL Image
328.        img = np.array(img)
329.        if (img.shape[0] != self.size) or (img.shape[1] != self.size):
330.        img = misc.imresize(img, (self.size, self.size))
```

```
331.        img_lab = color.rgb2lab(np.array(img)) # np array
332.        # img_lab = img_lab[13:13+224, 13:13+224, :]

333.        if self.types == 'classify':
334.        X_a = np.ravel(img_lab[:,:,1])
335.        X_b = np.ravel(img_lab[:,:,2])
336.        img_ab = np.vstack((X_a, X_b)).T
337.        _, ind = self.nbrs.kneighbors(img_ab)
338.        ab_class = np.reshape(ind, (self.size,self.size))
339.        # print(ab_class.shape, ab_class.dtype, np.amax(ab_class), np.amin(ab_class))
340.        ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)

341.        img = (img - 127.5) / 127.5 # -1 to 1
342.        img = torch.FloatTensor(np.transpose(img, (2,0,1)))
343.        img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))

344.        img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
345.        img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

346.        if self.types == 'classify':
347.        if self.show_ab:
348.        return img_l, ab_class, img_ab
349.        return img_l, ab_class
350.        elif self.types == 'raw':
351.        if img.size(1) == 479 or img.size(2) == 479:
352.        print(mypath)
353.        return img_l, img
354.        # if self.show_ab:
355.        #     return img_l, img_ab, None
356.        else:
357.        return img_l, img_ab

358.        def __len__(self):
359.        return len(self.path)


360.        if __name__ == '__main__':
361.        data_root = '/home/users/u5612799/DATA/SCReplay/'
362.        # normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
363.        #                                  std=[0.229, 0.224, 0.225])

364.        image_transform = transforms.Compose([
365.        transforms.CenterCrop(224),
366.        transforms.ToTensor(),
367.        ])

368.        lfw = SC2_Dataset(data_root, mode='train',
369.        transform=image_transform, large=True, types='raw')

370.        data_loader = data.DataLoader(lfw,
371.        batch_size=1,
372.        shuffle=False,
373.        num_workers=4)

374.        for i, (data, target) in enumerate(data_loader):
375.        print(i, len(lfw))
```

transform.py

```
1.  import numpy as np
2.  import torch
3.  from PIL import Image
4.  import collections


5.  class Scale(object):
6.  def __init__(self, size, interpolation=Image.BILINEAR):
7.  assert isinstance(size, int) or (isinstance(size, collections.Iterable) and len(size)
    == 2)
8.  self.size = size
9.  self.interpolation = interpolation

10. def __call__(self, img):
11. if isinstance(self.size, int):
12. w, h = img.size
13. if (w <= h and w == self.size) or (h <= w and h == self.size):
14. return img
15. if w < h:
16. ow = self.size
17. oh = int(self.size * h / w)
18. return img.resize((ow, oh), self.interpolation)
19. else:
20. oh = self.size
21. ow = int(self.size * w / h)
22. return img.resize((ow, oh), self.interpolation)
23. else:
24. # import ipdb; ipdb.set_trace()
25. return img.resize(self.size, self.interpolation)


26. class ToParallel(object):
27. def __init__(self, transforms):
28. self.transforms = transforms

29. def __call__(self, img):
30. yield img
31. for t in self.transforms:
32. yield t(img)


33. class ToLabel_32(object):
34. def __call__(self, inputs):
35. tensor = torch.from_numpy(np.array(inputs)).long()
36. return tensor

37. class ToLabel(object):
38. def __call__(self, inputs):
39. tensors = []
40. for i in inputs:
41. tensors.append(torch.from_numpy(np.array(i)).long())
42. return tensors


43. class ReLabel(object):
44. def __init__(self, olabel, nlabel):
45. self.olabel = olabel
46. self.nlabel = nlabel
```

```
47. def __call__(self, inputs):
48. # assert isinstance(input, torch.LongTensor), 'tensor needs to be LongTensor'
49. for i in inputs:
50. i[i == self.olabel] = self.nlabel
51. return inputs


52. class ToSP(object):
53. def __init__(self, size):
54. self.scale2 = Scale(size/2, Image.NEAREST)
55. self.scale4 = Scale(size/4, Image.NEAREST)
56. self.scale8 = Scale(size/8, Image.NEAREST)
57. self.scale16 = Scale(size/16, Image.NEAREST)
58. self.scale32 = Scale(size/32, Image.NEAREST)

59. def __call__(self, input):
60. input2 = self.scale2(input)
61. input4 = self.scale4(input)
62. input8 = self.scale8(input)
63. input16 = self.scale16(input)
64. input32 = self.scale32(input)
65. inputs = [input, input2, input4, input8, input16, input32]
66. # inputs = [input]

67. return inputs


68. class HorizontalFlip(object):
69. """Horizontally flips the given PIL.Image with a probability of 0.5."""

70. def __call__(self, img):
71. return img.transpose(Image.FLIP_LEFT_RIGHT)


72. class VerticalFlip(object):
73. def __call__(self, img):
74. return img.transpose(Image.FLIP_TOP_BOTTOM)

75. def uint82bin(n, count=8):
76. """returns the binary of integer n, count refers to amount of bits"""
77. return ''.join([str((n >> y) & 1) for y in range(count-1, -1, -1)])

78. def labelcolormap(N):
79. cmap = np.zeros((N, 3), dtype=np.uint8)
80. for i in range(N):
81. r = 0
82. g = 0
83. b = 0
84. id = i
85. for j in range(7):
86. str_id = uint82bin(id)
87. r = r ^ (np.uint8(str_id[-1]) << (7-j))
88. g = g ^ (np.uint8(str_id[-2]) << (7-j))
89. b = b ^ (np.uint8(str_id[-3]) << (7-j))
90. id = id >> 3
91. cmap[i, 0] = r
92. cmap[i, 1] = g
93. cmap[i, 2] = b
94. return cmap
```

```
def colormap(n):
cmap = np.zeros([n, 3]).astype(np.uint8)

for i in np.arange(n):
r, g, b = np.zeros(3)

for j in np.arange(8):
        r = r + (1 << (7-j))*((i & (1 << (3*j))) >> (3*j))
        g = g + (1 << (7-j))*((i & (1 << (3*j+1))) >> (3*j+1))
        b = b + (1 << (7-j))*((i & (1 << (3*j+2))) >> (3*j+2))

        cmap[i, :] = np.array([r, g, b])

        return cmap


        class Colorize(object):
        def __init__(self, n=22):
        self.cmap = labelcolormap(22)
        self.cmap = torch.from_numpy(self.cmap[:n])

        def __call__(self, gray_image):
        size = gray_image.size()
        color_image = torch.ByteTensor(3, size[1], size[2]).fill_(0)

        for label in range(0, len(self.cmap)):
        mask = (label == gray_image[0]).cpu()
        color_image[0][mask] = self.cmap[label][0]
        color_image[1][mask] = self.cmap[label][1]
        color_image[2][mask] = self.cmap[label][2]

        return color_image
```

gan_model.py

```
import math
import torch
import torch.nn as nn
import torch.nn.functional as F


class ConvGen(nn.Module):
'''Generator'''
def __init__(self):
super(ConvGen, self).__init__()

self.conv1 = nn.Conv2d(1, 64, 3, stride=2, padding=1, bias=False)
self.bn1 = nn.BatchNorm2d(64)
self.relu1 = nn.LeakyReLU(0.1)

self.conv2 = nn.Conv2d(64, 128, 3, stride=2, padding=1, bias=False)
self.bn2 = nn.BatchNorm2d(128)
self.relu2 = nn.LeakyReLU(0.1)

self.conv3 = nn.Conv2d(128, 256, 3, stride=2, padding=1, bias=False)
```

```
16. self.bn3 = nn.BatchNorm2d(256)
17. self.relu3 = nn.LeakyReLU(0.1)

18. self.conv4 = nn.Conv2d(256, 512, 3, stride=2, padding=1, bias=False)
19. self.bn4 = nn.BatchNorm2d(512)
20. self.relu4 = nn.LeakyReLU(0.1)

21. self.conv5 = nn.Conv2d(512, 512, 3, stride=2, padding=1, bias=False)
22. self.bn5 = nn.BatchNorm2d(512)
23. self.relu5 = nn.LeakyReLU(0.1)

24. self.deconv6 = nn.ConvTranspose2d(512, 512, 3, stride=2, padding=1, output_padding=1,
    bias=False)
25. self.bn6 = nn.BatchNorm2d(512)
26. self.relu6 = nn.ReLU()

27. self.deconv7 = nn.ConvTranspose2d(512, 256, 3, stride=2, padding=1, output_padding=1,
    bias=False)
28. self.bn7 = nn.BatchNorm2d(256)
29. self.relu7 = nn.ReLU()

30. self.deconv8 = nn.ConvTranspose2d(256, 128, 3, stride=2, padding=1, output_padding=1,
    bias=False)
31. self.bn8 = nn.BatchNorm2d(128)
32. self.relu8 = nn.ReLU()

33. self.deconv9 = nn.ConvTranspose2d(128, 64, 3, stride=2, padding=1, output_padding=1,
    bias=False)
34. self.bn9 = nn.BatchNorm2d(64)
35. self.relu9 = nn.ReLU()

36. self.deconv10 = nn.ConvTranspose2d(64, 3, 3, stride=2, padding=1, output_padding=1,
    bias=False)
37. self.bn10 = nn.BatchNorm2d(3)
38. self.relu10 = nn.ReLU()

39. self._initialize_weights()

40. def forward(self, x):
41. h = x
42. h = self.conv1(h)
43. h = self.bn1(h)
44. h = self.relu1(h) # 64,112,112 (if input is 224x224)
45. pool1 = h

46. h = self.conv2(h)
47. h = self.bn2(h)
48. h = self.relu2(h) # 128,56,56
49. pool2 = h

50. h = self.conv3(h) # 256,28,28
51. h = self.bn3(h)
52. h = self.relu3(h)
53. pool3 =h

54. h = self.conv4(h) # 512,14,14
55. h = self.bn4(h)
56. h = self.relu4(h)
57. pool4 = h
```

```
58. h = self.conv5(h) # 512,7,7
59. h = self.bn5(h)
60. h = self.relu5(h)

61. h = self.deconv6(h)
62. h = self.bn6(h)
63. h = self.relu6(h) # 512,14,14
64. h += pool4

65. h = self.deconv7(h)
66. h = self.bn7(h)
67. h = self.relu7(h) # 256,28,28
68. h += pool3

69. h = self.deconv8(h)
70. h = self.bn8(h)
71. h = self.relu8(h) # 128,56,56
72. h += pool2

73. h = self.deconv9(h)
74. h = self.bn9(h)
75. h = self.relu9(h) # 64,112,112
76. h += pool1

77. h = self.deconv10(h)
78. h = F.tanh(h) # 3,224,224

79. return h

80. def _initialize_weights(self):
81. for m in self.modules():
82. if isinstance(m, nn.Conv2d):
83. n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
84. m.weight.data.normal_(0, math.sqrt(2. / n))
85. if isinstance(m, nn.ConvTranspose2d):
86. n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
87. m.weight.data.normal_(0, math.sqrt(2. / n))

88. class ConvDis(nn.Module):
89. '''Discriminator'''
90. def __init__(self, large=False):
91. super(ConvDis, self).__init__()

92. self.conv1 = nn.Conv2d(3, 64, 3, stride=2, padding=1, bias=False)
93. self.bn1 = nn.BatchNorm2d(64)
94. self.relu1 = nn.LeakyReLU(0.1)

95. self.conv2 = nn.Conv2d(64, 128, 3, stride=2, padding=1, bias=False)
96. self.bn2 = nn.BatchNorm2d(128)
97. self.relu2 = nn.LeakyReLU(0.1)

98. self.conv3 = nn.Conv2d(128, 256, 3, stride=2, padding=1, bias=False)
99. self.bn3 = nn.BatchNorm2d(256)
100.       self.relu3 = nn.LeakyReLU(0.1)

101.       self.conv4 = nn.Conv2d(256, 512, 3, stride=2, padding=1, bias=False)
102.       self.bn4 = nn.BatchNorm2d(512)
103.       self.relu4 = nn.LeakyReLU(0.1)
```

```
104.        self.conv5 = nn.Conv2d(512, 512, 3, stride=2, padding=1, bias=False)
105.        self.bn5 = nn.BatchNorm2d(512)
106.        self.relu5 = nn.LeakyReLU(0.1)

107.        if large:
108.        self.conv6 = nn.Conv2d(512, 512, 15, stride=1, padding=0, bias=False)
109.        else:
110.        self.conv6 = nn.Conv2d(512, 512, 7, stride=1, padding=0, bias=False)
111.        self.bn6 = nn.BatchNorm2d(512)
112.        self.relu6 = nn.LeakyReLU(0.1)

113.        self.conv7 = nn.Conv2d(512, 1, 1, stride=1, padding=0, bias=False)

114.        self._initialize_weights()

115.        def forward(self, x):
116.        h = x
117.        h = self.conv1(h)
118.        h = self.bn1(h)
119.        h = self.relu1(h) # 64,112,112 (if input is 224x224)

120.        h = self.conv2(h)
121.        h = self.bn2(h)
122.        h = self.relu2(h) # 128,56,56

123.        h = self.conv3(h) # 256,28,28
124.        h = self.bn3(h)
125.        h = self.relu3(h)

126.        h = self.conv4(h) # 512,14,14
127.        h = self.bn4(h)
128.        h = self.relu4(h)

129.        h = self.conv5(h) # 512,7,7
130.        h = self.bn5(h)
131.        h = self.relu5(h)

132.        h = self.conv6(h)
133.        h = self.bn6(h)
134.        h = self.relu6(h) # 512,1,1

135.        h = self.conv7(h)
136.        h = F.sigmoid(h)

137.        return h

138.        def _initialize_weights(self):
139.        for m in self.modules():
140.        if isinstance(m, nn.Conv2d):
141.        n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
142.        m.weight.data.normal_(0, math.sqrt(2. / n))
143.        if isinstance(m, nn.ConvTranspose2d):
144.        n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
145.        m.weight.data.normal_(0, math.sqrt(2. / n))
```

gan_main.py

```
1.  # from transform import ReLabel, ToLabel, ToSP, Scale
```

```
from gan_model import *
from utils import *

import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
from torch.utils import data
import torch.nn.functional as F
import torchvision
from torchvision import datasets, models, transforms
from skimage import color

import time
import os
import sys
from PIL import Image
import argparse
import numpy as np
import matplotlib.pyplot as plt


parser = argparse.ArgumentParser(description='Colorization using GAN')
parser.add_argument('path', type=str,
help='Root path for dataset')
parser.add_argument('--dataset', type=str,
help='which dataset?', choices=['sc2','flower','bob'])
parser.add_argument('--large', action="store_true",
help='Use larger images?')
parser.add_argument('--batch_size', default=4, type=int,
help='Batch size: default 4')
parser.add_argument('--lr', default=1e-4, type=float,
help='Learning rate for optimizer')
parser.add_argument('--weight_decay', default=0, type=float,
help='Weight decay for optimizer')
parser.add_argument('--num_epoch', default=20, type=int,
help='Number of epochs')
parser.add_argument('--lamb', default=100, type=int,
help='Lambda for L1 Loss')
parser.add_argument('--test', default='', type=str,
help='Path to the model, for testing')
parser.add_argument('--model_G', default='', type=str,
help='Path to resume for Generator model')
parser.add_argument('--model_D', default='', type=str,
help='Path to resume for Discriminator model')

# parser.add_argument('-p', '--plot', action="store_true",
#                     help='Plot accuracy and loss diagram?')
parser.add_argument('-s','--save', action="store_true",
help='Save model?')
parser.add_argument('--gpu', default=0, type=int,
help='Which GPU to use?')

def main():
global args, date
args = parser.parse_args()
date = '1220'

os.environ["CUDA_VISIBLE_DEVICES"]=str(args.gpu)
```

```
model_G = ConvGen()
model_D = ConvDis(large=args.large)

start_epoch_G = start_epoch_D = 0
if args.model_G:
print('Resume model G: %s' % args.model_G)
checkpoint_G = torch.load(resume)
model_G.load_state_dict(checkpoint_G['state_dict'])
start_epoch_G = checkpoint_G['epoch']
if args.model_D:
print('Resume model D: %s' % args.model_D)
checkpoint_D = torch.load(resume)
model_D.load_state_dict(checkpoint_D['state_dict'])
start_epoch_D = checkpoint_D['epoch']
assert start_epoch_G == start_epoch_D
if args.model_G == '' and args.model_D == '':
print('No Resume')
start_epoch = 0

model_G.cuda()
model_D.cuda()

# optimizer
optimizer_G = optim.Adam(model_G.parameters(),
lr=args.lr, betas=(0.5, 0.999),
eps=1e-8, weight_decay=args.weight_decay)
optimizer_D = optim.Adam(model_D.parameters(),
lr=args.lr, betas=(0.5, 0.999),
eps=1e-8, weight_decay=args.weight_decay)
if args.model_G:
optimizer_G.load_state_dict(checkpoint_G['optimizer'])
if args.model_D:
optimizer_D.load_state_dict(checkpoint_D['optimizer'])

# loss function
global criterion
criterion = nn.BCELoss()
global L1
L1 = nn.L1Loss()

# dataset
# data_root = '/home/users/u5612799/DATA/Spongebob/'
data_root = args.path
dataset = args.dataset
if dataset == 'sc2':
from load_data import SC2_Dataset as myDataset
elif dataset == 'flower':
from load_data import Flower_Dataset as myDataset
elif dataset == 'bob':
from load_data import Spongebob_Dataset as myDataset
else:
        raise ValueError('dataset type not supported')

        if args.large:
        image_transform = transforms.Compose([transforms.CenterCrop(480),
        transforms.ToTensor()])
        else:
        image_transform = transforms.Compose([transforms.CenterCrop(224),
```

```
106.        transforms.ToTensor()])

107.        data_train = myDataset(data_root, mode='train',
108.        transform=image_transform,
109.        types='raw',
110.        shuffle=True,
111.        large=args.large
112.        )

113.        train_loader = data.DataLoader(data_train,
114.        batch_size=args.batch_size,
115.        shuffle=False,
116.        num_workers=4)

117.        data_val = myDataset(data_root, mode='test',
118.        transform=image_transform,
119.        types='raw',
120.        shuffle=True,
121.        large=args.large
122.        )

123.        val_loader = data.DataLoader(data_val,
124.        batch_size=args.batch_size,
125.        shuffle=False,
126.        num_workers=4)

127.        global val_bs
128.        val_bs = val_loader.batch_size

129.        # set up plotter, path, etc.
130.        global iteration, print_interval, plotter, plotter_basic
131.        iteration = 0
132.        print_interval = 5
133.        plotter = Plotter_GAN_TV()
134.        plotter_basic = Plotter_GAN()

135.        global img_path
136.        size = ''
137.        if args.large: size = '_Large'
138.        img_path = 'img/%s/GAN_%s%s_%dL1_bs%d_%s_lr%s/' \
139.        % (date, args.dataset, size, args.lamb, args.batch_size, 'Adam', str(args.lr))
140.        model_path = 'model/%s/GAN_%s%s_%dL1_bs%d_%s_lr%s/' \
141.        % (date, args.dataset, size, args.lamb, args.batch_size, 'Adam', str(args.lr))
142.        if not os.path.exists(img_path):
143.        os.makedirs(img_path)
144.        if not os.path.exists(model_path):
145.        os.makedirs(model_path)

146.        # start loop
147.        start_epoch = 0

148.        for epoch in range(start_epoch, args.num_epoch):
149.        print('Epoch {}/{}'.format(epoch, args.num_epoch - 1))
150.        print('-' * 20)
151.        if epoch == 0:
152.        val_lerrG, val_errD = validate(val_loader, model_G, model_D, optimizer_G,
    optimizer_D, epoch=-1)
153.        # train
```

```
154.       train_errG, train_errD = train(train_loader, model_G, model_D, optimizer_G,
    optimizer_D, epoch, iteration)
155.       # validate
156.       val_lerrG, val_errD = validate(val_loader, model_G, model_D, optimizer_G,
    optimizer_D, epoch)

157.       plotter.train_update(train_errG, train_errD)
158.       plotter.val_update(val_lerrG, val_errD)
159.       plotter.draw(img_path + 'train_val.png')

160.       if args.save:
161.       print('Saving check point')
162.       save_checkpoint({'epoch': epoch + 1,
163.       'state_dict': model_G.state_dict(),
164.       'optimizer': optimizer_G.state_dict(),
165.       },
166.       filename=model_path+'G_epoch%d.pth.tar' \
167.       % epoch)
168.       save_checkpoint({'epoch': epoch + 1,
169.       'state_dict': model_D.state_dict(),
170.       'optimizer': optimizer_D.state_dict(),
171.       },
172.       filename=model_path+'D_epoch%d.pth.tar' \
173.       % epoch)



174.       def train(train_loader, model_G, model_D, optimizer_G, optimizer_D, epoch,
    iteration):
175.       errorG = AverageMeter() # will be reset after each epoch
176.       errorD = AverageMeter() # will be reset after each epoch
177.       errorG_basic = AverageMeter() # basic will be reset after each print
178.       errorD_basic = AverageMeter() # basic will be reset after each print
179.       errorD_real = AverageMeter()
180.       errorD_fake = AverageMeter()
181.       errorG_GAN = AverageMeter()
182.       errorG_R = AverageMeter()

183.       model_G.train()
184.       model_D.train()

185.       real_label = 1
186.       fake_label = 0

187.       for i, (data, target) in enumerate(train_loader):
188.       data, target = Variable(data.cuda()), Variable(target.cuda())

189.       ########################
190.       # update D network
191.       ########################
192.       # train with real
193.       model_D.zero_grad()
194.       output = model_D(target)
195.       label = torch.FloatTensor(target.size(0)).fill_(real_label).cuda()
196.       labelv = Variable(label)
197.       errD_real = criterion(torch.squeeze(output), labelv)
198.       errD_real.backward()
199.       D_x = output.data.mean()
```

```
200.        # train with fake
201.        fake =  model_G(data)
202.        labelv = Variable(label.fill_(fake_label))
203.        output = model_D(fake.detach())
204.        errD_fake = criterion(torch.squeeze(output), labelv)
205.        errD_fake.backward()
206.        D_G_x1 = output.data.mean()

207.        errD = errD_real + errD_fake
208.        optimizer_D.step()

209.        ########################
210.        # update G network
211.        ########################
212.        model_G.zero_grad()
213.        labelv = Variable(label.fill_(real_label))
214.        output = model_D(fake)
215.        errG_GAN = criterion(torch.squeeze(output), labelv)
216.        errG_L1 = L1(fake.view(fake.size(0),-1), target.view(target.size(0),-1))

217.        errG = errG_GAN + args.lamb * errG_L1
218.        errG.backward()
219.        D_G_x2 = output.data.mean()
220.        optimizer_G.step()

221.        # store error values
222.        errorG.update(errG.data[0], target.size(0), history=1)
223.        errorD.update(errD.data[0], target.size(0), history=1)
224.        errorG_basic.update(errG.data[0], target.size(0), history=1)
225.        errorD_basic.update(errD.data[0], target.size(0), history=1)
226.        errorD_real.update(errD_real.data[0], target.size(0), history=1)
227.        errorD_fake.update(errD_fake.data[0], target.size(0), history=1)

228.        errorD_real.update(errD_real.data[0], target.size(0), history=1)
229.        errorD_fake.update(errD_fake.data[0], target.size(0), history=1)
230.        errorG_GAN.update(errG_GAN.data[0], target.size(0), history=1)
231.        errorG_R.update(errG_L1.data[0], target.size(0), history=1)


232.        if iteration % print_interval == 0:
233.        print('Epoch%d[%d/%d]:        Loss_D:        %.4f(R%0.4f+F%0.4f)        Loss_G:
    %0.4f(GAN%.4f+R%0.4f) D(x): %.4f D(G(z)): %.4f / %.4f' \
234.        % (epoch, i, len(train_loader),
235.        errorD_basic.avg, errorD_real.avg, errorD_fake.avg,
236.        errorG_basic.avg, errorG_GAN.avg, errorG_R.avg,
237.        D_x, D_G_x1, D_G_x2
238.        ))
239.        # plot image
240.        plotter_basic.g_update(errorG_basic.avg)
241.        plotter_basic.d_update(errorD_basic.avg)
242.        plotter_basic.draw(img_path + 'train_basic.png')
243.        # reset AverageMeter
244.        errorG_basic.reset()
245.        errorD_basic.reset()
246.        errorD_real.reset()
247.        errorD_fake.reset()
248.        errorG_GAN.reset()
249.        errorG_R.reset()
```

```
250.        iteration += 1

251.        return errorG.avg, errorD.avg


252.        def validate(val_loader, model_G, model_D, optimizer_G, optimizer_D, epoch):
253.        errorG = AverageMeter()
254.        errorD = AverageMeter()

255.        model_G.eval()
256.        model_D.eval()

257.        real_label = 1
258.        fake_label = 0

259.        for i, (data, target) in enumerate(val_loader):
260.        data, target = Variable(data.cuda()), Variable(target.cuda())
261.        ########################
262.        # D network
263.        ########################
264.        # validate with real
265.        output = model_D(target)
266.        label = torch.FloatTensor(target.size(0)).fill_(real_label).cuda()
267.        labelv = Variable(label)
268.        errD_real = criterion(torch.squeeze(output), labelv)

269.        # validate with fake
270.        fake =  model_G(data)
271.        labelv = Variable(label.fill_(fake_label))
272.        output = model_D(fake.detach())
273.        errD_fake = criterion(torch.squeeze(output), labelv)

274.        errD = errD_real + errD_fake

275.        ########################
276.        # G network
277.        ########################
278.        labelv = Variable(label.fill_(real_label))
279.        output = model_D(fake)
280.        errG_GAN = criterion(torch.squeeze(output), labelv)
281.        errG_L1 = L1(fake.view(fake.size(0),-1), target.view(target.size(0),-1))

282.        errG = errG_GAN + args.lamb * errG_L1

283.        errorG.update(errG.data[0], target.size(0), history=1)
284.        errorD.update(errD.data[0], target.size(0), history=1)

285.        if i == 0:
286.        vis_result(data.data, target.data, fake.data, epoch)

287.        if i % 50 == 0:
288.        print('Validating Epoch %d: [%d/%d]' \
289.        % (epoch, i, len(val_loader)))

290.        print('Validation: Loss_D: %.4f Loss_G: %.4f '\
291.        % (errorD.avg, errorG.avg))

292.        return errorG.avg, errorD.avg
```

```
293.        def vis_result(data, target, output, epoch):
294.        '''visualize images for GAN'''
295.        img_list = []
296.        for i in range(min(32, val_bs)):
297.        l = torch.unsqueeze(torch.squeeze(data[i]), 0).cpu().numpy()
298.        raw = target[i].cpu().numpy()
299.        pred = output[i].cpu().numpy()

300.        raw_rgb = (np.transpose(raw, (1,2,0)).astype(np.float64) + 1) / 2.
301.        pred_rgb = (np.transpose(pred, (1,2,0)).astype(np.float64) + 1) / 2.

302.        grey = np.transpose(l, (1,2,0))
303.        grey = np.repeat(grey, 3, axis=2).astype(np.float64)
304.        img_list.append(np.concatenate((grey, raw_rgb, pred_rgb), 1))

305.        img_list = [np.concatenate(img_list[4*i:4*(i+1)], axis=1) for i in
    range(len(img_list) // 4)]
306.        img_list = np.concatenate(img_list, axis=0)

307.        plt.figure(figsize=(36,27))
308.        plt.imshow(img_list)
309.        plt.axis('off')
310.        plt.tight_layout()
311.        plt.savefig(img_path + 'epoch%d_val.png' % epoch)
312.        plt.clf()

313.        if __name__ == '__main__':
314.        main()
```

loss.py

```
1.  import torch
2.  import torch.nn.functional as F
3.  import torch.nn as nn


4.  # this may be unstable sometimes.Notice set the size_average
5.  def CrossEntropy2d(input, target, weight=None, size_average=True):
6.  # input:(n, c, h, w) target:(n, h, w)
7.  n, c, h, w = input.size()
8.  # log_p: (n, c, h, w)
9.  log_p = F.log_softmax(input)
10. # log_p: (n*h*w, c)
11. log_p = log_p.transpose(1, 2).transpose(2, 3).contiguous().view(-1, c)
12. log_p = log_p[target.view(n, h, w, 1).repeat(1, 1, 1, c).view(-1, c) >= 0]
13. log_p = log_p.view(-1, c)

14. # target: (n*h*w,)
15. mask = target >= 0
16. target = target[mask]
17. loss = F.nll_loss(log_p, target, weight=weight, size_average=False)
18. if size_average:
19. loss /= mask.data.sum()
20. return loss


21. def BCE2d(input, target, weight=None, size_average=True):
```

```
# input:(n, 1, h, w) target:(n, h, w)
n, c, h, w = input.size()
# log_p: (n, c, h, w)
BCEWithLogits = nn.BCEWithLogitsLoss(weight=weight, size_average=False)
log_p = input
# log_p: (n*h*w, c)
log_p = log_p.transpose(1, 2).transpose(2, 3).contiguous().view(-1, c)
log_p = log_p[target.view(n, h, w, 1).repeat(1, 1, 1, c) >= 0]
log_p = log_p.view(-1, c)
# target: (n*h*w,)
mask = target >= 0
target = target[mask]
# import pdb; pdb.set_trace()

loss = BCEWithLogits(torch.squeeze(log_p), target.type(torch.FloatTensor).cuda())
# import pdb; pdb.set_trace()

if size_average:
loss /= mask.data.sum()

return loss
```

load_data.py

```
import os
import os.path as osp
import sys
import numpy as np
import pickle
from PIL import Image
import matplotlib.pyplot as plt
# plt.switch_backend('agg')
import time

import torch
import torchvision
from torchvision import transforms
from torch.utils import data
import scipy.io as io
import scipy.misc as misc
import glob
import csv
from skimage import color
from transform import ReLabel, ToLabel, ToSP, Scale

def pil_loader(path):
with open(path, 'rb') as f:
with Image.open(f) as img:
return img.convert('RGB')


class lfw_Dataset(data.Dataset):
def __init__(self, root,
shuffle=False,
small=False,
mode='test',
transform=None,
```

```
30. target_transform=None,
31. types='',
32. show_ab=False,
33. loader=pil_loader):

34. tic = time.time()
35. self.root = root
36. self.loader = loader
37. self.image_transform = transform
38. self.imgpath = glob.glob(root + 'lfw_funneled/*/*')
39. self.types = types
40. self.show_ab = show_ab # show ab channel in classify mode

41. # read split
42. self.train_people = set()
43. with open(self.root + 'peopleDevTrain.txt', 'r') as f:
44. reader = csv.reader(f, delimiter='\t')
45. for i, row in enumerate(reader):
46. if i == 0:
47. continue
48. self.train_people.add(row[0])
49. assert self.train_people.__len__() == 4038

50. self.test_people = set()
51. with open(self.root + 'peopleDevTest.txt', 'r') as f:
52. reader = csv.reader(f, delimiter='\t')
53. for i, row in enumerate(reader):
54. if i == 0:
55. continue
56. self.test_people.add(row[0])
57. assert self.test_people.__len__() == 1711

58. self.path = []
59. if mode == 'train':
60. for item in self.imgpath:
61. if item.split('/')[-2] in self.train_people:
62. self.path.append(item)
63. elif mode == 'test':
64. for item in self.imgpath:
65. if item.split('/')[-2] in self.test_people:
66. self.path.append(item)

67. np.random.seed(0)
68. if shuffle:
69. perm = np.random.permutation(len(self.path))
70. self.path = [self.path[i] for i in perm]

71. if types == 'classify':
72. ab_list = np.load('data/pts_in_hull.npy')
73. self.nbrs = NearestNeighbors(n_neighbors=1, algorithm='ball_tree').fit(ab_list)

74. print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))

75. def __getitem__(self, index):
76. mypath = self.path[index]
77. img = self.loader(mypath) # PIL Image
78. img = np.array(img)[13:13+224, 13:13+224, :]

79. img_lab = color.rgb2lab(np.array(img)) # np array
```

```
# img_lab = img_lab[13:13+224, 13:13+224, :]

if self.types == 'classify':
X_a = np.ravel(img_lab[:,:,1])
X_b = np.ravel(img_lab[:,:,2])
img_ab = np.vstack((X_a, X_b)).T
_, ind = self.nbrs.kneighbors(img_ab)
ab_class = np.reshape(ind, (224,224))
# print(ab_class.shape, ab_class.dtype, np.amax(ab_class), np.amin(ab_class))
ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)

img = (img - 127.5) / 127.5 # -1 to 1
img = torch.FloatTensor(np.transpose(img, (2,0,1)))
img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))

img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

if self.types == 'classify':
if self.show_ab:
return img_l, ab_class, img_ab
return img_l, ab_class
elif self.types == 'raw':
return img_l, img
        else:
        return img_l, img_ab

        def __len__(self):
        return len(self.path)

        class Flower_Dataset(data.Dataset):
        def __init__(self, root,
        shuffle=False,
        small=False,
        mode='test',
        transform=None,
        target_transform=None,
        types='',
        show_ab=False,
        loader=pil_loader):

        tic = time.time()
        self.root = root
        self.loader = loader
        self.image_transform = transform
        self.imgpath = glob.glob(root + 'jpg/*.jpg')
        self.types = types
        self.show_ab = show_ab # show ab channel in classify mode

        # read split
        split_file = io.loadmat(root + 'datasplits.mat')

        self.train_file       =       set([str(i).zfill(4)       for       i       in
    np.hstack((split_file['trn1'][0], split_file['val1'][0]))])
        self.test_file = set([str(i).zfill(4) for i in split_file['tst1'][0]])
        assert self.train_file.__len__() == 1020
        assert self.test_file.__len__() == 340

        self.path = []
```

```
128.        if mode == 'train':
129.        for item in self.imgpath:
130.        if item.split('/')[-1][6:6+4] in self.train_file:
131.        self.path.append(item)
132.        elif mode == 'test':
133.        for item in self.imgpath:
134.        if item.split('/')[-1][6:6+4] in self.test_file:
135.        self.path.append(item)

136.        self.path = sorted(self.path)

137.        np.random.seed(0)
138.        if shuffle:
139.        perm = np.random.permutation(len(self.path))
140.        self.path = [self.path[i] for i in perm]

141.        if types == 'classify':
142.        ab_list = np.load('data/pts_in_hull.npy')
143.        self.nbrs                   =                   NearestNeighbors(n_neighbors=1,
    algorithm='ball_tree').fit(ab_list)
144.        print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))

145.        def __getitem__(self, index):
146.        mypath = self.path[index]
147.        img = self.loader(mypath) # PIL Image
148.        img = np.array(img)
149.        img = misc.imresize(img, (224, 224))

150.        img_lab = color.rgb2lab(np.array(img)) # np array
151.        # img_lab = img_lab[13:13+224, 13:13+224, :]

152.        if self.types == 'classify':
153.        X_a = np.ravel(img_lab[:,:,1])
154.        X_b = np.ravel(img_lab[:,:,2])
155.        img_ab = np.vstack((X_a, X_b)).T
156.        _, ind = self.nbrs.kneighbors(img_ab)
157.        ab_class = np.reshape(ind, (224,224))
158.        ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)
159.        img = (img - 127.5) / 127.5 # -1 to 1
160.        img = torch.FloatTensor(np.transpose(img, (2,0,1)))
161.        img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))

162.        img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
163.        img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

164.        if self.types == 'classify':
165.        if self.show_ab:
166.        return img_l, ab_class, img_ab
167.        return img_l, ab_class
168.        elif self.types == 'raw':
169.        return img_l, img
170.        # if self.show_ab:
171.        #     return img_l, img_ab, None
172.        else:
173.        return img_l, img_ab

174.        def __len__(self):
175.        return len(self.path)
```

```
176.        class Spongebob_Dataset(data.Dataset):
177.        def __init__(self, root,
178.        shuffle=False,
179.        small=False,
180.        mode='test',
181.        transform=None,
182.        target_transform=None,
183.        types='',
184.        show_ab=False,
185.        large=False,
186.        loader=pil_loader):

187.        tic = time.time()
188.        self.root = root
189.        self.loader = loader
190.        self.image_transform = transform
191.        if large:
192.        self.size = 480
193.        self.imgpath = glob.glob(root + 'img_480/*.png')
194.        else:
195.        self.size = 224
196.        self.imgpath = glob.glob(root + 'img/*.png')
197.        self.types = types
198.        self.show_ab = show_ab # show ab channel in classify mode

199.        # read split
200.        self.train_file = set()
201.        with open(self.root + 'train_split.csv', 'r') as f:
202.        reader = csv.reader(f, delimiter='\t')
203.        for i, row in enumerate(reader):
204.        if i == 0:
205.        continue
206.        self.train_file.add(str(row[0]).zfill(4))

207.        assert self.train_file.__len__() == 1392

208.        self.test_file = set()
209.        with open(self.root + 'test_split.csv', 'r') as f:
210.        reader = csv.reader(f, delimiter='\t')
211.        for i, row in enumerate(reader):
212.        if i == 0:
213.        continue
214.        self.test_file.add(str(row[0]).zfill(4))
215.        assert self.test_file.__len__() == 348

216.        self.path = []
217.        if mode == 'train':
218.        for item in self.imgpath:
219.        if item.split('/')[-1][6:6+4] in self.train_file:
220.        self.path.append(item)
221.        elif mode == 'test':
222.        for item in self.imgpath:
223.        if item.split('/')[-1][6:6+4] in self.test_file:
224.        self.path.append(item)

225.        self.path = sorted(self.path)

226.        np.random.seed(0)
227.        if shuffle:
```

```
228.        perm = np.random.permutation(len(self.path))
229.        self.path = [self.path[i] for i in perm]

230.        if types == 'classify':
231.        ab_list = np.load('data/pts_in_hull.npy')
232.        self.nbrs                    =                    NearestNeighbors(n_neighbors=1,
    algorithm='ball_tree').fit(ab_list)

233.        print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))

234.        def __getitem__(self, index):
235.        mypath = self.path[index]
236.        img = self.loader(mypath) # PIL Image
237.        img = np.array(img)
238.        if (img.shape[0] != self.size) or (img.shape[1] != self.size):
239.        img = misc.imresize(img, (self.size, self.size))

240.        img_lab = color.rgb2lab(np.array(img)) # np array
241.        # img_lab = img_lab[13:13+224, 13:13+224, :]

242.        if self.types == 'classify':
243.        X_a = np.ravel(img_lab[:,:,1])
244.        X_b = np.ravel(img_lab[:,:,2])
245.        img_ab = np.vstack((X_a, X_b)).T
246.        _, ind = self.nbrs.kneighbors(img_ab)
247.        ab_class = np.reshape(ind, (self.size,self.size))
248.        # print(ab_class.shape, ab_class.dtype, np.amax(ab_class), np.amin(ab_class))
249.        ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)

250.        img = (img - 127.5) / 127.5 # -1 to 1
251.        img = torch.FloatTensor(np.transpose(img, (2,0,1)))
252.        img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))

253.        img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
254.        img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

255.        if self.types == 'classify':
256.        if self.show_ab:
257.        return img_l, ab_class, img_ab
258.        return img_l, ab_class
259.        elif self.types == 'raw':
260.        return img_l, img
261.        # if self.show_ab:
262.        #     return img_l, img_ab, None
263.        else:
264.        return img_l, img_ab
265.        def __len__(self):
266.        return len(self.path)

267.        class SC2_Dataset(data.Dataset):
268.        def __init__(self, root,
269.        shuffle=False,
270.        small=False,
271.        mode='test',
272.        transform=None,
273.        target_transform=None,
274.        types='',
275.        show_ab=False,
276.        large=False,
```

```
277.        loader=pil_loader):

278.        tic = time.time()
279.        self.root = root
280.        self.loader = loader
281.        self.image_transform = transform
282.        if large:
283.        self.size = 480
284.        self.imgpath = glob.glob(root + 'img_480/*.png')
285.        else:
286.        self.size = 224
287.        self.imgpath = glob.glob(root + 'img/*.png')
288.        self.types = types
289.        self.show_ab = show_ab # show ab channel in classify mode

290.        # read split
291.        self.train_file = set()
292.        with open(self.root + 'train_split.csv', 'r') as f:
293.        reader = csv.reader(f, delimiter='\t')
294.        for i, row in enumerate(reader):
295.        if i == 0:
296.        continue
297.        self.train_file.add(str(row[0]).zfill(4))
298.        assert self.train_file.__len__() == 1383

299.        self.test_file = set()
300.        with open(self.root + 'test_split.csv', 'r') as f:
301.        reader = csv.reader(f, delimiter='\t')
302.        for i, row in enumerate(reader):
303.        if i == 0:
304.        continue
305.        self.test_file.add(str(row[0]).zfill(4))
306.        assert self.test_file.__len__() == 345

307.        self.path = []
308.        if mode == 'train':
309.        for item in self.imgpath:
310.        if item.split('/')[-1][6:6+4] in self.train_file:
311.        self.path.append(item)
312.        elif mode == 'test':
313.        for item in self.imgpath:
314.        if item.split('/')[-1][6:6+4] in self.test_file:
315.        self.path.append(item)
316.        self.path = sorted(self.path)
317.        np.random.seed(0)
318.        if shuffle:
319.        perm = np.random.permutation(len(self.path))
320.        self.path = [self.path[i] for i in perm]

321.        if types == 'classify':
322.        ab_list = np.load('data/pts_in_hull.npy')
323.        self.nbrs                    =                    NearestNeighbors(n_neighbors=1,
    algorithm='ball_tree').fit(ab_list)

324.        print('Load %d images, used %fs' % (self.path.__len__(), time.time()-tic))

325.        def __getitem__(self, index):
326.        mypath = self.path[index]
327.        img = self.loader(mypath) # PIL Image
```

```
328.        img = np.array(img)
329.        if (img.shape[0] != self.size) or (img.shape[1] != self.size):
330.        img = misc.imresize(img, (self.size, self.size))
331.        img_lab = color.rgb2lab(np.array(img)) # np array
332.        # img_lab = img_lab[13:13+224, 13:13+224, :]

333.        if self.types == 'classify':
334.        X_a = np.ravel(img_lab[:,:,1])
335.        X_b = np.ravel(img_lab[:,:,2])
336.        img_ab = np.vstack((X_a, X_b)).T
337.        _, ind = self.nbrs.kneighbors(img_ab)
338.        ab_class = np.reshape(ind, (self.size,self.size))
339.        # print(ab_class.shape, ab_class.dtype, np.amax(ab_class), np.amin(ab_class))
340.        ab_class = torch.unsqueeze(torch.LongTensor(ab_class), 0)

341.        img = (img - 127.5) / 127.5 # -1 to 1
342.        img = torch.FloatTensor(np.transpose(img, (2,0,1)))
343.        img_lab = torch.FloatTensor(np.transpose(img_lab, (2,0,1)))
344.        img_l = torch.unsqueeze(img_lab[0],0) / 100. # L channel 0-100
345.        img_ab = (img_lab[1::] + 0) / 110. # ab channel -110 - 110

346.        if self.types == 'classify':
347.        if self.show_ab:
348.        return img_l, ab_class, img_ab
349.        return img_l, ab_class
350.        elif self.types == 'raw':
351.        if img.size(1) == 479 or img.size(2) == 479:
352.        print(mypath)
353.        return img_l, img
354.        # if self.show_ab:
355.        #     return img_l, img_ab, None
356.        else:
357.        return img_l, img_ab
358.        def __len__(self):
359.        return len(self.path)

360.        if __name__ == '__main__':
361.        data_root = '/home/users/u5612799/DATA/SCReplay/'
362.        # normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
363.        # std=[0.229, 0.224, 0.225]
364.        image_transform = transforms.Compose([
365.        transforms.CenterCrop(224),
366.        transforms.ToTensor(),
367.        ])

368.        lfw = SC2_Dataset(data_root, mode='train',
369.        transform=image_transform, large=True, types='raw')
370.        data_loader = data.DataLoader(lfw,
371.        batch_size=1,
372.        shuffle=False,
373.        num_workers=4)

374.        for i, (data, target) in enumerate(data_loader):
375.        print(i, len(lfw))
```