

❖ Step 1 — Load Dataset & Convert to Lab

```

1 import kagglehub
2
3 # Download latest version
4 path = kagglehub.dataset_download("theblackmamba31/landscape-image-colorization")
5
6 print("Path to dataset files:", path)

```

Using Colab cache for faster access to the 'landscape-image-colorization' dataset.
Path to dataset files: /kaggle/input/landscape-image-colorization

```

1 import cv2
2 import numpy as np
3 import glob
4
5 def load_images(path):
6     files = glob.glob(path + "/*.jpg")
7
8     L_list = []
9     ab_list = []
10
11    for f in files:
12        img = cv2.imread(f)
13        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
14        img = cv2.resize(img, (256, 256))
15
16        lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
17        L = lab[:, :, 0] / 255.0           # Normalize
18        ab = lab[:, :, 1:] / 128.0         # Normalize
19
20        L_list.append(L.reshape(256,256,1))
21        ab_list.append(ab.reshape(256,256,2))
22
23    return np.array(L_list), np.array(ab_list)
24

```

```
1 train_L, train_ab = load_images("/content/train_images")
```

💬 3. ECCV 2016 Model Architecture

UNet-like encoder-decoder

Softmax over 313 ab bins

Convert ab colors into 313 bins

Paper provides points → You can download the bin centers:

```

1 import numpy as np
2 import os
3 import glob
4
5 # The 'path' variable from cell 579pssGiWTfD holds the root directory of the downloaded dataset.
6 # We need to find the pts_in_hull.npy file within this directory or its subdirectories.
7
8 pts_in_hull_files = glob.glob(os.path.join(path, '**', 'pts_in_hull.npy'), recursive=True)
9
10 if pts_in_hull_files:
11     pts_in_hull_path = pts_in_hull_files[0]
12     pts_in_hull = np.load(pts_in_hull_path)
13     print(f"Loaded pts_in_hull from: {pts_in_hull_path}")
14     print(f"Loaded pts_in_hull with shape: {pts_in_hull.shape}")
15 else:
16     raise FileNotFoundError("pts_in_hull.npy not found in the dataset directory or its subdirectories.")

-----
FileNotFoundException                                Traceback (most recent call last)
/tmp/ipython-input-1759789337.py in <cell line: 0>()
14     print(f"Loaded pts_in_hull with shape: {pts_in_hull.shape}")
15 else:
--> 16     raise FileNotFoundError("pts_in_hull.npy not found in the dataset directory or its subdirectories")

FileNotFoundException: pts_in_hull.npy not found in the dataset directory or its subdirectories.

```

Next steps: [Explain error](#)

↙ ECCV16 Model Code (Simplified Keras)

```

1 from keras.layers import *
2 from keras.models import Model
3
4 def eccv16_model():
5     L_in = Input(shape=(256,256,1))
6
7     x = Conv2D(64,3,activation='relu',padding='same')(L_in)
8     x = Conv2D(64,3,activation='relu',padding='same',strides=2)(x)
9
10    x = Conv2D(128,3,activation='relu',padding='same')(x)
11    x = Conv2D(128,3,activation='relu',padding='same',strides=2)(x)
12
13    x = Conv2D(256,3,activation='relu',padding='same')(x)
14    x = Conv2D(256,3,activation='relu',padding='same')(x)
15    x = Conv2D(256,3,activation='relu',padding='same',strides=2)(x)
16
17    x = Conv2D(512,3,activation='relu',padding='same')(x)
18    x = Conv2D(512,3,activation='relu',padding='same')(x)
19    x = Conv2D(512,3,activation='relu',padding='same')(x)
20
21    x = Conv2D(256,3,activation='relu',padding='same')(x)
22    x = UpSampling2D()(x)
23    x = Conv2D(128,3,activation='relu',padding='same')(x)
24    x = UpSampling2D()(x)
25    x = Conv2D(64,3,activation='relu',padding='same')(x)
26    x = UpSampling2D()(x)
27
28    out = Conv2D(313,1,activation='softmax')(x)
29
30    return Model(L_in, out)
31

```

↙ 4. ECCV16 Training Procedure

Loss Function

ECCV16 uses:

✓ Cross entropy on quantized AB labels ✓ Class-rebalancing weights

```

1 model = eccv16_model()
2 model.compile(
3     optimizer='adam',
4     loss='categorical_crossentropy',
5     metrics=['accuracy']
6 )

```

↙ Training Loop

```

1 model.fit(
2     train_L,
3     train_bins,      # 313-channel encoded labels
4     batch_size=32,
5     epochs=50
6 )
7

-----
NameError: name 'train_bins' is not defined
Traceback (most recent call last)
/tmp/ipython-input-3658512317.py in <cell line: 0>()
      1 model.fit(
      2     train_L,
----> 3     train_bins,
      4     batch_size=8,
      5     epochs=5

NameError: name 'train_bins' is not defined

```

Next steps: [Explain error](#)

⌄ 5. SIGGRAPH17 (User Guided) Model

Same base network + TWO additional inputs:

1 Local hints (ab strokes)

Shape: (256,256,2)

2 Global hint vector (216 dims)

Extracted using global features.

```
1 L_input      = Input(shape=(256,256,1))      # grayscale
2 local_hint   = Input(shape=(256,256,2))      # user strokes
3 global_hint  = Input(shape=(216,))           # global stats
4
```

⌄ 6. SIGGRAPH17 Training Loss

Uses two losses:

✓ L2 loss (mean squared error) ✓ Classification loss on bins

Final loss = (MSE + CE)

```
1 model.compile(
2     optimizer='adam',
3     loss=['mse','categorical_crossentropy'],
4     loss_weights=[1.0, 0.3]
5 )
6
```

⌄ 7. Saving the Model

```
1 model.save("eccv16_colorization.h5")
2
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)` . This file for

⌄ 8. Inference (Colorization)

```
1 pred = model.predict(L_img.reshape(1,256,256,1))
2 ab_img = decode_313_bins(pred)
3
4 lab = np.concatenate((L_img*255, ab_img), axis=-1)
5 rgb = cv2.cvtColor(lab.astype(np.uint8), cv2.COLOR_LAB2RGB)
6
```

```
NameError: name 'L_img' is not defined
-----  

Traceback (most recent call last)  

/ttmp/ipython-input-1204642008.py in <cell line: 0>()  

----> 1 pred = model.predict(L_img.reshape(1,256,256,1))
      2 ab_img = decode_313_bins(pred)
      3
      4 lab = np.concatenate((L_img*255, ab_img), axis=-1)
      5 rgb = cv2.cvtColor(lab.astype(np.uint8), cv2.COLOR_LAB2RGB)
```

NameError: name 'L_img' is not defined

Next steps: [Explain error](#)

🔥 9. Training Hardware

Realistic training time:

Model Dataset GPU Epochs Time

ECCV16 1M images RTX 4090 20 ~3–4 hours

SIGGRAPH17 1M images RTX 4090 20 ~6–8 hours

▼ *Eccv16 Siggraph17 Colorization*

```

1 """
2 ECCV16 + SIGGRAPH17 Colorization Training Script
3 Single-file training + inference + saving script for both models.
4
5 Usage examples:
6   # Train ECCV16 automatic model
7   python eccv16_siggraph17_colorization.py --mode eccv16 \
8     --dataset /path/to/images --pts pts_in_hull.npy --epochs 40 --batch 32
9
10  # Train SIGGRAPH17 user-guided model (uses same dataset; generates synthetic hints)
11 python eccv16_siggraph17_colorization.py --mode siggraph \
12   --dataset /path/to/images --pts pts_in_hull.npy --epochs 40 --batch 16
13
14 Notes:
15 - Expects images (.jpg/.png) in a single folder (no subfolders) for simplicity.
16 - Requires pts_in_hull.npy (313 ab-bin centers) for ECCV-style quantization.
17 - Uses TensorFlow / Keras.
18 - Save outputs: eccv16_weights.h5 and siggraph17_weights.h5 and SavedModel dirs.
19
20 Author: Generated by ChatGPT for user
21 """
22
23 import os
24 import argparse
25 import glob
26 import random
27 import math
28 import numpy as np
29 from pathlib import Path
30 from tqdm import tqdm
31
32 import tensorflow as tf
33 from tensorflow.keras import layers, Model
34 from tensorflow.keras.optimizers import Adam
35
36 import cv2
37
38 # ----- Utilities -----
39
40 def load_image_paths(dataset_dir, exts=(".jpg", ".jpeg", ".png")):
41     files = []
42     for ext in exts:
43         files.extend(glob.glob(os.path.join(dataset_dir, f"**/*{ext}"), recursive=True))
44     return sorted(files)
45
46
47 def read_and_resize(path, target_size=(256,256)):
48     img = cv2.imread(path)
49     if img is None:
50         raise ValueError(f"Unable to read image: {path}")
51     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
52     img = cv2.resize(img, target_size, interpolation=cv2.INTER_AREA)
53     return img
54
55
56 # ----- color utilities -----
57
58 def rgb2lab(img_rgb):
59     lab = cv2.cvtColor(img_rgb.astype(np.uint8), cv2.COLOR_RGB2LAB)
60     return lab.astype(np.float32)
61
62
63 def lab2rgb(lab):
64     lab = lab.astype(np.uint8)
65     rgb = cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
66     return rgb
67
68
69 # pts_in_hull.npy helper: contains 313 centers (ab values) used in ECCV paper
70 # You must provide pts_in_hull.npy from Zhang et al. repo or precomputed centers.
71
72 def load_pts(pts_path):
73     pts = np.load(pts_path)
74     if pts.shape[1] != 2:

```

```

    74     pts.shape[1] := 2.
75     raise ValueError("pts_in_hull.npy should be shape (313,2)")
76     return pts
77
78
79 def ab_to_q(ab, pts):
80     # ab: (... , 2) with a,b in typical LAB ranges (a approx [-128,127])
81     # pts: (313,2)
82     # Output: argmin bin index per pixel
83     h, w, _ = ab.shape
84     flat = ab.reshape(-1,2)
85     # compute L2 distance to centers
86     # avoid huge memory for big images; use chunking
87     dists = np.linalg.norm(flat[:,None,:] - pts[None,:,:], axis=2) # (h*w,313)
88     q = np.argmin(dists, axis=1)
89     q = q.reshape(h,w)
90     return q
91
92
93 def q_to_ab_map(pred_probs, pts):
94     # pred_probs: (H, W, 313) softmax probabilities
95     # return ab channels as weighted sum of centers
96     H,W,_ = pred_probs.shape
97     probs = pred_probs.reshape(-1, pred_probs.shape[-1]) # (H*W,313)
98     ab_flat = probs.dot(pts) # (H*W,2)
99     ab = ab_flat.reshape(H,W,2)
100    return ab
101
102
103 # ----- Data generator -----
104
105 class ImageFolderGenerator(tf.keras.utils.Sequence):
106     def __init__(self, paths, pts=None, batch_size=16, target_size=(256,256), shuffle=True, mode='eccv'):
107         self.paths = paths
108         self.batch_size = batch_size
109         self.target_size = target_size
110         self.shuffle = shuffle
111         self.mode = mode
112         self.pts = pts
113         self.on_epoch_end()
114
115     def __len__(self):
116         return math.ceil(len(self.paths)/self.batch_size)
117
118     def on_epoch_end(self):
119         if self.shuffle:
120             random.shuffle(self.paths)
121
122     def __getitem__(self, idx):
123         batch_paths = self.paths[idx*self.batch_size : (idx+1)*self.batch_size]
124         ls = []
125         bins = []
126         abs_reg = []
127         local_hints = []
128         global_feats = []
129
130         for p in batch_paths:
131             img = read_and_resize(p, self.target_size)
132             lab = rgb2lab(img) # L in [0,255], a,b roughly in [0,255]
133
134             L = lab[:,:,:0] / 255.0 # normalize 0-1
135             a = lab[:,:,:1] - 128.0 # center around zero roughly
136             b = lab[:,:,:2] - 128.0
137             ab = np.stack([a,b], axis=-1)
138
139             ls.append(L.reshape(self.target_size[0], self.target_size[1], 1).astype(np.float32))
140             abs_reg.append(ab.astype(np.float32))
141
142         if self.mode in ['eccv', 'siggraph']:
143             if self.pts is None:
144                 raise ValueError("pts centers required for ECCV-style quantization")
145             q = ab_to_q(ab, self.pts) # (H,W)
146             # convert to one-hot 313 channels
147             onehot = np.eye(len(self.pts), dtype=np.float32)[q] # (H,W,313)
148             bins.append(onehot)
149
150         if self.mode == 'siggraph':
151             # generate synthetic local hints: random sparse points/strokes with ab color values
152             hint = np.zeros((self.target_size[0], self.target_size[1], 2), dtype=np.float32)
153             num_points = random.randint(1, 20)
154             h,w = self.target_size
155             for i in range(num_points):
156                 x = random.randint(0, w-1)

```

```

157     y = random.randint(0, h-1)
158     # spread the hint into small gaussian blob
159     rr = np.arange(h)[:,None]
160     cc = np.arange(w)[None,:]
161     sigma = random.uniform(1.0, 6.0)
162     gauss = np.exp(-((rr-y)**2 + (cc-x)**2)/(2*sigma*sigma))
163     hint[:, :, 0] += gauss * ab[y,x,0]
164     hint[:, :, 1] += gauss * ab[y,x,1]
165     local_hints.append(hint)
166     # global features: simple per-channel mean & std + histogram bins (trivial)
167     gfeat = np.array([ab[:, :, 0].mean(), ab[:, :, 1].mean(), ab[:, :, 0].std(), ab[:, :, 1].std()], dtype=np.float32)
168     # pad to 216 dims as in paper by zeros (paper used 216-dim global descriptor)
169     gpad = np.zeros((216,), dtype=np.float32)
170     gpad[:gfeat.shape[0]] = gfeat
171     global_feats.append(gpad)
172
173     inputs = {'L': np.array(Ls)}
174     outputs = {}
175
176     if len(bins) > 0:
177         outputs['q'] = np.array(bins)
178     if len(abs_reg) > 0:
179         outputs['ab_reg'] = np.array(abs_reg)
180     if self.mode == 'siggraph':
181         inputs['local_hint'] = np.array(local_hints)
182         inputs['global_hint'] = np.array(global_feats)
183
184     return inputs, outputs
185
186
187 # ----- Models -----
188
189 def conv_block(x, filters, kernel=3, strides=1):
190     x = layers.Conv2D(filters, kernel, strides=strides, padding='same')(x)
191     x = layers.BatchNormalization()(x)
192     x = layers.ReLU()(x)
193     return x
194
195
196 def eccv16_model(input_shape=(256,256,1), n_bins=313):
197     L_in = layers.Input(shape=input_shape, name='L')
198     x = conv_block(L_in, 64)
199     x = conv_block(x, 64)
200     x = layers.MaxPool2D(2)(x)
201
202     x = conv_block(x, 128)
203     x = conv_block(x, 128)
204     x = layers.MaxPool2D(2)(x)
205
206     x = conv_block(x, 256)
207     x = conv_block(x, 256)
208     x = conv_block(x, 256)
209     x = layers.MaxPool2D(2)(x)
210
211     x = conv_block(x, 512)
212     x = conv_block(x, 512)
213     x = conv_block(x, 512)
214
215     x = conv_block(x, 256)
216     x = layers.UpSampling2D()(x)
217     x = conv_block(x, 128)
218     x = layers.UpSampling2D()(x)
219     x = conv_block(x, 64)
220     x = layers.UpSampling2D()(x)
221
222     out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)
223
224     model = Model(inputs=L_in, outputs=out, name='eccv16')
225     return model
226
227
228 def siggraph17_model(input_shape=(256,256,1), n_bins=313):
229     # Inputs: L, local_hint (H,W,2), global_hint (216,)
230     L_in = layers.Input(shape=input_shape, name='L')
231     local_hint = layers.Input(shape=(input_shape[0], input_shape[1], 2), name='local_hint')
232     global_hint = layers.Input(shape=(216,), name='global_hint')
233
234     # Simple encoder for L
235     x = conv_block(L_in, 64)
236     x = conv_block(x, 64)
237     x = layers.MaxPool2D(2)(x)
238
239     x = conv_block(x, 128)

```

```

240     x = conv_block(x, 128)
241     x = layers.MaxPool2D(2)(x)
242
243     x = conv_block(x, 256)
244     x = conv_block(x, 256)
245     x = conv_block(x, 256)
246
247     # incorporate local hint: concat/residual
248     lh = layers.Conv2D(32, 1, padding='same')(local_hint)
249     # downsample local hint a few times to match spatial dims
250     lh_down = layers.MaxPool2D(4)(lh)
251     x = layers.concatenate([x, lh_down])
252
253     x = conv_block(x, 512)
254
255     # global hint processing
256     g = layers.Dense(512, activation='relu')(global_hint)
257     g = layers.Dense(np.prod(x.shape[1:3]) * 16, activation='relu')(g)
258     g = layers.Reshape((x.shape[1], x.shape[2], 16))(g)
259     x = layers.concatenate([x, g])
260
261     x = conv_block(x, 256)
262     x = layers.UpSampling2D()(x)
263     x = conv_block(x, 128)
264     x = layers.UpSampling2D()(x)
265     x = conv_block(x, 64)
266     x = layers.UpSampling2D()(x)
267
268     q_out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)
269     ab_reg = layers.Conv2D(2, 1, activation='linear', name='ab_reg')(x)
270
271     model = Model(inputs=[L_in, local_hint, global_hint], outputs=[q_out, ab_reg], name='siggraph17')
272     return model
273
274
275 # ----- Training utilities -----
276
277 def compile_eccv(model, lr=1e-4):
278     model.compile(optimizer=Adam(lr), loss='categorical_crossentropy')
279     return model
280
281
282 def compile_siggraph(model, lr=1e-4):
283     # two outputs: q (categorical) and ab_reg (MSE)
284     losses = {'q': 'categorical_crossentropy', 'ab_reg': 'mse'}
285     loss_weights = {'q': 1.0, 'ab_reg': 1.0}
286     model.compile(optimizer=Adam(lr), loss=losses, loss_weights=loss_weights)
287     return model
288
289
290 # ----- Inference helpers -----
291
292 def eccv_infer(model, L_input, pts):
293     # L_input: (H,W,1) float 0-1
294     pred = model.predict(L_input[None,...])[0]  # (H,W,313)
295     ab = q_to_ab_map(pred, pts)  # (H,W,2)
296     L = (L_input[:, :, 0]*255.0).astype(np.float32)
297     lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
298     lab[:, :, 0] = L
299     lab[:, :, 1] = ab[:, :, 0] + 128.0
300     lab[:, :, 2] = ab[:, :, 1] + 128.0
301     rgb = lab2rgb(lab)
302     return rgb
303
304
305 def siggraph_infer(model, L_input, local_hint, global_hint, pts=None):
306     q_pred, ab_reg = model.predict([L_input[None,...], local_hint[None,...], global_hint[None,...]])
307     q_pred = q_pred[0]
308     ab_reg = ab_reg[0]
309     if pts is not None:
310         ab_q = q_to_ab_map(q_pred, pts)
311         # fuse regression and quantized result by simple avg
312         ab = 0.5*ab_reg + 0.5*ab_q
313     else:
314         ab = ab_reg
315     L = (L_input[:, :, 0]*255.0).astype(np.float32)
316     lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
317     lab[:, :, 0] = L
318     lab[:, :, 1] = ab[:, :, 0] + 128.0
319     lab[:, :, 2] = ab[:, :, 1] + 128.0
320     rgb = lab2rgb(lab)
321     return rgb
322

```

```

322
323 # ----- Main trainer -----
325
326 def main():
327     parser = argparse.ArgumentParser()
328     parser.add_argument('--dataset', required=True, help='Path to folder of images (recursive)')
329     parser.add_argument('--pts', required=True, help='Path to pts_in_hull.npy (313x2)')
330     parser.add_argument('--mode', choices=['eccv', 'siggraph', 'both'], default='eccv')
331     parser.add_argument('--epochs', type=int, default=30)
332     parser.add_argument('--batch', type=int, default=16)
333     parser.add_argument('--save_dir', default='models')
334     parser.add_argument('--lr', type=float, default=1e-4)
335     parser.add_argument('--img_size', type=int, default=256)
336     args = parser.parse_args()
337
338     os.makedirs(args.save_dir, exist_ok=True)
339
340     pts = load_pts(args.pts)
341
342     paths = load_image_paths(args.dataset)
343     if len(paths) == 0:
344         raise ValueError('No images found in dataset path')
345
346     print(f"Found {len(paths)} images. Preparing generator...")
347
348     if args.mode in ['eccv', 'both']:
349         gen_eccv = ImageFolderGenerator(paths, pts=pts, batch_size=args.batch, target_size=(args.img_size, args.img_size))
350         model_eccv = eccv16_model(input_shape=(args.img_size, args.img_size, 1), n_bins=pts.shape[0])
351         model_eccv = compile_eccv(model_eccv, lr=args.lr)
352
353         cb = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'eccv16_best.h5'), save_best_only=True, monitor='val_loss')
354         print('Training ECCV16...')
355         model_eccv.fit(gen_eccv, epochs=args.epochs, callbacks=[cb])
356         print('Saving ECCV16 final weights...')
357         model_eccv.save(os.path.join(args.save_dir, 'eccv16_final.h5'))
358         model_eccv.save(os.path.join(args.save_dir, 'eccv16_savedmodel'), save_format='tf')
359
360     if args.mode in ['siggraph', 'both']:
361         gen_sig = ImageFolderGenerator(paths, pts=pts, batch_size=max(1, args.batch//2), target_size=(args.img_size, args.img_size))
362         model_sig = siggraph17_model(input_shape=(args.img_size, args.img_size, 1), n_bins=pts.shape[0])
363         model_sig = compile_siggraph(model_sig, lr=args.lr)
364
365         cb2 = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'siggraph17_best.h5'), save_best_only=True, monitor='val_loss')
366         print('Training SIGGRAPH17...')
367         model_sig.fit(gen_sig, epochs=args.epochs, callbacks=[cb2])
368         print('Saving SIGGRAPH17 final weights...')
369         model_sig.save(os.path.join(args.save_dir, 'siggraph17_final.h5'))
370         model_sig.save(os.path.join(args.save_dir, 'siggraph17_savedmodel'), save_format='tf')
371
372     print('Done.')
373
374
375 if __name__ == '__main__':
376     main()
377

```

Hardcoded Path

```

1 import os
2 import argparse
3 import glob
4 import random
5 import math
6 import numpy as np
7 from pathlib import Path
8 from tqdm import tqdm
9
10 import tensorflow as tf
11 from tensorflow.keras import layers, Model
12 from tensorflow.keras.optimizers import Adam
13
14 import cv2
15
16 # ----- Utilities -----
17
18 def load_image_paths(dataset_dir, exts=('.jpg', '.jpeg', '.png')):
19     files = []
20     for ext in exts:
21         files.extend(glob.glob(os.path.join(dataset_dir, f'**/*{ext}'), recursive=True))

```

```

22     return sorted(files)
23
24
25 def read_and_resize(path, target_size=(256, 256)):
26     img = cv2.imread(path)
27     if img is None:
28         raise ValueError(f'Unable to read image: {path}')
29     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
30     img = cv2.resize(img, target_size, interpolation=cv2.INTER_AREA)
31     return img
32
33
34 # ----- color utilities -----
35
36 def rgb2lab(img_rgb):
37     lab = cv2.cvtColor(img_rgb.astype(np.uint8), cv2.COLOR_RGB2LAB)
38     return lab.astype(np.float32)
39
40
41 def lab2rgb(lab):
42     lab = lab.astype(np.uint8)
43     rgb = cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
44     return rgb
45
46
47 # pts_in_hull.npy helper: contains 313 centers (ab values) used in ECCV paper
48 # You must provide pts_in_hull.npy from Zhang et al. repo or precomputed centers.
49
50 def load_pts(pts_path):
51     pts = np.load(pts_path)
52     if pts.shape[1] != 2:
53         raise ValueError('pts_in_hull.npy should be shape (313,2)')
54     return pts
55
56
57 def ab_to_q(ab, pts):
58     # ab: (... , 2) with a,b in typical LAB ranges (a approx [-128,127])
59     # pts: (313, 2)
60     # Output: argmin bin index per pixel
61     h, w, _ = ab.shape
62     flat = ab.reshape(-1, 2)
63     # compute L2 distance to centers
64     # avoid huge memory for big images; use chunking
65     dists = np.linalg.norm(flat[:, None, :] - pts[None, :, :], axis=2)  # (h*w, 313)
66     q = np.argmin(dists, axis=1)
67     q = q.reshape(h, w)
68     return q
69
70
71 def q_to_ab_map(pred_probs, pts):
72     # pred_probs: (H, W, 313) softmax probabilities
73     # return ab channels as weighted sum of centers
74     H, W, _ = pred_probs.shape
75     probs = pred_probs.reshape(-1, pred_probs.shape[-1])  # (H*W, 313)
76     ab_flat = probs.dot(pts)  # (H*W, 2)
77     ab = ab_flat.reshape(H, W, 2)
78     return ab
79
80
81 # ----- Data generator -----
82
83 class ImageFolderGenerator(tf.keras.utils.Sequence):
84
85     def __init__(self, paths, pts=None, batch_size=16, target_size=(256, 256), shuffle=True, mode='eccv'):
86         self.paths = paths
87         self.batch_size = batch_size
88         self.target_size = target_size
89         self.shuffle = shuffle
90         self.mode = mode
91         self pts = pts
92         self.on_epoch_end()
93
94     def __len__(self):
95         return math.ceil(len(self.paths) / self.batch_size)
96
97     def on_epoch_end(self):
98         if self.shuffle:
99             random.shuffle(self.paths)
100
101    def __getitem__(self, idx):
102        batch_paths = self.paths[idx * self.batch_size:(idx + 1) * self.batch_size]
103        ls = []

```

```

104     bins = []
105     abs_reg = []
106     local_hints = []
107     global_feats = []
108
109     for p in batch_paths:
110         img = read_and_resize(p, self.target_size)
111         lab = rgb2lab(img) # L in [0,255], a,b roughly in [0,255]
112
113         L = lab[:, :, 0] / 255.0 # normalize 0-1
114         a = lab[:, :, 1] - 128.0 # center around zero roughly
115         b = lab[:, :, 2] - 128.0
116         ab = np.stack([a, b], axis=-1)
117
118         Ls.append(L.reshape(self.target_size[0], self.target_size[1], 1).astype(np.float32))
119         abs_reg.append(ab.astype(np.float32))
120
121     if self.mode in ['eccv', 'siggraph']:
122         if self.pts is None:
123             raise ValueError('pts centers required for ECCV-style quantization')
124         q = ab_to_q(ab, self.pts) # (H,W)
125         # convert to one-hot 313 channels
126         onehot = np.eye(len(self.pts), dtype=np.float32)[q] # (H,W,313)
127         bins.append(onehot)
128
129     if self.mode == 'siggraph':
130         # generate synthetic local hints: random sparse points/strokes with ab color values
131         hint = np.zeros((self.target_size[0], self.target_size[1], 2), dtype=np.float32)
132         num_points = random.randint(1, 20)
133         h, w = self.target_size
134         for i in range(num_points):
135             x = random.randint(0, w - 1)
136             y = random.randint(0, h - 1)
137             # spread the hint into small gaussian blob
138             rr = np.arange(h)[:, None]
139             cc = np.arange(w)[None, :]
140             sigma = random.uniform(1.0, 6.0)
141             gauss = np.exp(-((rr - y) ** 2 + (cc - x) ** 2) / (2 * sigma * sigma))
142             hint[:, :, 0] += gauss * ab[y, x, 0]
143             hint[:, :, 1] += gauss * ab[y, x, 1]
144         local_hints.append(hint)
145         # global features: simple per-channel mean & std + histogram bins (trivial)
146         gfeat = np.array([ab[:, :, 0].mean(), ab[:, :, 1].mean(), ab[:, :, 0].std(), ab[:, :, 1].std()], dtype=np.float32)
147         # pad to 216 dims as in paper by zeros (paper used 216-dim global descriptor)
148         gpad = np.zeros((216,), dtype=np.float32)
149         gpad[:gfeat.shape[0]] = gfeat
150         global_feats.append(gpad)
151
152     inputs = {'L': np.array(Ls)}
153     outputs_dict = {'q': np.array(bins)}
154
155     if self.mode == 'siggraph':
156         outputs_dict['ab_reg'] = np.array(abs_reg)
157         inputs['local_hint'] = np.array(local_hints)
158         inputs['global_hint'] = np.array(global_feats)
159         return inputs, outputs_dict
160     else: # For eccv mode (single output), return the tensor directly
161         return inputs, outputs_dict['q']
162
163
164 # ----- Models -----
165
166 def conv_block(x, filters, kernel=3, strides=1):
167     x = layers.Conv2D(filters, kernel, strides=strides, padding='same')(x)
168     x = layers.BatchNormalization()(x)
169     x = layers.ReLU()(x)
170     return x
171
172
173 def eccv16_model(input_shape=(256, 256, 1), n_bins=313):
174     L_in = layers.Input(shape=input_shape, name='L')
175     x = conv_block(L_in, 64)
176     x = conv_block(x, 64)
177     x = layers.MaxPool2D(2)(x)
178
179     x = conv_block(x, 128)
180     x = conv_block(x, 128)
181     x = layers.MaxPool2D(2)(x)
182
183     x = conv_block(x, 256)
184     x = conv_block(x, 256)
185     x = conv_block(x, 256)

```

```

186     x = layers.MaxPool2D(2)(x)
187
188     x = conv_block(x, 512)
189     x = conv_block(x, 512)
190     x = conv_block(x, 512)
191
192     x = conv_block(x, 256)
193     x = layers.UpSampling2D()(x)
194     x = conv_block(x, 128)
195     x = layers.UpSampling2D()(x)
196     x = conv_block(x, 64)
197     x = layers.UpSampling2D()(x)
198
199     out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)
200
201 model = Model(inputs=L_in, outputs=out, name='eccv16')
202 return model
203
204
205 def siggraph17_model(input_shape=(256, 256, 1), n_bins=313):
206     # Inputs: L, local_hint (H,W,2), global_hint (216,)
207     L_in = layers.Input(shape=input_shape, name='L')
208     local_hint = layers.Input(shape=(input_shape[0], input_shape[1], 2), name='local_hint')
209     global_hint = layers.Input(shape=(216,), name='global_hint')
210
211     # Simple encoder for L
212     x = conv_block(L_in, 64)
213     x = conv_block(x, 64)
214     x = layers.MaxPool2D(2)(x)
215
216     x = conv_block(x, 128)
217     x = conv_block(x, 128)
218     x = layers.MaxPool2D(2)(x)
219
220     x = conv_block(x, 256)
221     x = conv_block(x, 256)
222     x = conv_block(x, 256)
223
224     # incorporate local hint: concat/residual
225     lh = layers.Conv2D(32, 1, padding='same')(local_hint)
226     # downsample local hint a few times to match spatial dims
227     lh_down = layers.MaxPool2D(4)(lh)
228     x = layers.concatenate([x, lh_down])
229
230     x = conv_block(x, 512)
231
232     # global hint processing
233     g = layers.Dense(512, activation='relu')(global_hint)
234     g = layers.Dense(np.prod(x.shape[1:3]) * 16, activation='relu')(g)
235     g = layers.Reshape((x.shape[1], x.shape[2], 16))(g)
236     x = layers.concatenate([x, g])
237
238     x = conv_block(x, 256)
239     x = layers.UpSampling2D()(x)
240     x = conv_block(x, 128)
241     x = layers.UpSampling2D()(x)
242     x = conv_block(x, 64)
243     x = layers.UpSampling2D()(x)
244
245     q_out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)
246     ab_reg = layers.Conv2D(2, 1, activation='linear', name='ab_reg')(x)
247
248 model = Model(inputs=[L_in, local_hint, global_hint], outputs=[q_out, ab_reg], name='siggraph17')
249 return model
250
251
252 # ----- Training utilities -----
253
254 def compile_eccv(model, lr=1e-4):
255     model.compile(optimizer=Adam(lr), loss='categorical_crossentropy')
256     return model
257
258
259 def compile_siggraph(model, lr=1e-4):
260     # two outputs: q (categorical) and ab_reg (MSE)
261     losses = {'q': 'categorical_crossentropy', 'ab_reg': 'mse'}
262     loss_weights = {'q': 1.0, 'ab_reg': 1.0}
263     model.compile(optimizer=Adam(lr), loss=losses, loss_weights=loss_weights)
264     return model
265
266
267 # ----- Inference helpers -----

```

```

268
269 def eccv_infer(model, L_input, pts):
270     # L_input: (H,W,1) float 0-1
271     pred = model.predict(L_input[None, ...])[0]  # (H,W,313)
272     ab = q_to_ab_map(pred, pts)  # (H,W,2)
273     L = (L_input[:, :, 0] * 255.0).astype(np.float32)
274     lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
275     lab[:, :, 0] = L
276     lab[:, :, 1] = ab[:, :, 0] + 128.0
277     lab[:, :, 2] = ab[:, :, 1] + 128.0
278     rgb = lab2rgb(lab)
279     return rgb
280
281
282 def siggraph_infer(model, L_input, local_hint, global_hint, pts=None):
283     q_pred, ab_reg = model.predict([L_input[None, ...], local_hint[None, ...], global_hint[None, ...]])
284     q_pred = q_pred[0]
285     ab_reg = ab_reg[0]
286     if pts is not None:
287         ab_q = q_to_ab_map(q_pred, pts)
288         # fuse regression and quantized result by simple avg
289         ab = 0.5 * ab_reg + 0.5 * ab_q
290     else:
291         ab = ab_reg
292     L = (L_input[:, :, 0] * 255.0).astype(np.float32)
293     lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
294     lab[:, :, 0] = L
295     lab[:, :, 1] = ab[:, :, 0] + 128.0
296     lab[:, :, 2] = ab[:, :, 1] + 128.0
297     rgb = lab2rgb(lab)
298     return rgb
299
300
301 # ----- Main trainer -----
302
303 def main():
304     # ----- HARDCODED DATASET PATHS -----
305     DATASET_PATH = "/kaggle/input/landscape-image-colorization/landscape_Images/color"
306     PTS_PATH = "/content/pts_in_hull.npy"
307     SAVE_DIR = "models"
308     MODE = "both"  # eccv, siggraph, both
309     EPOCHS = 30
310     BATCH = 16
311     IMG_SIZE = 256
312     LR = 1e-4
313
314     # (Argparse removed; using hardcoded paths)
315     # parser = argparse.ArgumentParser()
316     # parser.add_argument('--dataset', required=True, help='Path to folder of images (recursive)')
317     # parser.add_argument('--pts', required=True, help='Path to pts_in_hull.npy (313x2)')
318     # parser.add_argument('--mode', choices=['eccv','siggraph','both'], default='eccv')
319     # parser.add_argument('--epochs', type=int, default=30)
320     # parser.add_argument('--batch', type=int, default=16)
321     # parser.add_argument('--save_dir', default='models')
322     # parser.add_argument('--lr', type=float, default=1e-4)
323     # parser.add_argument('--img_size', type=int, default=256)
324     # args = parser.parse_args()
325     # Using hardcoded variables instead:
326     class Args:
327         pass
328     args = Args()
329     args.dataset = DATASET_PATH
330     args.pts = PTS_PATH
331     args.mode = MODE
332     args.epochs = EPOCHS
333     args.batch = BATCH
334     args.save_dir = SAVE_DIR
335     args.lr = LR
336     args.img_size = IMG_SIZE
337
338     os.makedirs(args.save_dir, exist_ok=True)
339
340     pts = load_pts(args.pts)
341
342     paths = load_image_paths(args.dataset)
343     if len(paths) == 0:
344         raise ValueError('No images found in dataset path')
345
346     print(f'Found {len(paths)} images. Preparing generator...')
347
348     if args.mode in ['eccv', 'both']:
349         gen_eccv = ImageFolderGenerator(paths, pts=pts, batch_size=args.batch, target_size=(args.img_size, args.img_si

```

```

350     model_eccv = eccv16_model(input_shape=(args.img_size, args.img_size, 1), n_bins=pts.shape[0])
351     model_eccv = compile_eccv(model_eccv, lr=args.lr)
352
353     cb = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'eccv16_best.h5'), save_best_only=True, mode='auto')
354     print('Training ECCV16...')
355     model_eccv.fit(gen_eccv, epochs=args.epochs, callbacks=[cb])
356     print('Saving ECCV16 final weights...')
357     model_eccv.save(os.path.join(args.save_dir, 'colorization_release_v2-9b330a0b.pth'))
358     model_eccv.save(os.path.join(args.save_dir, 'eccv16_savedmodel'), save_format='tf')
359
360     if args.mode in ['siggraph', 'both']:
361         gen_sig = ImageFolderGenerator(paths, pts=pts, batch_size=max(1, args.batch // 2), target_size=(args.img_size, args.img_size))
362         model_sig = siggraph17_model(input_shape=(args.img_size, args.img_size, 1), n_bins=pts.shape[0])
363         model_sig = compile_siggraph(model_sig, lr=args.lr)
364
365         cb2 = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'siggraph17_best.h5'), save_best_only=True, mode='auto')
366         print('Training SIGGRAPH17...')
367         model_sig.fit(gen_sig, epochs=args.epochs, callbacks=[cb2])
368         print('Saving SIGGRAPH17 final weights...')
369         model_sig.save(os.path.join(args.save_dir, 'siggraph17-df00044c.pth'))
370         model_sig.save(os.path.join(args.save_dir, 'siggraph17_savedmodel'), save_format='tf')
371
372     print('Done.')
373
374
375 if __name__ == '__main__':
376     main()

```

```

Found 7129 images. Preparing generator...
Training ECCV16...
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class does not implement `self._warn_if_super_not_called()`. This is required by Keras.
Epoch 1/30
/usr/local/lib/python3.12/dist-packages/keras/src/models/functional.py:241: UserWarning: The structure of `inputs` doesn't match the structure of `outputs`.
Expected: L
Received: inputs=['Tensor(shape=(None, 256, 256, 1))']
  warnings.warn(msg)
5/446 [=====] 8:10:43 67s/step - loss: 5.8100

```

› Task

List the contents of the directory [/kaggle/input/landscape-image-colorization/](#).

↳ 2 cells hidden

› Task

Fix the `main` function in the training script (cell [FJH7AyYgZv91](#)) by removing the unused `parser.add_argument` calls and correcting the `IMG_SIZE` assignment to use the variable directly instead of calling it as a function, then execute the cell.

↳ 2 cells hidden

› Task

List the files and subdirectories within the [/kaggle/input/landscape-image-colorization/](#) directory again to definitively find the correct path for `pts_in_hull.npy`.

↳ 2 cells hidden

