📌 Step 1 — Load Dataset & Convert to Lab

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("theblackmamba31/landscape-image-colorization")

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
```

```
import cv2
import numpy as np
import glob

def load_images(path):
    files = glob.glob(path + "/*.jpg")

    L_list = []
    ab_list = []

    for f in files:
        img = cv2.imread(f)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (256, 256))

        lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
        L = lab[:, :, 0] / 255.0              # Normalize
        ab = lab[:, :, 1:] / 128.0            # Normalize

        L_list.append(L.reshape(256,256,1))
        ab_list.append(ab.reshape(256,256,2))

    return np.array(L_list), np.array(ab_list)

```

```
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:

```
import numpy as np
import os
import glob

# The 'path' variable from cell 579pssGiWTfD holds the root directory of the downloaded dataset.
# We need to find the pts_in_hull.npy file within this directory or its subdirectories.

pts_in_hull_files = glob.glob(os.path.join(path, '**', 'pts_in_hull.npy'), recursive=True)

if pts_in_hull_files:
    pts_in_hull_path = pts_in_hull_files[0]
    pts_in_hull = np.load(pts_in_hull_path)
    print(f"Loaded pts_in_hull from: {pts_in_hull_path}")
    print(f"Loaded pts_in_hull with shape: {pts_in_hull.shape}")
else:
    raise FileNotFoundError("pts_in_hull.npy not found in the dataset directory or its subdirectories.")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         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.")

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

Next steps: Explain error

📌 ECCV16 Model Code (Simplified Keras)

```
from keras.layers import *
from keras.models import Model

def eccv16_model():
    L_in = Input(shape=(256,256,1))

    x = Conv2D(64,3,activation='relu',padding='same')(L_in)
    x = Conv2D(64,3,activation='relu',padding='same',strides=2)(x)

    x = Conv2D(128,3,activation='relu',padding='same')(x)
    x = Conv2D(128,3,activation='relu',padding='same',strides=2)(x)

    x = Conv2D(256,3,activation='relu',padding='same')(x)
    x = Conv2D(256,3,activation='relu',padding='same')(x)
    x = Conv2D(256,3,activation='relu',padding='same',strides=2)(x)

    x = Conv2D(512,3,activation='relu',padding='same')(x)
    x = Conv2D(512,3,activation='relu',padding='same')(x)
    x = Conv2D(512,3,activation='relu',padding='same')(x)

    x = Conv2D(256,3,activation='relu',padding='same')(x)
    x = UpSampling2D()(x)
    x = Conv2D(128,3,activation='relu',padding='same')(x)
    x = UpSampling2D()(x)
    x = Conv2D(64,3,activation='relu',padding='same')(x)
    x = UpSampling2D()(x)

    out = Conv2D(313,1,activation='softmax')(x)

    return Model(L_in, out)

```

🧪 4. ECCV16 Training Procedure

Loss Function

ECCV16 uses:

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

```
model = eccv16_model()
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Training Loop

```
model.fit(
    train_L,
    train_bins,      # 313-channel encoded labels
    batch_size=32,
    epochs=50
)

```

```
---------------------------------------------------------------------------
NameError                                 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:

1️⃣ **Local hints (ab strokes)**

Shape: (256,256,2)

2️⃣ **Global hint vector (216 dims)**

Extracted using global features.

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

```

🧪 6. SIGGRAPH17 Training Loss

Uses two losses:

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

Final loss = (MSE + CE)

```
model.compile(
    optimizer='adam',
    loss=['mse','categorical_crossentropy'],
    loss_weights=[1.0, 0.3]
)

```

💾 7. Saving the Model

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

```

```
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)

```
pred = model.predict(L_img.reshape(1,256,256,1))
ab_img = decode_313_bins(pred)

lab = np.concatenate((L_img*255, ab_img), axis=-1)
rgb = cv2.cvtColor(lab.astype(np.uint8), cv2.COLOR_LAB2RGB)

```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/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

```
"""
ECCV16 + SIGGRAPH17 Colorization Training Script
Single-file training + inference + saving script for both models.

Usage examples:
    # Train ECCV16 automatic model
    python eccv16_siggraph17_colorization.py --mode eccv16 \
        --dataset /path/to/images --pts pts_in_hull.npy --epochs 40 --batch 32

    # Train SIGGRAPH17 user-guided model (uses same dataset; generates synthetic hints)
    python eccv16_siggraph17_colorization.py --mode siggraph \
        --dataset /path/to/images --pts pts_in_hull.npy --epochs 40 --batch 16

Notes:
- Expects images (.jpg/.png) in a single folder (no subfolders) for simplicity.
- Requires pts_in_hull.npy (313 ab-bin centers) for ECCV-style quantization.
- Uses TensorFlow / Keras.
- Save outputs: eccv16_weights.h5 and siggraph17_weights.h5 and SavedModel dirs.

Author: Generated by ChatGPT for user
"""

import os
import argparse
import glob
import random
import math
import numpy as np
from pathlib import Path
from tqdm import tqdm

import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.optimizers import Adam

import cv2

# ---------------------------- Utilities ----------------------------

def load_image_paths(dataset_dir, exts=(".jpg", ".jpeg", ".png")):
    files = []
    for ext in exts:
        files.extend(glob.glob(os.path.join(dataset_dir, f"**/*{ext}"), recursive=True))
    return sorted(files)


def read_and_resize(path, target_size=(256,256)):
    img = cv2.imread(path)
    if img is None:
        raise ValueError(f"Unable to read image: {path}")
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, target_size, interpolation=cv2.INTER_AREA)
    return img


# ---------------------------- color utilities ----------------------------

def rgb2lab(img_rgb):
    lab = cv2.cvtColor(img_rgb.astype(np.uint8), cv2.COLOR_RGB2LAB)
    return lab.astype(np.float32)


def lab2rgb(lab):
    lab = lab.astype(np.uint8)
    rgb = cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
    return rgb


# pts_in_hull.npy helper: contains 313 centers (ab values) used in ECCV paper
# You must provide pts_in_hull.npy from Zhang et al. repo or precomputed centers.

def load_pts(pts_path):
    pts = np.load(pts_path)
    if pts.shape[1] != 2:
```

```
    if pts.shape[1] != 2:
        raise ValueError("pts_in_hull.npy should be shape (313,2)")
    return pts


def ab_to_q(ab, pts):
    # ab: (...,2) with a,b in typical LAB ranges (a approx [-128,127])
    # pts: (313,2)
    # Output: argmin bin index per pixel
    h, w, _ = ab.shape
    flat = ab.reshape(-1,2)
    # compute L2 distance to centers
    # avoid huge memory for big images; use chunking
    dists = np.linalg.norm(flat[:,None,:] - pts[None,:,:], axis=2)  # (h*w,313)
    q = np.argmin(dists, axis=1)
    q = q.reshape(h,w)
    return q


def q_to_ab_map(pred_probs, pts):
    # pred_probs: (H, W, 313) softmax probabilities
    # return ab channels as weighted sum of centers
    H,W,_ = pred_probs.shape
    probs = pred_probs.reshape(-1, pred_probs.shape[-1])  # (H*W,313)
    ab_flat = probs.dot(pts)  # (H*W,2)
    ab = ab_flat.reshape(H,W,2)
    return ab


# ----------------------------- Data generator -----------------------------

class ImageFolderGenerator(tf.keras.utils.Sequence):
    def __init__(self, paths, pts=None, batch_size=16, target_size=(256,256), shuffle=True, mode='eccv'):
        self.paths = paths
        self.batch_size = batch_size
        self.target_size = target_size
        self.shuffle = shuffle
        self.mode = mode
        self.pts = pts
        self.on_epoch_end()

    def __len__(self):
        return math.ceil(len(self.paths)/self.batch_size)

    def on_epoch_end(self):
        if self.shuffle:
            random.shuffle(self.paths)

    def __getitem__(self, idx):
        batch_paths = self.paths[idx*self.batch_size : (idx+1)*self.batch_size]
        Ls = []
        bins = []
        abs_reg = []
        local_hints = []
        global_feats = []

        for p in batch_paths:
            img = read_and_resize(p, self.target_size)
            lab = rgb2lab(img)  # L in [0,255], a,b roughly in [0,255]

            L = lab[:,:,0] / 255.0  # normalize 0-1
            a = lab[:,:,1] - 128.0  # center around zero roughly
            b = lab[:,:,2] - 128.0
            ab = np.stack([a,b], axis=-1)

            Ls.append(L.reshape(self.target_size[0], self.target_size[1], 1).astype(np.float32))
            abs_reg.append(ab.astype(np.float32))

            if self.mode in ['eccv', 'siggraph']:
                if self.pts is None:
                    raise ValueError("pts centers required for ECCV-style quantization")
                q = ab_to_q(ab, self.pts)  # (H,W)
                # convert to one-hot 313 channels
                onehot = np.eye(len(self.pts), dtype=np.float32)[q]  # (H,W,313)
                bins.append(onehot)

            if self.mode == 'siggraph':
                # generate synthetic local hints: random sparse points/strokes with ab color values
                hint = np.zeros((self.target_size[0], self.target_size[1], 2), dtype=np.float32)
                num_points = random.randint(1, 20)
                h,w = self.target_size
                for i in range(num_points):
                    x = random.randint(0, w-1)
                    y = random.randint(0, h-1)
                    # spread the hint into small gaussian blob
                    rr = np.arange(h)[:,None]
                    cc = np.arange(w)[None,:]
                    sigma = random.uniform(1.0, 6.0)
                    gauss = np.exp(-((rr-y)**2 + (cc-x)**2)/(2*sigma*sigma))
                    hint[:,:,0] += gauss * ab[y,x,0]
                    hint[:,:,1] += gauss * ab[y,x,1]
                local_hints.append(hint)
                # global features: simple per-channel mean & std + histogram bins (trivial)
                gfeat = np.array([ab[:,:,0].mean(), ab[:,:,1].mean(), ab[:,:,0].std(), ab[:,:,1].std()], dtype=np.float
                # pad to 216 dims as in paper by zeros (paper used 216-dim global descriptor)
                gpad = np.zeros((216,), dtype=np.float32)
                gpad[:gfeat.shape[0]] = gfeat
                global_feats.append(gpad)

        inputs = {'L': np.array(Ls)}
        outputs = {}

        if len(bins) > 0:
            outputs['q'] = np.array(bins)
        if len(abs_reg) > 0:
            outputs['ab_reg'] = np.array(abs_reg)
        if self.mode == 'siggraph':
            inputs['local_hint'] = np.array(local_hints)
            inputs['global_hint'] = np.array(global_feats)

        return inputs, outputs


# ----------------------------- Models -----------------------------

def conv_block(x, filters, kernel=3, strides=1):
    x = layers.Conv2D(filters, kernel, strides=strides, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)
    return x


def eccv16_model(input_shape=(256,256,1), n_bins=313):
    L_in = layers.Input(shape=input_shape, name='L')
    x = conv_block(L_in, 64)
    x = conv_block(x, 64)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 128)
    x = conv_block(x, 128)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 256)
    x = conv_block(x, 256)
    x = conv_block(x, 256)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 512)
    x = conv_block(x, 512)
    x = conv_block(x, 512)

    x = conv_block(x, 256)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 128)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 64)
    x = layers.UpSampling2D()(x)

    out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)

    model = Model(inputs=L_in, outputs=out, name='eccv16')
    return model


def siggraph17_model(input_shape=(256,256,1), n_bins=313):
    # Inputs: L, local_hint (H,W,2), global_hint (216,)
    L_in = layers.Input(shape=input_shape, name='L')
    local_hint = layers.Input(shape=(input_shape[0], input_shape[1], 2), name='local_hint')
    global_hint = layers.Input(shape=(216,), name='global_hint')

    # Simple encoder for L
    x = conv_block(L_in, 64)
    x = conv_block(x, 64)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 128)
    x = conv_block(x, 128)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 256)
    x = conv_block(x, 256)
    x = conv_block(x, 256)

    # incorporate local hint: concat/residual
    lh = layers.Conv2D(32, 1, padding='same')(local_hint)
    # downsample local hint a few times to match spatial dims
    lh_down = layers.MaxPool2D(4)(lh)
    x = layers.Concatenate()([x, lh_down])

    x = conv_block(x, 512)

    # global hint processing
    g = layers.Dense(512, activation='relu')(global_hint)
    g = layers.Dense(np.prod(x.shape[1:3]) * 16, activation='relu')(g)
    g = layers.Reshape((x.shape[1], x.shape[2], 16))(g)
    x = layers.Concatenate()([x, g])

    x = conv_block(x, 256)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 128)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 64)
    x = layers.UpSampling2D()(x)

    q_out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)
    ab_reg = layers.Conv2D(2, 1, activation='linear', name='ab_reg')(x)

    model = Model(inputs=[L_in, local_hint, global_hint], outputs=[q_out, ab_reg], name='siggraph17')
    return model


# ----------------------------- Training utilities -----------------------------

def compile_eccv(model, lr=1e-4):
    model.compile(optimizer=Adam(lr), loss='categorical_crossentropy')
    return model


def compile_siggraph(model, lr=1e-4):
    # two outputs: q (categorical) and ab_reg (MSE)
    losses = {'q': 'categorical_crossentropy', 'ab_reg': 'mse'}
    loss_weights = {'q': 1.0, 'ab_reg': 1.0}
    model.compile(optimizer=Adam(lr), loss=losses, loss_weights=loss_weights)
    return model


# ----------------------------- Inference helpers -----------------------------

def eccv_infer(model, L_input, pts):
    # L_input: (H,W,1) float 0-1
    pred = model.predict(L_input[None,...])[0]  # (H,W,313)
    ab = q_to_ab_map(pred, pts)  # (H,W,2)
    L = (L_input[:,:,0]*255.0).astype(np.float32)
    lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
    lab[:,:,0] = L
    lab[:,:,1] = ab[:,:,0] + 128.0
    lab[:,:,2] = ab[:,:,1] + 128.0
    rgb = lab2rgb(lab)
    return rgb


def siggraph_infer(model, L_input, local_hint, global_hint, pts=None):
    q_pred, ab_reg = model.predict([L_input[None,...], local_hint[None,...], global_hint[None,...]])
    q_pred = q_pred[0]
    ab_reg = ab_reg[0]
    if pts is not None:
        ab_q = q_to_ab_map(q_pred, pts)
        # fuse regression and quantized result by simple avg
        ab = 0.5*ab_reg + 0.5*ab_q
    else:
        ab = ab_reg
    L = (L_input[:,:,0]*255.0).astype(np.float32)
    lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
    lab[:,:,0] = L
    lab[:,:,1] = ab[:,:,0] + 128.0
    lab[:,:,2] = ab[:,:,1] + 128.0
    rgb = lab2rgb(lab)
    return rgb
```

```

# ---------------------------- Main trainer ----------------------------

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--dataset', required=True, help='Path to folder of images (recursive)')
    parser.add_argument('--pts', required=True, help='Path to pts_in_hull.npy (313x2)')
    parser.add_argument('--mode', choices=['eccv','siggraph','both'], default='eccv')
    parser.add_argument('--epochs', type=int, default=30)
    parser.add_argument('--batch', type=int, default=16)
    parser.add_argument('--save_dir', default='models')
    parser.add_argument('--lr', type=float, default=1e-4)
    parser.add_argument('--img_size', type=int, default=256)
    args = parser.parse_args()

    os.makedirs(args.save_dir, exist_ok=True)

    pts = load_pts(args.pts)

    paths = load_image_paths(args.dataset)
    if len(paths) == 0:
        raise ValueError('No images found in dataset path')

    print(f"Found {len(paths)} images. Preparing generator...")

    if args.mode in ['eccv','both']:
        gen_eccv = ImageFolderGenerator(paths, pts=pts, batch_size=args.batch, target_size=(args.img_size,args.img_size
        model_eccv = eccv16_model(input_shape=(args.img_size,args.img_size,1), n_bins=pts.shape[0])
        model_eccv = compile_eccv(model_eccv, lr=args.lr)

        cb = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'eccv16_best.h5'), save_best_only=True, mor
        print('Training ECCV16...')
        model_eccv.fit(gen_eccv, epochs=args.epochs, callbacks=[cb])
        print('Saving ECCV16 final weights...')
        model_eccv.save(os.path.join(args.save_dir, 'eccv16_final.h5'))
        model_eccv.save(os.path.join(args.save_dir, 'eccv16_savedmodel'), save_format='tf')

    if args.mode in ['siggraph','both']:
        gen_sig = ImageFolderGenerator(paths, pts=pts, batch_size=max(1,args.batch//2), target_size=(args.img_size,args
        model_sig = siggraph17_model(input_shape=(args.img_size,args.img_size,1), n_bins=pts.shape[0])
        model_sig = compile_siggraph(model_sig, lr=args.lr)

        cb2 = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'siggraph17_best.h5'), save_best_only=True
        print('Training SIGGRAPH17...')
        model_sig.fit(gen_sig, epochs=args.epochs, callbacks=[cb2])
        print('Saving SIGGRAPH17 final weights...')
        model_sig.save(os.path.join(args.save_dir, 'siggraph17_final.h5'))
        model_sig.save(os.path.join(args.save_dir, 'siggraph17_savedmodel'), save_format='tf')

    print('Done.')


if __name__ == '__main__':
    main()

```

Hardcoded Path

```
import os
import argparse
import glob
import random
import math
import numpy as np
from pathlib import Path
from tqdm import tqdm

import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.optimizers import Adam

import cv2

# ---------------------------- Utilities ----------------------------

def load_image_paths(dataset_dir, exts=('.jpg', '.jpeg', '.png')):
    files = []
    for ext in exts:
        files.extend(glob.glob(os.path.join(dataset_dir, f'**/*{ext}'), recursive=True))
    return sorted(files)


def read_and_resize(path, target_size=(256, 256)):
    img = cv2.imread(path)
    if img is None:
        raise ValueError(f'Unable to read image: {path}')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, target_size, interpolation=cv2.INTER_AREA)
    return img


# ----------------------------- color utilities -----------------------------

def rgb2lab(img_rgb):
    lab = cv2.cvtColor(img_rgb.astype(np.uint8), cv2.COLOR_RGB2LAB)
    return lab.astype(np.float32)


def lab2rgb(lab):
    lab = lab.astype(np.uint8)
    rgb = cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
    return rgb


# pts_in_hull.npy helper: contains 313 centers (ab values) used in ECCV paper
# You must provide pts_in_hull.npy from Zhang et al. repo or precomputed centers.

def load_pts(pts_path):
    pts = np.load(pts_path)
    if pts.shape[1] != 2:
        raise ValueError('pts_in_hull.npy should be shape (313,2)')
    return pts


def ab_to_q(ab, pts):
    # ab: (...,2) with a,b in typical LAB ranges (a approx [-128,127])
    # pts: (313,2)
    # Output: argmin bin index per pixel
    h, w, _ = ab.shape
    flat = ab.reshape(-1, 2)
    # compute L2 distance to centers
    # avoid huge memory for big images; use chunking
    dists = np.linalg.norm(flat[:, None, :] - pts[None, :, :], axis=2)  # (h*w,313)
    q = np.argmin(dists, axis=1)
    q = q.reshape(h, w)
    return q


def q_to_ab_map(pred_probs, pts):
    # pred_probs: (H, W, 313) softmax probabilities
    # return ab channels as weighted sum of centers
    H, W, _ = pred_probs.shape
    probs = pred_probs.reshape(-1, pred_probs.shape[-1])  # (H*W,313)
    ab_flat = probs.dot(pts)  # (H*W,2)
    ab = ab_flat.reshape(H, W, 2)
    return ab


# ----------------------------- Data generator -----------------------------

class ImageFolderGenerator(tf.keras.utils.Sequence):

    def __init__(self, paths, pts=None, batch_size=16, target_size=(256, 256), shuffle=True, mode='eccv'):
        self.paths = paths
        self.batch_size = batch_size
        self.target_size = target_size
        self.shuffle = shuffle
        self.mode = mode
        self.pts = pts
        self.on_epoch_end()

    def __len__(self):
        return math.ceil(len(self.paths) / self.batch_size)

    def on_epoch_end(self):
        if self.shuffle:
            random.shuffle(self.paths)

    def __getitem__(self, idx):
        batch_paths = self.paths[idx * self.batch_size:(idx + 1) * self.batch_size]
        Ls = []
        bins = []
        abs_reg = []
        local_hints = []
        global_feats = []

        for p in batch_paths:
            img = read_and_resize(p, self.target_size)
            lab = rgb2lab(img)  # L in [0,255], a,b roughly in [0,255]

            L = lab[:, :, 0] / 255.0  # normalize 0-1
            a = lab[:, :, 1] - 128.0  # center around zero roughly
            b = lab[:, :, 2] - 128.0
            ab = np.stack([a, b], axis=-1)

            Ls.append(L.reshape(self.target_size[0], self.target_size[1], 1).astype(np.float32))
            abs_reg.append(ab.astype(np.float32))

            if self.mode in ['eccv', 'siggraph']:
                if self.pts is None:
                    raise ValueError('pts centers required for ECCV-style quantization')
                q = ab_to_q(ab, self.pts)  # (H,W)
                # convert to one-hot 313 channels
                onehot = np.eye(len(self.pts), dtype=np.float32)[q]  # (H,W,313)
                bins.append(onehot)

            if self.mode == 'siggraph':
                # generate synthetic local hints: random sparse points/strokes with ab color values
                hint = np.zeros((self.target_size[0], self.target_size[1], 2), dtype=np.float32)
                num_points = random.randint(1, 20)
                h, w = self.target_size
                for i in range(num_points):
                    x = random.randint(0, w - 1)
                    y = random.randint(0, h - 1)
                    # spread the hint into small gaussian blob
                    rr = np.arange(h)[:, None]
                    cc = np.arange(w)[None, :]
                    sigma = random.uniform(1.0, 6.0)
                    gauss = np.exp(-((rr - y) ** 2 + (cc - x) ** 2) / (2 * sigma * sigma))
                    hint[:, :, 0] += gauss * ab[y, x, 0]
                    hint[:, :, 1] += gauss * ab[y, x, 1]
                local_hints.append(hint)
                # global features: simple per-channel mean & std + histogram bins (trivial)
                gfeat = np.array([ab[:, :, 0].mean(), ab[:, :, 1].mean(), ab[:, :, 0].std(), ab[:, :, 1].std()], dtype
                # pad to 216 dims as in paper by zeros (paper used 216-dim global descriptor)
                gpad = np.zeros((216,), dtype=np.float32)
                gpad[:gfeat.shape[0]] = gfeat
                global_feats.append(gpad)

        inputs = {'L': np.array(Ls)}
        outputs_dict = {'q': np.array(bins)}

        if self.mode == 'siggraph':
            outputs_dict['ab_reg'] = np.array(abs_reg)
            inputs['local_hint'] = np.array(local_hints)
            inputs['global_hint'] = np.array(global_feats)
            return inputs, outputs_dict
        else:  # For eccv mode (single output), return the tensor directly
            return inputs, outputs_dict['q']


# ----------------------------- Models -----------------------------

def conv_block(x, filters, kernel=3, strides=1):
    x = layers.Conv2D(filters, kernel, strides=strides, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)
    return x


def eccv16_model(input_shape=(256, 256, 1), n_bins=313):
    L_in = layers.Input(shape=input_shape, name='L')
    x = conv_block(L_in, 64)
    x = conv_block(x, 64)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 128)
    x = conv_block(x, 128)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 256)
    x = conv_block(x, 256)
    x = conv_block(x, 256)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 512)
    x = conv_block(x, 512)
    x = conv_block(x, 512)

    x = conv_block(x, 256)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 128)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 64)
    x = layers.UpSampling2D()(x)

    out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)

    model = Model(inputs=L_in, outputs=out, name='eccv16')
    return model


def siggraph17_model(input_shape=(256, 256, 1), n_bins=313):
    # Inputs: L, local_hint (H,W,2), global_hint (216,)
    L_in = layers.Input(shape=input_shape, name='L')
    local_hint = layers.Input(shape=(input_shape[0], input_shape[1], 2), name='local_hint')
    global_hint = layers.Input(shape=(216,), name='global_hint')

    # Simple encoder for L
    x = conv_block(L_in, 64)
    x = conv_block(x, 64)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 128)
    x = conv_block(x, 128)
    x = layers.MaxPool2D(2)(x)

    x = conv_block(x, 256)
    x = conv_block(x, 256)
    x = conv_block(x, 256)

    # incorporate local hint: concat/residual
    lh = layers.Conv2D(32, 1, padding='same')(local_hint)
    # downsample local hint a few times to match spatial dims
    lh_down = layers.MaxPool2D(4)(lh)
    x = layers.Concatenate()([x, lh_down])

    x = conv_block(x, 512)

    # global hint processing
    g = layers.Dense(512, activation='relu')(global_hint)
    g = layers.Dense(np.prod(x.shape[1:3]) * 16, activation='relu')(g)
    g = layers.Reshape((x.shape[1], x.shape[2], 16))(g)
    x = layers.Concatenate()([x, g])

    x = conv_block(x, 256)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 128)
    x = layers.UpSampling2D()(x)
    x = conv_block(x, 64)
    x = layers.UpSampling2D()(x)

    q_out = layers.Conv2D(n_bins, 1, activation='softmax', name='q')(x)
    ab_reg = layers.Conv2D(2, 1, activation='linear', name='ab_reg')(x)

    model = Model(inputs=[L_in, local_hint, global_hint], outputs=[q_out, ab_reg], name='siggraph17')
    return model


# ----------------------------- Training utilities -----------------------------

def compile_eccv(model, lr=1e-4):
    model.compile(optimizer=Adam(lr), loss='categorical_crossentropy')
    return model


def compile_siggraph(model, lr=1e-4):
    # two outputs: q (categorical) and ab_reg (MSE)
    losses = {'q': 'categorical_crossentropy', 'ab_reg': 'mse'}
    loss_weights = {'q': 1.0, 'ab_reg': 1.0}
    model.compile(optimizer=Adam(lr), loss=losses, loss_weights=loss_weights)
    return model


# ----------------------------- Inference helpers -----------------------------
```

```

def eccv_infer(model, L_input, pts):
    # L_input: (H,W,1) float 0-1
    pred = model.predict(L_input[None, ...])[0]  # (H,W,313)
    ab = q_to_ab_map(pred, pts)  # (H,W,2)
    L = (L_input[:, :, 0] * 255.0).astype(np.float32)
    lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
    lab[:, :, 0] = L
    lab[:, :, 1] = ab[:, :, 0] + 128.0
    lab[:, :, 2] = ab[:, :, 1] + 128.0
    rgb = lab2rgb(lab)
    return rgb


def siggraph_infer(model, L_input, local_hint, global_hint, pts=None):
    q_pred, ab_reg = model.predict([L_input[None, ...], local_hint[None, ...], global_hint[None, ...]])
    q_pred = q_pred[0]
    ab_reg = ab_reg[0]
    if pts is not None:
        ab_q = q_to_ab_map(q_pred, pts)
        # fuse regression and quantized result by simple avg
        ab = 0.5 * ab_reg + 0.5 * ab_q
    else:
        ab = ab_reg
    L = (L_input[:, :, 0] * 255.0).astype(np.float32)
    lab = np.zeros((L.shape[0], L.shape[1], 3), dtype=np.float32)
    lab[:, :, 0] = L
    lab[:, :, 1] = ab[:, :, 0] + 128.0
    lab[:, :, 2] = ab[:, :, 1] + 128.0
    rgb = lab2rgb(lab)
    return rgb


# ----------------------------- Main trainer -----------------------------

def main():
    # ----------- HARDCODED DATASET PATHS -----------
    DATASET_PATH = "/kaggle/input/landscape-image-colorization/landscape Images/color"
    PTS_PATH = "/content/pts_in_hull.npy"
    SAVE_DIR = "models"
    MODE = "both"  # eccv, siggraph, both
    EPOCHS = 30
    BATCH = 16
    IMG_SIZE = 256
    LR = 1e-4

    # (Argparse removed; using hardcoded paths)
    # parser = argparse.ArgumentParser()()
    # parser.add_argument('--dataset', required=True, help='Path to folder of images (recursive)')
    # parser.add_argument('--pts', required=True, help='Path to pts_in_hull.npy (313x2)')
    # parser.add_argument('--mode', choices=['eccv','siggraph','both'], default='eccv')
    # parser.add_argument('--epochs', type=int, default=30)
    # parser.add_argument('--batch', type=int, default=16)
    # parser.add_argument('--save_dir', default='models')
    # parser.add_argument('--lr', type=float, default=1e-4)
    # parser.add_argument('--img_size', type=int, default=256)
    # args = parser.parse_args()
    # Using hardcoded variables instead:
    class Args:
        pass
    args = Args()
    args.dataset = DATASET_PATH
    args.pts = PTS_PATH
    args.mode = MODE
    args.epochs = EPOCHS
    args.batch = BATCH
    args.save_dir = SAVE_DIR
    args.lr = LR
    args.img_size = IMG_SIZE

    os.makedirs(args.save_dir, exist_ok=True)

    pts = load_pts(args.pts)

    paths = load_image_paths(args.dataset)
    if len(paths) == 0:
        raise ValueError('No images found in dataset path')

    print(f'Found {len(paths)} images. Preparing generator...')

    if args.mode in ['eccv', 'both']:
        gen_eccv = ImageFolderGenerator(paths, pts=pts, batch_size=args.batch, target_size=(args.img_size, args.img_s
        model_eccv = eccv16_model(input_shape=(args.img_size, args.img_size, 1), n_bins=pts.shape[0])
        model_eccv = compile_eccv(model_eccv, lr=args.lr)

        cb = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'eccv16_best.h5'), save_best_only=True, mc
        print('Training ECCV16...')
        model_eccv.fit(gen_eccv, epochs=args.epochs, callbacks=[cb])
        print('Saving ECCV16 final weights...')
        model_eccv.save(os.path.join(args.save_dir, 'colorization_release_v2-9b330a0b.pth'))
        model_eccv.save(os.path.join(args.save_dir, 'eccv16_savedmodel'), save_format='tf')

    if args.mode in ['siggraph', 'both']:
        gen_sig = ImageFolderGenerator(paths, pts=pts, batch_size=max(1, args.batch // 2), target_size=(args.img_size,
        model_sig = siggraph17_model(input_shape=(args.img_size, args.img_size, 1), n_bins=pts.shape[0])
        model_sig = compile_siggraph(model_sig, lr=args.lr)

        cb2 = tf.keras.callbacks.ModelCheckpoint(os.path.join(args.save_dir, 'siggraph17_best.h5'), save_best_only=Tru
        print('Training SIGGRAPH17...')
        model_sig.fit(gen_sig, epochs=args.epochs, callbacks=[cb2])
        print('Saving SIGGRAPH17 final weights...')
        model_sig.save(os.path.join(args.save_dir, 'siggraph17-df00044c.pth'))
        model_sig.save(os.path.join(args.save_dir, 'siggraph17_savedmodel'), save_format='tf')

    print('Done.')


if __name__ == '__main__':
    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 `PyDat
  self._warn_if_super_not_called()
Epoch 1/30
/usr/local/lib/python3.12/dist-packages/keras/src/models/functional.py:241: UserWarning: The structure of `inputs` doesn't m
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