Chapter 14
Robust Image Enhancement

Yanhua Huang

Abstract Deep generative models such as GAN and Unet have achieved significant progress over classic methods in several computer vision tasks like super-resolution and segmentation. However, such learning-based methods lack robustness and interpretability, which limits their applications in real-world situations. In this chapter, we discuss a robust way for image enhancement that can combine a number of interpretable techniques through deep reinforcement learning. We first present some background about image enhancement. Then we formulate the image enhancement as a pipeline modeled by MDP. Finally, we show how to implement an agent on this MDP with PPO algorithm. The experimental environment is constructed by a real-world dataset that contains 5000 photographs with both the raw images and adjusted versions by experts. Codes are available at: https://github.com/deep-reinforcement-learning-book/Chapter14-Robust-Image-Enhancement.

Keywords Image processing · Image enhancement · Robust learning

14.1 Image Enhancement

Image enhancement belongs to image processing techniques. Its principal objective is to make the processed images more suitable for the needs of various applications. Typical image enhancement techniques contain denoising, deblurring, and brightness improvement. Real-world images always need multiple image enhancement techniques. Figure 14.1 shows an enhancement pipeline that consists of brightness improvements and denoising. Professional photo editing software, such as Adobe Photoshop, allows powerful image retouching but is not efficient and requires expertise in photo editing for users. In large-scale situations like recommendation systems, the subjective quality of images is vital for user experience, where an automatic image enhancement method that satisfies effectiveness, robustness, and

Y. Huang (✉)
Xiaohongshu Technology Co., Ltd., Shanghai, China

H. Dong et al. (eds.), *Deep Reinforcement Learning*,
https://doi.org/10.1007/978-981-15-4095-0_14

Fig. 14.1 An example of image enhancement pipeline. The raw image in the left is underexposed with JPEG compression noise

efficiency is needed. In particular, robustness is the most important condition, especially in user-generated content platforms, e.g., Facebook and Twitter, even if 1% of enhancement results are bad it will hurt millions of users.

Unlike image classification or segmentation that has a unique ground truth, the training data of image enhancement relies on human experts. As a result, no large-scale public dataset for image enhancement is available. Classical methods are mainly based on gamma correction and histogram equalization that enhance the image with the help of prior expert knowledge. These methods do not require a large amount of data either. Gamma correction takes advantage of nonlinearity in human perception such as our capacity to perceive light and color (Poynton 2012). Histogram equalization achieves the idea that allows areas of lower local contrast to gain a higher contrast for better distribution on the pixel histogram, which is useful when backgrounds and foregrounds are both bright or both dark such as X-ray images. Although these methods are fast and simple, the lack of consideration of contextual information limits their performance.

Recently, learning-based methods, which try to approximate the mapping from the input image to the desired pixel values with CNN, have achieved great success (Bychkovsky et al. 2011; Ulyanov et al. 2018; Kupyn et al. 2018; Wang et al. 2019). However, such methods are not without issues. First of all, it is hard to train a comprehensive neural network that can handle multiple enhancement situations. Besides, pixel-to-pixel mapping lacks robustness, e.g., it does not perform very well when dealing with some detailed information such as hair and characters (Zhang et al. 2019; Nataraj et al. 2019). Some researchers have proposed to apply deep reinforcement learning to image enhancement by formulating the enhancement procedure as a sequence of iterative decision-making problems to address the challenges above (Yu et al. 2018; Park et al. 2018; Furuta et al. 2019). In this chapter, we follow these methods and propose a new MDP formulation for image enhancement. We demonstrate our approach on a dataset containing 5000 pairs of images with code examples, for providing a quick hands-on learning process.

Before discussing the algorithm, we introduce two Python libraries *Pillow* (Clark 2015) and *scikit-image* (Van der Walt et al. 2014) that provide a number of friendly interfaces to implement image enhancement. One can install them directly from PyPI as follows:

```
pip install Pillow
pip install scikit-image
```

Here is an example code for contrast adjustment by Pillow's sub-module ImageEnhance.

```
from PIL import ImageEnhance

def adjust_contrast(image_rgb, contrast_factor):
    """Adjust contrast
    Args:
        image_rgb (PIL.Image): RGB image
        contrast_factor (float): color balance factor range from 0
            to 1.
    Return:
        PIL.Image object
    """
    enhancer = ImageEnhance.Contrast(image_rgb)
    return enhancer.enhance(contrast_factor)
```

14.2 Reinforcement Learning for Robust Processing

When applying reinforcement learning to image enhancement, one needs to first consider how to construct an MDP in this domain. An idea that naturally emerges is to consider processing pixels to be states and different image enhancement technologies to be actions in the context of reinforcement learning. This formulation provides a combination method of several controllable primary enhancers to achieve robust and effective results. In this section, we discuss such a reinforcement learning-based color enhancement method. For simplicity, we only take global enhancement actions. Note that it is natural to adapt to general enhancement algorithms by adding region proposal modules (Ren et al. 2015).

Suppose that the training dataset contains N pairs of RGB images $\{(l_i, h_i)\}_{i=1}^{N}$ where l_i is the low-quality raw image and h_i is the high-quality retouched image. In order to maintain the data distribution, the initial state S_0 should be sampled from $\{l_i\}_{i=1}^{N}$ uniformly. In each step, the agent takes a predefined action such as contrast adjustment with a certain factor and then applies it to the current state. Note that the current state and selected action fully determine the transition, i.e., no environment uncertainty exists. Following previous works (Park et al. 2018; Furuta et al. 2019), we use the improvement on CIELAB color space as the transition reward function:

$$||L(h) - L(S_t)||_2^2 - ||L(h) - L(S_{t+1})||_2^2 \tag{14.1}$$

where h is the corresponding high-quality image of S_0 and L maps images from RGB color space to CIELAB color space.

Another important thing is the terminal condition during learning and evaluation. Unlike reinforcement learning applications on games where the terminal state can be determined by the environment, agents in image enhancement need to decide an exit time by themselves. Park et al. (2018) proposed a DQN-based agent that exits when all predicted Q-values are negative. However, the overestimation problem of function approximation in Q-learning might lead to less robust results during inference. We address this issue by training an explicit policy and adding a "NO-OP" action to represent the exit choice. Table 14.1 lists all predefined actions, where the action with index 0 represents "NO-OP."

Training a convolutional neural network from scratch needs a large amount of retouched image pairs. Instead of using raw image states as observations, we consider the activation of the last convolutional layer in ResNet50 pre-trained on the ILSVRC classification dataset (Russakovsky et al. 2015), which is a significant deep feature that improves many other visual recognition tasks (Ren et al. 2016; Redmon et al. 2016). Inspired by previous work (Park et al. 2018; Lee et al. 2005), we further consider the histogram information when constructing observations. Specifically, we calculate the histogram statistics of the state in RGB color space over ranges $(0, 255)$, $(0, 255)$, $(0, 255)$, and CIELAB color space over ranges $(0, 100)$, $(-60, 60)$, $(-60, 60)$. These three features are concatenated as $2048 + 2000$ dimensional observations. We select PPO (Schulman et al. 2017) as the policy optimization algorithm. PPO is an actor-critic method that achieves significant results on a number of tasks. The network consists of three parts: three-layers feature extractor serving as a backbone, one-layer actor, and one-layer critic. All layers are fully connected, where the outputs of the layers in feature extractor are 2048, 512, and 128 units with ReLU activation, respectively.

Table 14.1 The action set for global color enhancement

Index	Description
0	No operation
1	Contrast $\times 0.95$
2	Contrast $\times 1.05$
3	Saturation $\times 0.95$
4	Saturation $\times 1.05$
5	Brightness $\times 0.95$
6	Brightness $\times 1.05$
7	Red and green $\times 0.95$
8	Red and green $\times 1.05$
9	Green and blue $\times 0.95$
10	Green and blue $\times 1.05$
11	Red and blue $\times 0.95$
12	Red and blue $\times 1.05$

Table 14.2 Hyper-parameters of PPO for image enhancement

Hyper-parameter	Value
Optimizer	Adam
Learning rate	1e−5
Clip norm	1.0
GAE λ	0.95
Episodes per iter	4
Optimization per iter	2
Max iter	10,000
Entropy factor	1e−2
Reward scale	0.1
Reward clip	[−1, 1]
γ	0.95

We evaluated our method on the MIT-Adobe FiveK (Bychkovsky et al. 2011) dataset including 5000 raw images, each with five retouched images produced by different experts (A/B/C/D/E). Following previous work (Park et al. 2018; Wang et al. 2019), we only use the retouched images by Expert C, which randomly selected 4500 images for training and the rest 500 images for testing. The raw images are DNG format while the retouched images are TIFF format. We convert all of them to JPEG format with quality 100 and color space sRGB by Adobe Lightroom. For efficient training, we resized images such that the maximal side consists of 512 pixels for each image. Hyper-parameters are provided in Table 14.2.

From now on, we demonstrate how to implement the algorithm above. First of all, we need to construct an environment object.

```
class Env(object):
    """Training env wrapper of image processing RL problem"""
    def __init__(self, src, max_episode_length=20,
        reward_scale=0.1):
        """
        Args:
            src (list[str, str]): list of raw and retouched path,
                initial
                            state will sample from it uniformly
            max_episode_length (int): max number of actions can be
                taken
        """
        self._src = src
        self._backbone = backbone
        self._preprocess = preprocess
        self._rgb_state = None
        self._lab_state = None
        self._target_lab = None
        self._current_diff = None
        self._count = 0
        self._max_episode_length = max_episode_length
        self._reward_scale = reward_scale
        self._info = dict()
```

With the ResNet API from TensorFlow, we build the observation by function `_state_feature` as follows:

```
backbone = tf.keras.applications.ResNet50(include_top=False,
    pooling='avg')
preprocess = tf.keras.applications.resnet50.preprocess_input

def get_lab_hist(lab):
   """Get hist of lab image"""
   lab = lab.reshape(-1, 3)
   hist, _ = np.histogramdd(lab, bins=(10, 10, 10),
                      range=((0, 100), (-60, 60), (-60, 60)))
   return hist.reshape(1, 1000) / 1000.0

def get_rgb_hist(lab):
   """Get hist of lab image"""
   lab = lab.reshape(-1, 3)
   hist, _ = np.histogramdd(lab, bins=(10, 10, 10),
                      range=((0, 255), (0, 255), (0, 255)))
   return hist.reshape(1, 1000) / 1000.0

def _state_feature(self):
   s = self._preprocess(self._rgb_state)
   s = tf.expand_dims(s, axis=0)
   context = self._backbone(s).numpy().astype('float32')
   hist_rgb = get_rgb_hist(self._rgb_state).astype('float32')
   hist_lab = get_lab_hist(self._lab_state).astype('float32')
   return np.concatenate([context, hist_rgb, hist_lab], 1)
```

Then we define the transition function `_transit` following Table 14.2, and implement reward function `_reward` with Eq. (14.1), to construct same interfaces as OpenAI Gym (Brockman et al. 2016):

```
def step(self, action):
   """One step"""
   self._count += 1
   self._rgb_state = self._transit(action)
   self._lab_state = rgb2lab(self._rgb_state)
   reward = self._reward()
   done = self._count >= self._max_episode_length or action == 0
   return self._state_feature(), reward, done, self._info

def reset(self):
   """Reset"""
   self._count = 0
   raw, retouched = map(Image.open, random.choice(self._src))
   self._rgb_state = np.asarray(raw)
   self._lab_state = rgb2lab(self._rgb_state)
   self._target_lab = rgb2lab(np.asarray(retouched))
   self._current_diff = self._diff(self._lab_state)
   self._info['max_reward'] = self._current_diff
   return self._state_feature()
```

In contrast to the implementation in Sect. 5.10.6, we apply the PPO (Schulman et al. 2017) algorithm in the discrete case. Note that we use LogSoftmax as the activation function in the actor network, which provides better numerical stability when calculating the surrogate objective. For the PPO agent, we first define its initialization and act function:

```
class Agent(object):
   """PPO Agent"""
   def __init__(self, feature, actor, critic, optimizer,
             epsilon=0.1, gamma=0.95, c1=1.0, c2=1e-4,
                gae_lambda=0.95):
      """
      Args:
         feature (tf.keras.Model): backbone of actor and critic
         actor (tf.keras.Model): actor network
         critic (tf.keras.Model): critic network
         optimizer (tf.keras.optimizers.Optimizer): optimizer
            for NNs
         epsilon (float): epsilon in clip
         gamma (float): reward discount
         c1 (float): factor of value loss
         c2 (float): factor of entropy
      """
      self.feature, self.actor, self.critic = feature, actor,
         critic
      self.optimizer = optimizer

      self._epsilon = epsilon
      self.gamma = gamma
      self._c1 = c1
      self._c2 = c2
      self.gae_lambda = gae_lambda

   def act(self, state, greedy=False):
      """
      Args:
         state (numpy.array): 1 * 4048
         greedy (bool): whether select action greedily

      Returns:
         action (int): selected action
         logprob (float): log prob of the selected action
         value (float): value of the current state
      """
      feature = self.feature(state)
      logprob = self.actor(feature)
      if greedy:
         action = tf.argmax(logprob[0]).numpy()
         return action, 0, 0
      else:
         value = self.critic(feature)
         logprob = logprob[0].numpy()
```

```
        action = np.random.choice(range(len(logprob)),
            p=np.exp(logprob))
        return action, logprob[action], value.numpy()[0, 0]
```

During sampling, we record the trajectories with the GAE (Schulman et al. 2015) algorithm

```
def sample(self, env, sample_episodes, greedy=False):
    """ Sample trajectories from given env
    Args:
        env: environment
        sample_episodes (int): how many episodes will be sampled
        greedy (bool): whether select action greedily
    """
    trajectories = [] # s, a, r, logp
    e_reward = 0
    e_reward_max = 0
    for _ in range(sample_episodes):
        s = env.reset()
        values = []
        while True:
            a, logp, v = self.act(s, greedy)
            s_, r, done, info = env.step(a)
            e_reward += r
            values.append(v)
            trajectories.append([s, a, r, logp, v])
            s = s_
            if done:
                e_reward_max += info['max_reward']
                break
        episode_len = len(values)
        gae = np.empty(episode_len)
        reward = trajectories[-1][2]
        gae[-1] = last_gae = reward - values[-1]
        for i in range(1, episode_len):
            reward = trajectories[-i - 1][2]
            delta = reward + self.gamma * values[-i] - values[-i -
                1]
            gae[-i - 1] = last_gae = \
                delta + self.gamma * self.gae_lambda * last_gae
        for i in range(episode_len):
            trajectories[-(episode_len - i)][2] = gae[i] + values[i]
    e_reward /= sample_episodes
    e_reward_max /= sample_episodes
    return trajectories, e_reward, e_reward_max
```

Finally, the optimization part is provided as follows:

```
def _train_func(self, b_s, b_a, b_r, b_logp_old, b_v_old):
    all_params = self.feature.trainable_weights + \
                 self.actor.trainable_weights + \
                 self.critic.trainable_weights
    with tf.GradientTape() as tape:
```

```
        b_feature = self.feature(b_s)
        b_logp, b_v = self.actor(b_feature), self.critic(b_feature)

        entropy = -tf.reduce_mean(
            tf.reduce_sum(b_logp * tf.exp(b_logp), axis=-1))
        b_logp = tf.gather(b_logp, b_a, axis=-1, batch_dims=1)
        adv = b_r - b_v_old
        adv = (adv - tf.reduce_mean(adv)) /
              (tf.math.reduce_std(adv) + 1e-8)

        c_b_v = b_v_old + tf.clip_by_value(b_v - b_v_old,
                                           -self._epsilon, self._epsilon)
        vloss = 0.5 * tf.reduce_max(tf.stack(
            [tf.pow(b_v - b_r, 2), tf.pow(c_b_v - b_r, 2)],
                 axis=1), axis=1)
        vloss = tf.reduce_mean(vloss)

        ratio = tf.exp(b_logp - b_logp_old)
        clipped_ratio = tf.clip_by_value(
            ratio, 1 - self._epsilon, 1 + self._epsilon)
        pgloss = -tf.reduce_mean(tf.reduce_min(tf.stack(
            [clipped_ratio * adv, ratio * adv], axis=1), axis=1))

        total_loss = pgloss + self._c1 * vloss - self._c2 * entropy
    grad = tape.gradient(total_loss, all_params)
    self.optimizer.apply_gradients(zip(grad, all_params))
    return entropy

def optimize(self, trajectories, opt_iter):
    """ Optimize based on given trajectories """
    b_s, b_a, b_r, b_logp_old, b_v_old = zip(*trajectories)
    b_s = np.concatenate(b_s, 0)
    b_a = np.expand_dims(np.array(b_a, np.int64), 1)
    b_r = np.expand_dims(np.array(b_r, np.float32), 1)
    b_logp_old = np.expand_dims(np.array(b_logp_old, np.float32),
        1)
    b_v_old = np.expand_dims(np.array(b_v_old, np.float32), 1)
    b_s, b_a, b_r, b_logp_old, b_v_old = map(
        tf.convert_to_tensor, [b_s, b_a, b_r, b_logp_old, b_v_old])
    for _ in range(opt_iter):
        entropy = self._train_func(b_s, b_a, b_r, b_logp_old,
            b_v_old)

    return entropy.numpy()
```

where the value loss clipping and advantage normalization are followed by Dhariwal et al. (2017). Figure 14.2 shows an example result.

Fig. 14.2 An example result of global enhancement on the MIT-Adobe FiveK dataset. The global brightness is increased while some areas like sky in the upper right corner need local enhancement

References

Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) OpenAI gym. Preprint. arXiv:160601540

Bychkovsky V, Paris S, Chan E, Durand F (2011) Learning photographic global tonal adjustment with a database of input/output image pairs. In: Conference on computer vision and pattern recognition 2011. IEEE, Piscataway, pp 97–104

Clark A (2015) Pillow (PIL fork) documentation. https://github.com/python-pillow/Pillow

Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, Schulman J, Sidor S, Wu Y, Zhokhov P (2017) OpenAI baselines. GitHub, GitHub repository

Furuta R, Inoue N, Yamasaki T (2019) Fully convolutional network with multi-step reinforcement learning for image processing. In: Proceedings of the AAAI conference on artificial intelligence, vol 33, pp 3598–3605

Kupyn O, Budzan V, Mykhailych M, Mishkin D, Matas J (2018) DeblurGAN: Blind motion deblurring using conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 8183–8192

Lee S, Xin J, Westland S (2005) Evaluation of image similarity by histogram intersection. Color research & application: endorsed by inter-society color council, the colour group (Great Britain), Canadian society for color, color science association of Japan, Dutch society for the study of color, the Swedish colour centre foundation, colour society of Australia, centre. Français de la Couleur 30(4):265–274

Nataraj L, Mohammed TM, Manjunath B, Chandrasekaran S, Flenner A, Bappy JH, Roy-Chowdhury AK (2019) Detecting GAN generated fake images using co-occurrence matrices. J Electron Imaging 2019:532-1

Park J, Lee JY, Yoo D, So Kweon I (2018) Distort-and-recover: color enhancement using deep reinforcement learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5928–5936

Poynton C (2012) Digital video and HD: algorithms and interfaces. Elsevier, Amsterdam

Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 779–788

Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, pp 91–99

Ren S, He K, Girshick R, Zhang X, Sun J (2016) Object detection networks on convolutional feature maps. IEEE Trans Pattern Anal Mach Intell 39(7):1476–1481

Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M (2015) ImageNet large scale visual recognition challenge. Int J Comput Vision 115(3):211–252

Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2015) High-dimensional continuous control using generalized advantage estimation. Preprint. arXiv:150602438
Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. Preprint. arXiv:170706347
Ulyanov D, Vedaldi A, Lempitsky V (2018) Deep image prior. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 9446–9454
Van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, Gouillart E, Yu T (2014) Scikit-image: image processing in python. PeerJ 2:e453
Wang R, Zhang Q, Fu CW, Shen X, Zheng WS, Jia J (2019) Underexposed photo enhancement using deep illumination estimation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6849–6857
Yu K, Dong C, Lin L, Change Loy C (2018) Crafting a toolchain for image restoration by deep reinforcement learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2443–2452
Zhang S, Zhen A, Stevenson RL (2019) GAN based image deblurring using dark channel prior. Preprint. arXiv:190300107