14 鲁棒的图像增强

深度生成模型相较于经典的算法，在超分辨率、图像分割等计算机视觉任务中，取得了显著的进展。然而，这种基于学习的方法缺乏鲁棒性和可解释性，限制了它们在现实世界中的应用。本章将讨论一种鲁棒的图像增强方法，它可以通过深度强化学习与许多可解释的技术进行结合。我们将先从一些图像增强的背景知识进行介绍，接着将图像增强的过程看作一个由马尔可夫决策过程（Markov Decision Process，MDP）建模的处理流程。最后，我们将展示如何通过近端策略优化（Proximal Policy Optimization，PPO）算法构建智能体来处理这个 MDP 过程。实验环境由一个真实世界的数据集构建，包含 5000 张照片，其中包括原始图像和专家调整后的版本。项目代码链接见读者服务。

14.1 图像增强

图像增强技术属于图像处理技术。它的主要目标是使处理后的图像更适合各种应用的需要。典型的图像增强技术包括去噪、去模糊和亮度改善。现实世界中的图像总是需要多种图像增强技术。图 14.1 显示了一个包括亮度改善和去噪的图像增强流程。专业的照片编辑软件，如 Adobe Photoshop，提供强大的修图能力，但效率不高，需要用户在照片编辑方面具备专业知识。在诸如推荐系统这样的大规模场景中，图像的主观质量对用户体验至关重要，因此需要一种满足有效性、鲁棒性和效率的自动图像增强方法。其中鲁棒性是最重要的条件，尤其是在用户生成内容的平台上，比如 Facebook 和 Twitter，即使 1% 的较差情况（Bad Case）也会伤害数百万用户的使用体验。

与图像分类或分割有着其独特的真实值（Ground Truth）不同，图像增强的训练数据依赖于人类专家。因此，图像增强并没有大规模的公共图像增强数据集。经典的图像增强方法主要基于

图 14.1 一个图像增强流程的案例。左侧的原始图像存在 JPEG 压缩噪声并且曝光不足（见彩插）

的是伽马校正和直方图均衡化，以及先验的专家知识。这些方法也不需要大量的数据。伽马校正利用了人类感知的非线性，比如，我们感知光和颜色的能力。直方图均衡化实现了允许局部对比度较低的区域获得更高的对比度，以更好地分布在像素直方图上的思想，这在背景和前景为全亮或全暗（如 X 射线图像）时非常有用。这些方法虽然快速、简单，但是缺乏对上下文信息的考虑，限制了它们的应用。

最近，有学者使用基于学习的方法，试图用 CNN 拟合从输入图像到所需像素值的映射，并取得了很大的成功 (Bychkovsky et al., 2011; Kupyn et al., 2018; Ulyanov et al., 2018; Wang et al., 2019)。然而，这种方法也存在问题。首先，很难训练出一个能处理多种增强情况的综合神经网络。此外，像素到像素的映射缺乏鲁棒性，例如，在处理诸如头发和字符等细节信息时，它的表现不是很好 (Nataraj et al., 2019; Zhang et al., 2019)。一些研究者提出，将深度强化学习应用于图像增强，将增强过程描述为一系列策略迭代问题，以解决上述问题。在本章中，我们遵循这些方法，并提出一种新的 MDP 公式来进行图像增强。我们在一个包含 5000 对图像的数据集上用代码示例演示了我们的方法，以提供快速的实际学习过程。

在讨论算法之前，我们先介绍两个 Python 库：Pillow (Clark, 2015) 和 scikit-image (Van der Walt et al., 2014)。它们提供了许多友好的接口来实现图像增强。可以使用如下代码直接从 PyPI 安装它们：

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

下面是 Pillow 的子模块 ImageEnhance 调整对比度的示例代码。

```
from PIL import ImageEnhance

def adjust_contrast(image_rgb, contrast_factor):
    # 调整对比度
    # 参数:
    # image_rgb (PIL.Image):  RGB 图像
```

```
    # contrast_factor (float):  颜色平衡因子范围从 0 到 1.
    # 返回:
    # PIL.Image 对象
    #
    enhancer = ImageEnhance.Contrast(image_rgb)
    return enhancer.enhance(contrast_factor)
```

14.2　用于鲁棒处理的强化学习

在将强化学习应用于图像增强时，首先需要考虑如何构造该领域的马尔可夫决策过程。一个自然出现的想法是将像素处理为状态，将不同的图像增强技术视为强化学习的动作。该构想提供了几种可控的初级增强算法的组合方法，以获得稳健、有效的结果。在本节中，我们将讨论这种基于强化学习的颜色增强方法。为了简单起见，我们只采取全局增强操作。值得一提的是，通过区域候选模块 (Ren et al., 2015) 来适应一般的增强算法也是很自然的想法。

假设训练集包含 N 对 RGB 图像 $\{(l_i, h_i)\}_{i=1}^{N}$，其中 l_i 为低质量原始图像，h_i 是高质量修复图像。为了保持数据分布，初始状态 S_0 应从 $\{l_i\}_{i=1}^{N}$ 中均匀采样。在每步中，智能体会执行一个预定义对动作，如调整对比度，再将它应用于当前状态。需要注意的是，当前状态和选择动作完全决定了状态转移。也就是说，环境没有不确定性。我们在之前工作的基础上 (Furuta et al., 2019; Park et al., 2018) 上继续研究，并使用了 CIELAB 颜色空间作为转移奖励函数。

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

其中 h 是对应的高质量图像 S_0，L 是 RGB 颜色空间到 CIELAB 颜色空间到映射。

另一个重点是定义学习和评估时的终结状态。在游戏的强化学习应用中，终结状态可以由环境决定。而与此不同的是，在图像增强中的智能体需要由自己决定退出时机。文献 (Park et al., 2018) 提出了一个基于 DQN 的智能体，它在所有动作预测的 Q 值都为负数时会退出。然而，Q-Learning 中由函数近似引起的过估计问题可能会导致推理过程的鲁棒性降低。我们通过训练一个明确的策略并增加一个用于退出选择的“无操作”动作来处理这个问题。表 14.1 列出了所有预定义的动作，其中索引为 0 的动作表示“无操作”动作。

从零开始训练一个卷积神经网络需要大量的原始-修复图像对。因此，我们不使用原始图像状态作为观测值的方案，而是考虑使用在 ILSVRC 分类数据集 (Russakovsky et al., 2015) 上预训练的 ResNet50 网络中的最后一层卷积层的激活值作为深层特征输入。这样的深层特征十分重要，它可以提升许多其他视觉识别任务的效果 (Redmon et al., 2016; Ren et al., 2016)。受到前人工作 (Lee et al., 2005; Park et al., 2018) 的启发，我们在构造观测信息时进一步考虑使用直方图信息。具体来说，我们计算了 RGB 颜色空间在 $(0, 255)$, $(0, 255)$, $(0, 255)$ 范围内和 CIELab 颜色空间在 $(0, 100)$,

表 14.1 全局图像增强动作集

索引	简介	索引	简介
0	无操作	7	红、绿色调整 ×0.95
1	对比度 ×0.95	8	红、绿色调整 ×1.05
2	对比度 ×1.05	9	蓝、绿色调整 ×0.95
3	饱和度 ×0.95	10	蓝、绿色调整 ×1.05
4	饱和度 ×1.05	11	红、蓝色调整 ×0.95
5	亮度 ×0.95	12	红、蓝色调整 ×1.05
6	亮度 ×1.05		

$(-60, 60)$, $(-60, 60)$ 范围内的统计信息。这三个特征连成 $2048 + 2000$ 维的观测信息。接着，我们选择 PPO (Schulman et al., 2017) 作为策略优化算法。PPO 是一种 Actor-Critic 算法，它在一系列任务上已经取得了显著的成果。它的网络由 3 部分组成：3 层特征抽取作为主干网络、1 层行动者（Actor）网络和 1 层批判者（Critic）网络。所有层都是全连接的，其中特征抽取器中各层的输出分别为 2048、512 和 128 个单元，并都使用了 ReLU 作为激活函数。

我们在 MIT-Adobe FiveK (Bychkovsky et al., 2011) 数据集上对我们的方法进行了评估。其中包括 5000 张原始图像，而每张原始图像又有 5 个不同专家（A/B/C/D/E）修复后的图像。继之前的工作 (Park et al., 2018; Wang et al., 2019) 之后，我们只使用专家 C 修复的图像，并随机选择 4500 张图像进行训练，剩下的 500 张图像用于测试。原始图像是 DNG 格式的，而修复图像是 TIFF 格式的。我们使用 Adobe Lightroom 将它们都转换为质量为 100、颜色空间为 sRGB 的 JPEG 格式。为了更有效地训练，我们也调整了图像大小，使得每张图像的最大边为 512 像素。具体的超参数在表 14.2 中列出。

表 14.2 用于图像增强的 PPO 超参数

超参数	值	超参数	值
优化器	Adam	每次迭代的优化数	2
学习率	1e-5	最大迭代数	10000
梯度范数裁剪	1.0	熵因子	1e-2
GAE λ	0.95	奖励缩放	0.1
每次迭代的片段数	4	γ	0.95

接下来开始，我们将演示如何实现上述算法，首先需要构建一个环境对象。

```
class Env(object):
    # 训练环境
```

```
    def __init__(self, src, max_episode_length=20, reward_scale=0.1):
        # 参数:
        # src (list[str, str]):  原始图像和处理图像路径的列表，初始状态将从中均匀采样
        # max_episode_length (int):  最大可执行动作数量
        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()
```

通过使用 TensorFlow 的 ResNet API，我们可以通过 `_state_feature` 函数构建观测数据。过程如下所示：

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

def get_lab_hist(lab):
    # 获取 Lab 图像的直方图
    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(rgb):
    # 获取 RGB 图像的直方图
    rgb = rgb.reshape(-1, 3)
    hist, _ = np.histogramdd(rgb, 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)
```

接着，我们构建和 OpenAI Gym (Brockman et al., 2016) 相同的接口。其中，我们按照表 14.1 定义转移函数 `_transit`，并依据公式 (14.1) 构建奖励函数 `_reward`。

```
def step(self, action):
    # 执行单步
    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):
    # 重置环境
    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()
```

这里的 PPO 与 5.10.6 节实现有所不同。我们将 PPO (Schulman et al., 2017) 算法用于离散动作情况。需要注意的是，我们将 LogSoftmax 作为行动者网络的激活函数，这样能在计算替代目标时提供更好的数值稳定性。对 PPO 智能体，我们先定义它的初始化函数和行为函数：

```
class Agent(object):
    # PPO 智能体
    def __init__(self, feature, actor, critic, optimizer,
                 epsilon=0.1, gamma=0.95, c1=1.0, c2=1e-4, gae_lambda=0.95):
        # 参数:
        # feature (tf.keras.Model):  行动者和批判者的基础网络
```

```
        # actor (tf.keras.Model):  行动者网络
        # critic (tf.keras.Model):  批判者网络
        # optimizer (tf.keras.optimizers.Optimizer):  优化器
        # epsilon (float):  裁剪操作中的 epsilon
        # gamma (float):  奖励折扣
        # c1 (float):  价值损失系数
        # c2 (float):  熵系数
        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):
        # 参数:
        # state (numpy.array):  1 * 4048 维的状态
        # greedy (bool):  是否要选取贪心动作
        # Returns:
        # action (int):  所选择的动作
        # logprob (float):  所选动作的概率对数
        # value (float):  当前状态的价值
        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]
```

在采样过程中，我们通过 GAE (Schulman et al., 2015) 算法记录轨迹。

```
def sample(self, env, sample_episodes, greedy=False):
    # 从给定环境中采样一个轨迹
    # 参数:
    # env:  给定的环境
    # sample_episodes (int):  要采样多少片段
    # greedy (bool):  是否选取贪心动作
    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
```

最后，策略优化的部分如下所示，其中价值损失裁剪和优势标准化遵循文献 (Dhariwal et al., 2017) 的描述。

```
def _train_func(self, b_s, b_a, b_r, b_logp_old, b_v_old):
    # 训练函数
    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):
    # 基于给定轨迹数据进行优化
    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()
```

最终经过训练后，智能体学到了图像增强的策略。图 14.2 展示了一个训练效果样例。

原始图像　　本章效果　　专家数据

图 14.2　一个在 MIT-Adobe FiveK 数据集上使用全局增强的效果样例。当右上角的天空等区域需要局部增强时，全局亮度会增加（见彩插）

参考文献

BROCKMAN G, CHEUNG V, PETTERSSON L, et al., 2016. OpenAI gym[J]. arXiv:1606.01540.

BYCHKOVSKY V, PARIS S, CHAN E, et al., 2011. Learning photographic global tonal adjustment with a database of input/output image pairs[C]//CVPR 2011. IEEE: 97-104.

CLARK A, 2015. Pillow (pil fork) documentation[Z].

DHARIWAL P, HESSE C, KLIMOV O, et al., 2017. OpenAI baselines[J]. GitHub, GitHub repository.

FURUTA R, INOUE N, YAMASAKI T, 2019. Fully convolutional network with multi-step reinforcement learning for image processing[C]//Proceedings of the AAAI Conference on Artificial Intelligence: volume 33. 3598-3605.

KUPYN O, BUDZAN V, MYKHAILYCH M, et al., 2018. DeblurGAN: Blind motion deblurring using conditional adversarial networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 8183-8192.

LEE S, XIN J, WESTLAND S, 2005. Evaluation of image similarity by histogram intersection[J]. 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 T M, MANJUNATH B, et al., 2019. Detecting GAN generated fake images using co-occurrence matrices[J]. Journal of Electronic Imaging.

PARK J, LEE J Y, YOO D, et al., 2018. Distort-and-recover: Color enhancement using deep reinforcement learning[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 5928-5936.

REDMON J, DIVVALA S, GIRSHICK R, et al., 2016. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 779-788.

REN S, HE K, GIRSHICK R, et al., 2015. Faster R-CNN: Towards real-time object detection with region proposal networks[C]//Advances in Neural Information Processing Systems. 91-99.

REN S, HE K, GIRSHICK R, et al., 2016. Object detection networks on convolutional feature maps[J]. IEEE transactions on pattern analysis and machine intelligence, 39(7): 1476-1481.

RUSSAKOVSKY O, DENG J, SU H, et al., 2015. Imagenet Large Scale Visual Recognition Challenge[J]. International Journal of Computer Vision (IJCV), 115(3): 211-252.

SCHULMAN J, MORITZ P, LEVINE S, et al., 2015. High-dimensional continuous control using generalized advantage estimation[J]. arXiv preprint arXiv:1506.02438.

SCHULMAN J, WOLSKI F, DHARIWAL P, et al., 2017. Proximal policy optimization algorithms[J]. arXiv:1707.06347.

ULYANOV D, VEDALDI A, LEMPITSKY V, 2018. Deep image prior[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 9446-9454.

VAN DER WALT S, SCHÖNBERGER J L, NUNEZ-IGLESIAS J, et al., 2014. scikit-image: image processing in python[J]. PeerJ, 2: e453.

WANG R, ZHANG Q, FU C W, et al., 2019. Underexposed photo enhancement using deep illumination estimation[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 6849-6857.

ZHANG S, ZHEN A, STEVENSON R L, 2019. GAN based image deblurring using dark channel prior[J]. arXiv preprint arXiv:1903.00107.