5 月 5 日-5 月 15 日工作汇报

Ku Jui

May 2023

Contents

1 文献阅读

1.1 Pre-knowledge

模型中的损失函数是用来衡量模型的预测值和真实值 (Ground Truth) 之间的差异程度的函数，它可以反映模型的优化方向和性能指标[1]。它是一种衡量模型预测结果与真实结果之间差异的方法，用于指导模型参数的更新。在训练过程中，通过不断最小化损失函数

[1]https://zhuanlan.zhihu.com/p/375968083

来优化模型参数，使模型能够更好地拟合数据[2]。因此，需要使用合适的损失函数，当模型在数据集上进行训练时，该函数可以适当地惩罚模型 [3]。

不同的损失函数适用于不同的任务和数据分布，例如回归问题常用的有均方误差损失函数 (MSE，也叫做 $\mathcal{L}_2$ 损失函数) 和 $\mathcal{L}_1$ 损失函数，分类问题常用的有交叉熵损失函数 (Cross Entropy Loss) 等[4]。损失函数的选择会影响模型的收敛速度和精度，因此需要根据具体情况选择合适的损失函数[5]。

目前常用的损失函数是从相关视觉任务中借用的，但这些损失函数可能并不完全适用于低照度图像增强 LLIE。因此，需要设计更适合 LLIE 的损失函数，以更好地驱动深度网络的优化。这可以通过研究人类对图像质量的视觉感知来实现，使用深度神经网络来近似人类视觉感知，并将这些理论应用于损失函数的设计。损失函数可以分为两个大类：回归问题和分类问题。

1.1.1 loss function for CV

$\mathcal{L}_1$-loss

平均绝对误差 (MAE) 损失，也称 $\mathcal{L}_1$ 范数损失，计算实际值和预测值之间绝对差之和的平均值。

$$\mathcal{L}_1 = \frac{1}{N}\sum_{i=1}^{N} \|\hat{y}_i - y_i\|_1 \tag{1}$$

适用于回归问题，MAE loss 对异常值更具鲁棒性，尤其是当目标变量的分布有离群值时 (小值或大值与平均值相差很大)。

函数：*torch.nn.L1Loss*

$\mathcal{L}_2$-loss

均方误差 (MSE) 损失，也称为 $\mathcal{L}_2$ 范数损失，计算实际值和预测值之间平方差的平均值[6]。

$$\mathcal{L}_1 = \frac{1}{N}\sum_{i=1}^{N} \|\hat{y}_i - y_i\|_2^2 \tag{2}$$

[2]https://zhuanlan.zhihu.com/p/436809988

[3]https://zhuanlan.zhihu.com/p/473113939

[4]https://blog.csdn.net/weixin_57643648/article/details/122704657

[5]https://www.zhihu.com/tardis/zm/art/136047113?source_id=1005

[6]https://blog.csdn.net/yanyuxiangtoday/article/details/119788949

平方意味着较大的误差比较小的误差会产生更大的惩罚，所以 $\mathcal{L}_2 - loss$ 的收敛速度要比 $\mathcal{L}_1 - loss$ 要快得多。但是，$\mathcal{L}_2 - loss$ 对异常点更敏感，鲁棒性差于 $\mathcal{L}_1 - loss$。

$\mathcal{L}_1 - loss$ 损失函数相比于 L2 损失函数的鲁棒性更好。因为以 $\mathcal{L}_2 - loss$ 范数将误差平方化 (如果误差大于 1，则误差会放大很多)，模型的误差会比以 $\mathcal{L}_1 - loss$ 范数大的多，因此模型会对这种类型的样本更加敏感，这就需要调整模型来最小化误差。但是很大可能这种类型的样本是一个异常值，模型就需要调整以适应这种异常值，那么就导致训练模型的方向偏离目标了[7]。

对于大多数回归问题，一般是使用 $\mathcal{L}_2 - loss$ 而不是 $\mathcal{L}_1 - loss$。

函数: *torch.nn.MSELoss*

loss 正则化

正则化的基本思想是通过在损失函数中加入额外信息，以便防止过拟合和提高模型泛化性能。无论哪一种正则化方式，基本思想都是希望通过限制权重的大小，使得模型不能任意拟合训练数据中的随机噪声，正则化实际是在损失函数中加入刻画模型复杂程度的指标[8]。

对应的 L1 正则损失函数:

$$\mathcal{L}_{norm1} = \mathcal{L}_1(\hat{y}, y) + \lambda \sum_{\omega} \|\omega\|_1 \tag{3}$$

对应的 L2 正则损失函数：

$$\mathcal{L}_{norm2} = \mathcal{L}_2(\hat{y}, y) + \lambda \sum_{\omega} \|\omega\|_2^2 \tag{4}$$

假设 $\mathcal{L}_1(\hat{y}, y)$ 和 $\mathcal{L}_2(\hat{y}, y)$ 是未加正则项的损失，λ 是一个超参，用于控制正则化项的大小，惩罚项 ω 用于惩罚大的权重，隐式地减少自由参数的数量。

正则化是如何降低过拟合现象的？

正则化之所以能够降低过拟合的原因在于，正则化是结构风险最小化的一种策略实现。给损失函数加上正则化项，能使得新得到的优化目标函数 $h = f + normal$ ，需要在 f 和 $normal$ 中做一个权衡 (trade-off)，如果还像原来只优化 f 的情况下，那可能得到一组解比较复杂，使得正则项 $normal$ 比较大，那么 h 就不是最优的，因此可以看出加正则项能让解更加简单，符合奥卡姆剃刀理论，同时也比较符合在偏差和方差 (方差表示模型的复杂度) 分析中，通过降低模型复杂度，得到更小的泛化误差，降低过拟合程度[9]。

[7] https://zhuanlan.zhihu.com/p/137073968

[8] https://blog.csdn.net/weixin_41960890/article/details/104891561

[9] https://zhuanlan.zhihu.com/p/35356992

PyTorch 实现：L2 正则项是通过 optimizer 优化器的参数 weight_decay(float, optional) 添加的，用于设置权值衰减率，即正则化中的超参 λ，默认值为 0。

```
optimizer = torch.optim.SGD(model.parameters(),lr=0.01,weight_decay=0.01)
```

Smooth $\mathcal{L}_1$ loss

Smooth $\mathcal{L}_1$ 损失函数是由 Girshick R 在 Fast R-CNN 中提出的，主要用在目标检测中防止梯度爆炸。它是一个分段函数，在 $[-1,1]$ 之间是 $\mathcal{L}_2$ 损失，其他区间就是 $\mathcal{L}_1$ 损失。这样即解决了 $\mathcal{L}_1$ 损失在 0 处不可导的问题，也解决了 $\mathcal{L}_2$ 损失在异常点处梯度爆炸的问题[10]。

$$\text{smooth } \mathcal{L}_1 \text{ loss} = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}\dfrac{\|\hat{y}_i - y_i\|_2^2}{2\beta}, \|\hat{y}_i - y_i\| < \beta, \\ \|\hat{y}_i - y_i\|_1 - \dfrac{1}{2}\beta, \|\hat{y}_i - y_i\| \geq \beta.\end{cases} \tag{5}$$

一般取 $\beta = 1$。smooth $\mathcal{L}_1$ 和 $\mathcal{L}_1$-loss 函数的区别在于，smooth $\mathcal{L}_1$ 在 0 点附近使用 $\mathcal{L}_2$ 使得它更加平滑, 它同时拥有 $\mathcal{L}_2$-loss 和 $\mathcal{L}_1$-loss 的部分优点。

函数：*torch.nn.SmoothL1Loss*

Huber loss

$\mathcal{L}_2$-loss 但容易受离群点的影响，$\mathcal{L}_1$-loss 对离群点更加健壮但是收敛慢，Huber Loss 则是一种将 MSE 与 MAE 结合起来，取两者优点的损失函数，也被称作 Smooth Mean Absolute Error Loss。其原理很简单，就是在误差接近 0 时使用 $\mathcal{L}_2$-loss，误差较大时使用 $\mathcal{L}_1$-loss

$$J_{Huber}(\delta) = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}\dfrac{1}{2}\|\hat{y}_i - y_i\|_2^2, \|\hat{y}_i - y_i\| < \delta, \\ \delta\left(\|\hat{y}_i - y_i\|_1 - \dfrac{1}{2}\delta\right), \|\hat{y}_i - y_i\| \geq \delta.\end{cases} \tag{6}$$

残差比较小时，Huber Loss 是二次函数；残差比较大时，Huber Loss 是线性函数 (残差，即观测值和预测值之间的差值)。与 $\mathcal{L}_2$-loss 相比，Huber 损失对数据中的异常值不那么敏感。使函数二次化的小误差值是多少取决于“超参数”δ，它可以调整。当 $\delta = 1$ 时，退化成 smooth $\mathcal{L}_1$ Loss。

函数：*torch.nn.HuberLoss*

[10]https://zhuanlan.zhihu.com/p/261059231

log-MSE

$$J_{log-MSE} = 10 \log_{10} \left(\frac{1}{N} \sum_{i=1}^{N} \|\hat{y}_i - y_i\|_2^2 \right) \tag{7}$$

Perceptual loss

感知损失 (Perceptual Loss) 是一种用于比较两个看起来相似的图像的损失函数，这一损失函数由 Johnson et al. [1] 提出。它用于比较图像之间的高层次差异，如内容和风格差异[11]。它已被广泛用作图像合成任务 (包括图像超分辨率和风格转换) 中的有效损失项。

感知损失函数用于比较两个看起来相似的不同图像，例如同一张照片，但偏移了一个像素。该函数用于比较图像之间的高级差异，例如内容和样式差异。感知损失函数与每像素损失函数非常相似，因为两者都用于训练前馈神经网络以进行图像转换任务。感知损失函数是一个更常用的组件，因为它通常提供有关风格迁移的更准确的结果。

简而言之，感知损失函数的工作原理是将所有像素之间的所有平方误差相加并取平均值。这与每像素损失函数形成对比，后者对像素之间的所有绝对误差求和 [1]。

作者认为感知损失函数不仅在生成高质量图像方面更准确，而且在优化后也快了三倍。神经网络模型在图像上进行训练，其中感知损失函数基于从已训练网络中提取的高级特征进行优化。

$$\ell_{feat}^{\phi,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2 \tag{8}$$

其中 $\hat{y}$ 为输出图像，y 为目标图像，ϕ 为损失网络。$\phi_j(x)$ 为处理图像 x 时损失网络 ϕ 的第 j 层的激活情况，如果 j 是一个卷积层，那么 $\phi_j(x)$ 将是形状 $C_j \times H_j \times W_j$ 的特征映射，特征重建损失是特征表示之间的欧式距离，如 eq 8。

Fig.1表示经过训练以将输入图像转换为输出图像的神经网络。用于图像分类的预训练损失网络有助于通知损失函数。预先训练的网络有助于定义测量图像之间内容和风格的感知差异所需的感知损失函数。

对于图像数据来说，网络在提取特征的过程中，较浅层通常提取边缘、颜色、亮度等低频信息，而网络较深层则提取一些细节纹理等高频信息，再深一点的网络层则提取一些具有辨别性的关键特征，也就是说，网络层越深提取的特征越抽象越高级。

感知损失就是通过一个固定的网络 (通常使用预训练的 VGG16 或者 VGG19)，分别以真实图像 (Ground Truth)、网络生成结果 (Prediciton) 作为其输入，得到对应的输出特征：feature_gt、feature_pre，然后使用 feature_gt 与 feature_pre 构造损失 (通常为 $\mathcal{L}_2$-loss)，

[11] https://deepai.org/machine-learning-glossary-and-terms/perceptual-loss-function

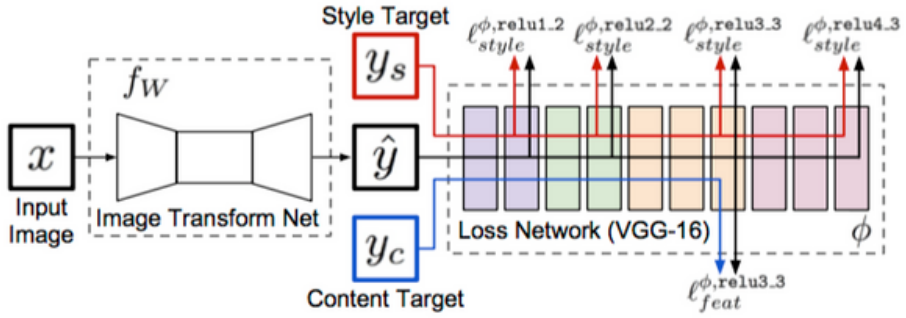

Figure 1: System overview. We train an image transformation network to transform input images into output images. We use a loss network pretrained for image classification to define perceptual loss functions that measure perceptual differences in content and style between images. The loss network remains fixed during the training process.

逼近真实图像与网络生成结果之间的深层信息，也就是感知信息，相比普通的 $\mathcal{L}_2$-loss 而言，可以增强输出特征的细节信息[12]。

SSIM loss

SSIM 损失函数是一种用于衡量两幅图像之间差距的损失函数。它考虑了亮度、对比度和结构指标，这就考虑了人类视觉感知，一般而言，SSIM 得到的结果会比 $\mathcal{L}_1$-loss，$\mathcal{L}_2$-loss 的结果更有细节[13]。

每个像素 p 的SSIM被定义为

$$\begin{aligned}\text{SSIM}(p) &= \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \cdot \frac{2\sigma_{xy} + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \\ &= l(p) \cdot cs(p)\end{aligned} \tag{9}$$

其中省略了均值和标准偏差对像素 p 的依赖性，均值和标准差是用标准偏差为 σ_G, G_{σ_G}

$$\begin{aligned}&\varepsilon(p) = 1 - \text{SSIM}(p) : \\ &\mathcal{L}^{\text{SSIM}}(P) = \frac{1}{N}\sum_{p\in P} 1 - \text{SSIM}(p).\end{aligned} \tag{10}$$

eq. 9表明 SSIM(p) 需要关注像素 p 的邻域，这个领域的大小取决于 G_{σ_G}，网络的卷积性质允许我们将 SSIM 损失写为

$$\mathcal{L}^{\text{SSIM}}(P) = 1 - \text{SSIM}(\tilde{p}). \tag{11}$$

其中 $\tilde{p}$ 是 P 的中心像素。

[12]https://blog.csdn.net/qq_43665602/article/details/127077484

[13]https://blog.csdn.net/u013289254/article/details/99694412

MS-SSIM loss

多尺度结构相似性 (MS-SSIM) 损失函数是基于多层 (图片按照一定规则，由大到小缩放) 的 SSIM 损失函数，相当于考虑了分辨率[14]。它是一种更为复杂的 SSIM 损失函数，可以更好地衡量图像之间的相似性。

$$\text{MS-SSIM}(p) = l_M^{\alpha}(p) \cdot \prod_{j=1}^{M} cs_j^{\beta_j}(p) \tag{12}$$

其中 M, j 描述的是比例，设 $\alpha = \beta_j = 1$，对于 $j = 1, \cdots, M$ 类似 eq. 11，利用中心像素 $\tilde{p}$ 处计算的损失来近似贴片 P 的损失：

$$\mathcal{L}^{\text{MS-SSIM}}(P) = 1 - \text{MS-SSIM}(\tilde{p}) \tag{13}$$

Cross-entropy loss function

交叉熵损失函数是一种常用的分类问题损失函数。在二分类问题中，它的定义如下：

$$\mathcal{L}(\hat{y}, y) = -\big(y \log_{\hat{y}} + (1-y)\log(1-\hat{y})\big) \tag{14}$$

其中，$\hat{y}$ 表示模型预测的概率值 (即分类器输出)，y 表示样本真实的类别标签。对于正例样本 ($y = 1$)，交叉熵损失函数的值等于 $log\hat{y}$；对于反例样本 ($y = 0$)，交叉熵损失函数的值等于 $\log(1-\hat{y})$。因此，交叉熵损失函数的目标是最小化模型预测与实际标签之间的差距，从而让模型能够更准确地进行分类。

交叉熵损失函数可以推广到多分类问题中，此时它的表达式略有不同。在多分类问题中，交叉熵损失函数可以写成以下形式：

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{K} y_i \log \hat{y}_i \tag{15}$$

其中，K 表示类别的数量，y_i 表示第 i 个类别的真实标签，$\hat{y}_i$ 表示模型对于第 i 个类别的预测概率值。交叉熵损失函数的目标仍然是最小化预测与实际标签之间的差距，从而让模型能够更准确地进行分类。

1.1.2 细节

Table 1给出了最近几年主流的基于深度学习的 LLIE 方案，并从不同角度对其进行了划分。

[14]https://blog.csdn.net/u013289254/article/details/99694412

	Method	Learning	Network Structure	Loss Function	Training Data	Testing Data	Evaluation Metric	Format	Platform	Retinex
2017	LLNet	SL	SSDA	SRR loss	simulated by Gamma Correction & Gaussian Noise	simulated self-selected	PSNR SSIM	RGB	Theano	
2018	LightenNet	SL	four layers	L_2 loss	simulated by random illumination values	simulated self-selected	PSNR MAE SSIM User Study	RGB	Caffe MATLAB	✓
	Retinex-Net	SL	multi-scale network	L_1 loss invariable reflectance loss smoothness loss	LOL simulated by adjusting histogram	self-selected	-	RGB	TensorFlow	✓
	MBLLEN	SL	multi-branch fusion	SSIM loss perceptual loss region loss	simulated by Gamma Correction & Poisson Noise	simulated self-selected	PSNR SSIM AB VIF LOE TOMI	RGB	TensorFlow	
	SCIE	SL	frequency decomposition	L_2 loss L_1 loss SSIM loss	SCIE	SCIE	PSNR FSIM Runtime FLOPs	RGB	Caffe MATLAB	
	Chen et al.	SL	U-Net	L_1 loss	SID	SID	PSNR SSIM	Raw	TensorFlow	
	Deepexposure	RL	policy network GAN	deterministic policy gradient adversarial loss	MIT-Adobe FiveK	MIT-Adobe FiveK	PSNR SSIM	Raw	TensorFlow	
2019	Chen et al.	SL	siamese network	L_1 loss self-consistency loss	DRV	DRV	PSNR SSIM MAE	Raw	TensorFlow	
	Jiang and Zheng	SL	3D U-Net	L_1 loss	SMOID	SMOID	PSNR SSIM MSE	Raw	TensorFlow	
	DeepUPE	SL	illumination map	L_1 loss smoothness loss color loss	retouched image pairs	MIT-Adobe FiveK	PSNR SSIM User Study	RGB	TensorFlow	✓
	KinD	SL	three subnetworks U-Net	reflectance similarity loss illumination smoothness loss mutual consistency loss L_1 loss L_2 loss SSIM loss texture similarity loss illumination adjustment loss	LOL	LOL LIME NPE MEF	PSNR SSIM LOE NIQE	RGB	TensorFlow	✓
	Wang et al.	SL	two subnetworks pointwise Conv	L_1 loss	simulated by camera imaging model	IP100 FNF38 MPI LOL NPE	PSNR SSIM NIQE	RGB	Caffe	✓
	Ren et al.	SL	U-Net like network RNN dilated Conv	L_2 loss perceptual loss adversarial loss	MIT-Adobe FiveK with Gamma correction & Gaussion noise	simulated self-selected DPED	PSNR SSIM Runtime	RGB	Caffe	
	EnlightenGAN	UL	U-Net like network	adversarial loss self feature preserving loss	unpaired real images	NPE LIME MEF DICM VV BBD-100K ExDARK	User Study NIQE Classification	RGB	PyTorch	
	ExCNet.	ZSL	fully connected layers	energy minimization loss	real images	$IE_{ps}D$	User Study CDIQA LOD	RGB	PyTorch	
2020	Zero-DCE	ZSL	U-Net like network	spatial consistency loss exposure control loss color constancy loss illumination smoothness loss	SICE	SICE NPE LIME MEF DICM VV DARK FACE	User Study PI PNSR SSIM MAE Runtime Face detection	RGB	PyTorch	
	DRBN	SSL	recursive network	SSIM loss perceptual loss adversarial loss	LOL images selected by MOS	LOL	PSNR SSIM SSIM-GC	RGB	PyTorch	
	Lv et al.	SL	U-Net like network	Huber loss SSIM loss perceptual loss illumination smoothness loss	simulated by a retouching module	LOL SICE DeepUPE	User Study PSNR SSIM VIF LOE NIQE #P Runtime Face detection	RGB	TensorFlow	✓
	Fan et al.	SL	four subnetworks U-Net like network feature modulation	mutual smoothness loss reconstruction loss illumination smoothness loss cross entropy loss consistency loss SSIM loss gradient loss ratio learning loss	simulated by illumination adjustment, slight color distortion, and noise simulation	simulated self-selected	PSNR SSIM NIQE	RGB	-	✓
	Xu et al.	SL	frequency decomposition U-Net like network	L_2 loss perceptual loss	SID in RGB	SID in RGB self-selected	PSNR SSIM	RGB	PyTorch	
	EEMEFN	SL	U-Net like network edge detection network	L_1 loss weighted cross-entropy loss	SID	SID	PSNR SSIM	Raw	TensorFlow PaddlePaddle	
	DLN	SL	residual learning interactive factor back projection network	SSIM loss total variation loss	simulated by illumination adjustment, slight color distortion, and noise simulation	simulated LOL	User Study PSNR SSIM NIQE	RGB	PyTorch	
	LPNet	SL	pyramid network	L_1 loss perceptual loss luminance loss	LOL SID in RGB MIT-Adobe FiveK	LOL SID in RGB MIT-Adobe FiveK MEF NPE DICM VV	PSNR SSIM NIQE #P FLOPs Runtime	RGB	PyTorch	
	SIDGAN	SL	U-Net	CycleGAN loss	SIDGAN	SIDGAN	PSNR SSIM TPSNR TSSIM ATWE	Raw	TensorFlow	
	RRDNet	ZSL	three subnetworks	retinex reconstruction loss texture enhancement loss noise estimation loss	-	NPE LIME MEF DICM	NIQE CPCQI	RGB	PyTorch	✓
	TBEFN	SL	three stages U-Net like network	SSIM loss perceptual loss smoothness loss	SCIE LOL	SCIE LOL DICM MEF NPE VV	PSNR SSIM NIQE Runtime #P FLOPs	RGB	TensorFlow	✓
	DSLR	SL	Laplacian pyramid U-Net like network	L_2 loss Laplacian loss color loss	MIT-Adobe FiveK	MIT-Adobe FiveK self-selected	PSNR SSIM NIQMC NIQE BTMQI CaHDC	RGB	PyTorch	

Table 1: Summary of essential characteristics of representative deep learning-based methods, including learning strategies, network structures, loss functions, training datasets, testing datasets, evaluation metrics, data formats of input, and whether the models are Retinex-based or not. "simulated" means the testing data are simulated by the same approach as the synthetic training data. "self-selected" stands for the real-world images selected by the authors. "#P" represents the number of trainable parameters. "-" means this item is not available or not indicated in the paper.

2 个人工作进展

2.1 华为杯赛题想法

2.1.1 AI 交互预测决策规划

Challenge

基于数据驱动的方法来实现安全可靠的预测决策规划算法，往往会涉及以下挑战：

1. AI 模型建模拓扑和交通参与者：为了有效地建模城区场景中的拓扑结构和交通参与者，需要设计适合的模型来表示道路网络、车辆、非机动车和行人等元素，并捕捉它们之间的关系。

2. 考虑交互和自车类人安全性：在设计 AI 模型时，需要充分考虑自车与其他交通参与者的交互作用，以及确保自车行为符合人类驾驶员的安全性标准。模型的输出应该满足预测、决策和规划的一致性和时序稳定性，而不仅仅是对自车进行无交互的开环预测。

3. 有效性和安全性的证明：传统的开环平均指标难以有效证明 AI 模型的有效性。在闭环仿真器中测试时，由于仿真环境与真实环境存在行为差异，模型的有效性和安全性需要寻找高效的证明方法。

4. 捕捉场景差异特征：数据驱动的方法需要 AI 模型能够捕捉到不同场景的差异特征，学习到场景特定的行为模式，而不仅仅是学习平均行为。这样可以使模型更加准确地预测和规划针对不同场景的行为。

5. 有效应用和看护算法：经过数据驱动后的 AI 模型需要与有效的应用程序和监控算法结合，以实现更安全智能的自动驾驶。这意味着需要设计相应的决策算法和监控机制，确保自动驾驶系统的安全性和可靠性。

以上是基于数据驱动的安全可靠预测决策规划算法所面临的挑战。解决这些挑战需要综合考虑模型设计、仿真测试、数据采集和处理、决策算法等多个方面，并进行深入的研究和创新。

2.1.2 适用多场景的通用化时序预测算法

Challenge

将人工智能技术应用于时序预测任务时，面临以下主要挑战：

数据的多样性和平衡性：云计算中的数据具有丰富的来源、庞大的规模和迥异的特性。为了设计通用的时序预测算法，需要提出并论证数据多样性和平衡性的指标，并根据这些指标进行数据收集，以避免算法过度拟合于特定数据集。

应对序列多样性：针对具有序列多样性的数据集，设计通用化的时序预测算法是一个挑战。这些算法需要处理不同长度、趋势性、周期性和采样粒度的时间序列，并在所有序列上都能给出较好的预测结果。

为了应对这些挑战，可以基于对数据特征的分析选择合适的算法类型，也可以利用神经网络等机器学习模型的强大表达能力，通过单个模型实现通用化预测的目的。需要进行深入研究和创新，不断改进和优化算法，以提高时序预测的准确性和适应性。

2.1.3 通过穿戴设备监测高糖发生次数

Challenge

2.2 梳理损失函数

梳理了一些常见的损失函数，如 $\mathcal{L}_1$-loss, $\mathcal{L}_2$-loss, Smooth $\mathcal{L}_1$ loss, Huber loss, Perceptual loss 等一些在上周的 paper 梳理过程中，梳理出来的一些损失函数的原理及相关功能。不难发现，这些损失函数在原理上存在很大的共通性，也有详细的论文说明这些函数的原理和工作机制。但进一步的梳理仍然是必要的，许多损失函数是从其他图像处理任务中借用过来的，而在 LLIE 任务中，更好的损失函数或许有待发现。还有一些不同网络架构下的损失函数待梳理，比如 adversarial loss、region loss、reflectance loss、consistency loss、color loss、Laplacian loss 等。

2.3 论文代码复现

复现 **KinD** 代码

KinD[15]采用的是类 U-Net 网络，且对比使用了多种损失函数，采用的数据集相对较少，且环境配置相对简单，具体见 Tab. 1。同时，KinD [14] 目前引用数量为 543，采用的网络架构较为经典，即 Retinex-based 的思想。

Requirement

```
Python
Tensorflow >= 1.10.0
numpy, PIL
```

[15]https://github.com/zhangyhuaee/KinD

Train

KinD Network 分为三部分 (Fig. 2)：(1) 图像分解网络：Layer Decomposition Net;(2) 反射分量纠正网络：Reflectance Restoration Net;(3) 光照分量纠正网络：Illumination Adjustment Net。

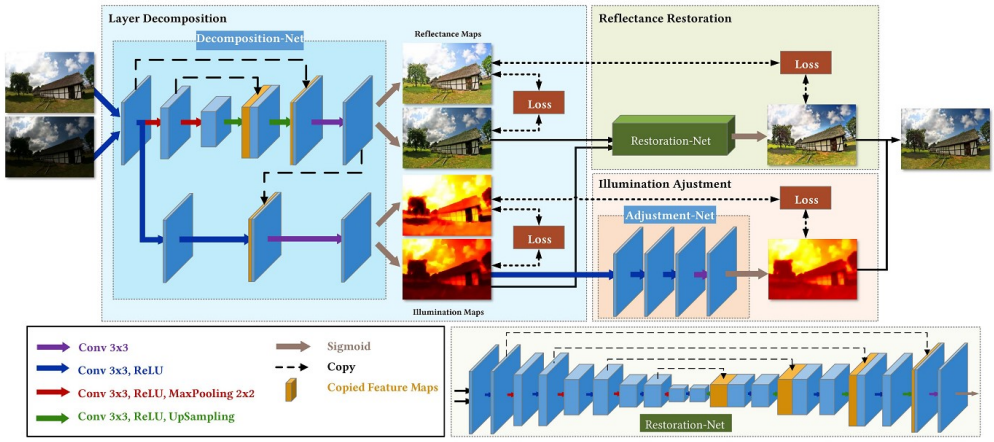

Figure 2: The architecture of our KinD network. Two branches correspond to the reflectance and illumination, respectively. From the perspective of functionality, it also can be divided into three modules, including layer decomposition, reflectance restoration, and illumination adjustment.

以暗光/正常光照图像 (I_{low}/I_{high}) 对作为训练样本,Layer Decomposition Net 对 (I_{low}/I_{high}) 依次进行分解，得到光照分量 L_{low}、L_{high} 和反射分量 R_{low}、R_{high}。再通过 Reflectance Restoration Net 和 Illumination Adjustment Net 得到 $\tilde{R}_{low}$ 和 $\tilde{L}_{low}$ 。

Layer Decomposition Net

Layer Decomposition Net 有两个分支，一个分支用于预测反射分量，另一个分支用于预测光照分量，反射分量分支以五层 Unet 网络为主要网络结构，后接一个卷积层和 Sigmoid 层。光照分量分支由三个卷积层构成，其中还利用了反射分量分支中的特征图，具体细节可参考论文。

Layer Decomposition Net 对 (I_l/I_h) 依次进行分解，得到光照分量 L_l、L_h 和反射分量 R_l、R_h。

从 Fig. 2可以看出，KinD 的损失函数主要由三部分损失构成，它们分别是层分解部分损失、反射重建部分损失以及亮度调整部分损失。

层分解 (Layer Decomposition Net) 部分损失定义如下：

$$\mathcal{L}^{LD} = \mathcal{L}^{LD}_{rec} + 0.01\mathcal{L}^{LD}_{rs} + 0.15\mathcal{L}^{LD}_{is} + 0.2\mathcal{L}^{LD}_{mc} \tag{16}$$

其中，$\mathcal{L}_{rs}^{LD} = \|R_l - R_h\| 1$ 表示反射相似性损失 (Reflectance Similarity)，即短曝光与长曝光图形的反射图应该是相同的；

$$\mathcal{L}_{is}^{LD} = \left\| \frac{\nabla L_l}{\max(|\nabla I_l|, \varepsilon)} \right\|_1 + \left\| \frac{\nabla L_h}{\max(|\nabla I_h|, \varepsilon)} \right\|_1$$

表示亮度平滑损失约束 (Illumination Smoothness)，它度量了亮度图与输入图像之间的相对结构，边缘区域惩罚较小，平滑区域惩罚较大；$\mathcal{L}_{mc}^{LD} = \|M \circ \exp(-c \cdot M)\|_1, M = \nabla \mathcal{L}_l + \nabla \mathcal{L}_h$ 表示相互一致性约束 (Mutual Consistency)，它意味着强边缘得以保留，弱边缘被抑制；$\mathcal{L}_{rec}^{LD} = \|I_l - R_t \circ \mathcal{L}_l\|_1 + \|I_h - R_h \circ \mathcal{L}_h\|_1$ 表示重建损失 (Reconstruction Error)。反射部分损失定义如下：

$$\mathcal{L}^{RR} := \left\| \hat{R} - R_h \right\|_2^2 - SSIM(\hat{R}, R_h) + \left\| \nabla \hat{R} - \nabla R_h \right\|_2^2 \tag{17}$$

亮度调整部分损失定义如下：

$$\mathcal{L}^{IA} := \left\| \hat{L} - L_t \right\|_2^2 + \left\| \left| \nabla \hat{L} \right| - \nabla L_t \right\|_2^2 \tag{18}$$

```
# 设置一些训练所需的参数
batch_size = 10
patch_size = 48

# 创建一个TensorFlow会话
sess = tf.Session()

# 定义输入的占位符，这些占位符用于接收低分辨率和高分辨率的输入图像。
input_low = tf.placeholder(tf.float32, [None, None, None, 3], name='input_low')
input_high = tf.placeholder(tf.float32, [None, None, None, 3], name='input_high')

# 使用定义的模型构建计算图,DecomNet_simple是一个分解网络模型，它接受输入图像并输出反射和亮度组件
[R_low, I_low] = DecomNet_simple(input_low)
[R_high, I_high] = DecomNet_simple(input_high)

```

```
    # 将反射和亮度组件拼接起来形成输出图像
    # 这一步操作将反射和亮度组件进行通道拼接，以生成输出图像。
    I_low_3 = tf.concat([I_low, I_low, I_low], axis=3)
    I_high_3 = tf.concat([I_high, I_high, I_high], axis=3)
    output_R_low = R_low
    output_R_high = R_high
    output_I_low = I_low_3
    output_I_high = I_high_3

    # 定义损失函数

    def mutual_i_loss(input_I_low, input_I_high):
    # 互信息损失函数的定义
     ...

    def mutual_i_input_loss(input_I_low, input_im):
    # 输入互信息损失函数的定义
     ...

    recon_loss_low = tf.reduce_mean(tf.abs(R_low * I_low_3 - input_low))
    recon_loss_high = tf.reduce_mean(tf.abs(R_high * I_high_3 - input_high
        ))
    equal_R_loss = tf.reduce_mean(tf.abs(R_low - R_high))
    i_mutual_loss = mutual_i_loss(I_low, I_high)
    i_input_mutual_loss_high = mutual_i_input_loss(I_high, input_high)
    i_input_mutual_loss_low = mutual_i_input_loss(I_low, input_low)

    loss_Decom = 1*recon_loss_high + 1*recon_loss_low + 0.01*
        equal_R_loss + 0.2*i_mutual_loss + 0.15*i_input_mutual_loss_high
        + 0.15*i_input_mutual_loss_low

    # recon_loss_low 和 recon_loss_high 是重构损失，用于衡量输出图像与输
        入图像之间的差异。
    # equal_R_loss 是反射一致性损失，用于衡量两个不同尺度下的反射分量
```

```
    之间的一致性。
# i_mutual_loss 是亮度互信息损失，用于鼓励亮度分量之间的一致性。
# i_input_mutual_loss_high 和 i_input_mutual_loss_low 是输入亮度与反射之间的互信息损失，用于鼓励输入图像与反射分量之间的一致性。
```

最后定义优化器和训练操作

```
# 我们使用 Adam 优化器来最小化损失函数，其中只更新 DecomNet 模型的可训练变量。
lr = tf.placeholder(tf.float32, name='learning_rate')
optimizer = tf.train.AdamOptimizer(learning_rate=lr, name='AdamOptimizer')
var_Decom = [var for var in tf.trainable_variables() if 'DecomNet' in var.name]
train_op_Decom = optimizer.minimize(loss_Decom, var_list=var_Decom)
sess.run(tf.global_variables_initializer())
saver_Decom = tf.train.Saver(var_list=var_Decom)

# 加载数据集
# 这里使用 glob 函数获取训练集的低分辨率和高分辨率图像的文件名，并进行排序。
train_low_data = []
train_high_data = []
train_low_data_names = glob('./LOLdataset/our485/low/*.png')
train_low_data_names.sort()
train_high_data_names = glob('./LOLdataset/our485/high/*.png')
train_high_data_names.sort()

# 定义了一些辅助变量和文件夹路径
# epoch 表示训练的总轮数，learning_rate 表示学习率，sample_dir 是保存样本图像的文件夹路径，checkpoint_dir 是保存模型检查点的文件夹路径。
epoch = 2000
learning_rate = 0.0001
sample_dir = './Decom_net_train/'
```

```
23      checkpoint_dir = './checkpoint/decom_net_train/'
```

最后开始训练循环

```
1       for epoch in range(start_epoch, epoch):
2       for batch_id in range(start_step, numBatch):
3       # 获取一个批次的训练数据
4       ...
5
6       # 执行训练操作，计算损失
7       _, loss = sess.run([train_op, train_loss], feed_dict={input_low:
            batch_input_low, input_high: batch_input_high, lr: learning_rate})
8
9       # 打印训练进度和损失
10      print("%s Epoch: [%2d] [%4d/%4d] time: %4.4f, loss: %.6f" % (
            train_phase, epoch + 1, batch_id + 1, numBatch, time.time() -
            start_time, loss))
11
12      # 每隔100个批次保存模型和样本图像
13      if (epoch + 1) % 100 == 0:
14       print('Saving sample images...')
15       sample_results = sess.run([output_R_low, output_R_high,
             output_I_low, output_I_high], feed_dict={input_low:
             sample_input_low, input_high: sample_input_high})
16       save_images(sample_results, [batch_size, 1], sample_dir + 'train_%d.png
             ' % (epoch + 1))
17
18       print('Saving model...')
19       saver.save(sess, checkpoint_dir + 'model.ckpt', global_step=epoch + 1)
20
21      # 每隔500个批次降低学习率
22      if (epoch + 1) % 500 == 0:
23       learning_rate /= 10
```

在每个 epoch 的训练过程中，当遍历完一个批次后，计算并打印损失值。

当达到一定条件时，例如每隔 100 个批次，保存模型和样本图像。首先，我用当前模

型对一部分样本进行推断，并将结果保存为图像文件。然后，保存模型的检查点，以便在需要时恢复模型。

另外，每隔 500 个批次，将学习率除以 10，以实现学习率的衰减。

Reflectance Restoration Net

Illumination Adjustment Net

Evaluation

3 下周工作计划

(1) 继续复现并详细分析 KinD 的项目代码，目前架构中的三个结构 (Fig. 2) 只完成其中一个结构 (Layer Decomposition Net) 的训练和代码分析，因此，下周工作计划是分析和复现后续两个结构的代码。

(2) 继续整理各类损失函数的区别。按照 Table1中整理的不同文献所采用的损失函数，弄清楚各类损失函数的作用。详细阅读 [15], [16]，再看看在损失函数方面有没有更好的想法。

References

[1] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 694–711. Springer, 2016.

[2] Chen Wei, Wenjing Wang, Wenhan Yang, and Jiaying Liu. Deep retinex decomposition for low-light enhancement. *arXiv preprint arXiv:1808.04560*, 2018.

[3] Jianrui Cai, Shuhang Gu, and Lei Zhang. Learning a deep single image contrast enhancer from multi-exposure images. *IEEE Transactions on Image Processing*, 27(4):2049–2062, 2018.

[4] Haiyang Jiang and Yinqiang Zheng. Learning to see moving objects in the dark. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7324–7333, 2019.

[5] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input/output image pairs. In *CVPR 2011*, pages 97–104. IEEE, 2011.

[6] Chen Chen, Qifeng Chen, Minh N Do, and Vladlen Koltun. Seeing motion in the dark. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3185–3194, 2019.

[7] Xiaojie Guo, Yu Li, and Haibin Ling. Lime: Low-light image enhancement via illumination map estimation. *IEEE Transactions on image processing*, 26(2):982–993, 2016.

[8] Shuhang Wang, Jin Zheng, Hai-Miao Hu, and Bo Li. Naturalness preserved enhancement algorithm for non-uniform illumination images. *IEEE transactions on image processing*, 22(9):3538–3548, 2013.

[9] Chulwoo Lee, Chul Lee, Young-Yoon Lee, and Chang-Su Kim. Power-constrained contrast enhancement for emissive displays based on histogram equalization. *IEEE transactions on image processing*, 21(1):80–93, 2011.

[10] Chulwoo Lee, Chul Lee, and Chang-Su Kim. Contrast enhancement based on layered difference representation of 2d histograms. *IEEE transactions on image processing*, 22(12):5372–5384, 2013.

[11] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020.

[12] Yuen Peng Loh and Chee Seng Chan. Getting to know low-light images with the exclusively dark dataset. *Computer Vision and Image Understanding*, 178:30–42, 2019.

[13] Ye Yuan, Wenhan Yang, Wenqi Ren, Jiaying Liu, Walter J Scheirer, and Zhangyang Wang. UG^{2+} track 2: A collective benchmark effort for evaluating and advancing image understanding in poor visibility environments. *arXiv preprint arXiv:1904.04474*, 2019.

[14] Yonghua Zhang, Jiawan Zhang, and Xiaojie Guo. Kindling the darkness: A practical low-light image enhancer. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 1632−1640, New York, NY, USA, 2019. Association for Computing Machinery.

[15] Yuming Fang, Hanwei Zhu, Yan Zeng, Kede Ma, and Zhou Wang. Perceptual quality assessment of smartphone photography. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3677–3686, 2020.

[16] Hossein Talebi and Peyman Milanfar. Nima: Neural image assessment. *IEEE transactions on image processing*, 27(8):3998–4011, 2018.